



Eötvös Loránd Tudományegyetem
Informatikai Kar
Információs Rendszerek Tanszék

VisualQuery:

vizuális lekérdezés szerkesztő program

TDK dolgozat

Témavezető
dr. Kiss Attila Elemér
tanszékvezető egyetemi docens

Készítette
Rácz Gábor
MSc 2. évfolyam

Budapest, 2011

A dolgozat a TÁMOP 4.2.1/B-09/1/KMR-2010-0003 számú pályázatának
támogatásával készült.

Tartalomjegyzék

1. Bevezető	4
2. Szemantikus Web	6
2. 1. RDF	7
2. 2. Szókészlet-leíró nyelvek	11
2. 2. 1. RDF Schema ¹²	12
2. 2. 2. OWL	14
2. 2. 3. Következtetések	15
2. 3. SPARQL	16
2. 3. 1. WHERE záradék	18
2. 3. 2. FROM záradék	20
2. 3. 3. Végpontok	21
2. 3. 4. Lekérdezés típusa	22
2. 3. 5. Eredménymódosítók	24
3. Probléma felvetése	25
3. 1. Vizuális lekérdezés szerkesztő programok	27
3. 2. Kapcsolódó munkák	28
3. 2. 1. gFacet	28
3. 2. 2. Nitelight	29
3. 2. 3. iSPARQL	31
3. 2. 4. SPARQLViz	32
4. VisualQuery	34
4. 1. Felhasználói felület	35
4. 1. 1. Query fül	35
4. 1. 2. Menü	35
4. 1. 3. Lekérdezés típusok	36
4. 1. 4. Vázon	37
4. 1. 5. Tulajdonság ablak	38
4. 1. 6. Options fül	39
4. 2. Funkciók	39
4. 2. 1. Prefix	39
4. 2. 2. FROM záradék	40
4. 2. 3. WHERE záradék	40

4. 2. 4. Eredménymódosítók.....	44
4. 2. 5. Lekérdezés típusa	44
4. 3. Technikai megvalósítás	45
4. 3. 1. Programozási nyelv	45
4. 3. 2. Gráfkezelő API.....	46
4. 3. 3. Csomagok, osztályok	47
5. Összegzés	50
6. További tervek.....	52
7. Irodalomjegyzék.....	53
8. Függelékek	54
8. 1. Bor ontológia.....	54
8. 2. Futtatáshoz szükséges teendők.....	55
8. 2. 1. Appletként	55
8. 2. 2. Asztali alkalmazásként	56
8. 2. 3. Webes elérhetőség.....	56

1. Bevezető

Napjainkban egyre inkább teret hódít magának a Szemantikus Web, amely azt tűzte ki célul, hogy az interneten található és az időközben keletkező hatalmas mennyiségű adatot a számítógép által értelmezhetővé tegye, szemantikával lássa el.

Az ehhez szükséges technológiát a World Wide Web Consortium (röviden: W3C) dolgozta ki, fejleszti és tartja karban. A szervezet a következőképpen fogalmazta meg a célját:

„A W3C célja, hogy az internet az együttműködést, a tudás megosztását, és ezáltal a globális szinten megvalósuló bizalom kiépülését szolgálja.”¹

Mindemellett a szervezet víziója az „Egytelen Web” kialakítása, ezt hozza közelebb a Szemantikus Web.

A dolgozat célja, hogy áttekintést adjon a Szemantikus Web alapvető céljairól, technikáiról. Továbbá, hogy bemutassa a dolgozat keretében elkészült VisualQuery alkalmazást, mely egy vizuális lekérdezőszerkesztő program speciálisan a SPARQL nyelvhez.

A SPARQL szerkesztő programok többsége szöveges, a jelenlegi vizuális lekérdezőszerkesztők pedig még nem kiforrottak. A VisualQuery több formázási lehetőséget biztosít (a csúcsok méretezhetőek, az élcímkék helyzete változtatható), így átláthatóbb ábrákat lehet összerakni, mint az elérhető alkalmazások nagy részében. Valamint külön táblázatokban összeszedve láthatóak a prefixek és az adatforrások, ezáltal azok könnyen áttekinthetőek.

Tehát az alkalmazás megkönnyíti a SPARQL lekérdezések létrehozását, melyeket vizuális elemekből építhetünk fel. Az ilyen formában elkészített lekérdezések a későbbiekben könnyebben megérthetőek, ezért dokumentációs célokra is használhatóak.

A dolgozat a következőképpen épül fel. A 2. fejezetben megismerkedhetünk a Szemantikus Web technológiával. A célokkal, a fontosabb építőelemekkel, mint az Erőforrás Leíró Keretrendszer, a szókészlet-leíró nyelvek (RDFS, OWL), a SPARQL lekérdezőnyelv.

¹ Forrás: A W3C küldetése: <http://www.w3c.hu/forditasok/w3cmision.html>

Majd a 3. fejezetben szó lesz az alapvető problémáról, mely a dolgozatot inspirálta, vagyis arról, hogy hogyan lehet a SPARQL lekérdezéseket elrejteni a hozzá nem értő felhasználók elől, hogyan lehet megkönnyíteni azok összeállítását.

A kínálgató megoldások közül a vizuális lekérdezésszerkesztő programokat részletesen is szemügyre vesszük: megnézzük hogyan működnek, és milyen alkalmazások érhetők el a témában.

A 4. fejezetben pedig a problémának egy megoldását ismertetem, a VisualQuery-t, egy ilyen vizuális lekérdezésszerkesztő programot, melyet a dolgozat mellett készítettem el. Szó lesz a felhasználói felületről, a megvalósított funkciókról, valamint a megvalósítás részleteiről: a választott programozási nyelvről, gráf kezelő programkönyvtárról, az osztályok szerkezetéről. Ezek után az 5. fejezetben egy összegzést olvashatunk, melyben összefoglalom a dolgozatban leírtakat, a tapasztalatokat, az eredményeket.

Végül a 6. fejezetben leírom a további terveket, fejlesztési irányokat, feladatokat, melyek a VisualQuery készítése során felmerültek, és még nem kerültek megvalósításra.

2. Szemantikus Web

Manapság egyre többször hallani, egyre többet lehet találkozni a Szemantikus Web kifejezéssel. De mit is jelent ez tulajdonképpen, hogyan lehet szemantikus az internet?

A W3C a következőt írja:

„The Semantic Web is about two things. It is about **common formats for integration and combination of data** drawn from diverse sources, where on the original Web mainly concentrated on the interchange of documents. It is also **about language for recording how the data relates to real world objects**. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing.”²

Vagyis a Szemantikus Web két dologról szól:

1. Közös formátumot biztosít a változatos forrásokból – például weboldalakról, adatbázisokból, szöveges dokumentumokból – származó adatok integrációjára és kombinációjára, valamint
2. definiál egy nyelvet, mellyel leírhatjuk, hogy az adatok miképpen kapcsolódnak a világbeli objektumokhoz.

Egy másik forrás szerint:

„A szemantikus web lényege, hogy az interneten található információkat a keresőrendszerek ne csupán sekélyesen, hanem mélyen, valódi tartalomként kezeljék.”³

Gondoljunk csak bele! Rákeresünk például a francia borokra. Ekkor azokat is meg szeretnénk találni, amelyekről nem szerepel explicit leírva, hogy franciák, csak annyi, hogy Bordeaux-ból származnak. Számunkra világos, hogy ekkor franciák is (hiszen Bordeaux Franciaország területén található), de a keresőknek ezt az extra tudást valahogy át kell tudnunk adni.

Erre egy megoldás, hogy szemantikus annotációkkal látjuk el a forrásokat. Ezek az annotációk plusz információt jelentenek a keresőprogramoknak, melyek ezáltal képesek lesznek hatékonyabban értelmezni az adatokat.

² Forrás: W3C Semantic Web Activity: <http://www.w3.org/2001/sw/>

³ Forrás: Gottdank Tibor: Szemantikus web 118. oldal (ComputerBooks Kiadó, Budapest, 2005.)

Nem elég azonban pusztán annotálni szövegeket. Valahogyan azt is le kell tudnunk írni, hogy az annotációk mit jelentenek, milyen kapcsolatban állnak egymással és magával a megcímkézett objektummal.

A W3C Szemantikus Web technológiája ezeknek a feladatoknak az elvégzésére nyújt eszközöket.

A fejezet további részében ezeket az eszközöket tekintjük át részletesebben.

2. 1.RDF⁴

„Az RDF (Resource Description Framework) egy adateleíró nyelv, amellyel erőforrásokról szóló információkat ábrázolhatunk a weben.”⁴

Vagyis a fentebb említett szemantikus annotációk megadására tökéletesen alkalmas. Továbbá mivel az erőforrás fogalma általánosítható, gyakorlatilag szinte bármit tekinthetünk erőforrásnak: személyeket, weboldalakat, könyveket, számítógépeket, intézményeket, ...

Maga az elv nagyon egyszerű: minden erőforráshoz rendelünk egy egységes azonosítót (Uniform Resource Identifier, URI), majd egyszerű állításokkal kapcsoljuk össze az erőforrásokat. Az állításokat hármassok formájában adjuk meg (alany, állítmány, tárgy) vagy (alany, tulajdonság, érték) formájában.

Hadd szemléltessem néhány egyszerű példával ezeknek az állításoknak a működését!

(A példák egy része a W3C által kiadott étel és bor ontológiára⁵ vonatkozik, az ontológia osztályhierarchiája és a fontosabb tulajdonságok megtalálhatók a függelékek között.

A másik része pedig a DBpedia-ra⁶ támaszkodik, amely a Wikipedia adatainak RDF átírata.)

1. példa (bor ontológia)

Tegyük fel, hogy:

- Van egy bor, amit a <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#MountadamPinotNoir> URI egyértelműen azonosít.

⁴ Pataki Ernő. Az RDF bevezető tankönyve. (2005 fordítás) <http://www.w3c.hu/forditasok/RDF/REC-rdf-primer-20040210.html>

⁵ W3C bor ontológia elérése: <http://www.w3.org/TR/owl-guide/wine.rdf>

⁶ DBpedia leírás: <http://dbpedia.org/About>

- Azt a tulajdonságot, hogy egy objektum cukortartalom alapján száraz, félszáraz vagy édes a `http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#hasSugar` URI azonosítja;
- A `http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Dry`, a száraz cukortartalmat írja le.

Ekkor annak a leírása, hogy

„A bor cukortartalma alapján száraz.”

A következő hármassal adható meg:

```
<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#MountadamPinotNoir>
<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#hasSugar>
<http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#Dry>
```

A hármás megfelel az (alany, tulajdonság, érték) sémának.

2. példa (DBpedia)

Tegyük fel, hogy:

- A zsiráfot a `http://dbpedia.org/resource/Giraffe`;
- Az állatokat a `http://dbpedia.org/ontology/Animal`;
- Azt, hogy valami beletartozik egy csoportba (osztályba) a `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` URI azonosítja.

Ekkor annak a leírása, hogy

„A zsiráf az egy állat.”

A következő hármassal adható meg:

```
<http://dbpedia.org/resource/Giraffe>
< http://www.w3.org/1999/02/22-rdf-syntax-ns#type >
<http://dbpedia.org/ontology/Animal>
```

A példákön több dolgot is megfigyelhetünk

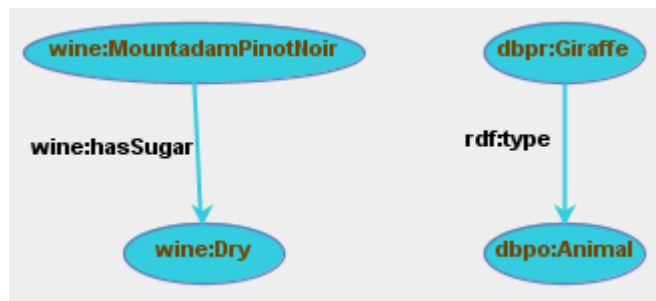
- Maguk az erőforrás-leírók nagyon hosszúak is lehetnek. Éppen ezért kidolgozták annak a lehetőségét, hogy a gyakran előforduló, vagy túl hosszú URI-k prefixeit egy rövid karaktersorozattal helyettesítsük. Persze ez is leírható a hármások segítségével, egy speciális jelöléssel:

@prefix dbpr: <http://dbpedia.org/resource/>

- A hármasok a szerkezetükből adódóan jól reprezentálhatók irányított gráfokkal: az alany az él kezdőpontjának, az állítmány az élnek és a tárgy az él végpontjának a címkéje.

A fenti példák gráfként, az alábbi prefixekkel:

- wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
- dbpr: <http://dbpedia.org/resource/>
- dbpo: <http://dbpedia.org/ontology/>
- rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>



1. ábra Egyszerű RDF gráfok

Természetesen több állítást is összefűzhetünk (ekkor az állítások közé '.'-ot kell tenni), ezáltal bonyolultabb gráfok keletkezhetnek: egy csúcsból több él is kiindulhat, illetve az egyik él végpontja kezdőpontja lehet egy másik élnek.

3. példa (DBpedia)

Vezessünk be ehhez két új prefixet!

- foaf: <http://xmlns.com/foaf/0.1/>
- rdfs: <http://www.w3.org/2000/01/rdf-schema#>

Ekkor az állítás:

dbpr:Giraffe rdf:type dbpo:Animal .

dbpr:Lion rdf:type dbpo:Animal .

dbpr:Giraffe foaf:name „Giraffe”@en .

dbpo:Animal rdfs:label „animal”@en .

Vagyis:

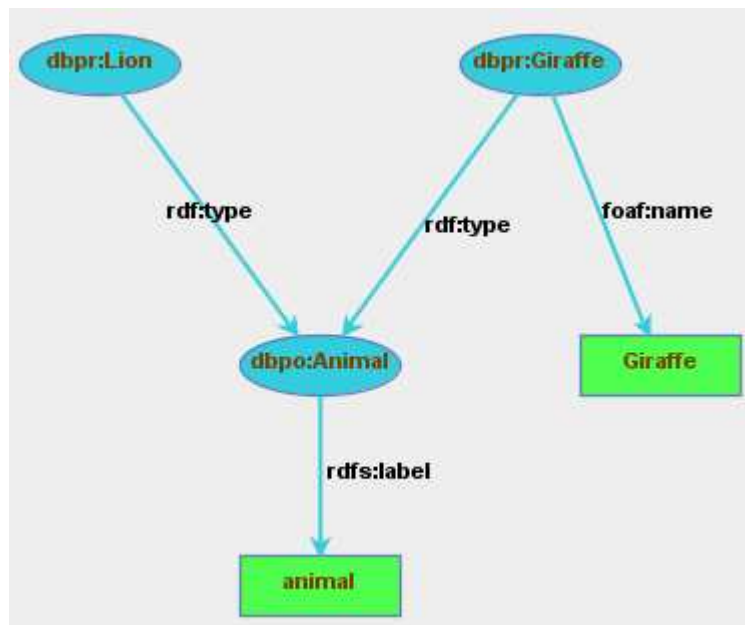
A zsiráf az egy állat.

