Logikai alapú ontológiakezelés

Networkshop 2003

Lukácsy Gergely* Benkő Tamás† Krauth Péter‡ Szeredi Péter§

2003

Kivonat

Napjaink informatikájában egyre nagyobb szerepet kapnak az ontológiák. Az ontológiák különösen fontosak az orvostudományban, a nyelvészetben, valamint az intelligens Web-kereső rendszerekben. Fontos szerepet kaphatnak ugyanakkor az ontológiák abban is, hogy a hagyományos információ-forrásokat, pl. adatbázisokat hatékonyabban és kényelmesebben kérdezéshessük le. A SILK eszközkészlet heterogén adatforrások integrációját támogatja, az objektumorientált módszertan elemeit (UML, OCL) ötvözve logikai alapú megvalósítási módszerekkel. Az ontológiák felhasználása iránt mutatkozó növekvő igény szükségessé tette a SILK továbbfejlesztését az ontológia-integráció irányába. A SILK rendszert alkalmassá tettük RDF sémában leírt ontológiák beolvasására, ezek konzisztenciaellenőrzésére és a sémák összehasonlítására. Az új rendszer alkalmas RDF sémában leírt ontológiák integrációjára, azaz egyesített ontológiák (részben automatikus) létrehozására. Lehetővé vált, hogy az RDF sémában leírt ontológiákat akár más típusú adatforrásból kinyert metainformációkkal is összevessük. A cikkben ismertetjük, hogy milyen kihívásokat jelentett a rendszer átalakítása, valamint bemutatjuk annak alkalmazását az orvosbiológia területén.

1. Bevezetés

Először röviden ismertetjük az ontológiák fogalmát és kapcsolatukat a szemantikus Internettel valamint a Leíró logikák világával. A következő szakaszban megvizsgáljuk a ma létező adat- és rendszerintegrálási stratégiákat, majd bemutatjuk az általunk kifejlesztett, heterogén adatforrások integrációját támogató SILK rendszert. Ezután ismertetjük, hogy hogyan képzeljük a SILK eszközkészlet továbbfejlesztését a fogalmi (ontológia) szintű integráció irányába és milyen kezdeti fejlesztéseket végeztünk el ennek érdekében. Végül bemutatjuk, hogy a végrehajtott változások mennyire tették alkalmassá a rendszert orvosi ontológiákkal kapcsolatban felmerülő problémák kezelésére.

2. Ontológiák

2.1. Bevezetés

Ontológián egy adott tudományterületen fellelhető tudás formális leírását értjük ismeretcsere és újrahasznosítás céljából. Ez jelenti fogalmak és kapcsolatok megadását, a fogalmak hierarchikus elrendezését, szabályok megadását. Egy ontológia többnyire hierarchikus szótár formájában jelenik meg. Ezeket hívjuk *pehelysúlyú* ontológiáknak.

*lukacsyg@iqsoft.hu
†benko@iqsoft.hu
‡pkrauth@kfki.com
§szeredi@iqsoft.hu

Napjaink informatikájában egyre nagyobb szerepet kapnak az ontológiák. Az ontológiák különösen fontosak az orvostudomány területén, valamint az intelligens Web-kereső rendszerekben. Ez utóbbiak működéséhez ugyanis létfontosságú, hogy a világhálón elérhető információkat a számítógép által is értelmezhető jelentéstartalommal ruházzuk fel („Semantic Web" kezdeményzés). Az orvosbiológiai ontológiákkal külön fejezetben foglalkozunk.

Fontos szerepet kaphatnak ugyanakkor az ontológiák abban is, hogy a hagyományos információforrásokat, pl. adatbázisokat hatékonyabban és kényelmesebben kérdezéshessük le.

Fontos látni, hogy egy helyes ontológia megalkotása interdiszciplináris tevékenység. Egy orvosi ontológia például igényli mind a mérnöki, mind az orvosi tudás használatát.

Az ontológiák esetében különösen fontos az, hogy formálisan legyenek leírva. Az ontológiák használatával ugyanis azt szeretnénk elérni, hogy bizonyos (eddig nem automatizálható) folyamatokat a gépek sokkal intelligensebben tudjanak elvégezni, mint korábban. Az ontológiákat leíró formalizmusoknak több fajtája különböztethető meg. A hagyományos, LISP alapú ontológia leíró nyelvekre/rendszerekre példa a KIF, Ontolingua, CycL, OKBC, KM. *Keret-alapú* rendszerek közé tartoznak a *leíró logikán* (Description Logics) alapuló rendszerek, mint például a DAML+OIL, illetve az UML alapúak, mint például a SILK rendszer. Ez utóbbi, az UML korlát leíró résznyelvét, az OCL-t (Object Constraint Language) használja fel az ontológiákban jelenlévő belső összefüggések leírására.

2.2. Az ontológiákban rejlő lehetőségek

Ebben a részben egy példa segítségével mutatjuk be, hogy miért fontosak és mire képesek az ontológiák.

Képzeljünk el egy adatforrást, amely vírusokról tartalmaz információkat. A példa szempontjából teljesen minden, hogy maguk az adatok milyen módon vannak letárolva. Az egyszerűség és szemléletesség kedvéért legyenek ezek most egy relációs adatbázisban. Az alábbi egy részlet egy ilyen, fiktív, vírusokat leíró adatbázisból:

vírusnév	**törzs**	**azonosító**	**jellemző**
Herpesvirus 1, Human	Simplexvirus	0134	Primary infection occurs mainly in infants and young children...
HIV	Lentiviruses	0022	It is acknowledged to be the agent responsible for...
SIV	Retroviridae	0023	The genetic organization of SIV is virtually identical to HIV...
...	...	...	...

1. ábra. Egy példa adatbázis, amely vírusokat ír le

Több dolgot érdemes megfigyelni. Egyrészt a fenti adatbázis (és ez jellemzően így van) szabadszöveget tartalmaz, ezért semmi garancia arra, hogy nem szerepel egy-két elírás a fenti adatok között. Persze ez mindaddig nem gond, amíg egy ilyen adatforrást orvosok használnak: ők úgyis észreveszik az elírásokat és gond nélkül kitalálják, hogy mi volt az adott rekordot létrehozó személy szándéka. Sajnos, ha az adatforrást valamilyen automatizmus keretében, gépek felhasználásával szeretnénk feldolgozni, korántsem ilyen egyszerű a helyzet.

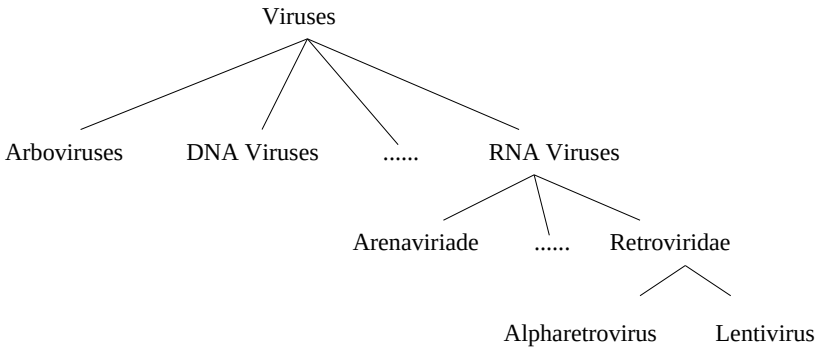
Másrészt, ha eltekintünk a nyelvtani hibáktól egy ilyen adatforrás lehet túl precíz, illetve tartalmazhat inkonzisztenciákat is. Precízségre példa, hogy a fenti adatforrás a HIV vírust a Lentivírusok családjába sorolja. Ez ugyan igaz, de ez így sajnos számos gondot okozhat. Például, ha csak a fenti adatforrásra támaszkodunk, akkor arra a kérdésre, hogy igaz-e, hogy a HIV vírus Retrovírus, nemleges lesz a válasz. Persze, hiszen honnan tudná egy gép, amely végrehajtja a lekérdezést, hogy a Lentivírusok család része a Retrovírusoknak? Nyilvánvaló ugyanakkor, hogy a precíz megfogalmazás éppenhogy követendő, mintsem elvetendő módja adatok leírásának.

Az adott szakember fejében persze biztosan jelen van egy egyszerű ontológia (a víruscsaládok egymás közti hierarchikus elrendezése) és ő ez alapján tölti fel az adatbázist. Ez azonban explicit

módon nem jelenik meg. Szükséges lehet tehát ennek az ismeretnek valamilyen formában történő megadása és így a keresés minőségének javítása.

Tekintsük most a 2. ábrán látható ontológiát. A példa a MeSH-ből [3] származik, a szemléltetés kedvéért sok helyen egyszerűsítettünk. Az előzőek alapján tudjuk, hogy a HIV vírus a Lentivírusok családjába tartozik. Most már azt is láthatjuk, hogy a Lentivírusok az RNS vírusokon belüli Retrovírusok csoportjába tartoznak. Fontos, hogy maga az ontológia nem terjed ki a konkrét vírusok leírására, hanem csak a víruscsaládok közti tartalmazási relációkat írja le.

Az ontológiát felhasználva részletesebb válaszokat tudunk adni bizonyos kérdésekre. Például, ha azt kérdezzük, hogy igaz-e, hogy a SIV vírus RNS vírus vagy hogy a HIV vírus egy Retrovírus, akkor az ontológia alapján mindkét esetben igenlő választ adhatunk. Ne feledjük, hogy mindez egyáltalán nem következik magából az adatforrásból.



2. ábra. Vírusok ontológiája

2.2.1. Ontológiák explicit megadása

Ontológiák felhasználásával kapcsolatban felmerül két fontos kérdés. Az egyik, hogy milyen formában érdemes leírni egy, a 2. ábrán látotthoz hasonló ontológiát? A másik, hogy hogyan kapcsoljuk össze az adatforrást és a hozzá kapcsolódó ontológiát?

Az első kérdésre válaszunk lehetne egy tetszőleges olyan formalizmus, amely képes leírni hierarchikus struktúrákat. A helyzet ennél bonyolultabb, mert mint említettük egy ontológia jóval több lehet, mint egyszerű hierarchikus leírás (taxonómia). Ennél fogva fontos többek között az is, hogy az adott leírás milyen mértékben támogatja az ontológián történő következtetést. Mi egy olyan formalizmust használunk ontológiák reprezentálására, amely az ún. leíró logikákon alapul. Ennek előnyeiről részletesen írunk később.

Tegyük fel, hogy sikerült megtalálni a megfelelő formalizmust az ontológiák leírására. Hátra van még, hogy 1. ábrán látható adatbázist összekapcsoljuk a vírustörzsek hierarchiáját leíró 2. ábrán megtalálható ontológiával. Bárhogyan is oldjuk meg az összekapcsolást, ügyelnünk kell arra, hogy az adatbázisban fellelhető vírustörzsek *kölcsönösen megfeleljenek* az ontológiában található vírustörzseknek. Ez nem triviális feladat. Mi történik ugyanis, ha...

- az ontológia más nyelvű, mint az adatforrás (ez például úgy fordulhat elő, hogy egy más által megkonstruált ontológiát használunk)
- az adatforrásban elírunk néhány dolgot
- ha az adatforrásban olyanra hivatkozunk, amit nem fed le az ontológia

...?

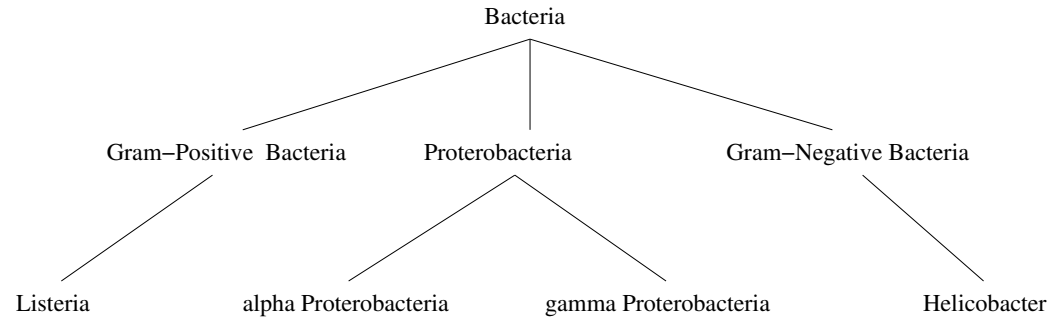
Annak, hogy pusztán az ontológia-adatforrás összekapcsolás kedvéért morfológiai elemzést végezzünk nyilvánvalóan semmi értelme. A legcélravezetőbb az, ha egy olyan nyelven írjuk le az

adatforrást, melyet kifejezetten arra terveztek, hogy ontológiákat lehessen hozzákapcsolni. Ilyen nyelv például az RDF [7], amelyre jellemző, hogy sokkal kötöttebb módon írhatjuk csak le benne ugyanazt az adatforrást, mint például, ha relációs adatbázist használnánk. RDF-ben annak a kijentésnek, hogy a HIV vírus a Lentivírusok törzsébe tartozik csak akkor van bármi értelme, ha a Lentivírusok fogalmat már definiáltuk az adatforráshoz tartozó ontológiában. Ezért mégha az adatforrás maga más nyelvű is, mint az ontológia, az adatforrás írója *kénytelen* az ontológiában definiált fogalmakat felhasználni és így automatikusan biztosítja az adatforrások gépek által történő feldolgozhatóságát.