Az oroszlán az egy állat.

A zsiráf neve „zsiráf”.

Az állatot reprezentáló csúcs címkéje „animal”.

Mindez gráfként:



2. ábra Összetett RDF gráf

Az ábrán látható, hogy többféle csúcs is előfordul. Valóban a csúcsoknak, éleknek van típusa. Háromféle típust különböztetünk meg:

- RDF URI referencia: erőforrás-azonosítók leírására (kék csúcsok az ábrán)
- Literál: sima szöveges adat leírására (zöld csúcs az ábrán)
- Blank node (üres csúcs): véges halmaz reprezentálására, URI-k és literálok összefogására alkalmas.

A hármasok tagjaira a típusok alapján a következő megszorítások érvényesek:

- Alany: URI vagy blank node
- Állítmány: URI
- Tárgy: URI, blank node vagy literál

Tehát ezen az elven működik az RDF. Az objektumokat erőforrásként értelmezzük és hármasok formájában írjuk le a tudásunkat az objektumról.

A leírásokat fájlokba menthetjük, erre különböző formátumok léteznek:

RDF/XML: XML szintaxis. Előnye, hogy a már létező XML eszközökkel feldolgozható.

N-Triples $\geq$ Turtle $\geq$ Notation-3: Ezek szöveges formátumok. A hármasok tagjait whitespace-el, a hármasokat `'.'`-tal elválasztva kell egymás után írni. Jól olvasható. A három formátum

csak a bevezetett szintaktikus rövidítésekben különbözik egymástól. A bővebb nyelvek több rövidítési lehetőséget tartalmaznak. Például azonos alanyú hármasok egymás után írása esetén, elég csak egyszer leírni az alanyt.

2. Szókészlet-leíró nyelvek

Az előzőekben láthattuk, hogyan is lehet szemantikus annotációkból képzett hármasokkal leírni a valós világ objektumait, tulajdonságait.

Ahhoz, hogy az annotációkból nagyobb, összetettebb gráfokat tudjunk alkotni – és ezáltal komplexebb tudást tudjunk leírni – fontos, hogy ugyanazt az objektumot mindig, vagy legalábbis az esetek döntő többségében ugyanazzal az erőforrás-leíróval azonosítsunk.

Igaz ugyan, hogy az erőforrásokat egyértelmű azonosítóval látjuk el, de egy erőforrást több ilyen azonosítóval is elláthatunk. Ebben az esetben a több forrásból származó állítások integrációjának a hatékonyságát nagyban rontja, ha nem tudjuk, hogy két URI ugyanazt az erőforrást azonosítja. Ezt a problémát „*identity resolution*”⁷ néven emlegetik, és már tekintélyes szakirodalommal rendelkezik, ugyanis más területeken is jelentkezik, például a relációs adatbázisoknál.

A hasonló nehézségek elkerülését segíti a szókészletek, ontológiák létrehozása.

Egy szókészletben megadhatjuk, hogy milyen URI-kat használunk az erőforrások azonosítására, illetve azt, hogy az egyes URI-kat pontosan milyen esetekben használjuk (mit jelentenek), és leírhatjuk az URI-k egymással való kapcsolatait is.

Ha létrehozunk egy szókészletet, és sokan el kezdik azt használni az adataik reprezentálására, akkor az adatok integrációja hatékonyabban elvégezhető.

Több publikus szókészlet létezik a legáltalánosabbtól, mely az egész világot próbálja leírni egészen a terület-specifikusakig, melyek egy-egy témakört próbálnak minél részletesebben megragadni.

Ilyen publikus, széles körben használt szókészletek például:

- Foaf (Friend of a Friend)⁸: Emberek közötti kapcsolatokhoz, szociális hálózatokhoz
- Dublin core⁹: erőforrások metaadataihoz
- DOAC (Description Of A Career)¹⁰: szakmai tapasztalatokhoz, önéletrajzokhoz

⁷ Identity resolution: http://en.wikipedia.org/wiki/Identity_resolution

⁸ FOAF Vocabulary Specification 0.98: <http://xmlns.com/foaf/spec/>

⁹ Dublin Core Metadata Element Set, Version 1.1: <http://dublincore.org/documents/dces/>

- DOAP (Description Of A Project)¹¹: szoftver projektek leírására

A szókészleteket is RDF-ben írhatjuk le, speciálisan az RDF-ek felé épülő nyelveken. Több ilyen nyelv is létezik, ebből nézünk meg most kettőt az RDF Sémát és a Web Ontology Language-t (röviden: OWL).

2. 2. 1.RDF Schema¹²

Az RDF Schema biztosít számunkra egy központi szókészletet. Ez a szókészlet metaosztályok gyűjteményének is tekinthető, és a <http://www.w3.org/2000/01/rdf-schema#> névtéren belül érhető el, amelyet általában az *rdfs* prefixel rövidítenek. Az alábbi táblázatokban néhány fontosabb osztályt látható a névtérből.

Az osztály neve	magyarázata
rdfs:Resource (Erőforrás)	Minden erőforrás közös osztálya.
rdfs:Literal (Literál)	A literális értékek osztálya (pl. karakterláncok és egész számok).
rdf:XMLLiteral (XML-literál)	Az XML kódrészletek literális ábrázolásainak osztálya.
rdfs:Class (Osztály)	Az osztályok osztálya.
rdf:Property (Tulajdonság)	Az RDF tulajdonságok osztálya.
rdfs:Datatype (Adattípus)	Az RDF adattípusok osztálya.
rdf:Statement (Kijelentés)	Az RDF kijelentések osztálya.
rdf:Bag (Zsák)	A rendezetlen konténerek osztálya.
rdf:Seq (Sorozat)	A rendezett konténerek osztálya.
rdf:Alt (Alternatíva-csoport)	Az alternatívákat tartalmazó konténerek osztálya.
rdfs:Container (Konténer)	Az összes RDF konténertípus osztálya.
rdfs:ContainerMembershipProperty (Konténertagság-tulajdonság)	A Konténertagság-tulajdonságok (<i>rdf:_1</i> , <i>rdf:_2</i> , ...) osztálya, mely tulajdonságok mindegyike a "tagja" (member) tulajdonság altulajdonsága.
rdf:List (Lista)	Az RDF Listák osztálya.

3. ábra RDFS osztályok¹²

¹⁰ Description of a Career: http://en.wikipedia.org/wiki/Description_of_a_Career

¹¹ Description of a Project: <http://trac.usefulinc.com/doap>

¹² Forrás: Pataki Ernő: Az RDF Szókészlet Leíró Nyelv 1.0: RDF Séma (2005, fordítás) <http://www.w3c.hu/forditasok/RDF/REC-rdf-schema-20040210.html>

A tulajdonság neve	magyarázata	érvényességi köre	értéktartomány
rdf:type (típusa)	Az alany egy osztály egyede.	rdfs:Resource	rdfs:Class
rdfs:subClassOf (alosztálya ~nak)	Az alany egy osztály alosztálya.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf (altulajdonsága ~nak)	Az alany egy tulajdonság altulajdonsága.	rdf:Property	rdf:Property
rdfs:domain (érvényességi köre)	Az alany (tulajdonság) érvényességi köre.	rdf:Property	rdfs:Class
rdfs:range (értéktartománya)	Az alany (tulajdonság) értéktartománya.	rdf:Property	rdfs:Class
rdfs:label (címkéje)	Az alany (erőforrás) ember által olvasható neve.	rdfs:Resource	rdfs:Literal
rdfs:comment (kommentárja)	Az alany (erőforrás) ember által olvasható magyarázata.	rdfs:Resource	rdfs:Literal
rdfs:member (tagja)	Az alany (erőforrás) tagja/eleme.	rdfs:Resource	rdfs:Resource
rdf:first (első eleme)	Az alany (RDF lista) első eleme.	rdf:List	rdfs:Resource
rdf:rest (maradék része)	Az alany (RDF lista) első eleme utáni maradék része.	rdf:List	rdf:List
rdfs:seeAlso (lásd továbbá)	További információ az alanyról (erőforrásról).	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy (definiálja)	Az alanyt (erőforrást) definiáló másik erőforrás.	rdfs:Resource	rdfs:Resource
rdf:value (értéke)	Idiomatikus tulajdonság olyan strukturált értékek leírásához, amilyen pl. "Az RDF bevezető tankönyve" egy példája mutat.	rdfs:Resource	rdfs:Resource
rdf:subject (alanya)	Az alany (RDF kijelentés) alanya.	rdf:Statement	rdfs:Resource
rdf:predicate (állítmánya)	Az alany (RDF kijelentés) állítmánya.	rdf:Statement	rdfs:Resource
rdf:object (tárgya)	Az alany (RDF kijelentés) tárgya.	rdf:Statement	rdfs:Resource

4. ábra RDFS tulajdonságok¹²

Példa a szókészlet létrehozására:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix ex: <http://example.org/#>.
```

Prefixek megadása

```
ex:Person rdf:type rdfs:Class.
ex:Female rdf:type rdfs:Class.
ex:Female rdfs:subClassOf ex:Person.
ex:Male rdf:type rdfs:Class.
ex:Male rdfs:subClassOf ex:Person.
```

Osztályhierarchia leírása

```
ex:name rdf:type rdf:Property.
ex:name rdfs:domain ex:Person.
```

Tulajdonságok definiálása

```
dbpedia:Katie_Melua rdf:type Female.
dbpedia:Katie_Melua ex:name „Ketevan Malua”.
```

Egyedek létrehozása

Látható, hogy egyszerű RDF hármasokat használunk. A példában leírtunk egy osztályhierarchiát:



5. ábra Egyszerű osztályhierarchia

Majd felvettük az *ex:name* tulajdonságot, melynek az értelmezési tartománya a *Person* osztály. Végül a népszerű énekest, Katie Meluát reprezentáló csúcshoz hozzárendeltük az *ex:name* tulajdonsággal a születési nevét, a „*Ketevan Malua*” értéket.

2. 2. 2. OWL¹³

Láthattuk az RDF Schema szókészletének legfontosabb elemeit. Most nézzük meg, hogy milyen pluszt nyújt ehhez a Web Ontology Language, azaz az OWL.

RDF Schemából átvett elemek:

- Class
- *rdfs:subClassOf*
- *rdf:Property*
- *rdf:subPropertyOf*
- *rdfs:domain*
- *rdfs:range*
- Individual

Újítás, hogy kiköthetjük osztályok, objektumok különbözőségét, egyenlőségét:

- *sameAs*
- *differentFrom*
- *equivalentClass*
- ...

A tulajdonságokról (állítványokról) is több dolgot megadhatunk:

- *ObjectProperty*: két erőforrást kapcsol össze
- *DatatypeProperty*: egy erőforráshoz rendel egy literált
- *TransitiveProperty*: a tulajdonság tranzitív, pl.: ha A rokona B-nek és B rokona C-nek, akkor A rokona lesz C-nek is.

¹³ OWL Web Ontology Language Overview, W3C Recommendation: <http://www.w3.org/TR/owl-features/>

- SymmetricProperty: szimmetrikus tulajdonság, pl.: ha A ismert B-t, akkor B is ismeri A-t.
- ...

A tulajdonságokra kardinális megszorításokat tehetünk:

- minCardinality: például minden embernek van legalább egy szülője, vagyis minden objektumhoz, amelyről azt állítjuk, hogy ember kell tartoznia legalább egy szülő tulajdonságnak, amely a szülőjét kapcsolja hozzá
- maxCardinality
- Cardinality
- ...

Részletesebb leírásért érdemes felkeresni a W3C ide vonatkozó oldalát:
<http://www.w3.org/TR/owl-features/>

2. 2. 3.Következtetések

Láthattuk, hogy a szókészletekben rengeteg mindent le lehet írni, de miért is jó ez nekünk?

A szókészletekben az osztályokról, objektumokról szerzett tudásunkat is tárolhatjuk. Ha ezt egyszer megteesszük, akkor a továbbiakban már használhatjuk az eltárolt tudást.

Példa

Tegyük fel, hogy adott az előbb látott osztályhierarchia, és definiáljuk a következő tulajdonságot

hasMother, amely megmondja, hogy egy személynek ki az anyukája.

Vagyis az értelmezési tartomány a Person osztály, az értékkészlet pedig a Woman osztály, hiszen az anyuka csak nő lehet.

Ezek után, ha egy dokumentumban csak annyi szerepel, hogy

- *Emma is a person.* (*x:Emma rdf:type x:Person*)
- *Emma mother is Amy.* (*x:Emma x:hasMother x:Amy*)

Akkor azt már ki lehet következtetni, hogy

Amy is a Woman (*x:Amy rdf:type x:Women*)

Hiszen Amy előfordul a hasMother tulajdonság értékeként, és a hasMother tulajdonság értékkészlete a Woman osztály.

Emlékszünk még a bevezető boros példára? Ha egy szókészletben megadjuk, hogy Bordeaux Franciaországban található, és a szókészlet elemeivel annotáljuk a borokra vonatkozó forrásokat, akkor a francia borok kérdése nem okoz többé gondot.

2. 3.SPARQL¹⁴

A SPARQL (Simple **P**rotocol and **R**DF **Q**uery **L**anguage) egy lekérdező nyelv, speciálisan az RDF adatokhoz. A működése gráfminta-illesztésen alapul, vagyis a felhasználó által megadott mintákat megpróbálja illeszteni az adott RDF adathalmazra.

A nyelv szerkezete nagyon hasonlít a relációs adatbázisok világából ismert SQL szerkezetére:

- Szintén deklaratív¹⁵, vagyis a felhasználó csak azt mondja meg, mit szeretne kiszámolni, de azt nem, hogy hogyan.
- Továbbá több kulcsszót is átvett onnét: select, distinct, limit, order by, ...

Többféle lehetőséget kínál a felhasználó számára

- Új RDF adathalmazt lehet létrehozni a már létezőkből (CONSTRUCT);
- Egy erőforrás környezetét lehet leírni vele (DESCRIBE);
- Tesztelni lehet egy gráfminta meglétét vagy éppen hiányát az adatgráfban (ASK);
- Képes kiemelni egy gráfmintára illeszkedő adatok bizonyos részeit (SELECT).

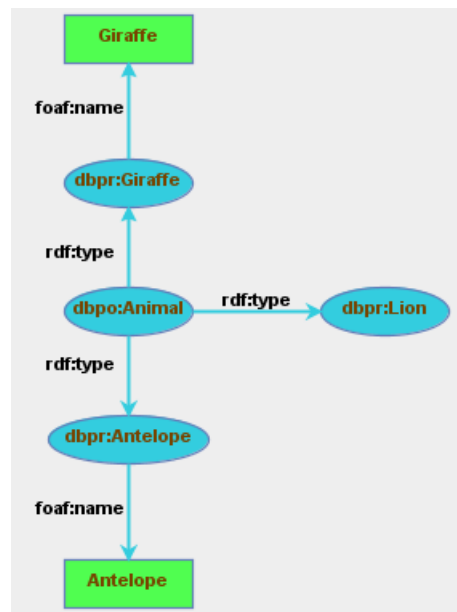
(A továbbiakban a SPARQL 1.0, W3C ajánlásban rögzített változattal foglalkozunk.)

¹⁴ SPARQL Query Language for RDF, W3C Recommendation: <http://www.w3.org/TR/rdf-sparql-query/>

¹⁵ Declarative Programming: http://en.wikipedia.org/wiki/Declarative_programming

A következő példa a mintaillesztést szemlélteti:

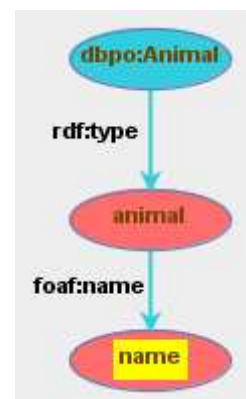
```
dbpr:Giraffe rdf:type dbpo:Animal .
dbpr:Giraffe foaf:name "Giraffe" .
dbpr:Antelope rdf:type dbppo:Animal .
dbpr:Antelope foaf:name "Antelope" .
dbpr:Lion rdf:type dbpo:Animal .
```



6. ábra Példa adathalmaz lekérdezéshez

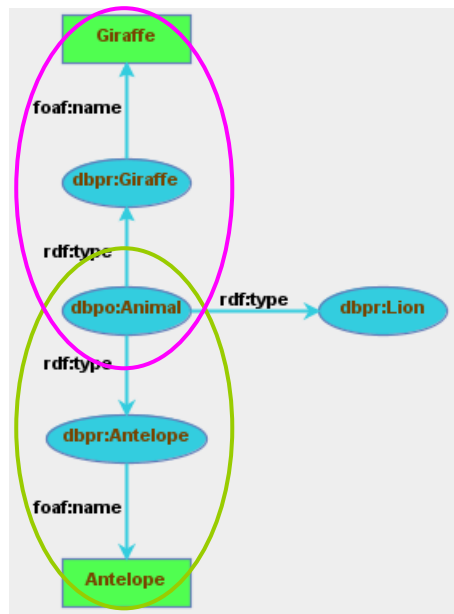
```
PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
```