Felmerülhet az is, hogy van-e értelme egy adatforráshoz több ontológiát is kapcsolni? A válasz az, hogy van. Képzeljük ugyanis el a 1. ábrán látható adatbázisnak azt a változatát, amely nem csak vírusokról, hanem baktériumokról is tartalmazza azt az információt, hogy az adott kórokozó mely törzsbe tartozik (3. ábra). Feltesszük, hogy a vírustörzsek ontológiájához hasonlóan a baktériumtörzsekhez is szeretnénk egy ilyet megadni. Ekkor ehhez érdemes egy külön ontológiát felvenni, hiszen egy ontológiában, egyszerre osztályozni a vírus- és baktériumtörzseket nem célszerű. Egy ilyen baktérium ontológiára mutat példát a 4. ábra. Megjegyezzük, hogy a fenti esetben praktikusan mégiscsak egy ontológiát veszünk majd fel, melyben a `vírusok` és `baktériumok` a `kórokozók` leszármazottjai. Az is igaz ugyanakkor, hogy mégha nehézségek árán is (erről szól az egyik fő mondanivalónk, az ontológiaintegráció), de mindig tudunk több ontológiából egy közöset csinálni.

kórokozónév	**törzs**	**azonosító**	**jellemző**
Herpesvirus 1, Human	Simplexvirus	0134	Primary infection occurs mainly in infants and young children...
Helicobacter pylori	Helicobacter	0322	A spiral bacterium active as a human gastric pathogen. It is a gram-negative...
HIV	Lentiviruses	0022	It is acknowledged to be the agent responsible for...
Vibrio cholerae	gamma Proteobacteria	0125	The etiologic agent of CHOLERA
Listeria monocytogenes	Listeria	0232	A species of gram-positive, rod-shaped bacteria widely distributed in nature...
SIV	Retroviridae	0023	The genetic organization of SIV is virtually identical to HIV...
...	...	...	...

3. ábra. Kibővített adatbázis: vírusokat és baktériumokat is leír



4. ábra. Baktériumok ontológiája

A fentebb említett RDF nyelv arra is lehetőséget ad, hogy több ontológiát kapcsoljunk egy adatforráshoz. Az RDF nyelvről dióhéjban annyit érdemes tudni, hogy segítségével állításokat fogalmazhatunk meg szinte bármiről. Az állításokban felhasznált fogalmak azonban definiálva kell, hogy legyenek az adott RDF forráshoz tartozó RDF sémában, azaz ontológiában. Az RDF nyelvről többet [7] alatt olvashatunk.

2.2.2. Adatforrások összekapcsolása ontológiák segítségével

Az előbbiekben példát láthattunk arra, hogy az ontológiák hogyan segíthetnek abban, hogy intelligensebb módon legyünk képesek válaszolni egy adatforrásra feltett kérdésre. Arra mutatunk most példát, hogy az ontológiák ennél sokkal többre is képesek. Nevezetesen, alkalmasak arra, hogy több adatbázist összekapcsoljanak oly módon, hogy lehetővé váljanak a komplex, egyidejű, több adatforrást is felhasználó lekérdezések.

Szóltunk arról, hogy néha érdemes egy adatforráshoz több ontológiát is kapcsolni. Egy másik érdekes dolog az lehet, amikor egy adatforráshoz más adatforráso(ko)n keresztül *közvetve* kapcsolódik egy vagy több ontológia. Ennek illusztrálásához tekintsünk újra a 1. ábrán látható adatforrásnak a megbeszélt módon módosított változatát. Ebben tehát baktériumok is szerepelnek és az adatbázishoz két ontológia is tartozik. Jelöljük ezt (3. ábra) az adatforrást `T`-vel.

Az alábbiakban bemutatunk egy új adatbázist (5. ábra), mely betegségekhez rendel kórokozókat. Vegyük észre, hogy az adatforrásban a kórokozó lehet mind vírus, mind baktérium. Értelmes kérdés lehet, hogy egy adott betegségért milyen kórokozó felelős, illetve, hogy egy adott kórokozó milyen betegséget okoz. Amennyiben sikerülne összekapcsolnunk ezt az adatforrást `T`-vel, rögtön azt is meg tudnánk mondani, hogy a kérdéses kórokozó mely törzsbe tartozik, illetve olyan kérdéseket is megválaszolhatnánk, hogy adott *kórokozó törzs* milyen betegségeket okozhat. Az így összekapcsolt adatforrást hívjuk `K`-nak.

betegség neve	**kórokozó**	**azonosító**
Cholera	Vibrio Cholerae	0012
Peptic ulcer disease	Helicobacter Pylori	0015
AIDS	HIV	0037
Meningitis	Listeria Monocytogenes	0034
African Swine Fever	African Swine Fever Virus	0222
...	...	...

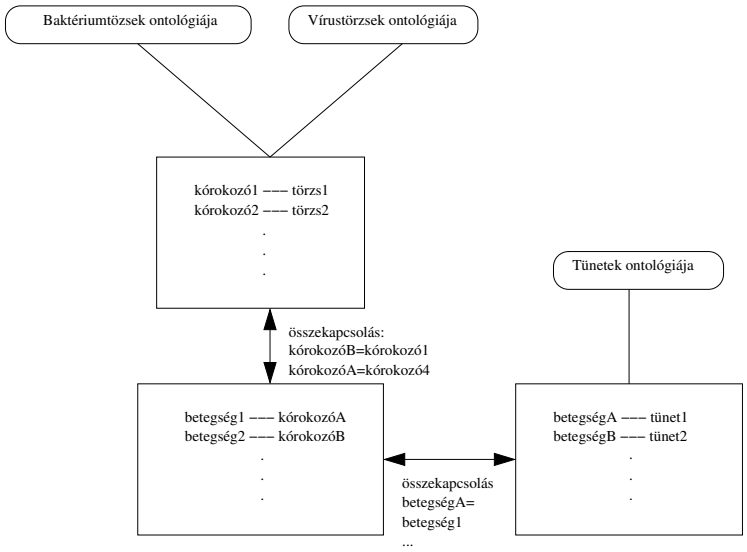
5. ábra. Egy példa adatbázis, mely betegségeket ír le

Ezt a fajta *összekapcsolást* járjuk most körül egy kicsit egy másik példán keresztül (a példa struktúráját tekintve azonos a most elmondottal). Képzeljünk el egy fiktív adatforrást, mely tünetekhez rendel betegségeket, valamint a tünetek egy hierarchikus ontológiáját (mondjuk a gyökérben az „általános rosszullét"-tel). Jogos igény lehet, hogy `K`-t és a tünet-betegség adatbázist összekapcsoljuk. Ekkor többek között olyan kérdések is megválaszolhatóvá válhatnak, hogy vajon adott tünetet milyen kórokozó okozhat vagy hogy adott kórokozó törzs milyen tünetekért felelős?

Ezt megtehetjük egy olyan szabályhalmazzal, mely a két adatforrásban megtalálható betegségeket kölcsönösen megfelelteti egymásnak. Felmerül a kérdés, hogy mi lesz az esetlegesen kimaradó (nem mindkét adatforrásban szereplő) betegségekkel, mi történik, ha nem egyszerű azonosságról van szó? Ezekről később részletesen szólunk.

Ezen szabályok segítségével egy hidat képeztünk a két adatforrás között, lehetővé téve, hogy olyan kérdéseket is megfogalmazhassunk, mely mindkét adatforrás egyidejű lékérdezését igényli. Az elmondottakat szemlélteti összefoglalóan a 6. ábra.

Adatforrások összekapcsolásakor persze felmerülnek bizonyos problémák. Többek között ilyen az adatforrások fizikai különbözősége, azaz például, hogy az egyik adatforrás relációs, a másik



6. ábra. Több adatforrás és ontológia összekapcsolása

objektum-orientált stb. Erre a későbbiekben bemutatásra kerülő SILK rendszer, illetve tetszőleges más adatintegrációs technika nyújthat megoldást.

Végezetül megjegyezzük, hogy az ontológiák azt is lehetővé teszik, hogy konkrét adatokat többféleképpen is csoportosítsunk. Erre példa, hogy a MeSH-ben a Helicobacter bakériumtörzs az általunk megadott Bakétium, Gram-negatív baktérium úton kívül elérhető a Baktérium, Proteobaktérium, Epszilon proteobaktérium úton is.

2.2.3. Ontológiák integrációja

Képzeljük egy, hogy adott két adatforrás és a hozzájuk tartozó ontológiák. Tegyük fel továbbá, hogy mindkét adatforrás azonos dolgokat ír le, például betegség-tünet megfeleltetéseket. Ekkor esetleg szeretnénk ebből a két adatforrásból egy közöset létrehozni, hiszen együtt esetleg több és hasznosabb információt tartalmaznak, mint önmagukban. A legjobb, ha úgy képzeljük el, hogy a két adatforrás két különböző kutatócsoport eredményeit tartalmazza. Ontológiák nélkül a feladat igen egyszerű. Összefésűljük a két adatbázist (a triviális duplikátumoktól esetleg rögtön meg is szabadulunk) és gyakorlatilag készen is vagyunk.

Ilyenkor azonban lehetséges, hogy inkonzisztens adatforrást hoztunk létre. Elképzelhető ugyanis, hogy az egyik adatforrás azt állítja, hogy egy adott betegség magas, míg a másik azt, hogy alacsony lázzal jár. Az is lehetséges, hogy az ellentmondás csak az ontológiák ismeretében deríthető ki. Tegyük fel például, hogy egy betegséghez az első adatforrásban az *idegesség* társul, mint tünet, míg a másikban a *levertség*. Ekkor, ha a tüneteket leíró ontológia kijelenti, hogy ezen két dolog kizárja egymást, akkor az ellentmondás felderíthető. Itt most példát látunk arra, hogy egy ontológia nem csak hierarchikus információkat tartalmazhat. Az ilyen típusu ellentmondásokat hívjuk **adat-inkonzisztenciának**.

A két adatforrás egyesítésekor szükséges az ontológiákat is integrálni. Hogyha feltesszük, hogy maguk az ontológiák külön-külön konzisztensek, akkor is elképzelhető, hogy együtt már ellentmondást tartalmaznak. Az **ontológia-integráció** még számos más gondot is felvet, erről később részletesen írunk. Triviális ontológia integrációra már mutattunk példát akkor, amikor megjegyeztük, hogy célszerű lehet a bakétium- és vírustörzseket leíró ontológiákat egy közös, `kórokozó` osztályon keresztül összekapcsolni.

A szakasz végén az ontológiaintegráció egy speciális felhasználására szeretnénk felhíni a figyelmet. Tegyük fel, hogy adott egy adatforrás és a hozzátartozó ontológia. Ekkor elképzelhető, hogy szeretnénk egy olyan „ablakot nyitni” az adatforrásra, melyen keresztül a saját ízlésünknek megfelelően láthatjuk az adatforrást. Ez megtehető a saját és az adatforráshoz tartozó ontológia megfelelő integrálásával.

2.3. Ontológiák és a szemantikus Internet

Az ontológiák kiemelkedő szerepet játszhatnak abban is, hogy a jelenleginél intelligensebb módon kereshessünk a Weben. Az alábbiakban ennek miként jét járjuk körül.

Megjegyezzük, hogy bár az olyan Internetes keresőknek, mint például a Yahoo, Open Directory Project stb. is sok közük van az ontológiákhoz, mi nem tekintjük azokat szigorúan ontológia-alapú keresőknek. Az ilyen rendszerek kézzel, emberek által kategorizált honlapokat kínálnak fel keresésre és bár a kategorizáció ontológia alapján történik, hiányzik az a fajta automatizmus, ami egy ontológia-alapú keresőt igazán érdekessé tesz számunkra. Nevezetesen az, hogy már az oldalak indexelésekor (automatikusan) történjen meg, bizonyos háttértudás alapján a lapok osztályozása.

2.3.1. Szó-ontológia keresők

Az intelligens Internetes keresés egyik lehetséges megközelítésében a feltett kérdést először morfológiailag elemezzük. Ezen elemzés eredményét felhasználva készítünk el egy olyan kérdést, mely reményeink szerint jobb, pontosabb, hasznosabb találatokat eredményez, mint az eredeti. Egy ilyen elemzéshez/átalakításhoz szükségünk lehet bizonyos **háttértudásra**, mely célszerűen egy ontológia formájában jelenik meg. Az ilyen alapú keresőket hívhatjuk **szó-ontológia** keresőknek. A név megtévesztő lehet, mert nincsen szó hagyományos értelemben vett keresőrendszerről. A szó-ontológia keresők általában egy hagyományos keresőmotorra épülnek rá, a felhasználó és a keresőgép közé.