```
SELECT ?name
WHERE{
  ?animal rdf:type dbpo:Animal .
  ?animal foaf:name ?name.
}
```



7. ábra Egyszerű lekérdezés

Figyeljük meg az 6. ábraán az adathalmazt, és a 7. ábraán a gráfmintát! Látható, hogy a minta kétféleképpen illeszthető az adathalmazra:



8. ábra Mintaillesztés

A lekérdezés eredményében azt az értéket, értékeket szeretnénk visszakapni, melyekre a *name* csúcsot illesztettük a mintából (az szerepel a SELECT utáni listában, illetve az ábrán piros színű csúcsban sárga háttérű *name* szöveg). Látható, hogy a zsiráfra és az antilopra illeszthető a minta, de az oroszlánra nem, mert ahhoz nem tartozik *foaf:name* tulajdonsággal kapcsolt csúcs. Ezért az eredmény:

Name
„Giraffe”
„Antelope”

1. Táblázat Mintaillesztés eredménye

2. 3. 1.WHERE záradék

Az előző példán megfigyelhetjük, hogy a már említett gráfmintákat a lekérdezés WHERE záradékában tudjuk megadni, ami minden lekérdezés esetén kötelező.

(A legegyszerűbb gráfminta:

?subject ?predicate ?object

erre minden hármas illeszkedik.)

A gráfminták lényegében hármások, melyek változókat is tartalmazhatnak és a turtle formátumhoz hasonló nyelven kell leírni őket, vagyis a hármások tagjait whitespace-el, a hármásokat '.'-tal kell elválasztani.

A minták megadására összetettebb eszközöket is biztosít a SPARQL számunkra:

- A változókra megkötéseket tehetünk a **FILTER** segítségével.

- o Megszoríthatjuk a típust (isURI, isBLANK, isLITERAL)

```
PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
WHERE{
  ?animal rdf:type dbpo:Animal .
  ?animal foaf:name ?name .
  FILTER isLITERAL(?name) .
}
```

Ekkor a ?name változót csak olyan csúcsra illeszthetjük, mely literál típusú (6. ábrán a zöld csúcsok).

- o Megadhatunk reguláris kifejezést, amire illeszkednie kell a változó értékének

```
PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
WHERE{
  ?animal rdf:type dbpo:Animal .
  ?animal foaf:name ?name .
  FILTER regex(?name, „^G”) .
}
```

Ekkor a ?name változót csak olyan csúcsra illeszthetjük, mely a 'G' karakterrel kezdődik.

- o A változó értékére aritmetikai, logikai összehasonlításokat tehetünk (<,>=,!=,||,&&, ...)

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
SELECT ?wine ?year
WHERE
{
  ?wine a wine:Wine .
  ?wine wine:hasVintageYear ?y .
  ?y wine:yearValue ?year .
  FILTER ((?year < 2006) && (?year > 2000)).
}
```

Ez a lekérdezés a bor ontológiából adja vissza a 2000 és 2006 között készült borokat.

- o Egyéb megkötések (BOUND, sameTerm, lang, datatype, str, langMATCHES, ...)

- Opcionális gráfmintákat is megadhatunk az **OPTIONAL** segítségével, melyek nem szűrik az adathalmazt, csak egyezés esetén kiterjesztik az eredményt.

```

PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?image
WHERE{
  ?animal rdf:type dbpo:Animal .
  ?animal foaf:name ?name .
  OPTIONAL{
    ?animal dbpo:thumbnail ?image .
  }
}

```

Ekkor a megszorító gráfminta nem változik, vagyis ugyanazokat a sorokat kapjuk vissza, mint az OPTIONAL rész elhagyásával, de ha az állatokhoz tartozik kép, akkor a képet is hozzáfűzzük az eredményhez.

- A mintákat csoportokba szervezhetjük, hogy a csoportok között további műveleteket hajthassunk végre (unió).

```

PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x ?name
WHERE{
  {?x rdf:type dbpo:Plant . }
  UNION
  {?x rdf:type dbpo:Animal . }
  ?x foaf:name ?name
}

```

Itt a növényeket és az állatokat kérdezzük le a DBpediáról.

- A különböző forrásokból származó hármasokra különböző mintákat adhatunk a **GRAPH** szerkezettel.

```

PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
SELECT ?wine ?predicate ?object
FROM NAMED <http://dbpedia.org/sparql>
WHERE
{
  ?wine a wine:Wine .
  GRAPH <http://dbpedia.org/sparql> {
    ?wine ?predicate ?object
  }
}

```

Ezt a lekérdezést a bor ontológiára futtatjuk. Kiválogatjuk a borokat, majd a

DBpediától lekérdezzük, mit tud az egyes borokról.

Ez egy jó példa az adatok különböző forrásból származó integrációjára.

2. 3. 2.FROM záradék

Lehetőség van több forrásból származó adatok együttes lekérdezésére. Ekkor az adatforrásokat külön fel kell sorolni, amit a FROM kulcsszóval tehetünk meg.

```

PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x ?name
FROM <http://dbpedia.org/sparql>
FROM <http://dbpedia2.org>
WHERE{
  ?x rdf:type dbpo:Plant .
  ?x foaf:name ?name
}

```

Ahhoz hogy az előbb látott módon, az eltérő forrásból származó adatokra eltérő mintát is meg tudjunk adni, nevesíteni kell az adatforrást. Ezt a FROM NAMED kulcsszóval tehetjük meg.

```

PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
SELECT ?wine ?predicate ?object
FROM NAMED <http://dbpedia.org/sparql>
WHERE
{
  ?wine a wine:Wine .
  GRAPH <http://dbpedia.org/sparql> {
    ?wine ?predicate ?object
  }
}

```

Az ilyen formában megadott forrásokra a WHERE záradékon belül használhatjuk a GRAPH kulcsszót.

Joggal kérdezhetnénk, hogy ha nem adunk meg adatforrást a lekérdezésben, akkor mi történik. A válaszhoz meg kell ismerkednünk a végpontokkal.

2. 3. 3.Végpontok

Ahhoz, hogy egy lekérdezést le tudjunk futtatni, szükség van egy értelmezőre és egy végrehajtóra. Az adathalmazokhoz hozzárendelhetünk végpontokat. Egy végponthoz egy adathalmaz tartozik, de egy adathalmazt több végponton keresztül is elérhetünk.

Abban az esetben, ha a lekérdezésben nem használunk FROM záradékot, akkor az adott végponthoz tartozó adathalmaz szolgál inputként.

Több publikus végpont is elérhető az interneten:

- DBPedia¹⁶: a Wikipediában tárolt információ RDF átirata.
- GeoNames¹⁷: földrajzi adatok országokról és egyéb helyekről.
- BBC Music¹⁸: kritikák és egyéb jellemzők zenei albumokról, előadókról.
- Gene Ontology¹⁹: a génkutatással kapcsolatos eredmények.

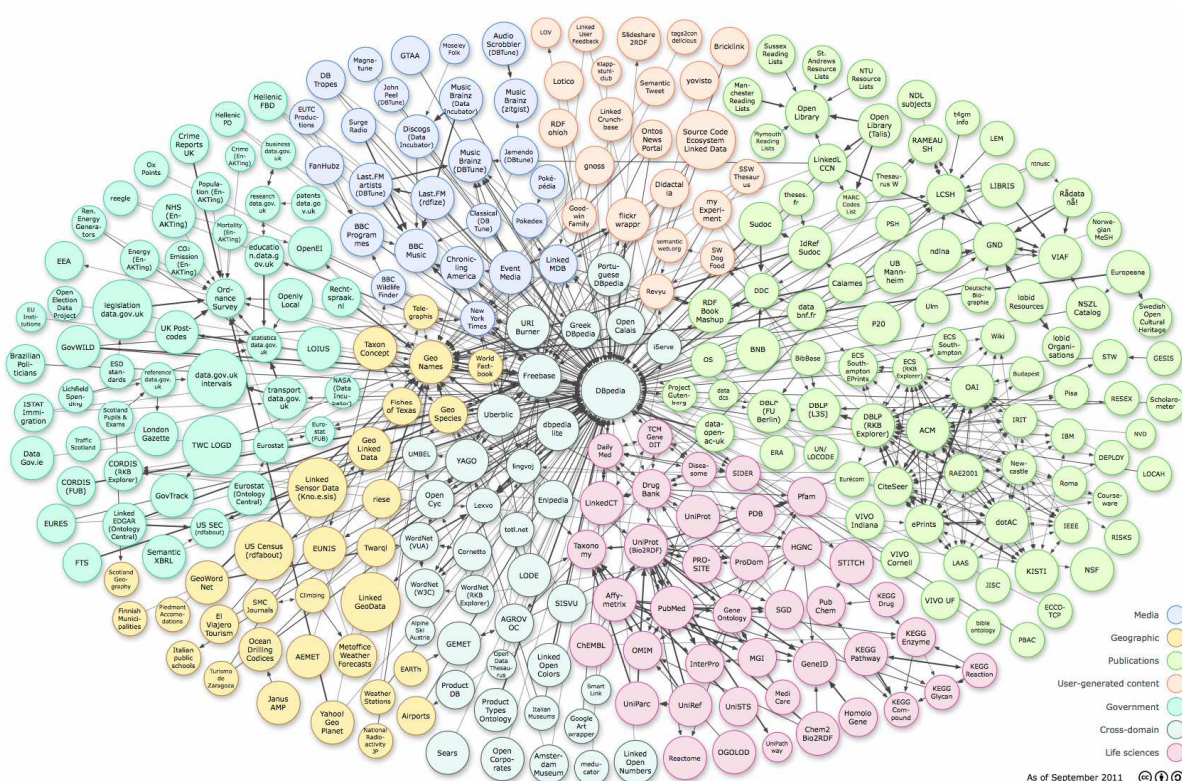
¹⁶ DBpedia leírás: <http://dbpedia.org/About>

¹⁷ GeoNames: <http://www.geonames.org/>

¹⁸ BBC Music: <http://thedatahub.org/dataset/bbc-music>

¹⁹ Gene Ontology: <http://thedatahub.org/dataset/bio2rdf-go>

A publikus végpontokról listákat is vezetnek. Több ilyen lista is létezik a leghíresebb talán a Linking Open Data, ahol egy cloud diagram formájában a végpontok közötti kapcsolatokat is szemléltetik.



9. ábra LOD felhő²⁰

2. 3. 4. Lekérdezés típusa

Új RDF adathalmazt lehet létrehozni a már létezőkből (CONSTRUCT);

Egy erőforrás környezetét lehet leírni vele (DESCRIBE);

Tesztelni lehet egy gráf minta meglétét vagy éppen hiányát az adatgráfban (ASK);

Képes kiemelni egy gráf mintára illeszkedő adatok bizonyos részeit (SELECT).

A SPARQL négyféle lekérdezés típust támogat:

- ASK: a WHERE záradékban megadott mintát teszteli. Kétféle eredményt adhat vissza:
 - o true – ha a minta megtalálható az adathalmazban,
 - o false – ha a minta nem található meg az adathalmazban.
- Emiatt a lekérdezésekhez nem tartoznak további eredménymódosítók.

²⁰ Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. <http://lod-cloud.net/>

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

ASK

WHERE

```
{
    ?wine rdf:type wine:Wine .
}
```

Lekérdezzük, hogy az adatok között vannak-e borok, a W3C bor ontológiája alapján.