Egy szó-ontológia kereső rendszer általában nagyon összetett. A morfológiai elemzés önmagában is komoly feladat (ugyanakkor látszik az is, hogy az ilyen keresők erősen nyelvspecifikusak), az ontológia alapján történő következtetés pedig erős logikai hátteret és komoly optimalizálási technikák használatát igényli.

A magyar Information & Knowledge Fusion (Információ és Tudás Tárház) projekt (IKF-H) részeként egy ilyen rendszer megvalósítását tűzték ki célul [1]. Következtető motornak a FaCT rendszert [2] használják.

2.3.2. Metainformációk a Weben

A szó-ontológia keresők legnagyobb hátránya, hogy az intelligenciát a Web fölé helyezik, így az Internet továbbra sem válik a gépek számára jobban érthetőbbé, mint eddig. A Szemantikus Internet elképzelés alapötlete, hogy kapcsoljunk **metainformációkat** Internetes erőforrásokhoz, mint például egy honlaphoz, a honlap egy részéhez, egy képhez, videóanyaghoz vagy bármihez, ami rendelkezik URI-val.

Metaadatnak nevezünk olyan adatot, ami adat egy adatról. A metaadat fogalma a könyvtártudományból származik. Metaadat például egy honlapról az, hogy ki készítette és mikor, egy állományról annak típusa és mérete, vagy egy képről az, hogy van rajta oroszlán, csimpánz és banán. Míg az *igazi* adat a tényleges honlap, állomány, illetve kép. Vegyük észre, hogy a határ adat és metaadat között nem igazán éles.

Az ötlet valójában nem új. A Weben a kezdetektől találhatunk metainformációkat, melyek elősegítik az Internet szemantikus oldalának kiaknázását. A `meta` nevű HTML elem segítségével metaadatot adhatunk meg honlapunkról. Ennek az elemnek két legfontosabb attribútuma a `description` és a `keywords`. Az előbbi segítségével honlapunk tartalmának összefoglalóját adhatjuk meg, melyet a keresőrobotok a találati listájukban mutatnak majd meg. A `keywords` attribútum felhasználásával specifikálhatunk olyan szavakat, melyek jellemzőek az oldalunkra, megkönnyítve így a keresést. Fontos tehát látni, hogy a szemantikus Web nem egy teljesen új elképzelés, gyökerei a kezdetekre nyúlnak vissza.

A legfőbb gond azonban az, hogy ezen `meta` elem szemantikája korlátos és nem is bővíthető. Felhasználásával csak magára a honlapra vonatkozó metainformációkat adhatunk meg. Nem tudjuk például leírni segítségével azt, hogy egy adott állomány *zene* és *énekel benne* Elton John.

A Szemantikus Internet elképzelés lehetővé teszi tehát, hogy tetszőleges Internetes erőforráshoz metainformációt társítsunk. Fontos azonban az, hogy mindenki számára (és itt elsősorban a gépekre gondoljunk) tiszta és egyértelmű legyen, hogy mit is jelent azt, hogy *zene*, valamint az,

hogy *énekel benne*. Fontos lehet továbbá a *zene* és az *énekel benne* viszonya más fogalmakhoz, mint például a *zeneszám*, *előadó* stb. azért, hogy egy esetleg másképpen megfogalmazott kérdésre is pontos választ adhassunk.

Egyszóval szükségünk van egy *ontológiára*, mely leírja egy adott területen megtalálható fogalmakat és azok összefüggéseit. Ehhez szükséges egy közös ontológia-leíró nyelv, mely illeszkedik a Webhez, szabványos és a gépek oldaláról nézve könnyen feldolgozható.

2.4. Ontológiák és a leíró logikák

A leíró logikák (Description Logics) napjaink egyik legnépszerűbb tudásreprezentációs formalizmusai. Szinte kivétel nélkül a predikátumkalkulus eldönthető részeit foglalják magukban. Ennek megfelelően egy DL következtető esetén nem fordulhat elő, hogy egy ügyesen megszerkesztett kérdés esetén végtelen ciklusba kerülünk. Ennek persze ára van: a leíró logikáknak kisebb a kifejezőerejük, mint az elsőrendű kalkulusnak. Ugyancsak probléma lehet, hogy a bonyolultabb (nagyobb kifejezőerővel rendelkező) DL rendszerek esetén a következtetési problémák exponenciális komplexitásuak (sőt, a létező algoritmusok gyakran NExpTime-beliek).

Mindezek ellenére a DL rendszerek nagyon alkalmasak adatbázis-sémák, ontológiák leírására és heterogén információforrások intergációjának támogatására is.

3. A SILK rendszer

3.1. Bevezetés

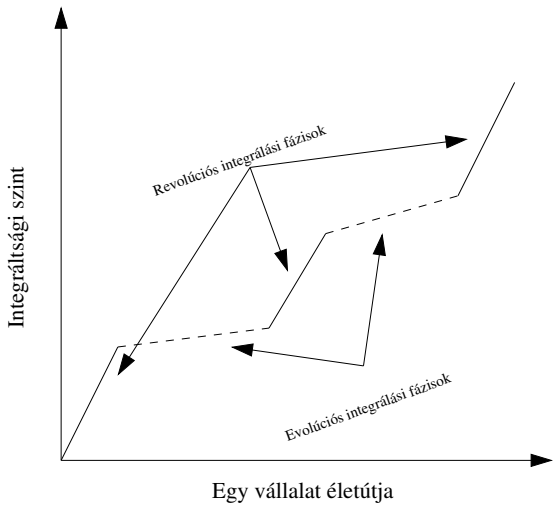
Az alábbiakban nagyon röviden bemutatjuk az elterjedt adatintegrációs stratégiákat, majd a következő fejezetben rátérünk a SILK-ben használt megközelítés és magának a SILK-nek az ismertetésére.

3.2. Adat- és rendszerintegráció

Számos stratégia képzelhető el adatintegrálásra. Az egyik az *adattárház* megközelítés. Ennek során az integrálandó adatbázisokból extrahálunk adatokat, melyeket felhasználunk egy ún. *származtatott adatbázis* létrehozására. Ez az adatbázis homogén alakban tárolja az eddig heterogén információkat, közvetlenül lekérdezhető. Ennek előnye, hogy a lekérdezés gyors. Hátrány egyrészt, hogy meg kell oldani a származtatott adatbázis frissítését, hiszen a lekérdezés nem az „eredeti" adatokon történik. Ez mindenképpen bizonyos késleltetést jelent (itt néha akár napokról van szó), ami néha elfogadható, néha nem. Másrészt, probléma, hogy sok mindent kétszer tárolunk: egyszer az eredeti adatbázisban, egyszer a származtatottban. Ennek megfelelően óriási tárolókapacitás szükséges ahhoz, hogy egy közös, nagy adatbázisban tároljunk mindent egyszerre. Összefoglalva: az adattárház megközelítés nagyon erőforrásigényes és a frissítések is gondot jelenthetnek, ugyanakkor a lekérdezések igen gyorsak tudnak lenni.

A *modelltárház* megközelítés egy ún. *virtuális adatbázist* épít. Ennek során csak az integrálandó adatforrásokat leíró modelleket, másképpen metaadatokat, szükséges valójában összeintegrálni. Relációs esetben metaadat az adatbázisban szereplő táblák nevei és adatai. XML forrás esetén a DTD stb. A metaadatok segítségével lehetővé válik az egyes adatforrások lekérdezése. A felhasználó a kérdéseit a virtuális adatbázishoz intézi: ő úgy látja, mintha valójában létezne a megfelelő adattárház. A tényleges lekérdezés az ún. *mediátor* segítségével történik. Egy mediátor a megkapott kérdést felbontja olyan részekre, melyek megválaszolásához már csak egy-egy adatforrás szükséges. Azaz egy kérdést elosztottan hajt végre.

Nagy előnye a modelltárház megközelítésnek, hogy az adatforrásokban történő esetleges változásokat valós időben képes átvezetni a rendszeren. Magyarul, ha bármi adat szintű változás történt egy adatforrásban, ez érezteti a hatását már a következő lekérdezéskor. További előny, hogy létrehozásához nem kell költsége többlet hardvereket venni, nem szükséges óriási mennyiségű információtömeggel közvetlenül dolgozni.



7. ábra. Evolúciós és revolúciós integrálás

Hátránya ugyanakkor a modelltárházaknak, hogy az összes megközelítés közül, lekérdezéskor, ennek a legnagyobb az erőforrásigénye és ennek megfelelően a teljesítménye alacsonyabb lehet bizonyos, több adatforrást átfogó kérdéseknél. Megfontolandó ugyanakkor, hogy mára megtanulta az informatikai szakma, hogy semmit sem szabad elvetni kizárólag annak sebességproblémái miatt. A fejlődés üteme szinte minden ilyen jellegű kérdést háttérbe szorít. Erre jó példa ma a JAVA és az XML óriási mértékű térhódítása. Előbbi kifejezetten lassú, utóbbi pedig terjengős. Ennek ellenére a hardverek felnőttek hozzájuk és így a fenti technológiáknak bizonyos hátrányait kiküszöbölték.

A fentieket úgy foglalhatnánk össze, hogy az adattárház megközelítés a „lassú összeintegrálás-gyors lekérdezés", míg a modelltárház megközelítés a „gyors összeintegrálás-lassú lekérdezés" elvet követi. A gyakorlatban érdemes e kettő véglet között megtalálni az aranyközéputat.

3.2.1. Rendszerintegrálás

Rendszerintegrálás kétféleképpen hajtható végre. Hirtelen, ugrásszerűen (a meglévő rendszerek kicserélésével, újra implementálásával) vagy fokozatosan, kis lépések sorozatának végrehajtásával, az eredetileg nem együttműködésre szánt rendszerek integrálásán keresztül. Előbbit *revolúciós*, utóbbit *evolúciós* integrálási stratégiának is hívják.

Egy vállalat életében a kétféle megközelítésmód ciklusan követi egymást, ahogyan azt a 7. ábra is mutatja.

3.3. A SILK rendszer bemutatása

A SILK rendszer vállalati információ-források integrálását segítő eszközök gyűjteménye. Az eszközkészletet a SILK (System Integration via Logic and Knowledge) EU projekt keretében, az 5. keretprogram IST alprogramjának támogatásával fejlesztette ki az IQSOFT Rt. francia, román és görög partnerekkel közösen. A három éves kutatási és fejlesztési munka 2002. októberében sikeresen lezárult.

A SILK rendszer egy ismeretkezelésen alapuló eszközkészlet, amelynek segítségével a heterogén adatforrások egységes rendszerré integrálása hatékonyabbá tehető. Modellalapú megközelítésmódjával lehetővé teszi mind az információforrások dinamikus lekérdezését (mediálás), mind azok homogénebb formájúra és tartalmúra történő átalakítását (integrálás). A rendszer adatforrások széles skáláját kezeli: közvetlenül képes lekérdezni relációs adatbázisokat, részben strukturált adatokat (mint az XML), valamint helyi alkalmazások vagy web-szolgáltatások formájában elérhető információ-forrásokat.

A SILK alapgondolata az, hogy a vállalat információs rendszereire vonatkozó ismereteket egy modelltárházban tárolja. Ezek az ismeretek többféle formalizmussal adhatóak meg, pl. leíró-logikában (Description Logics) vagy objektum-orientált módon (UML). A SILK az ismeretbázisban tárolja az objektum-modell jellemzőit: a fogalmak, osztályok szerkezetét, kapcsolataik leírását.

Emellett fontos szerepet kapnak az objektumokat, kapcsolatokat jellemző megszorítások, korlátok, amelyek megfogalmazására az OCL (az UML részét képező Object Constraint Language) nyelvet használja. A SILK rendszer ily módon lehetővé teszi, hogy a vállalat egyes felhasználói csoportjainak fogalmi modelljeit megfogalmazzuk és ezeket összekapcsoljuk az információs rendszerek modelljeivel.

A SILK rendszer eszközöket nyújt a modelltárház feltöltéséhez, a modellek és kapcsolatrendszerek felépítéséhez. Segítséget ad a modell- és adat-szintű ellentmondások felderítéséhez és kiküszöböléséhez, modellek hasonlóság-elemzéséhez és egyesítéséhez. Lehetőséget nyújt továbbá az heterogén információforrások lekérdezésére különböző absztrakciós szinteken (adatforrás- ill. fogalmi szinten). Az előállt modellek más integrációs módszerek (mint pl. adattárház vagy üzleti folyamat alapú technológiák) segítésére is felhasználhatók.

Lévén a SILK kizárólag modellekkel dolgozik az adatintegráción túl alkalmazások integrációjához is használható. Az egyes modellek kész, működő rendszereket reprezentálhatnak. A SILK segítségével létrehozhatunk egyesített modelleket, melyek működését tesztelhetjük. Ha meg vagyunk elégedve az eredménnyel az elkészült modellt exportálhatjuk. Ez lehet az új, integrált rendszer modellje, melyet ezekután ennek megfelelően valósíthatunk meg.