- **SELECT:** az eddigi példák során már találkoztunk ezzel a lekérdezés típussal. A gráfmintákban megadott változók illesztés során felvett értékeit kérdezhetjük le vele. Két módosító tartozik hozzá:

- o **DISTINCT:** azonos eredmények közül csak egyet kapunk vissza
- o **REDUCED:** nincs pontosan definiálva, azonos típusú eredmények közül legalább egyet, legfeljebb annyit kapunk vissza, amennyi az adathalmazban van

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

SELECT DISTINCT ?wine

WHERE

```
{
    ?wine rdf:type wine:Wine .
}
```

- **CONSTRUCT:** az adathalmazból a gráfmintára illesztés után új adathalmazt hozhatunk létre egy másik, speciális gráfminta alapján, melyben szintén hármasok szerepelnek, azonban csak változót, URI-t, literált és blank node-ot tartalmazhat. A változónak a WHERE záradékban is elő kell fordulnia.

```
PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

CONSTRUCT{

 ?wine ?p ?o

}

WHERE

```
{
    ?wine rdf:type wine:Wine .
    ?wine ?p ?o .
}
```

Az adathalmazból lekérdezzük a borokat, és az összes adatot, amit a borokról tudunk.

- **DESCRIBE:** a DESCRIBE működése nincs pontosan meghatározva. Arra lehet használni, hogy a felsorolt URI-król, illetve változókról megkapjuk a „szükséges” tudást.

```

PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-
20031209/wine#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
DESCRIBE ?wine
WHERE
{
    ?wine rdf:type wine:Wine .
}

```

2. 3. 5.Eredménymódosítók

Az SQL-hez hasonlóan a SPARQL is lehetőséget ad a lekérdezés eredményének módosítására. Ezt három kulcsszó megadásával tehetjük meg:

- **ORDER BY:** az eredmény sorrendjét befolyásolhatjuk, csökkenő vagy növekvő sorrendbe rendezhetjük a sorokat változók, vagy az azokból számított értékek szerint.
- **LIMIT:** Megadhatjuk, hogy az eredményből hány sort szeretnénk visszakapni.
- **OFFSET:** Megadhatjuk, hogy az eredménynek hányadik sorától következő sorokat akarjuk látni.

```

PREFIX wine: <http://www.w3.org/TR/2003/PR-owl-guide-
20031209/wine#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?wine
WHERE
{
    ?wine rdf:type wine:Wine .
}
ORDER BY DESC(?wine)
LIMIT 50
OFFSET 10

```

Az adathalmazban szereplő borokat akarjuk látni, az URI-k szerinti csökkenő sorrendben. Sorrendben a 10-edik bortól 50 darabot.

3. Probléma felvetése

Az előző fejezetben áttekintettük a Szemantikus Web fogalmát, céljait valamint az alapvető eszközöket, technológiákat. Most a problémával ismerkedünk meg, melynek kapcsán a dolgozat született.

Láthattuk, hogy a SPARQL lekérdező nyelv egy ideális eszköz az RDF adathalmazok lekérdezésére. Ez nem annyira meglepő, hiszen ezzel a céllal fejlesztették ki. Azonban a felhasználók nagy része nem tud, vagy nem akar SPARQL lekérdezéseket írni. Valamilyen könnyebben kezelhető felületet kell biztosítani számukra, melynek segítségével még mindig meg tudják találni azt az adatot, azokat az adatokat, amelyekre szükségük van.

Többféle megoldás létezik a probléma kezelésére. A [12] cikkben bemutatott négyfajta interfész:

1. Tudásbázis-specifikus interfész

Ebben az esetben pontosan ismernünk kell, hogy milyen adatok, tulajdonságok vannak az adathalmazban. Mindegyikhez felvesszünk egy szűrő mezőt a lehetséges értékekkel. Így pontosan meg lehet találni a kívánt adatot.

2. Aspektus alapú megközelítés

Ez az előző megoldásnak egy egyszerűsített változata. Itt nem készítünk mindenhez külön szűrőt, hanem csak a legfontosabb, leggyakoribb tulajdonságokhoz, típusokhoz. Ezáltal nem biztos, hogy egyértelműen ki tudunk választani egy-egy adatot, azonban jól lehet szűrni az adathalmazt. A kisebb halmazból pedig már a felhasználó választhatja ki manuálisan a megfelelőt.

3. Vizuális lekérdezőszerkesztő programok

Mivel az RDF adatok jól reprezentálhatók gráffal, a lekérdezések pedig gráfminták illesztésén alapulnak, ezért azok leírhatók gráfokkal. Ezeket vizuálisan is össze lehet építeni, mely nagy segítséget jelent a témában kevésbé járatos felhasználók számára, valamint azoknak, akik dokumentálni kívánják lekérdezéseiket.

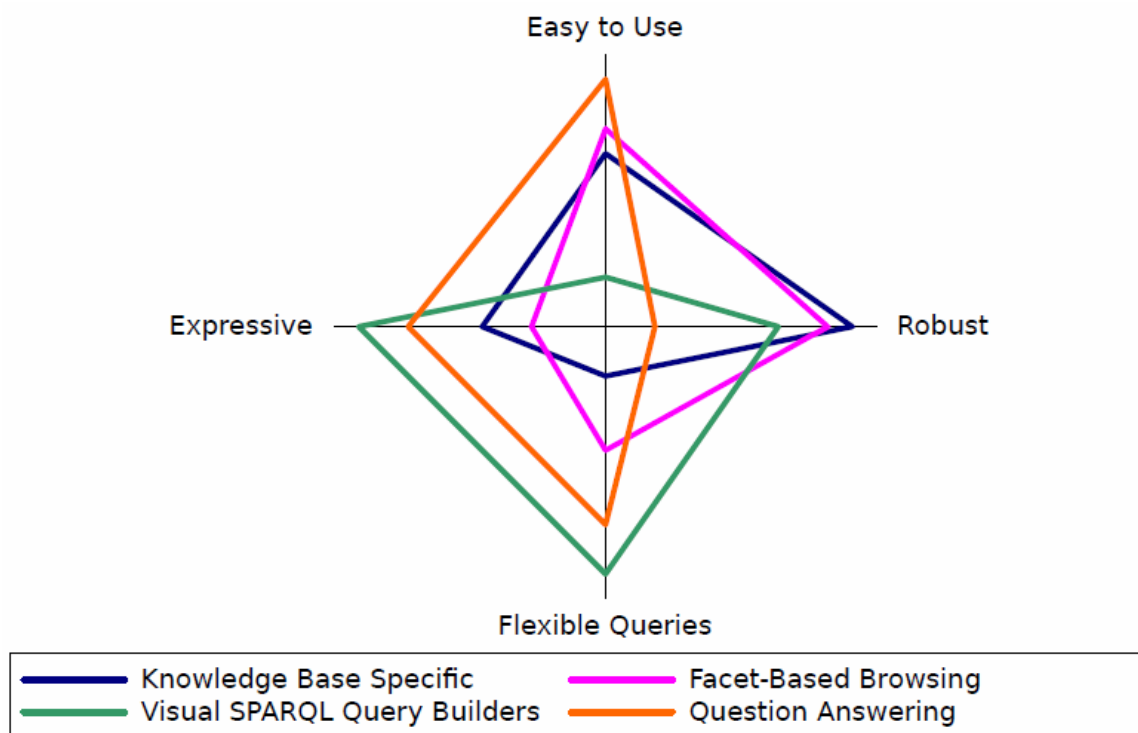
4. Természetes nyelven megfogalmazott kérdések megválaszolása

A felhasználó beír természetes nyelven egy kérdést, a rendszer feldolgozza, értelmezi és megadja rá a választ. A végső cél ennek az elérése, de a jelenlegi rendszerek nagyon kis számú, speciálisan megfogalmazott kérdéseket tudnak csak megválaszolni.

Mind a négy megközelítésnek vannak előnyei és hátrányai. Több szempont szerint lehet vizsgálni őket, például:

- mennyire könnyű használni (Easy to Use),
- mennyire kifejező, azaz magából a lekérdezésből mennyire látszik, hogy pontosan mit is keresünk (Expressive),
- mennyire flexibilis: a keresendő adatok körének megváltoztatása milyen hatással van a lekérdezésre (Flexible),
- mennyire robusztus: milyen kifejezőerővel rendelkezik az interfész, mennyire lehet testre szabni a kérdést (Robust).

A cikk szerzői az alábbi ábrán foglalták össze a megállapításaikat:



10. ábra A lekérdező interface-ek erőssége, gyengesége²¹

²¹ Jens Lehmann, Lorenz Bühmann: AutoSPARQL: Let Users Query Your Knowledge Base. Slides. ESCW 2011

Vagyis:

1. Tudásbázis-specifikus interfész csak kötött sémára illeszthető, ezáltal nem túl flexibilis. Azonban tetszőleges adatot meg tud találni, minden kérdést ki lehet fejezni, vagyis elég robosztus.
2. Az aspektus alapú megközelítés flexibilisebb az előzőnél, azonban nem biztos, hogy minden adatra pontosan rá tudunk keresni, ezért kevésbé robosztus, és így nem látszik a lekérdezésből, hogy pontosan mit is keresünk, vagyis nem is túl expresszív.
3. A vizuális lekérdezés szerkesztők nagyon kifejezők, hiszen egy ábra mindig szemléletesebb, mint a puszta szöveg, kellőképpen flexibilisek és robosztusak is. Azonban használatukhoz szükséges a lekérdező nyelvben való jártasság, hiszen aki vizuálisan össze tudja rakni a lekérdezést, az nagy valószínűséggel szövegesen is.
4. Mint láthatjuk a természetes nyelvű kérdések megválaszolása a legkevésbé robosztus. Ez nagyrészt a nyelvek többértelműségéből adódik. Gondoljunk csak bele egy-egy mondat, vagy egy-egy szó eltérő szöveggörnyezetben teljesen mást jelent! Jó példa erre a telefon:

Mikor csöngött utoljára a *telefon*? – Ebben a mondatban főnévként, alanyi szerepben jelenik meg.

Melyik állat az, amelyik minden sarkot *telefon*? – Ebben pedig ige és állítmányi szerepben fordul elő.

Ezek miatt pusztán egy mondatból nem biztos, hogy pontosan meg lehet állapítani a mondat lényegét.

3. 1.Vizuális lekérdezés szerkesztő programok

Dolgozatom tárgyaként a vizuális lekérdezés építő alkalmazásokat választottam. Készítettem egy ilyen alkalmazást a SPARQL nyelvhez.

Választásom fő oka a technika expresszivitása volt, hogy a szöveget képes vizuális formában megjeleníteni. Ez a tulajdonság nagyon hasznos, ha egy-egy lekérdezést újra és újra előveszünk, mert értelmezni, bővíteni esetleg módosítani kell. Ha valamilyen okból több felhasználó dolgozik közösen, akkor bizonyos, hogy ezek a problémák rendszeresen felmerülnek. Azonban, ha a dokumentáció tartalmaz egy képi megjelenítést is, akkor a lekérdezés újbóli megértése jóval könnyebb és kevesebb időt vesz igénybe.

Hosszabb lekérdezések esetén gyakran előfordul, hogy egyszerűbb a már meglévőket változtatni, mint az egészet újra megírni előlről. Ilyenkor jönnek jól a lekérdezés építők, melyek kellőképpen flexibilisek a feladatok elvégzéséhez.

A kifejezőképességért és a flexibilitásért cserébe csorbul ugyan a kifejezőerő, hiszen vannak olyan dolgok, melyeket vizuálisan nem, vagy csak nagyon körülményesen lehet megszerkeszteni, de erre jó megoldás, ha az összeépített lekérdezésből legeneráljuk a szöveges megfelelőt, így azt még tovább lehet alakítani, apróbb finomítások végezhetők rajta. Lehetséges, hogy ezeket az eszközöket nem a témában kezdő felhasználók fogják használni, de a technika mindenképpen szükséges és használható.

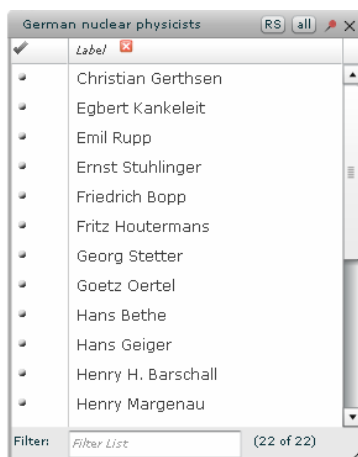
3. 2.Kapcsolódó munkák

SPARQL lekérdezések vizuális építésére már több alkalmazás készült. A témával többen foglalkoznak jelenleg is, azonban teljesen jó megoldás még nem született. Nem is születhet, hiszen minden egyes felhasználás során merülhetnek fel extra igények, melyekhez hozzá kell igazítani a programokat.

Nézzük meg most a legfontosabb alkalmazásokat, hogy milyen erősségeik, hátrányaik vannak. Miben hasonlítanak, és miben különböznek az általam készített megvalósításhoz képest.

3. 2. 1.gFacet²²

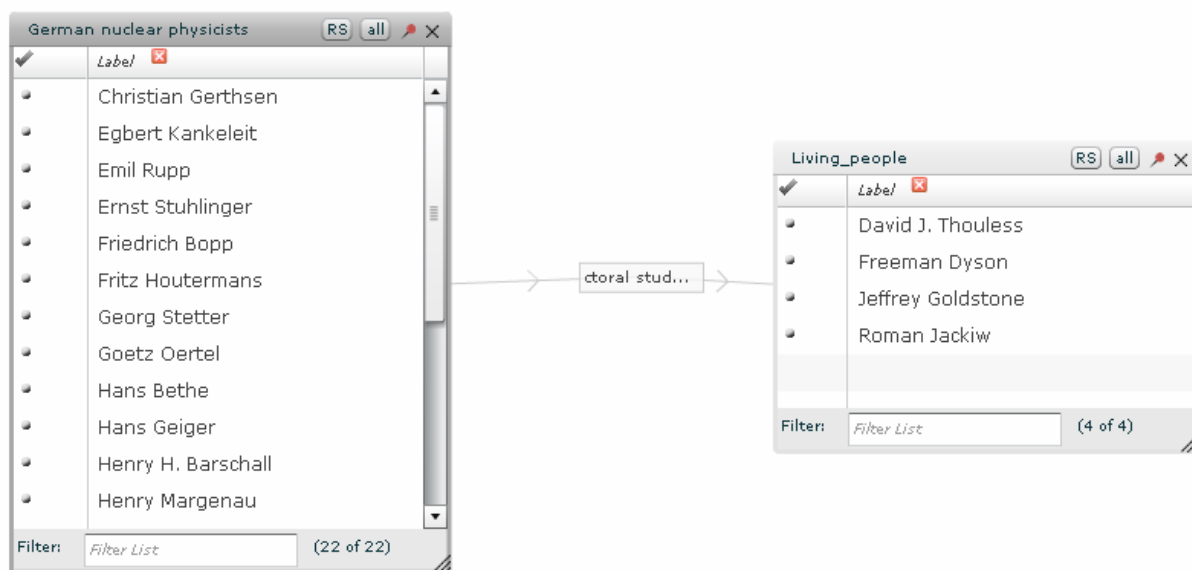
A gFacet egy Adobe Flex-el készült flash alkalmazás, mely kombinálja a gráf-alapú megjelenítést az aspektus elvű szűrési technikákkal. Segítségével könnyedén össze lehet rakni komplex lekérdezéseket is. A felhasználó úgynevezett nézőpontokat definiálhat például (német atomfizikusok, magyar egyetemek, ...).



11. ábra gFacet: német atomfizikusok

²² gFacet: <http://www.visualdataweb.org/gfacet.php>

Ezek a nézetek lesznek a gráf csúcsai. A csúcsokat rdf tulajdonságokkal (állításokkal) címkézett élekkel lehet összekötni.



12. ábra gFacet: német atomfizikusok és még élő doktoranduszaik

A nézetekre aztán mindenféle megszorításokat tehetünk, melyek szűkítik a nézetben megjelenő egyedek számát. Illetve az egyes csúcsokban ki is jelölhetünk egyedeket, melynek hatására a többi csúcs értékei is redukálódnak.

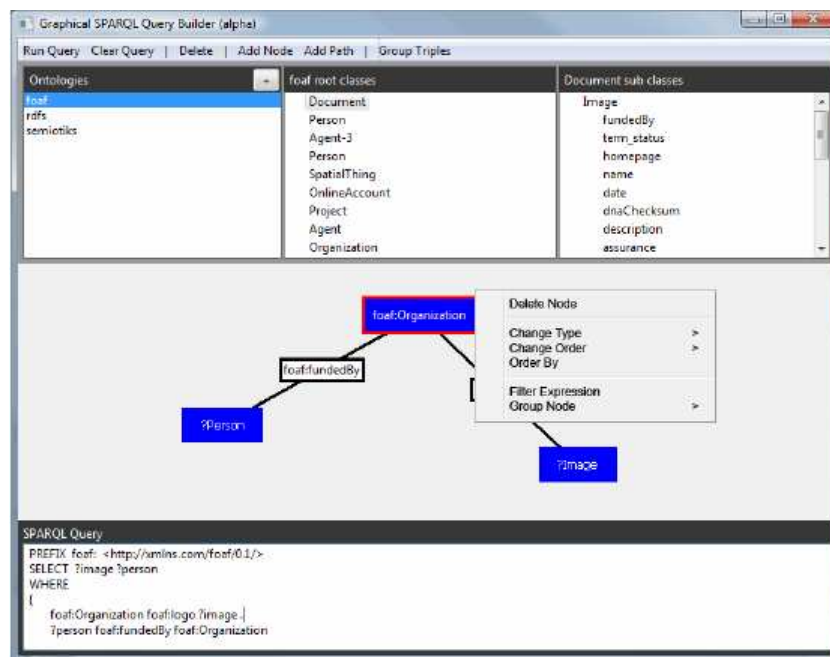
Láthatjuk, hogy az eszköz teljesen elrejtí a SPARQL lekérdezést a felhasználók elől, jól szűri az adatokat, könnyű változtatásokat eszközölni benne és nagyon látványos.

Ez egy aspektus elvű interfész.

3. 2. 2.Nitelight²³

A Nitelight-ot a southamptoni egyetemen fejlesztik. Grafikus elemeket feleltetnek meg a SPARQL-ben leírható konstrukcióknak.

²³ NiteLight: <http://nitelight.edefence.org/>



13. ábra Nitelight prototípus

[14]-ben szépen összeszedték a megjelenítés elemeit, hogy hogyan is lehet megjeleníteni a SPARQL-ben kifejezhető szerkezeteket. A program azonban még nem publikus, egy demót lehet csupán kipróbálni.

A Nitelight nagyon hasonlít az általam készített alkalmazáshoz:

- lehet rendezni a változókat,
- külön színek a különböző típusokhoz,
- van lehetőség csoportos gráfmintákat megadni,
- a gráfból legenerálódik a SPARQL lekérdezés,
- megadható a limit és offset záradék,
- többféle lekérdezés típus támogatott (ask, select, describe, construct).

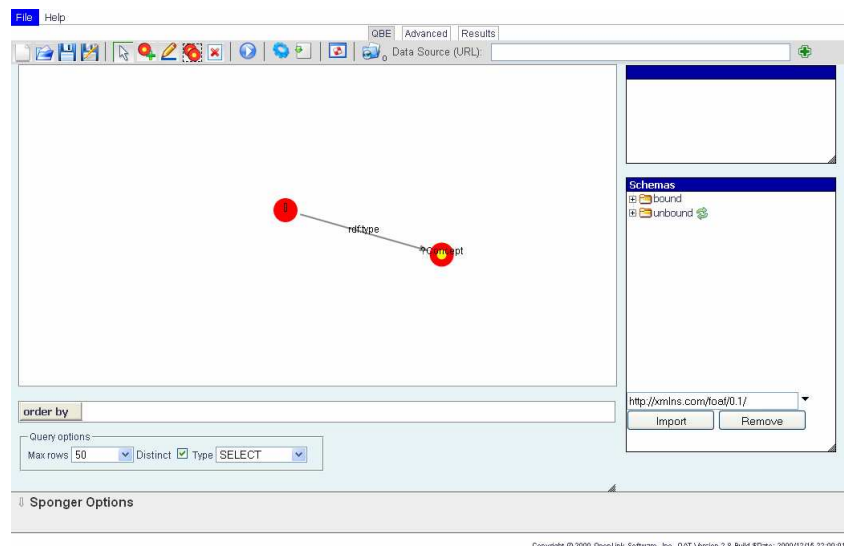
A hasonlóság mellett különbségek is mutatkoznak:

- A Nitelight-ban nem lehet prefixeket megadni,
 - sem a FROM záradékot,
 - a változók rendezése más elven működik,
 - a szűrőfeltételek sem ugyanúgy működnek.
-
- Azonban van beépített ontológiaböngésző,
 - a változóra vonatkozó megszorítások a változónál jelennek meg.

Mint említettem a program még nem elérhető, csak demó változatban, melyben több funkció nem használható, ezért az összehasonlításban elsősorban a cikkekre hagyatkoztam.

3. 2. 3.iSPARQL²⁴

Az OAT Interactive SPARQL (iSPARQL) Query Builder egy SVG alapú lekérdezés generátor és eredményböngésző alkalmazás, melyet az OpenLink fejlesztett ki.



14. ábra iSPARQL felület

Az elv megegyezik a Nitelight-nál bemutatottal, vagyis a SPARQL-ben megadható elemekhez vizuális megjelenést társítunk, ezáltal vizuálisan építhetjük össze a lekérdezést, melyet utána szöveges formában is megtekinthetünk.

Ez a program is támogatja a nyelvnek a legtöbb elemét, ha nem az összeset. Emellett kísérletet tesz a másik irány megvalósítására, ahol a szöveges formából generál vizuális lekérdezést.

Továbbá lehetőség van az összeállított lekérdezést tetszőleges végponthoz elküldeni, majd a végrehajtás után meg is nézhetjük annak eredményét egy külön felületen.

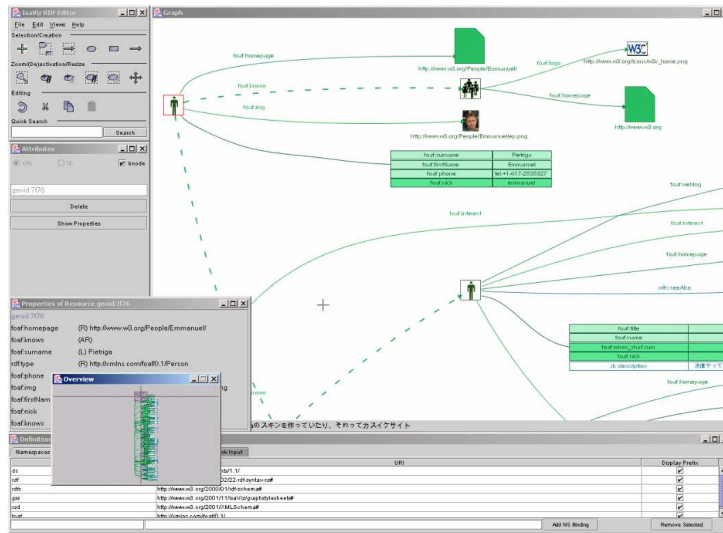
A szöveges szerkesztés során előre megírt mintákat szűrhetünk be, a leggyakrabban használt szókészletek alapból prefixekkel vannak ellátva.

Összehasonlítva az általam készített alkalmazással elmondható, hogy az iSPARQL többet tud, azonban vannak apró működésbeli különbségek (például mehet-e el két csoportos gráfmenta között, filter megadása), melyek a felhasználástól függően dönthetnek az egyik vagy a másik javára.

²⁴ iSPARQL: <http://oat.openlinksw.com/isparql/index.html>

3. 2. 4.SPARQLViz²⁵

A SPARQLViz egy plugin az IsaViz²⁶ rendszerhez. Az IsaViz egy vizuális környezet RDF adatok böngészésére és szerkesztésére. Gráfos megjelenítést használ, többféle nézet és eszköz támogatja a szerkesztést. A gráf és az egyes csúcsok nagymértékben testre szabhatók. A W3C fejleszti.



15. ábra IsaViz képernyőkép

A SPARQLViz ezt az alkalmazást egészíti ki a lekérdezések megadásának a lehetőségével. Azonban a lekérdezéseket inkább szöveges formában, varázslók segítségével lehet létrehozni, így azok nem igazán vizuálisak, de az eredmények annál inkább.

16. ábra SPARQLViz: ASK típusú lekérdezés

²⁵ SPARQLViz: <http://sparqlviz.sourceforge.net/>

²⁶ IsaViz: <http://www.w3.org/2001/11/IsaViz/>

A bemutatottakon kívül még több alkalmazás is foglalkozik a témával, azonban ezek a legtöbbször hivatkoztak, leggyakrabban használtak.

4. VisualQuery

A VisualQuery egy vizuális lekérdezés szerkesztő program a SPARQL nyelvhez. Segítségével a nyelvben leírható kifejezéseket képi formában állíthatjuk össze. Ezáltal jobban áttekinthető, könnyebben értelmezhető, dokumentálható lekérdezéseket kapunk. A vizuálisan megszerkesztett lekérdezés szöveges formára fordul le, melyet lehetőségünk van tovább szerkeszteni, majd a végleges változatot tetszőleges SPARQL végponthoz elküldeni.

Láthattuk, hogy a hasonló programokkal összevetve több előnnyel rendelkezik ez az alkalmazás.

A Nitelight-hoz képest a legnagyobb előnye, hogy jelenleg is elérhető és az összes leírt funkció működőképes. Továbbá több pluszt is nyújt: külön táblázatban felsorolhatjuk a használt prefixeket, a FROM záradékba kerülő elemeket. A változók sorba rendezése is egyszerűbb, a külön erre a célra kialakított, sorbarendezőkkel. Hiszen ha egy változót másik helyre szeretnénk beállítani, akkor csak a kívánt pozícióba kell húzni, a többi változó sorrendje automatikusan módosul.

Az igazi „vetélytárs” az iSPARQL. Az iSPARQL egy jól átgondolt, sok funkcióval rendelkező program. Támogatja a vizuális és a szöveges szerkesztést, valamint a lekérdezés eredményét meg is tudja jeleníteni, miután tetszőleges végpontra elküldte, és a végpont végrehajtotta azt.

A VisualQuery nem képes a szöveges lekérdezésből generálni a vizuális megfelelőt, az eredményt sem tudja megjeleníteni, azonban a vizuális szerkesztőfelülete több lehetőséget biztosít a felhasználók számára. A csúcsok átméretezhetőek, ezáltal a címkék nem lógnak ki a keretből, az élcímkék mozgathatóak, a különböző típusú csúcsokhoz különböző megjelenítés tartozik, vagyis szemléletesebb lekérdezéseket lehet összeállítani. Továbbá a csoportos gráfminták kezelése is átláthatóbb a VisualQuery esetén, amely nem engedi meg a csoportok közötti éleket, így egyértelmű, hogy mi generálódik az ábrából.

A következőkben áttekintjük a program felépítését, funkcióit, azokat a vizuális építőköveket, és kisegítő eszközöket, melyek rendelkezésre állnak a lekérdezések létrehozása, szerkesztése során.

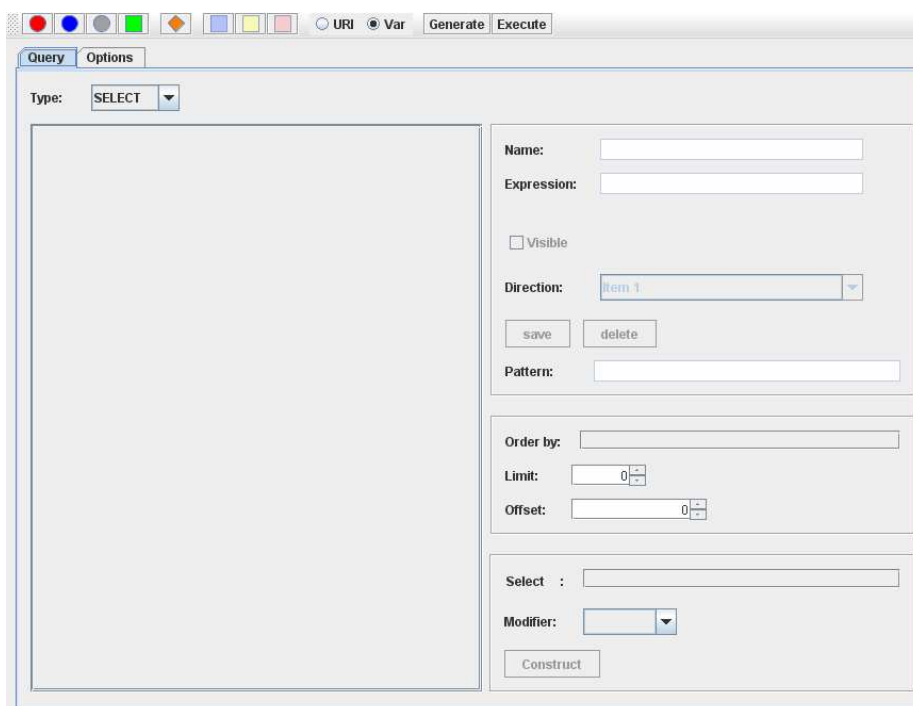
4. 1.Felhasználói felület

A felület első ránézésre bonyolultnak tűnik, azonban az elemek megismerése után tudni fogjuk, hogy mit hol keressünk.

Az opciók nem is egy, hanem két felületen vannak, melyek között a Query illetve Options fülökkel navigálhatunk.

4. 1. 1.Query fül

A Query fülön lehet összerakni a lekérdezést, a felületen található eszközök segítségével.



17. ábra VisualQuery képernyőkép (Query)

Nézzük meg az egyes elemeket részletesebben!

4. 1. 2.Menü

A menü a felső sorban található. Ezen helyezkednek el a vizuális építőkövek:



18. ábra VisualQuery menüsor

-  – Változó csúcs
-  – URI csúcs

-  – Blank node csúcs
-  – Literál csúcs
-  – FILTER csúcs
-  – csoport
-  – GRAPH csúcs
-  – OPTIONAL csúcs
- ☐ URI ☒ Var – az új él típusa változó vagy URI legyen
- szöveges lekérdezés generálása
- a lekérdezés elküldése a végpontnak

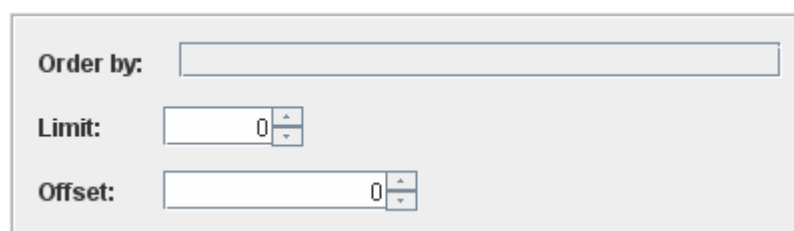
4. 1. 3.Lekérdezés típusok

A lekérdezés típusát a lekérdezés választó segítségével adhatjuk meg.



19. ábra VisualQuery lekérdezés választó

A típus változására dinamikusan változik a módosító ablak, hiszen ASK típusnál nem lehet módosítót megadni, ilyenkor azok elszürkülnek. Egyébként a rendezés sorrendjét, a LIMIT illetve az OFFSET záradékok segítségével állíthatjuk be.



20. ábra VisualQuery módosító ablak

A header ablak, amelyben a lekérdezés fejrészét állíthatjuk be, szintén változik. Ez eltérően viselkedik minden lekérdezés típusnál:

VisualQuery header window interface. It contains a 'Select' label followed by a text input field. Below it is a 'Modifier' label followed by a dropdown menu. At the bottom is a button labeled 'Construct'.

21. ábra VisualQuery header ablak

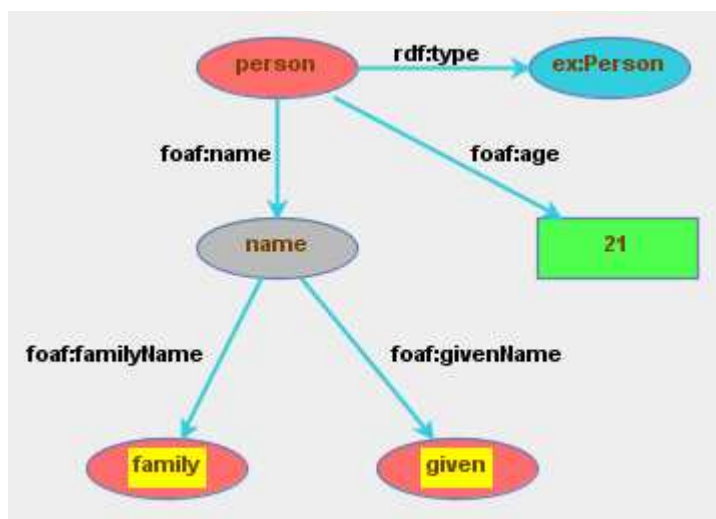
- SELECT és DESCRIBE esetén a kiválasztott változók sorrendjét állíthatjuk;
- SELECT esetén választhatunk számossági módosítót (REDUCED, DISTINCT);
- ASK-nál ez az ablak is elszürkül;
- CONSTRUCT-nál a gomb megnyomásával a CONSTRUCT-ban szereplő gráfmintát szerkeszthetjük.

VisualQuery Construct window interface. It features a large empty rectangular area for graph editing. Below this area is a 'Name:' label with a dropdown menu. At the bottom are three buttons: 'save', 'delete', and 'OK'.

22. ábra VisualQuery Construct

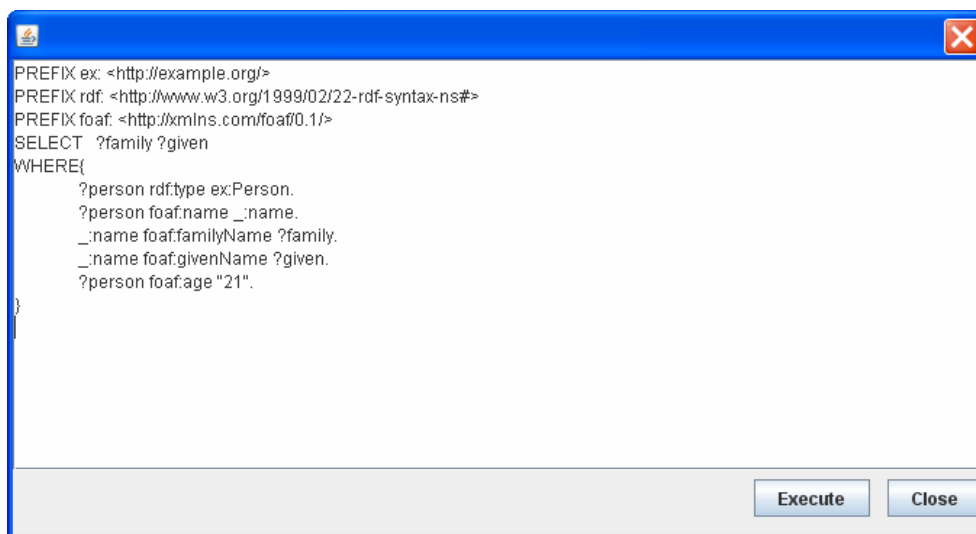
4. 1. 4.Vászon

A lekérdezéseket a vásznon állíthatjuk össze.



23. ábra VisualQuery vászon

A generált lekérdezést egy felugró ablakban nézhetjük meg.



24. ábra VisualQuery generált lekérdezés

4. 1. 5. Tulajdonság ablak

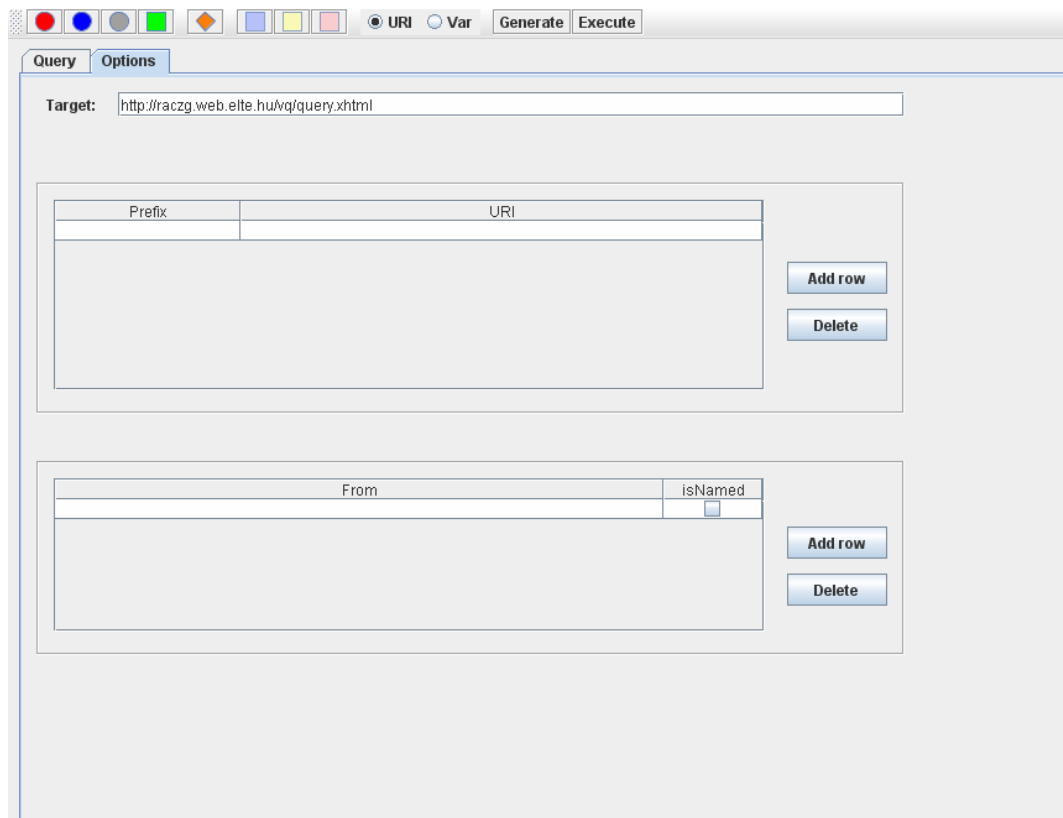
A vásznon egy tetszőleges elemre kattintva, a tulajdonság ablak segítségével, utólag is szerkeszthetjük az elem jellemzőit. Ezek a tulajdonságok típustól függően változhatnak, hiszen egy literálnál más opciókat kell szabályoznunk, mint egy URI esetén. Emiatt a tulajdonság ablak dinamikusan változik a vásznon kijelölt elemtől függően.

A screenshot of the 'VisualQuery' property window. It has a light gray background and contains the following fields and controls:

- Name:** A text input field containing the value 'person'.
- Suffix:** An empty text input field.
- Visible:** A checkbox that is currently unchecked.
- Direction:** A dropdown menu with a downward arrow.
- Buttons:** Two buttons labeled 'save' and 'delete'.
- Pattern:** A text input field containing the value '?person'.

25. ábra VisualQuery tulajdonság ablak

4. 1. 6.Options fül



26. ábra VisualQuery options ablak

A SPARQL végpontot, a használni kívánt prefixeket illetve a FROM záradékot egy új fülön az options ablakban adhatjuk meg:

- A végpontot egy beviteli mezőben,
- A prefixeket és a FROM záradékot pedig egy-egy táblázatban.

4. 2.Funkciók

A felület áttekintése után most ismerkedjünk meg a funkciókkal, az alkalmazás működésével, a SPARQL nyelv támogatott elemeivel!

4. 2. 1.Prefix

Mint láhattuk, lehetőségünk van prefixeket megadni az options fülön található táblázatban. A táblázat kitöltése értelemszerű: a prefix oszlopba kell írni a rövidítést, az URI oszlopba pedig a rövidítendő URI-t. Új sor hozzáadásához az 'Add row', sor törléséhez a 'Delete' gombot kell megnyomni.

Prefix	URI
ex	http://example.org/
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
foaf	http://xmlns.com/foaf/0.1/

27. ábra VisualQuery prefix megadás

A képen szereplő táblázatból a következő minta generálódik:

```
PREFIX ex: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

4. 2. 2.FROM záradék

A FROM záradékban szereplő adatforrásokat szintén az options fülön adhatjuk meg, a második táblázatban.

A from oszlopba kell írni az adatforrást, az isNamed oszlopban pedig beállíthatjuk, hogy nevesített forrás legyen-e, vagyis a FROM NAMED kulcsszót kapja-e meg. (Ekkor a WHERE záradékban tudunk rá hivatkozni a GRAPH szerkezetben.)

Sort hozzáadni és törölni hasonlóan lehet, mint a prefixnél az 'Add row' illetve a 'Delete' gombbal.

From	isNamed
http://dbpedia.org/sparql	☒
http://www.w3.org/TR/owl-guide/wine.rdf	☐

28. ábra VisualQuery adatforrás megadása

A képen szereplő táblázatból a következő minta generálódik:

```
FROM NAMED <http://dbpedia.org/sparql>
FROM <http://www.w3.org/TR/owl-guide/wine.rdf>
```

4. 2. 3.WHERE záradék

A lekérdezés legösszetettebb része a WHERE záradék. Az ebben szereplő kifejezéseket állíthatjuk össze vizuálisan.

Erre a Query fülön található vásznon van lehetőség, melyre a menüsoron található ikonokkal vihetjük fel a csúcsokat, a csúcsok összekötésével az éleket.

Minden csúcshoz másfajta tulajdonságok tartoznak, melyeket a tulajdonság ablakban dinamikusan változtathatunk.