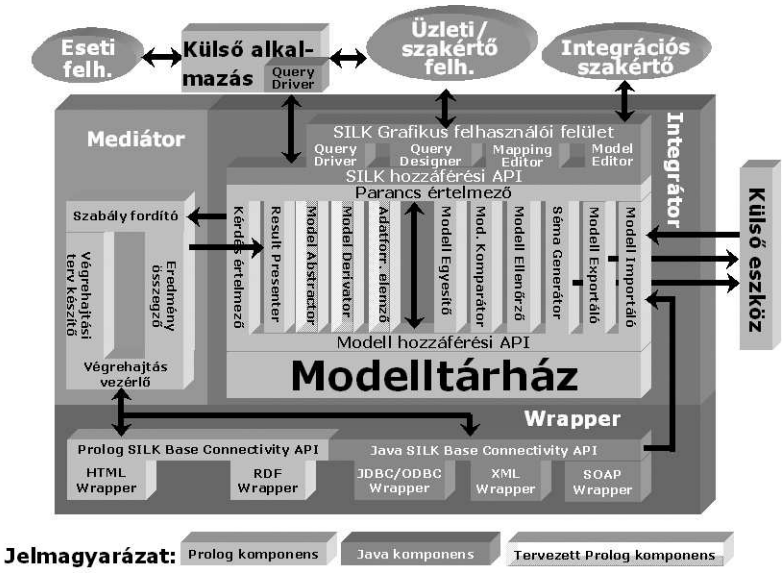
A SILK logikai alapokon működő szoftver, a rendszer belső moduljai Prolog használatával készültek (következtetés, elemzés, lekérdezés-optimalizálás). A rendszerhez készült grafikus kezelőfelületet és az adatforrásokhoz történő kapcsolódást Java környezetben implementáltuk.

Az elkészült rendszer közel 100 ezer sornyi forráskódból áll, ebből 40 ezer sornyi kód Prolog nyelven, a maradék JAVA-ban íródott. A teljes forráskód mérete (megjegyzésekkel együtt) eléri a 3 megabájtot.

3.4. A SILK felépítése

A SILK rendszer egy speciális modelltárházat használ az integrációs folyamat elvégzéséhez. A feldolgozható információforrás szinte bármi lehet. A SILK beépített támogatással rendelkezik relációs és objektum-orientált adatforrásokhoz (Oracle, DB2 stb), félig strukturált információforrásokhoz (HTML, XML stb.), valamint SOAP interfészű Web szolgáltatásokhoz.

A SILK rendszer felépítését a 8. ábrán láthatjuk.



8. ábra. A SILK rendszer felépítése

3.4.1. A SILK modelltárház felépítése

Ahhoz, hogy heterogén információforrásokat homogénebb alakba hozzunk szükséges, hogy rendelkezzünk az adatforrásokkal kapcsolatban minden lényeges információval. Ez magában foglalja az adatforrások metainformációit, az adatforrások egymáshoz való viszonyát stb. Ezen információk alkotják együtt a SILK rendszer *modelltárházát*. A strukturális metainformációkat a SILK objektum-orientált *modellekként* tárolja. A modellek között lévő relációkat *leképezések* írják le. A leképezések pontos jelentését *korlátok* határozzák meg. Ennek megfelelően a SILK modelltárház modelleket, megfeleltetéseket és korlátokat tartalmaz.

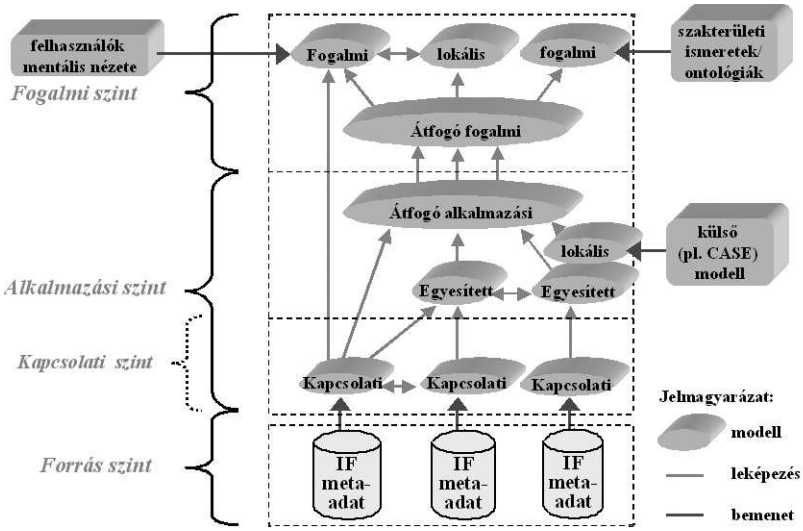
modellek: A SILK modelljei az UML csomagokhoz, és azokon belül az osztálydiagrammokhoz hasonló szervezésű, logikai információkkal kiegészített objektum modellek. A modellek egy adatforrás vagy rendszer strukturáját írják le vagy felhasználói ill. szakterületi/vállalati elvi (konceptuális) modelleket. Relációs adatforrás esetén ezek a táblák, oszlopok nevei, az oszlopok típusai, a kulcsok leírásai. Objektum-orientált adatforrás esetén a modell az osztályokat, attribútumokat, asszociációkat stb. tartalmazza. XML esetén az adatforrás modellje a DTD, míg Web szolgáltatások esetében a függvények/metódusok leírása.

korlátok: Sokszor szükséges olyan információk leírására, melyek túlmutatnak a strukturális leírásban rejlő lehetőségeken. Ilyen lehet például az, ha azt szeretnénk leírni, hogy az egyik adatforrásban szereplő név attribútum megfelel egy másikban lévő vezetéknév és keresztnév összefűzésének. Hasonló példa lehet, ha azt szeretnénk kifejezni, hogy egy bizonyos mennyiség nagyobb, mint egy másik (az igazgató fizetése mindig nagyobb, mint a beosztottaké) vagy tetszőleges, aritmetikai korlátot szeretnénk felállítani bizonyos mennyiségek között. Az ilyen korlátok leírására a SILK-ben egy OCL-szerű nyelvet használunk. A korlátokon történő következtetésekhez korlát-logikai technikákat használunk.

megfeleltetések: Amennyiben olyan kérdéseket szeretnénk feltenni, melyek megválaszolásához több információforrásból kell kinyerni az adatokat, szükséges megfeleltetések felvétele. Ezek leírják azt, hogy milyen relációban állnak egymással az információforrásokhoz tartozó modellek. Példa erre, hogy az egyik modellben szereplő `fizetés` attribútum valamilyen viszonyban áll egy másik modellben szereplő `jövedelem` attribútummal. A tényleges relációt már egy korlát írja le: például a `fizetés` attribútum értéke megfelel a `jövedelem` attribútum 1000-szeresének (ez annak felelhet meg, hogy egy cég egyik adatbázisában a fizetéseket más egységekben mérik, mint egy másikban).

A SILK modelltárházában a modellek különböző absztrakciós szinteken helyezkedhetnek el (9. ábra). A SILK két ilyen szintet különböztet meg élesen, ezeken belül további felosztások lehetségesek. Az **alkalmazási szintű** modellek egy működő rendszer, létező adatforrás struktúráját írják le. Ilyenek például a fentebb ismertetett modellek (ezeket hívhatjuk kapcsolati szintű modelleknek is utalva arra, hogy az adatforrásokhoz kapcsolódnak közvetlenül). Alkalmazás szintű modelleket hozhatunk ugyanakkor létre már meglévő modellek *egyesítésével*. Ez egy újabb kategorizálást tesz szükségessé: beszélhetünk egyesített modellekről, melyeket a modell-evolúciós folyamat eredményeképpen hoztunk létre, illetve nem egyesített modellekről. Egyesíthetünk csak kapcsolati szintű, csak már egyesített modelleket, de a kevert használati mód is megengedett. A SILK lehetőséget ad arra is, hogy alkalmazás-szintű modelleket közvetlenül olvassuk be, XMI-n keresztül, például Rational Rose-ból.

Egy **fogalmi szintű** modell a felhasználó elképzelését, nézetét írja le a rendszerről. Ezt úgy célszerű elképzelni, hogy bár az adatforrás egy bizonyos módon tárolja és strukturálja az adatokat mi esetleg egy egészen más megközelítésben is szeretnénk látni azt. Egy cégnél például egy igazgató más képet szeretne látni a vállalatról, mint a többi, az adatforrásokhoz hozzáférő dolgozó. A SILK fogalmi modell koncepciója hasonlít a relációs adatbázisoknál meglévő *nézet* fogalmához, de annál sokkal általánosabb és rugalmasabb. A fogalmi modellek leírásához a SILK felhasználja a leíró logikák egyes elemeit. Fogalmi szintű modellt a SILK grafikus felületének segítségével szerkeszthetünk és hozhatunk létre, illetve közvetlenül importálhatjuk azt, az alkalmazás szintű modelleknél elmondottaknak megfelelően.



9. ábra. A SILK Modelltárház szerkezete

3.4.2. A modell-evolúció folyamata

A modellevolúció során meglévő modellekből hozunk létre újakat. Mivel a SILK rendszerben a különböző adatforrásokhoz tartozó modellek homogén módon vannak letárolva a modelltárházban, az evolúció, illetve az egész integráció tekinthető egyfajta információmenedzsment folyamatnak is.

Az evolúció alapvető célja, hogy az eredeti modelleknél „jobb" modelleket hozzunk létre. Ezalatt két dolgot értünk. Egyrészt a modelleket közelebb vihetjük a felhasználó fogalmi modelljéhez, a világról alkotott nézetéhez. Másrészt pedig a modelleket közelebb vihetjük egymáshoz: egyesíthetjük őket. Ennek amellett, hogy megszüntetünk bizonyos redundanciákat megvan az az előnye is, hogy lehetővé válik ezáltal heterogén adatforrások felett az egyidejű, egységes lekérdezés lehetősége.

A SILK-ben a modell-evolúciós folyamat az alkalmazási környezet feltérképezésével kezdődik. Ebben a szakaszban a rendszer meghatározza, hogy mely modellekkel kell dolgoznia és inicializálja az integráció folyamatát. Az integráció nyilván létező modelleken történik. Ezek lehetnek adatforrások modelljei, valamely modellező eszközzel, kézzel készített modellek (a SILK képes XMI-ban leírt modellek beolvasására) vagy fogalmi modellek. Az integrálandó modellek általában tehát különböző absztrakciós szinteken vannak jelen.

Ezután az integrátor egy ciklikus működésbe kezd. Az első lépésnek a legelső ciklusban jellemzően nincsen még szerepe, mert beolvasáskor azt ellenőrizzük, hogy a kérdéses modell önmagában konzisztens-e.

Konzisztenciaellenőrzés: A modelleket ellenőrizzük, ellentmondás esetén interaktív módon történik a javítás. Például, legyen adott két modell. Tegyük fel, hogy az egyikben következtethető (a meglévő korlátok felhasználásával), hogy egy bizonyos attribútum értékének kisebbnek kell lennie valaminél. A másik modellben az derül ki, hogy egy másik attribútumnak nagyobbnak kell lennie az előző értéknél. Ha ekkor létezik egy megfeleltetés a két modell között, mely azt állítja, hogy a két attribútum megegyezik, akkor ez ellentmondás és az integráció nem folytatható.

Modellek összehasonlítása: A modelleket a rendszer automatikusan összehasonlítja és javaslatot tesz különböző modellek megfeleltetésére. A megfeleltetések között megkülönböztetünk *vízszintes*, illetve *függőleges* irányúakat, attól függően, hogy a modellek egymáshoz képest milyen absztrakciós szinten helyezkednek el. A függőleges irányú megfeleltetéseket sokszor *absztrakciónak* hívjuk. Az absztrakciók azt mutatják meg, hogy hogyan tudjuk közelebb

vinni a modelljeinket a fogalmi modellekhez, míg a megfeleltetések az azonos szintű modellek közti kapcsolódási pontokat írják le.

Modellek összekapcsolása: Az előző lépésben megkapott információk alapján ténylegesen létrehozzuk a modellek közti megfeleltetéseket.

Új modellek létrehozása: A kialakított megfeleltetések alapján új, egyesített modelleket hozunk létre.