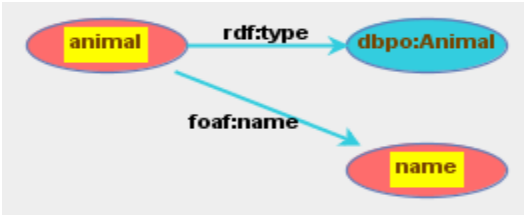
Tulajdonság neve	Típusa	Jelentése
Variable		
Name	Text	A változó neve, a '?' nélkül kell megadni.
Visible	Boolean	Megadja, hogy a változó értéke SELECT illetve DISTINCT típusú lekérdezés esetén szerepeljen-e a headerben.
Direction	Felsorolás	Megadja, hogy a lekérdezés ORDER BY záradékában szerepeljen-e a változó, illetve milyen irányban rendezzünk: • Üres – ne szerepeljen • ASCENDING – növekvő sorrend • DESCENDING – csökkenő sorrend
URI		
Name	Text	Az URI, melyet a csúcs reprezentáljon. Alkalmazható a prefixes alak is.
Visible	Boolean	Az URI megjelenjen-e a DESCRIBE típusú lekérdezés headerjében.
Blank node		
Name	Text	A blank node neve. _:name minta generálódik belőle.
Literal		
Name	Text	A szöveges csúcs értéke.
Suffix	Text	A szöveges csúcshoz fűzendő utótag.
Type	Felsorolás	Megadja, hogy a suffix mezőben szereplő értéket hogyan kell értelmezni: • Language: nyelvi tag, ekkor „name”@suffix • Datatype: adattípus, ekkor „name”^^<suffix> generálódik
FILTER		

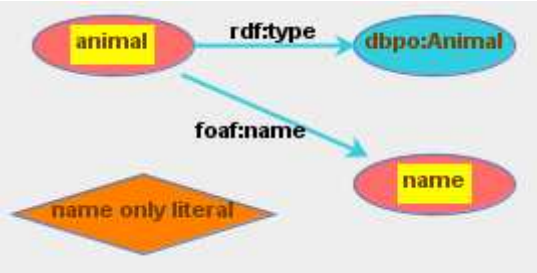
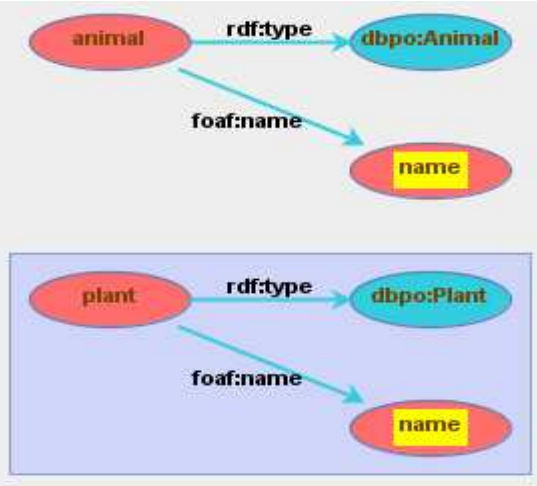
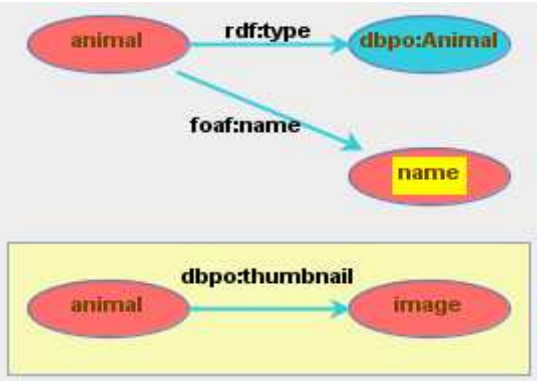
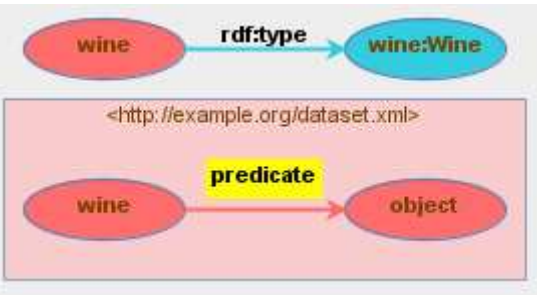
Name	Text	A filternek adhatunk egy megjelenítendő nevet, mely megjelenik a csúcson.
Expression	Text	A szűrőkifejezés. <i>FILTER expression</i> minta generálódik.
Group		
Name	Text	A csoportos mintát elnevezhetjük, hogy később tudjunk rá hivatkozni.
OPTIONAL		
Name	Text	A csoportos mintát elnevezhetjük, hogy később tudjunk rá hivatkozni.
GRAPH		
Name	Text	A csoportos mintát elnevezhetjük, hogy később tudjunk rá hivatkozni.
Parameter	Felsorolás	A megadott FROM záradék elemei, illetve a változók közül választhatjuk ki a GRAPH minta paraméterét.

2. Táblázat Csúcsok tulajdonságai

Nem csak csúcsokat, hanem éleket is rajzolhatunk a gráfba, ezeknek is vannak tulajdonságaik, ám azok megegyeznek a csúcsok esetén beállítható értékekkel.

A tulajdonságok után nézzük meg, hogy a WHERE záradékban megadható kifejezéseket hogyan is lehet grafikusán megszerkeszteni!

Egyszerű gráfmenta	
	<pre>WHERE{ ?animal rdf:type dbpo:Animal. ?animal foaf:name ?name. }</pre>
FILTER kifejezés	

	<pre>WHERE{ ?animal rdf:type dbpo:Animal. ?animal foaf:name ?name. FILTER isLITERAL(name) }</pre> A name only literal filter expression mezője: isLITERAL(name)
Group	
	<pre>WHERE{ ?animal rdf:type dbpo:Animal. ?animal foaf:name ?name. { ?plant rdf:type dbpo:Plant. ?plant foaf:name ?name. } }</pre>
OPTIONAL	
	<pre>WHERE{ ?animal rdf:type dbpo:Animal. ?animal foaf:name ?name. OPTIONAL { ?animal dbpo:thumbnail ?image. } }</pre>
GRAPH	
	<pre>WHERE{ ?wine rdf:type wine:Wine. GRAPH <http://example.org/dataset.xml> { ?wine ?predicate ?object. } }</pre>

3. Táblázat VisualQuery WHERE záradék példák

Az alkalmazás arra törekszik, hogy szintaktikailag helyes lekérdezéseket generáljon. A szintaktika mellett fontos a szemantika is, de annak ellenőrzése elég nehéz feladat.

A helyesség támogatása céljából a program bizonyos dolgokat nem enged meg. Következzen most a megszorítások listája:

- Literal csúcsból nem indulhat ki él, ugyanis Literal nem szerepelhet az alany helyén.
- Literal és Blank node típusú élet nem lehet húzni.
- Csoportos minták elemei között nem mehet él, ha mégis szükséges lenne, akkor az adott csúcsot újra fel kell venni a megfelelő helyen.
- A Filter típusú csúcsból nem indul és a csúcsba nem érkezik él.
- Az egyszerű hármasok az élek alapján generálódnak, vagyis a magában álló változó, URI, blank node, literál csúcsokból nem generálódik minta.
- Ha a Filter csúcshoz nem adunk meg szűrőkifejezést, akkor nem generálódik le.
- Ha a csoportos minták (group, optional, graph) nem tartalmazznak elemet, akkor nem generálódnak le.

A megszorítások listája a program fejlesztésével növekedni fog, ezáltal egyre jobb minőségű lekérdezések generálhatók a segítségével.

4. 2. 4.Eredménymódosítók

Az eredményt a módosító ablakban tudjuk befolyásolni.

Az Order by sorbarendezőben megjelennek a vásznon szereplő elemek közül azok, melyeknél megadható a Direction tulajdonság, és nem az üres érték van kiválasztva. Ezeknek a sorrendjét tudjuk itt meghatározni.

A Limitnél a LIMIT záradékot, az Offsetnél az OFFSET záradékot szabhatjuk testre. Amennyiben a Limit értéke 0, úgy egyik sem, ha az Offset értéke nulla, akkor az OFFSET záradék nem generálódik le.

4. 2. 5.Lekérdezés típusa

Az egyes lekérdezés típusok megadására az ablak dinamikusan változik.

- **SELECT**

A Select sorbarendezőben megadhatjuk a SELECT után szereplő változók sorrendjét. Azoknak a változóknak a neve szerepel a sorbarendezőben, melyeknél a Visible tulajdonság ki lett pipálva, ezek neve sárga háttérrel jelenik meg a vásznon.

A Modifierben a SELECT módosítóit adhatjuk meg, mely lehet DISTINCT, REDUCED vagy maradhat üresen.

- **ASK**

AZ ASK lekérdezés típusnak nincsenek extra beállításai.

- **DESCRIBE**

A Describe sorbarendezőben a SELECT-hez hasonlóan a változók és itt az URI-k sorrendjét is állíthatjuk.

- **CONSTRUCT**

Ebben az esetben aktiválhatjuk a 'Construct' gombot, melynek hatására feljön a Construct ablak, melyben a CONSTRUCT tagban szereplő gráfmintát adhatjuk meg, szintén grafikus formában.

Itt csak az egyszerű csúcsok szerepelnek a menü sorban, valamint csak a WHERE záradékban használt változók szerepelhetnek a mintában. A további generálási szabályok megegyeznek a WHERE záradéknál leírtakkal.

4. 3. Technikai megvalósítás

A felület áttekintése és a funkciók megismerése után következzen a háttér, a megvalósítás!

A leírás nem teljes körű, csak a legfontosabb elemeket emeli ki. A részletes leírást a készülő fejlesztői dokumentáció tartalmazza.

4. 3. 1. Programozási nyelv

Minden alkalmazás fejlesztése során kulcsfontosságú, hogy milyen programozási nyelven történik a megvalósítás. A jól megválasztott programozási nyelv nagymértékben lerövidítheti a fejlesztés idejét.

Az VisualQuery tervezése során a következő szempontok alapján választottam az általam ismert nyelvek közül:

- Mindenképpen objektumorientált nyelvre volt szükség, hiszen a vásznon megjelenő elemek mind-mind objektumok, melyek tulajdonságokkal rendelkeznek.
- Mindenhonnan elérhető alkalmazást szerettem volna készíteni, ezért kézenfekvő volt, hogy interneten keresztül is lehessen használni. Ezt szem előtt tartva a böngészőben futtatható alkalmazás jó választásnak tűnt.

- A nyelvhez tartoznia kell valamilyen gráfmegjelenítő, gráfkezelő API-nak, hogy ne kelljen mindent az elejéről megírni.
- Egy grafikus felületszerkesztőre is szükség van, hogy ne kelljen kódból elkészíteni a felhasználói felületet.

Ezek a szempontok igencsak leszűkítették az általam ismert programozási nyelvek körét.

Mivel elképzelhető, hogy a későbbiekben mégis szükség lesz egy asztali változatra is a programból, ezért a **Java** mellett döntöttem.

A böngészőben futtatható **Java Applet**et készítettem, amelyet a JAVA 1.6 – os változatával futtattam. Lehetőség van asztali környezetben is elindítani a programot, azonban ekkor az Execute funkció nem működik.

A fejlesztés a NetBeans IDE keretrendszer 7.0.1-es változatával történt, mely biztosította a felületszerkesztő eszközöket.

4. 3. 2. Gráfkezelő API

Szempon volt a létező gráfkezelő, -megjelnítő API. A Javához több ilyen programkönyvtár is elérhető:

- JUNG²⁷,
- HyperGraph²⁸,
- GVF²⁹,
- JGraph³⁰,
- Stb.

A <http://www.manageability.org/blog/stuff/open-source-graph-network-visualization-in-java/> oldalon található egy-egy rövid leírás az egyes nyílt forrású gráfmegjelenítő csomagokról, melyek Javában íródtak.

A rengeteg könyvtár kipróbálására, megismerésére idő hiányában nem volt lehetőségem. Ezért az általam már előzőleg használt **JGraph** mellett döntöttem.

A JGraph tartalmazza a fejlesztéshez szükséges eszközöket:

- Csúcs, él rajzolása
- Csúcsok csoportosítása

²⁷ Java Universal Network/Graph Framework: <http://jung.sourceforge.net/>

²⁸ The HyperGraph homepage: <http://hypergraph.sourceforge.net/>

²⁹ GVF- The Graph Visualization Framework: <http://gvf.sourceforge.net/>

³⁰ Java Graph Visualization Library: <http://www.jgraph.com/jgraph.html>

- Átméretezés
- Megjelenítési beállítások (szín, forma, betűtípus, betűméret, ..) szabályozása
- Interaktivitás (kijelölés, vonszolás) kezelése

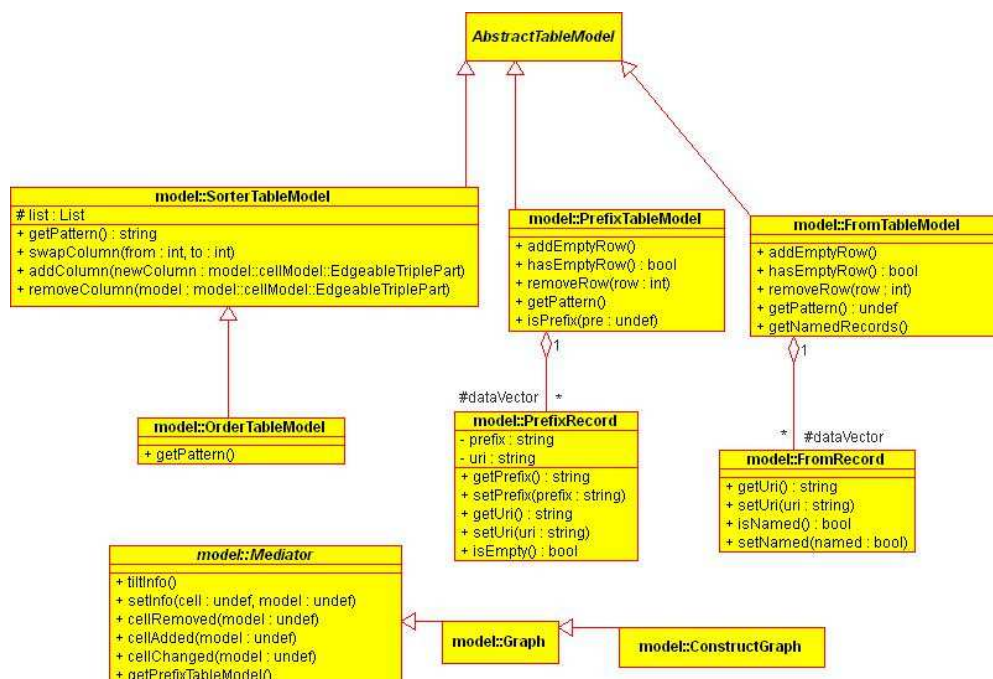
4. 3. 3.Csomagok, osztályok

Az osztályok modellezése során törekedtem a Modell-nézet-vezérlő³¹ szerkezeti minta kialakítására. A Java belső, névtelen osztályainak használata miatt néhol csorbul ez a minta.

• Model

A Model csomagban található az adatok tárolásáért felelős osztályok.

Az options fülön található Prefix és From táblázat modellje, a táblázat sorainak a modellje, valamint a vásznon szerkesztett gráf tárolására alkalmas osztály, mind-mind ebben a csomagban található.