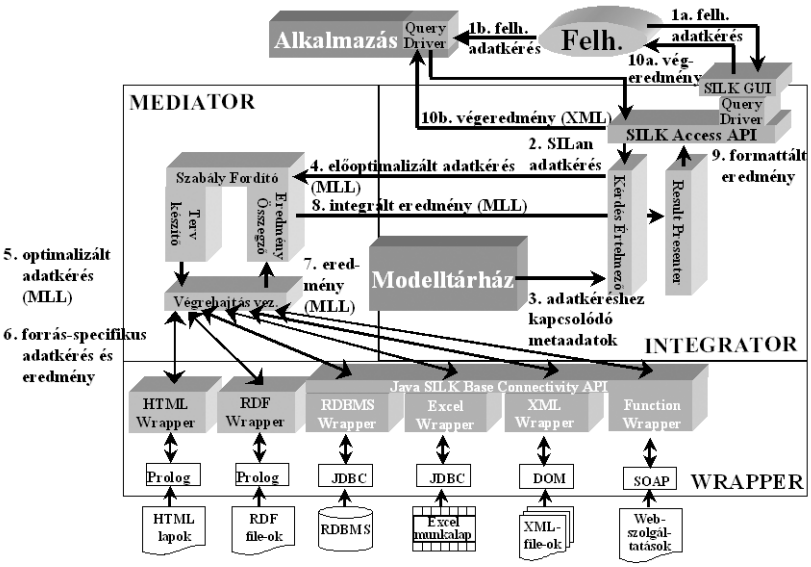
Mivel a létrehozott megfeleltetések inkonzisztenciát eredményezhetnek, újrakezdjük az eljárást. Ezt mindaddig csináljuk, míg ellentmondást találunk.

Az integrációs folyamatot a SILK nem automatikusan, hanem a felhasználóval közösen, lépésről-lépésre, interaktív módon hajtja végre.

A SILK mediátor komponense lehetővé teszi, hogy az újonnan létrehozott egyesített modelleket (vagy az újonnan létrejövő absztrakciók miatt magukat a fogalmi modelleket) lekérdezzük. A mediátor felhasználja a megfeleltetéseket, így összetett, több információforrást is felhasználó kérdések válnak megválaszolhatóvá.

3.5. A lekérdezés folyamata

A SILK-ben egy kérdés végrehajtása a 10. ábrán látható módon történik. A részleteket mellőzve egy kérdés végrehajtása lényegében a következőkből áll. A feltett magas szintű kérdést az integrátor elemzi és bizonyos átalakítások után átadja a mediátornak. A mediátor végrehajtási tervet készít, szétbontja a kérdést olyan egységekre, melyeket egy-egy wrapper már képes megválaszolni. A wrapperek végzik a tényleges adathozzáférést. Az alacsony szintről kapott válaszok végül felgyűrűznek magasabb szintre.



10. ábra. Egy lekérdezés végrehajtása

3.6. SILK példa

Az alábbiakban egy példát mutatunk a SILK integrációs folyamat egy részletére.

Legyen adott az alábbi modell a SILK modelltárházában. Ez egy fiktív vállalat pénzügyi részlegében alkalmazott adatbázis modellje lehet. A modellen belül egy osztály található. Itt az alkalmazókról tárolunk információkat, úgy mint a vezeték- és keresztnév, a fizetés, a befizetett adó

összege. Ezenfelül adott még két korlát is az értelemszerű jelentéssel és egy (összetett) elsődleges kulcs.

```
model Pénzügy {
    class Alkalmazott {
        attribute String vezetéknév;
        attribute String keresztnév;
        attribute Real fizetés, adó;
        constraint adó = 0.25*fizetés;
        constraint adó >= 10000;
        primary key (vezetéknév, keresztnév);
    };
};
```

Tekintsük ugyanennek a vállalatnak egy másik részlegét, mely szintén tárol a dolgozókról bizonyos információkat.

```
model Dolgozónyilvántartás {
    class Dolgozó {
        attribute String név;
        attribute String képesítés;
        attribute Real fizetés;
        constraint fizetés < 400;
        primary key név;
    };
};
```

Tegyük fel, hogy a két modell a fenti egy-egy osztályon kívül még tartalmaz másokat is. A SILK modell összehasonlítója ki fogja választani a fenti két osztályt, mint nagy valószínűséggel egymásnak megfelelőt. Ezt az attribútumok neveinek és típusainak hasonlósága alapján képes elvégezni. A modell összehasonlító javaslatot tesz egy alapértelmezett megfeleltetésre, melyet a felhasználó finomíthat, illetve ellenőrizhet a modell ellenőrzővel. Az alábbiakban megmutatjuk, hogy pontosan hogyan zajlik egy ilyen folyamat.

Az alapértelmezett megfeleltetés az alábbi.

```
correspondence (a: Pénzügy::Alkalmazott, d: Dolgozónyilvántartás::Dolgozó) {
    constraint d.név = unknown(a.vezetéknév,a.keresztnév)
        implies d.fizetés = a.fizetés;
}
```

Ezt a modell összehasonlító az elsődleges kulcs információkból következtette ki. Azt jelenti, hogy amennyiben igaz az, hogy a második modellbeli név valamilyen függvénye az első modellbeli vezeték- és keresztnévnek, akkor szükségképpen a két fizetés is megegyezik.

Mi tudjuk, hogy mi ez a függvény, ezért kijavíthatjuk a megfeleltetést:

```
correspondence (a: Pénzügy::Alkalmazott, d: Dolgozónyilvántartás::Dolgozó) {
    constraint d.név = a.vezetéknév.concat(a.keresztnév)
        implies d.fizetés = a.fizetés;
}
```

Tegyük fel, hogy a `concat` függvény éppen a kívánt összefűzést végzi el.

Ekkor azonban a modell ellenörző inkonzisztenciát fedez fel és megadja az ellentmondást okozó korlátokat is:

```
a.adó = 0.25*a.fizetés, a.adó >= 10000
d.fizetés = a. fizetés, d.fizetés < 400
```

Valóban, ha a két fizetés megegyezik (ez akkor igaz tehát, ha a dolgozók nevei azonosak), akkor az első sor szerint már a fizetés egynegyedének is nagyobbnak kell lennie, mint 10000. A második sorban lévő korlát viszont ellentmond ennek.

Az ellentmondás oka lehet például az, hogy a vállalat második adatbázisában a fizetést ezres egységekben tárolják. Ennek megfelelően javíthatjuk ki a megfeleltetésünket, melyben az ellenőrző már nem talál újabb ellentmondást.

```
correspondence (a: Pénzügy::Alkalmazott, d: Dolgozónyilvántartás::Dolgozó) {
    constraint d.név = a.vezetéknév.concat(a.keresztnév)
        implies d.fizetés*1000 = a.fizetés;
}
```

Miután adott a helyes megfeleltetés a modell egyesítő képes létrehozni az egyesített modellt, a megfelelő absztrakciókkal együtt.

4. A SILK továbbfejlesztése, a LOBO rendszer

4.1. Bevezetés

Lévén az ontológiák kiemelkedő szerepet játszhatnak hagyományos adatforrások hatékonyabb lekérdezésében, szükségesnek láttuk a SILK eszközkészlet továbbfejlesztését a fogalmi (ontológia) szintű integráció irányába. Ezt a rendszert LOBO-nak (LOgic Based management of Ontologies) neveztük el. A bővítés első lépéseként kifejlesztettünk egy modult, mely alkalmas RDF nyelvű adatforrások beolvasására, konzisztenciaellenőrzésére (adat és séma szinten egyaránt) és intelligens módon történő lekérdezésére. A modul a SILK-től függetlenül készült abban az értelemben, hogy a kommunikáció jól specifikált külső interfészeken keresztül történik. Ennek megvolt az az előnye, hogy a fejlesztés során nem kellett figyelembe venni az alaptechnológia által jelentett kötöttségeket és esetleges korlátozásokat. Az alábbiakban röviden bemutatjuk az RDFConsole felépítését és működését.

4.2. Az RDFConsole modul

A modul megalkotásának elsődleges célja volt, hogy lehetővé tegye a SILK számára RDF adatforrások kezelését. Ez magában foglalja az RDF adatforrás által reprezentált tudásbázis lekérdezhetőségét, az RDF adatokon való következtetést, metaadatok szolgáltatását az adatforrásról, valamint RDF adatok és sémák konzisztenciaellenőrzését. Lényegében egy olyan keretrendszer megalkotása volt a cél, mely az önmagában kevéssé kifejező RDF adatforrások fölé helyezve intelligens hozzáférést biztosít a külvilág számára, létrehozva ezáltal egy keretrendszert a hatékony RDF adatkezelés érdekében.

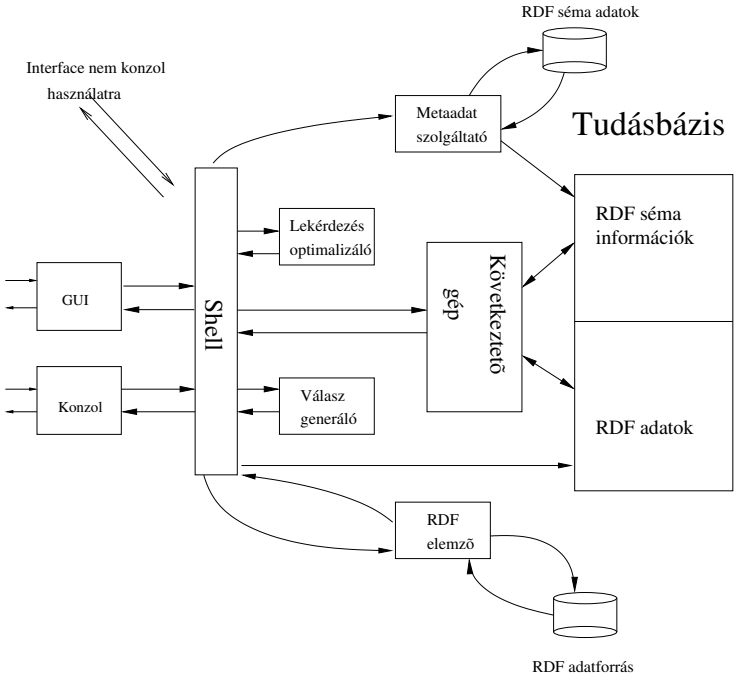
4.2.1. RDFConsole számokban

A modul Prolog és Java nyelven íródott. A teljes forráskód mérete eléri a 115 kbyte-ot. Ebből 65 kbyte Prolog, a maradék Java kód. A fejlesztés elején SWI Prolog-ot használtunk, mert ez a disztribúció könyvtár formájában elérhetővé tesz egy RDF elemzőt. Így lehetővé vált az, hogy a modul fontosabb részeinek gyorsan elkészítsük a prototípusait. Később áttértünk SICStus Prologra (ezzel párhuzamosan lecseréltük az RDF elemzőt is), így sokkal gördülékenyebben ment a SILK RDFConsole integráció.

A Java-kód a modulhoz tartozó GUI felépítéséhez és müködtetéséhez volt szükséges. A Prolog és Java közti kommunikációoz a SICStus Prolog jasper nevű könyvtárát használtuk.

4.2.2. RDFConsole architektúrája

Az 11. ábrán az RDFConsole felépítése látható. A következőben a fontosabb részek szerepét ismertetjük.



11. ábra. Az RDFConsole modul felépítése

Shell. Az RDFConsole shell az elsődleges interfész a program által nyújtott szolgáltatások eléréséhez. A shell szólítja meg az RDF elemzőt, hogy olvassa be az RDF adatforrást, illetve a metaadat szolgáltatót, hogy elemezze és építse be a tudásbázisba a séma-információkat. Az RDF elemzőtől kapott adatok alapján a shell inicializálja a tudásbázis RDF adatokat tartalmazó részét.

A shell fogadja és értelmezi az RDFLan-ban feltett kérdéseket, a lekérdezés optimalizálóval végrehatja a szükséges hatékonyságnövelő módosításokat. A shell fordul a következtető géphez egy már kioptimalizált kérdéssel, illetve a shell használja a válasz generálót a tényleges válasz elkészítéséhez.

A shell nyújtja továbbá a különböző interfészeket a külvilággal való kommunikációt elősegítendő.

Lekérdezés optimalizáló. A felhasználó vagy alkalmazás által feltett kérdések optimalizálását végzi ez a modul. Az optimális elrendezés a részkérdések megoldásszám szerinti növekvő sorrendje. Egy részkérdés megoldásainak száma függ magától a részkérdéstől statikusan, de függ a részkérdés többi részkérdéshez való viszonyától is, dinamikusan. Az optimalizációt tovább nehezíti, hogy fellép az ún. *statikus bizonytalanság* jelensége is. Ez azt jelenti, hogy egy részkérdés megoldásainak számát akkor is csak becsülni lehet, ha azt önmagában vizsgáljuk.

A lekérdezés optimalizáló modul képes a részkérdések sorrendjének cseréjére a meglévő becslések és beprogramozott heurisztikák alapján. A becsléseket futási időben állítja elő, illetve finomítja. Az optimalizáló figyelembe veszi a részkérdések dinamikus függőségeit is.

Tervezzük azt is, hogy a jövőben a modult képessé tesszük részkérdések módosítására is. Ekkor bizonyos esetekben további gyorsulás, kevesebb adatbázis-hozzáférés érhető el.

Válasz generáló. A válasz generáló feladata a következtető géptől kapott válasz mindenkori igényeknek megfelelő kozmetikázása. Erre azért van szükség, mert bizonyos helyzetekben az RDFConsole többet tud egy válaszként visszaadott erőforrásról, mint amit a kérdező fél (felhasználó vagy alkalmazás) újabb kérdések feltevése nélkül tudhat.

Következtetőgép. Az RDF sémákban leírt állítások felfoghatóak *korlátként*, illetve egyszerű leírásként is. Az RDF szabvány nem írja elő, hogy melyik szemlélet a helyes, hanem rábízza a döntést az adott alkalmazásra. Ennek megfeleően egy `rdfs:range` leírást felhasználhatunk arra, hogy eldöntsük vajon konzisztens módon használjuk-e a kérdéses tulajdonságot. Felhasználhatjuk azonban például arra is, hogy egy ismeretlen erőforrásról kikövetkeztessük annak típusát.

Az RDFConsole következtetőgépe nem korlátként tekint az RDF sémákban meglévő állításokra, hanem következtetni próbál segítségükkel. Kétfajta következtetést különböztetünk meg. Az *adatkövetkeztetésekhez* mind adat, mind sémainformáció szükséges. *Sémakövetkeztetéshez* elegendők maguk a sémák is. A SILK szempontjából adatkövetkeztetés történik a kérdések feldolgozásakor. Ilyenkor olyan válaszokat is kaphat az RDFConsole-tól, melyek nincsenek explicit leírva az adatforrásban. Sémakövetkeztetésnek két helyen van fontos szerepe. Egyrészt a SILK-nek továbbítandó metainformáció már tartalmazhat kikövetkeztetett elemeket. Másrészt az adatkövetkeztetések is felhasználhatnak kikövetkeztetett séma információkat.

Az adatkövetkeztetések figyelembe veszik az adatokon felül a sémában lévő osztály- és tulajdonsághierarchiákat, valamint a `rdfs:domain` és `rdfs:range` leírásokat. A sémakövetkeztetések a tulajdonságkorlátozásokat egyszerűsítik, illetve következtetik ki. Ez utóbbira például akkor van szükség, ha egy altulajdonság nem specifikál `rdfs:domain` és/vagy `rdfs:range` megkötést.

Tudásbázis. Az RDFConsole jelenleg saját tudásbázissal rendelketik, mely független a SILK modelltárházától. A tudásbázisban vannak tárolva az RDF adatok és sémák. Számos módon képzelhető el a tényleges tárolás, használhatunk valamilyen relációs adatbázis reprezentációt vagy magukat az RDF hármasokat is.

Az RDFConsole tudásbázisban az utóbbi módon történik a tárolás, azaz az RDF adatok és RDF séma adatok RDF hármasokként vannak reprezentálva. Ez a lehető legtömörebb (és így legcélszerűbb) módja az XML formában érkező RDF adatmodell tárolásának, amennyiben nem akarunk valamilyen más, nem szöveges leírást alkalmazni.

A tudásbázis fizikailag (dinamikus) Prolog tényállításokból áll, melyek megfelelnek az RDF hármasoknak. A lekérdezések a Prolog rendszer dinamikus adatbáziskezelőjének segítségével történnek.

Terveink között szerepel, hogy a dinamikus Prolog adatbáziskezelő helyett áttérünk a perzisztens Berkeley DB-re.

RDF elemző. Az RDF elemző feladata, hogy beolvasson egy RDF adatforrást és szolgáltassa a shellnek az adatforrás által leírt RDF adatmodellt, hármasokként reprezentálva. Ezt két lépésben hajtja végre. Az RDF állományt először egy közönséges XML elemző dolgozza fel. Az RDFConsole modul a *Xerces*[4] XML elemző C++ változatát használja. Második lépésként készítjük el az RDF hármasokat a Xercestől kapott adatok alapján.

Néhány ponton igyekeztünk követni a legújabb RDF szabványmódosításokat [5], ezért az RDFConsole nem fogad el olyan RDF adatforrásokat, amik még szintaktikailag helyesek voltak kicsit régebben, de most már nem. Erre példa, hogy az eredeti RDF szabvány kifejezetten támogatta azt, hogy egy XML elemen belül ne kelljen minden — feleslegesnek és automatikusan kitalálhatónak tűnő — névtér prefixet kitenni. Később kiderült, hogy ez számos gondot okoz és kötelezővé tették a használatukat.

Azt, hogy egy RDF adatforrás megfelel-e a szabványak a [6] helyen található, W3C-által kibocsájtott ellenőrző eszközzel tesztelhetjük.

Metaadat szolgáltató. A metaadat szolgáltató rész feladata, hogy az RDF sémákban található sémainformációkat a megfelelő formában eltárolja az RDFConsole tudásbázisában.

A metaadat szolgáltató szorosan együttműködik a következtető géppel. Ez azért fontos, mert az általa szolgáltatott metaadatokon számos következtetés érdemes elvégezni. Például, ha egy RDF tulajdonság nem specifikál értelmezési tartomány vagy értékkészlet megkötést nyilvánvalóan ki kell következtetni azt. Ezeket örökölheti az ősöktől, ugyanakkor az is elképzelhető, hogy így inkonzisztens megkötés keletkezne.

4.2.3. Adat- és séma szintű konzisztenciavizsgálat

Az RDFConsole képes az RDF adatforrások és sémák konzisztenciájának vizsgálatára. Ekkor az RDF sémákban leírt dolgokat az RDFConsole *korlátoknak* tekinti. Sajnos (sok egyéb mellett) az RDF nyelv nem támogatja az osztály/tulajdonság szintű diszjunktság fogalmát, ezért pusztán RDF alapokon nem tekinthetjük ellentmondásnak azt, ha egy helyen egy nő típusú erőforrás helyett férfi típusú szerepel stb. Emiatt szükségesnek láttuk egy olyan diszjunktságfogalom bevezetését, ami kellően erős ahhoz, hogy következtetni lehessen segítségével, ugyanakkor ne járjon az RDF szabvány szükségesnél nagyobb mértékű kibővítésével. Bevezettük az *implicit diszjunktság* fogalmát. Két RDF osztályt/tulajdonságot implicit diszjunktnak tekintünk, ha egyike sem leszármazottja a másiknak. Azaz például egy csomópont gyermekei implicit diszjunktak.

Ezen fogalom felhasználásával az alábbi konzisztenciákat képes felderíteni az RDFConsole.

adatszintű inkonzisztenciák (a sémák ehhez is kellenek)

- inkonzisztens erőforrások
- tulajdonságmegkötés megsértése
- nem létező tulajdonságok

sémainkonzisztenciák (csak a sémák kellenek hozzá)

- inkonzisztens tulajdonságok
- inkonzisztens korlátozás öröklődés

4.2.4. Az RDFConsole integrációja a SILK-hez

A 8. ábrán látható, hogy hogyan kapcsolódnak a különféle wrapperek a SILK rendszerhez. Az is látható a wrapperek egy része Java, másik részük Prolog nyelven íródott. Az RDFConsole modul a SILK-hez a hozzá készült RDF wrapperen keresztül kapcsolódik. Az RDF wrapper egy SICStus Prologban készült program, mely hidat képez a SILK és az RDFConsole interfészei között.

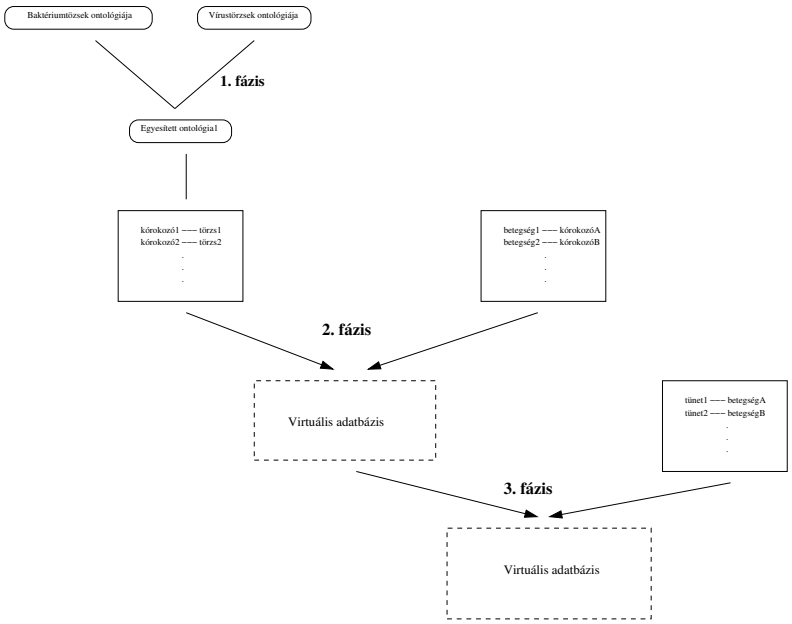
Az RDF wrapper kérésre képes az RDFConsole által szolgáltatott RDF sémainformációkat SILK csomagok/osztályok/asszociációk formájában eltárolnia a SILK modell tárházában. A wrapper speciális asszociációkat használ az RDF-beli slotok helyes reprezentálása érdekében.

Kérdés esetén a wrapper képes a SILan-ban (a SILK rendszer nyelvében) feltett kérdés RDFLan megfelelőjét előállítani és végrehajtatni azt az RDFConsole-lal.

5. A LOBO alkalmazása orvosi ontológiák területén

5.1. Általában az orvosi ontológiákról

A 90-es évektől kezdve egyre nagyobb figyelmet kap a számítástudomány és az információs technológiák egy új alkalmazási területe, az orvosbiológia. Néhány elképesztő technológiai áttörés után az orvosbiológia (és a kapcsolódó molekuláris biológia, génsebészet stb.) egyre több adatot kezdett el „termelni” és tett, valamint jelenleg is tesz elérhetővé az Interneten keresztül. E mellett kiderült az is, hogy az orvosi területen óriási mennyiségű információ vár már elektronikus feldolgozásra leletek, kórlapok, zárójelentések, beavatkozások leírása stb. formájában. Az adat-mennyiség létrehozásának sebességében bekövetkező exponenciális növekedést a jól ismert Moore-törvényhez hasonlóan lehet leírni. Például a génszekvenciákra vonatkozó információ minden 18 hónapban



12. ábra. A fázisok összefoglalása

megkétszereződik. Ennek megfelelően egyre nehezebbé válik a releváns adatok elérése, felhasználása.

Az orvosi ontológiákkal kapcsolatban felmerül néhány speciális tényező. Az ontológiák óriási méretén túl fontos látni, hogy a mögöttes biológiai komplexitás nagyobb kihívást jelent az ilyen típusú információk kezelésénél mint, mondjuk, a pénzügyi tranzakciók, időjárás-előrejelzés vagy a nukleáris fizika problémái. Ezeket a tendenciákat legéletszerűbben talán azzal lehet szemléltetni, hogy a molekuláris biológia és kémia megelőzi az áramlásdinamikát, az időjárás-előrejelzést és a virtuális nukleáris tesztelést a számítástechnikai erőforrás-igény vonatkozásában. Mindemellett még gond, hogy a meglévő tudás elosztottan van csak jelen, valamint, hogy a létező ontológiák célorientáltak, felépítésük egy adott szempont szerint történt. Erre példa, hogy egy betegség egy bizonyos orvosi ontológiában lehet tünet szerint kategorizálva, míg egy másikban kórokozó szerint. További probléma, hogy a létező ontológiák konzisztenciája kérdéses, valamint, hogy az orvosi ontológiák a természetükből adódóan nyelv-orientáltak.

Orvosi területen kiemelten fontos az ontológia-integrációs tevékenység, mert jelenleg nagy számú, nehezen összekapcsolható ontológia létezik. Ilyen az UMLS, GALEN, GENE ONTOLOGY, ICD, SNOMED stb.

5.2. A LOBO rendszer működésének bemutatása

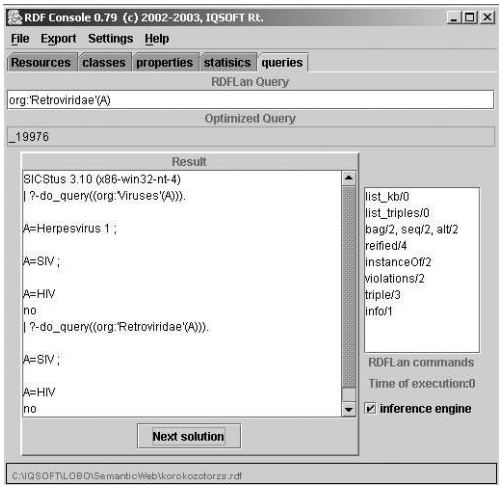
Az alábbiakban egy részletes példát mutatunk be, mely demonstrálja a LOBO-ban és az RDFConsoleban rejlő lehetőségeket. A Példa három fázisból áll. Minden fázis az adott fázisban felhasználandó adatforrások és/vagy ontológiák ismertetésével kezdődik. Az adatforrások és ontológiák leírásához az RDF nyelvet, valamint MySQL adatbázist használunk. Ezután kerül sor az adott fázisban elvégzendő feladatnak és feladat megoldásához vezető lépések sorozatának ismertetésére. A három fázist a 12. ábrán láthatjuk összefoglalva.

A cél különböző forrásokból származó orvosi adatbázisok összekapcsolása és ezzel egy olyan integrált rendszer létrehozása, mely intelligens módon képes válaszolni a feltett kérdésekre. Az

integrációt a LOBO rendszer segítségével végezzük. A kapott rendszerben (többszörösen integrált adatforrások és ontológiák gyűjteménye) az alábbi kérdésekre keressük a választ:

- mely vírusok, mely baktériumok vannak az adatbázisban?
- egy adott törzsbe kik tartoznak?
- egy adott betegséget mely baktérium- vagy vírustörzs okozhat?
- egy adott törzs milyen betegséget okozhat?
- egy tünet milyen betegségre utal?
- egy tünet milyen konrét baktériumra vagy vírusra utal?
- adott tünetet mely baktérium- vagy vírustörzs okozhat?
- adott baktérium- vagy vírustörzs milyen tüneteket okozhat?

A tesztet a SILK 2.3-as és az RDFConsole 0.79-es verziójával végeztük.



13. ábra. Lekérdezés az RDFConsole-ban

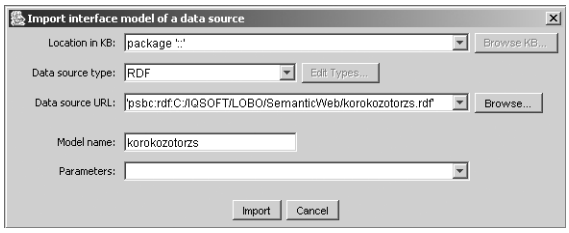
5.2.1. Első fázis - Vírusok- és bakériumok integrált ontológiája

Ebben a lépésben egy egyesített ontológiát hoztunk létre. Az egyesített ontológia egyben tartalmazza a 2. és 4. ábrán látott vírus- és baktériumtörzsek hierarcikus információit. A két ontológia nyilvánvaóan diszjunkt, ezért az integráció egy triviális lépésen (az organizmusok osztály bevezetésén) alapszik. Ezt a lépést kézzel végeztük, mert a LOBO ontológiaintegrációs képességének bemutatását egy komolyabb feladattal szeretnénk demonstrálni. Az A.1 függelékben megadjuk az egyesített ontológia RDF sémában történő leírását.

5.2.2. Második fázis

A második fázis elején a A.1 függelékben leírt ontológiát felhasználva készítettünk egy új adatforrást, mely konkrét kórokozókat sorol törzsekbe, illetve ad meg róluk néhány adatot. A besorolás megfelel a 3. ábrán látottaknak. Ez az adatforrás a A.2.1 függelékben látható. A valóságban bármilyen olyan adatforrás megfelelne, mely vírusokról tartalmaz információkat, feltéve, hogy a törzsbe történő besorolás is adott.

A kórokozókkal kapcsolatos adatforrás már önmagában is lekérdezhető. Néhány példalekérdezést mutatunk be, melyre az RDFConsole-t használtuk a SILK nélkül (13. ábra). Első kérdésünk



14. ábra. RDF-ben leírt ontológiák betöltése

az volt, hogy kik vírusok, második pedig az, hogy kik retrovírusok. Látható, hogy az RDFConsole kikövetkeztette, hogy a HIV vírus a Retrovírusok törzsébe tartozik.

Második lépésként elkészítettünk egy betegségeket nyilvántartó adatforrást, mely többek között betegségekhez rendel kórokozókat, a 5. ábrán látott adatforrás mintájára. Ez az adatforrást MySQL alapú, a létrehozásakor használt script a A.2.2 függelékben található. Ez is tetszőleges adatforrás lehetne, mely betegségekről tárol információkat, többek között a kórokozó kilétét is. Ez az adatforrás is lekérdezhető már önmagában is. Ennek bemutatását most mellőzzük.

Ennyi bevezetés után rátérhetünk a második fázis lényegére, a LOBO által támogatott komplex, egyidejű lekérdezésekre és adatforrás-integrációra. Ennek keretében bemutatjuk, hogy hogyan hoztunk létre olyan kérdéseket, melyek a fenti két adatforrást egyidejűleg kérdezik le. Végül ismertetjük, hogy hogyan integráltuk a kórokozókat és a betegségeket leíró adatforrásokat és ennek következtében hogyan vált így a lekérdezés még egyszerűbbé.

Egyidejű lekérdezés heterogén alapokon. Beolvastattuk mindkét adatforrás modelljét a LOBO rendszerbe (14. ábra). Az RDF adatforrás modelljét az RDF Wrapper szolgáltatta. A 15. ábrán látható, hogy a kórokozókat leíró adatforrás modellje 17 osztályból és egy tulajdonságból áll. A betegségeket leíró adatforrás modellje jóval kisebb.

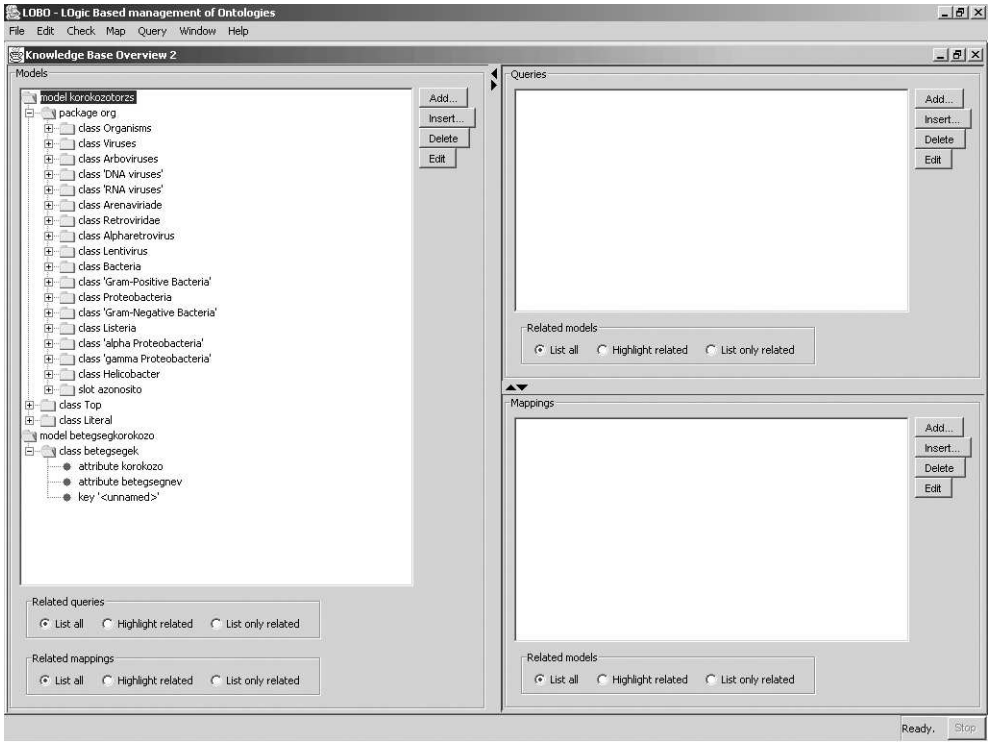
Ezután megfogalmaztunk egy olyan kérdést, melyre válaszul azokat a betegségeket kapjuk, melyeket adott törzsbe tartozó kórokozók okoznak. Azaz például azokat a betegségeket, melyeket vírusok okoznak vagy éppen retrovírusok. Nyilvánvaló, hogy ehhez mindkét adatforrás egyidejű lekérdezése szükséges. Az, hogy éppen melyik törzs által okozott betegségekre vagyunk kíváncsiak paraméterként adható meg.

A kérdés szerkesztésében a SILK *lekérdezés-tervező* modulja (16. ábra) segített bennünket. Látható, hogy a kérdés formája nagyon hasonlít a relációs nyelvek kapcsán megszokottakhoz. Fontos hangsúlyozni, hogy egy kérdésben szereplő modellek és modellelemek tetszőleges adatforrásból származhatnak. Ezért a kérdés változatlan maradhat akkor is, ha a lent lévő adatforrást esetleg teljesen újra cseréljük (például áttérünk relációsról RDF-re).

A lekérdező-tervező valójában SILan nyelvű kódot hoz létre, mely jelen esetben az alábbi (az alapértelmezett törzs az összes kórokozók törzse):

```
query MilyenBetegsegetOkoz {
    select s1.betegsegnev
    from s0: korokozotorzs::org::Torzs, s1: betegsegkorokozo::betegsegek
    where s0.URI=s1.korokozo
    with Torzs="Organisms";
};
```

A fenti kérdésre adott válaszból azonban hiányzik néhány betegség. Nevezetesen azok, melyeket olyan kórokozó okoz, mely nem szerepel A.2.1-ban. Ez lehetséges, hiszen két különálló adatforrásról van szó, senki sem garantálja azt, hogy mindkettőben ugyanazon kórokozók vannak jelen. Jelen esetben egy ilyen betegség van: az afrikai disznóláz. Tudjuk, hogy ezt az afrikai disznóláz vírus okozza. Azt persze nem tudjuk, hogy ezen vírus mely törzsbe tartozik, de azért annyit mi azért



15. ábra. Az integrálandó adatforrások ontológiái

mindenképpen tudunk róla, hogy kórokozó. Ezt érdemes tudomására hozni a LOBO rendszernek is, hiszen neki fogalma sincsen róla, hogy a második adatforrásban a kórokozók „megfelelnek" az első adatforrásban lévő kórokozóknak.

Ehhez össze kell kapcsolnunk a két adatforrásban megtalálható kórokozókat (pontosabban a kórokozók fogalmát) és ennek megfelelően létre kell hoznuk a két adatforrás egyesített modelljét.

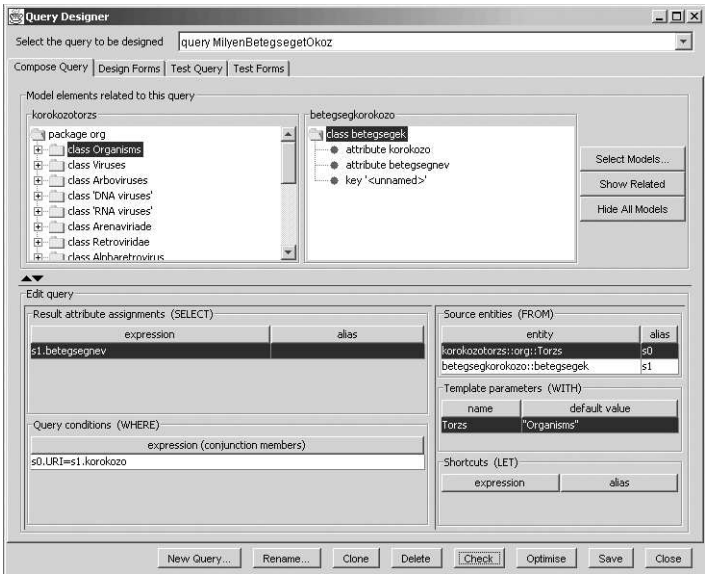
Adatforrások integrációja. Feladatunk tehát az, hogy egy olyan egyesített osztályt hozzunk létre, melyben az ott szereplő kórokozók magukban foglalják az összes olyan kórokozót, mely vagy az egyik vagy a másik adatforrásban szerepel. Ezt megtehetjük kézzel: a LOBO képes beolvasni SILan nyelvű állományokat. Használhatjuk ezenfelül a LOBO beépített *megfeleltetés tervező* komponensét. Ezen komponens segítségével interaktív módon van lehetőségünk a modellek közti negfeleltetések megadására. Itt megadhatjuk, hogy mely osztályok között adható meg valamiféle megfeleltetés, valamint megadhatjuk a megfeleltetést magát. Mindegy melyik utat választjuk, a beépített *modell egyesítő* képes ezen információk alapján létrehozni az egyesített modellt, a megfelelő absztrakciókkal együtt. A létrehozott kód az alábbi:

```
abstraction (s0: korokozotorzs::org::Organisms-> r0: egyesitett_k::Organisms) {
    constraint s0.URI=r0.nev;
};

abstraction (s0: korokozotorzs::org::Viruses-> r0: egyesitett_k::Viruses) {
    constraint s0.URI=r0.nev;
};

...

abstraction (s0: betegsegkorokozo::betegsegek-> r0: egyesitett_k::Organisms) {
```



16. ábra. A lekérdezés-tervező használat közben

```
    constraint s0.korokozo=r0.nev;
 };

model egyesitett_k {
  class Organisms {
      attribute String nev;
  };
  class Viruses: Organisms{};
  class Arboviruses: Viruses  {};
  class 'DNA viruses': Viruses  {};
  class 'RNA viruses': Viruses  {};
  class Arenaviriade: 'RNA viruses'  {};
  class Retroviridae: 'RNA viruses'  {};
  class Alpharetrovirus: Retroviridae  {};
  class Lentivirus: Retroviridae  {};
  class Bacteria: Organisms  {};
  class 'Gram-Positive Bacteria': Bacteria  {};
  class Proteobacteria: Bacteria  {};
  class 'Gram-Negative Bacteria': Bacteria  {};
  class Listeria: 'Gram-Positive Bacteria'  {};
  class 'alpha Proteobacteria': Proteobacteria  {};
  class 'gamma Proteobacteria': Proteobacteria  {};
  class Helicobacter: 'Gram-Negative Bacteria'  {};
};
```

Az absztrakciók szolgálnak arra, hogy a különböző törzsekben jelenlévő kórokozókat megfeleltesse egymásnak. Az utolsó absztrakció a MySQL adatforrásban lévő kórokozókat az egyesített modell `Organisms` osztályába sorolja. Persze lehet, hogy ennél többet tudunk az adott kórokozóról, de ennyit biztosan.

Amennyiben a 16-nak megfelelő kérdést az egyesített modellben lévő törzsekre vonatkoztatva tesszük fel, a válaszok között megtalálhatjuk az afrikai disznólázat is (17. ábra).

5.2.3. Harmadik fázis

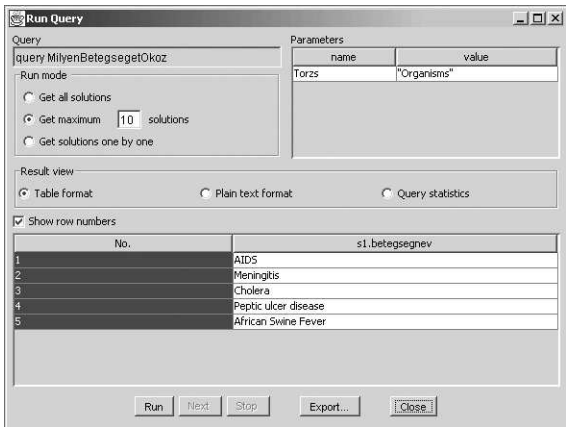
Ebben a fázisban a tünet-betegség adatforrást kapcsoljuk össze az előzőekkel. Az adatforrás XML alapú és így végül három különféle (RDF, MySQL, XML) adatbázisból áll össze az integrált rendszerünk. A tünet-betegség adatforrást az A.3 függelékben láthatjuk.

Természetesen előfordulhat, hogy olyan betegséget határoz meg az XML adatforrás tünet alapján, melyet a MySQL adatbázis nem tartalmaz. Ennek megfelően érdemes létrehozni egy egyesített modellt, mely a betegségeket nyilvántartó adatforrásban lévő betegségek és az XML adatforrás által leírt betegségek unióját reprezentálja. Ezt az alábbi SILan kóddak írhatjuk le:

```
abstraction (s0: betegsegkorokozo::betegsegek-> r0: egyesitett_b::Betegsegek) {
     constraint s0.betegsegnev=r0.nev;
 };

abstraction (s0: tunetbetegseg::tunet-> r0: egyesitett_b::Betegsegek) {
     constraint s0.betegsegnev=r0.nev;
 };

model egyesitett_b {
    class Betegsegek {
        attribute String nev;
    };
};
```



17. ábra. Milyen betegségeket okozhatnak a kórokozók?

5.2.4. Lekérdezések

A célul kitűzött kérdések közül most a legérdekesebbeket mutatjuk be. Elsőként arra vagyunk kiváncsiak, hogy egy adott tünet milyen kórokozótörzsre utal. Az ennek megfelelő SILan kérdés a következő:

```
query MilyenKorokozotorzsreUtalEgyTunet {
    select s0.typeOf()
    from s0: egyesitett_k::Organisms, s1: betegsegkorokozo::betegsegek,
    s2: tunetbetegseg::tunet
    where s0.nev=s1.korokozo and s1.betegsegnev=s2.betegsegnev and s2.tunetnev=Tunetnev
    with Tunetnev="T14";
```

```
};
```

A kérdés nagyon egyszerű szerkezetű. A paraméterként megkapott tünetnévhez megkeresi a hozzá tartozó betegségnevet és ez alapján a betegségeket leíró adatforrásban kikeresi a megfelelő kórokozót. Ennek a kórokozónak aztán meghatározza a törzsét.

Az alábbi kérdés az előző fordítottja. Azt kérdezzük, hogy egy adott törzs milyen tüneteket okozhat:

```
query AdottTorzsnekMilyenTuneteiVannak {
    select distinct s2.tunetnev
    from s0: egyesitett_k::Torzs, s1: betegsegkorokozo::betegsegek,
    s2: tunetbetegseg::tunet
    where s0.nev=s1.korokozo and s1.betegsegnev=s2.betegsegnev
    with Torzs="Organisms";
};
```

Azt érdemes megfigyelni, hogy a lekérdezésekben az egyesített modelleket használtuk.

6. Összefoglalás

Célunk a SILK rendszer továbbfejlesztése a fogalmi szintű integráció irányába. Ehhez első lépésként létrehoztuk az RDFConsole rendszert, mely képes RDF adatokat és sémákat intelligens módon kezelni. Ezt az alkalmazást kapcsoltuk össze a SILK rendszerrel az RDF wrapper segítségével. Ennek megfelelően a SILK által feldolgozható adatforrások repertoárja kibővült az RDF adatokkal. Az RDF sémákat, az RDF adatforrásokra vonatkozó metainformációknak tekintjük. Ezen metainformációk SILK modellekként élnek a rendszeren belül. A SILK felépítéséből adódóan egy RDF sémának megfelelő modell pontosan úgy van reprezentálva, mint pl. egy relációs adatforrásnak megfelelő. Így lehetővé válik, hogy a SILK segítségével olyan egyesített modelleket (ontológiákat) hozzunk létre, melyek egyszerre több különböző adatforrást, most már akár RDF-et is, reprezentálnak.

Továbblépési lehetőségként látjuk a DAML+OIL források támogatásának bevezetését. Az RDF sémák ugyanis lényegében csak taxonómiák leírására alkalmasak. Ennek megfelelően fontosnak érezzük, hogy a SILK képes legyen „teljes jogú" ontológiákkal is dolgozni. Úgy gondoljuk, összhangban a szakma nagy részével, hogy ilyenek leírására a (amúgy RDF alapú) DAML+OIL nyelv kifejezetten alkalmas.

Hivatkozások

[1] *Information & Knowledge Fusion Project*
http://ikf.mit.bme.hu

[2] *FaCT (Fast Classification of Terminologies)*
http://www.cs.man.ac.uk/ horrocks/FaCT/

[3] *Medical Subject Headings*
http://www.nlm.nih.gov/mesh/meshhome.html

[4] *Xerces: XML parsers in Java and C++*
http://xml.apache.org/

[5] *RDF Issue Tracking*
http://www.w3.org/2000/03/rdf-tracking/

[6] *RDF Validation Service*
http://www.w3.org/RDF/Validator/

[7] *Resource Description Framework (RDF) Model and Syntax Specification*
http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

A. Függelék: a felhasznált RDF példák forráskódjai

A.1. Első fázis - kórokozók ontológiája

```
<!--Baktériumok és vírusok ontológiája 2003.02.09-->

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdfs:Class rdf:ID="Organisms">
    <rdfs:comment>The most general class</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Viruses">
    <rdfs:comment>The most general virus class</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Organisms"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Arboviruses">
    <rdfs:comment>Arthropod-borne viruses. A non-taxonomic designation...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Viruses"/>
</rdfs:Class>

<rdfs:Class rdf:ID="DNA viruses">
    <rdfs:comment>Viruses whose nucleic acid is DNA</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Viruses"/>
</rdfs:Class>

<rdfs:Class rdf:ID="RNA viruses">
    <rdfs:comment>Viruses whose genetic material is RNA</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Viruses"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Arenaviriade">
    <rdfs:comment>A family of RNA viruses naturally infecting rodents...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#RNA viruses"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Retroviridae">
    <rdfs:comment>Family of RNA viruses that infects birds and mammals...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#RNA viruses"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Alpharetrovirus">
    <rdfs:comment>genus of the family RETROVIRIDAE with type C morphology...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Retroviridae"/>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="Lentivirus">
    <rdfs:comment>A genus of the family RETROVIRIDAE consisting of...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Retroviridae"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Bacteria">
    <rdfs:comment>The most general bacteria class</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Organisms"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Gram-Positive Bacteria">
    <rdfs:comment>Bacteria which retain the crystal violet stain when...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Bacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Proteobacteria">
    <rdfs:comment>A class of bacteria consisting of the purple bacteria...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Bacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Gram-Negative Bacteria">
    <rdfs:comment>Bacteria which lose crystal violet stain but are stained...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Bacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Listeria">
    <rdfs:comment>A genus of bacteria which may be found in the feces of...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Gram-Positive Bacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="alpha Proteobacteria">
    <rdfs:comment>A group generally comprised of those members of the...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Proteobacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="gamma Proteobacteria">
    <rdfs:comment>A group of the proteobacteria comprised of facultatively...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Proteobacteria"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Helicobacter">
    <rdfs:comment>A genus of gram-negative, spiral-shaped bacteria that is...</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Gram-Negative Bacteria"/>
</rdfs:Class>

<rdf:Property rdf:ID="azonosito">
    <rdfs:domain rdf:resource="#Organisms"/>
</rdf:Property>

</rdf:RDF>
```

A.2. Második fázis

A.2.1. Kórokozók besorolása

```
<!--Kórokozó-törzs adatforrás 2002.08.08-->

<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:org="bacivirus.rdfs#">

<rdf:Description rdf:about="Herpesvirus 1">
    <rdf:type rdf:resource="bacivirus.rdfs#Viruses"/>
    <org:azonosito>0134</org:azonosito>
</rdf:Description>

<rdf:Description rdf:about="Helicobacter pylori">
    <rdf:type rdf:resource="bacivirus.rdfs#Helicobacter"/>
    <org:azonosito>0322</org:azonosito>
</rdf:Description>

<rdf:Description rdf:about="HIV">
    <rdf:type rdf:resource="bacivirus.rdfs#Lentivirus"/>
    <org:azonosito>0022</org:azonosito>
</rdf:Description>

<rdf:Description rdf:about="Vibrio cholerae">
    <rdf:type rdf:resource="bacivirus.rdfs#gamma Proteobacteria"/>
    <org:azonosito>0125</org:azonosito>
</rdf:Description>

<rdf:Description rdf:about="Listeria monocytogenes">
    <rdf:type rdf:resource="bacivirus.rdfs#Listeria"/>
    <org:azonosito>0232</org:azonosito>
</rdf:Description>

<rdf:Description rdf:about="SIV">
    <rdf:type rdf:resource="bacivirus.rdfs#Retroviridae"/>
    <org:azonosito>0023</org:azonosito>
</rdf:Description>

</rdf:RDF>
```

A.2.2. Betegségek leírása

```
create database betegsekkorokozo;

use betegsegkorokozo;

create table betegsegek (
    betegsegnev varchar(30) primary key not null,
    korokozo varchar(30)
    );

insert into betegsegek values ('Cholera','Vibrio Cholerae');
```

```
insert into betegsegek values ('Peptic ulcer
disease','Helicobacter Pylori'); insert into betegsegek values
('AIDS','HIV'); insert into betegsegek values
('Meningitis','Listeria Monocytogenes'); insert into betegsegek
values ('African Swine Fever','African Swine Fever Virus');
```

A.3. Harmadik fázis - tünetek

```
<?xml version="1.0"?> <!DOCTYPE tunetek [
  <!ELEMENT tunetek (tunet*)>
  <!ELEMENT tunet (tunetnev,betegsegnev,azonosito)>
  <!ELEMENT tunetnev       (#PCDATA)>
  <!ELEMENT betegsegnev    (#PCDATA)>
  <!ELEMENT azonosito      (#PCDATA)>
  ]>
<tunetek>
    <tunet>
        <tunetnev>T21</tunetnev>
        <betegsegnev>Cholera</betegsegnev>
        <azonosito>001</azonosito>
    </tunet>
    <tunet>
        <tunetnev>T23</tunetnev>
        <betegsegnev>Peptic ulcer disease</betegsegnev>
        <azonosito>031</azonosito>
    </tunet>
    <tunet>
        <tunetnev>T12</tunetnev>
        <betegsegnev>Peptic ulcer disease</betegsegnev>
        <azonosito>032</azonosito>
    </tunet>
    <tunet>
        <tunetnev>T22</tunetnev>
        <betegsegnev>AIDS</betegsegnev>
        <azonosito>055</azonosito>
    </tunet>
    <tunet>
        <tunetnev>T14</tunetnev>
        <betegsegnev>Meningitis</betegsegnev>
        <azonosito>112</azonosito>
    </tunet>
    <tunet>
        <tunetnev>T18</tunetnev>
        <betegsegnev>Encephalitis</betegsegnev>
        <azonosito>112</azonosito>
    </tunet>
 </tunetek>
```

Tartalomjegyzék