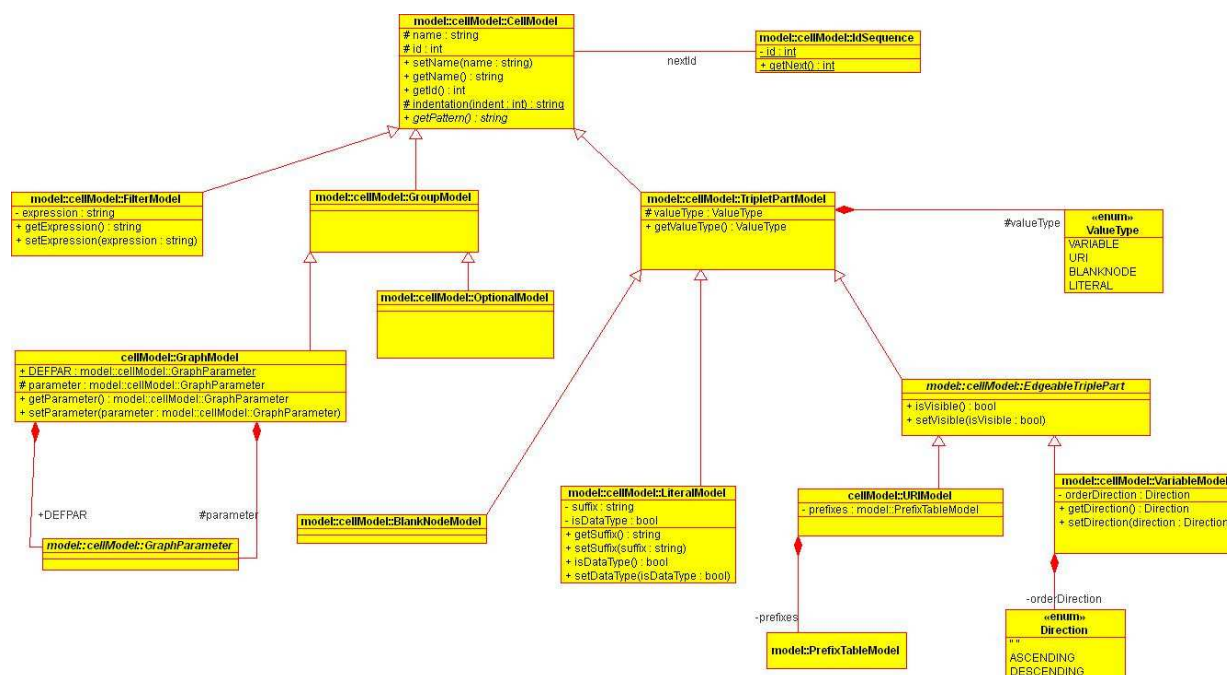
29. ábra VisualQuery Model osztálydiagram

Model.CellModel

A vásznon megjelenő elemek mindegyikéhez tartoznak tulajdonságok. Mint láthattuk Name tulajdonsággal minden elem rendelkezik, még ha esetenként eltérő is annak jelentése. Ezeket a tulajdonságokat tárolni kell a programban, ezért minden ilyen elemhez tartozik egy objektum. Minden elemtípushoz egy osztály.

³¹ Modell-nézet-vezérlő: <http://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91>

Ezek az osztályok a következő hierarchiába vannak szervezve:



30. ábra VisualQuery CellModel osztálydiagram

Vagyis minden a CellModel osztályból származik. A csoportos mintákat a GroupModel, míg az egyszerű mintákat az absztrakt TripletPartModel osztály fogja össze.

View

A view csomagban a megjelenítésért felelős osztályok kaptak helyet. A főablakot, azaz a menüt, a query illetve az options fület a MainPanel osztály tartalmazza.

A CONSTRUCT típusú lekérdezések CONSTRUCT részében szereplő gráf szerkesztéséhez létrehozott ablak a ConstructPanel osztályban van megadva.

A legenerált lekérdezést szöveges formában megjelenítő ablakot pedig a PatternDialog adja. Továbbá itt található a menüsoron megjelenő képek is, mint erőforrások.

Visualquery

Az alkalmazás elindításáért, Appletként való futtatásáért felelős VisualQuery osztály található itt, amely lényegében a MainPanel-t indítja el.

Az osztályok közül a Graph-ot, illetve a MainPanel-t emelném ki. Ezek adják a program vázát.

A Graph használja a JGraph gráf megjelenítő API segítségével létrehozott objektumot. Becsomagolja, kiegészíti, beállítja azt. Valamint kezeli az interakciókat (csúcsok kijelölése, törlés, él rajzolás, stb.).

A MainPanel pedig a felhasználói felület elemeit kapcsolja össze, tartja karban. Továbbá a lekérdezés szöveges változatának generálását is ez az osztály végzi.

5. Összegzés

A dolgozatban megismerkedhettünk az újabban egyre népszerűbb Szematikus Web kifejezés jelentésével, céljaival: az interneten keletkezett, keletkező adatmennyiség szemantikus annotációja, az annotációk jelentésének megadásával együtt, melynek segítségével a számítógép képesebb komplexebb módon értelmezni az adatokat a hatékonyabb, pontosabb keresési találatok elérésének érdekében.

Ezek után áttekintettük azokat az eszközöket, melyeket ezen célok megvalósításáért fejlesztettek ki. Az RDF erőforrás-leíró keretrendszert, mely segítségével tetszőleges objektumot erőforrásként értelmezve (alany, állítmány, tárgy) vagy (alany, tulajdonság, érték) hármasokkal le tudjuk írni az objektumról szerzett eddigi tudásunkat.

Az RDF-re épülő szókészlet-leíró nyelveket, az RDF Sémát és az OWL-t. Ezek az eszközök kibővítik a tudás leírására szolgáló technikákat. Segítségükkel meg tudjuk adni az annotációk jelentését, kapcsolatukat egymással, és az annotált objektumokkal, illetve bizonyos tulajdonságaikat (transzitivitás, funkcionalitás, stb.)

Miután megismertük a tudás leírására szolgáló eszközöket, megnéztük, hogyan lehet keresni az adatok között a speciálisan erre a célra kifejlesztett SPARQL lekérdezőnyelvvvel, amely a relációs adatbázisok világából jól ismert SQL nyelv sok vonását hordozza.

Ezek után azzal foglalkoztunk, hogy a SPARQL lekérdezéseket miképpen lehet elrejtetni a felhasználó elől, milyen felületeket lehet a technika felé építeni.

Ez volt a dolgozat létrejöttének kiindulópontja. Ugyanis a négy bemutatott lehetőség közül (tudásbázis-specifikus, aspektus-alapú, vizuális lekérdezés szerkesztők, természetes nyelvi kérdés megválaszolók) egy vizuális lekérdezés szerkesztő alkalmazás készült el, a VisualQuery.

Majd a témakörben született, hasonló alkalmazások közül a leggyakrabban használt, illetve legtöbbször hivatkozottakról olvashattunk egy rövid összefoglalót, összehasonlítva őket egymással, illetve a VisualQuery lehetőségeivel.

Az összehasonlítások után részletesebben láhattuk a VisualQuery felépítését. Mind a vizuális mind technikai felépítést. A felhasználó felületet, annak részeit: a vásznat, a menüsort, az options fület a két táblázattal és a végpont megadására szolgáló beviteli mezővel; az egyéb beállítási lehetőséget tartalmazó „ablakokat” (header-, tulajdonság-, módosító ablak). Az osztályok felépítését, hierarchiáját, csomagokra bontását. Mindezeket a megvalósított funkciók tükrében.

Összességében elmondható, hogy egy jól használható, lekérdezésszerkesztő program készült el. Melynek segítségével szemléletes, látványos SPARQL lekérdezéseket készíthetünk el vizuális formában, a kifejezőerő jelentősebb csorbulása nélkül. Hiszen a nyelvben leírható szerkezetek az alkalmazásban is megvalósíthatóak az UNIO művelettől eltekintve.

6. További tervek

A továbbiakra nézve több terv is szem előtt van.

Szó volt róla, hogy a vizuális lekérdezés építők nagyon alkalmasak dokumentációs célokra. Azonban ehhez szükség van mentés és exportálás funkciókra. Az appletek esetén a fájlok írása biztonsági kockázatot jelent a felhasználóknak, hiszen az appletek böngészőben futnak, és onnét kell a kliens gépén módosításokat végrehajtani. Erre persze megvannak a szabványos megoldások, de azok bonyolultságuk miatt még nem kerültek bele a programba.

Időközben a W3C emberei dolgoznak a SPARQL 1.1-es szabványon. A szabvány még fejlesztés alatt van, de már több végpont támogatja a szintaxist, ezért az új nyelvi elemeket szeretném beépíteni az alkalmazásba.

Az új elemeken kívül még az 1.0-ás ajánlásból is maradt olyan szerkezet, amit jelenleg nem tudunk leírni a VisualQuery-ben. Egészen pontosan az UNIO művelet, mellyel két csoportos gráfminitát köthetünk össze. Azonban ez a szerkezet helyettesíthető a megfelelően összerakott FILTER kifejezéssel.

A FILTER kifejezések szerkesztésének megkönnyítése is a célok között szerepel. Egy kifejezés szerkesztő formájában, mely tartalmazza a FILTER záradékba írható elemeket.

Továbbá folyamatosan lehet javítani a generált lekérdezés minőségét, a hibák szűrésével, például:

- URI-k validálása
- Prefixek használatának ellenőrzése
- Header-ek ellenőrzése (nem lehet üres változólista, nem lehet üres a CONSTRUCT mintája, nem lehet üres a WHERE záradék)

Egyéb segítséget lehet nyújtani a felhasználóknak, például a végpont megadása után lehet szűrni a végpont alatti modellben szereplő URI-kra, felkínálni azokat opcióként.

Mindezek mellett a vissza irány is érdekes lehet, azaz a szöveges formában megírt lekérdezés vizuális megjelenítése. Mint láthattuk az iSPARQL esetén már vannak előrelépések a témában, azonban még nem tökéletes a megvalósítás.

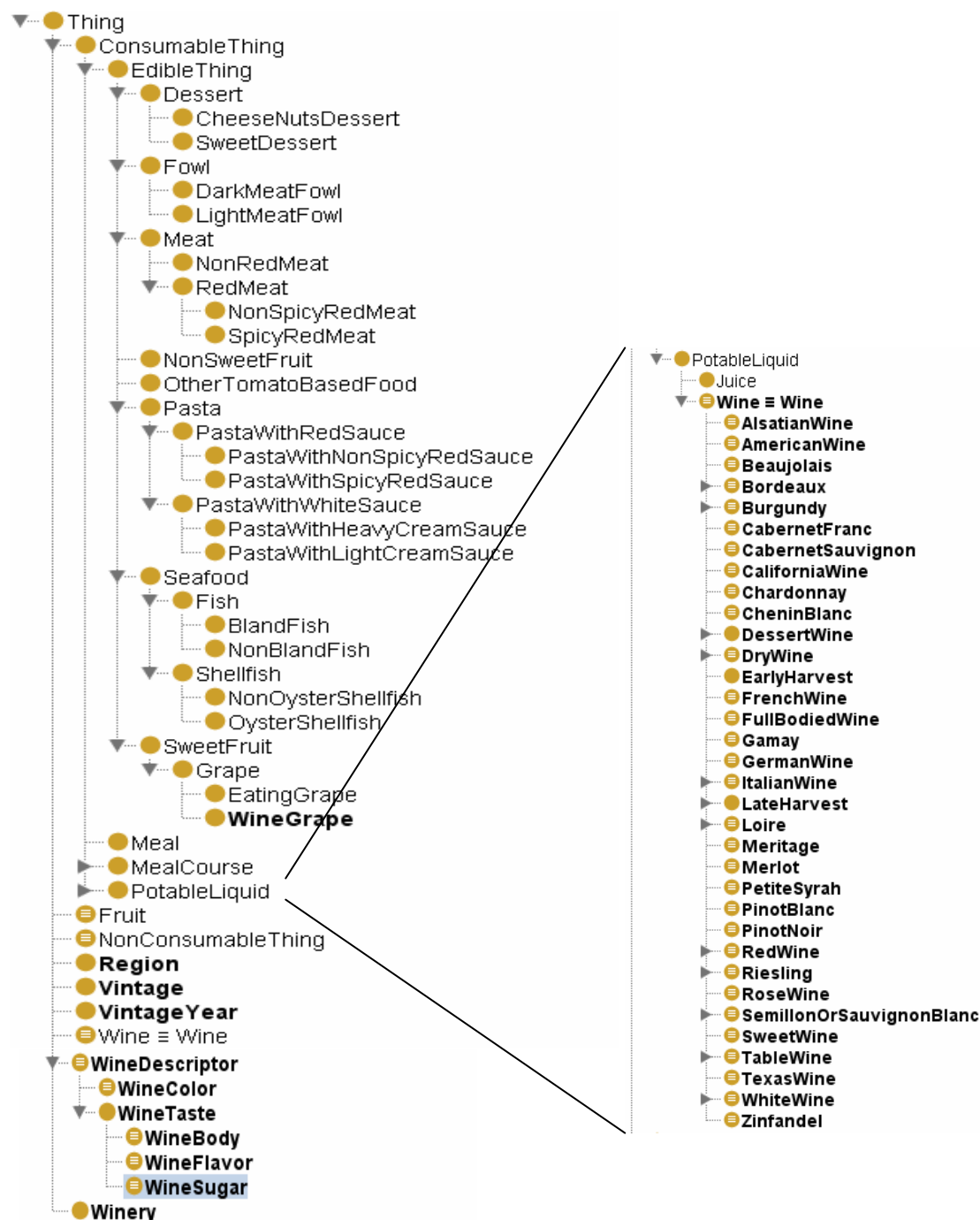
7. Irodalomjegyzék

- [1] A W3C küldetése <http://www.w3c.hu/forditasok/w3cmision.html>
- [2] W3C Semantic Web Activity <http://www.w3.org/2001/sw/>
- [3] Gottdank Tibor: Szemantikus web 118. oldal (ComputerBooks Kiadó, Budapest, 2005.)
- [4] Pataki Ernő: Az RDF bevezető tankönyve. (2005, fordítás)
<http://www.w3c.hu/forditasok/RDF/REC-rdf-primer-20040210.html>
- [5] DBpedia leírás. <http://dbpedia.org/About>
- [6] Identity resolution, Wikipedia oldal. http://en.wikipedia.org/wiki/Identity_resolution
- [7] Pataki Ernő: Az RDF Szókészlet Leíró Nyelv 1.0: RDF Séma (2005, fordítás)
<http://www.w3c.hu/forditasok/RDF/REC-rdf-schema-20040210.html>
- [8] OWL Web Ontology Language Overview, W3C Recommendation
<http://www.w3.org/TR/owl-features/>
- [9] SPARQL Query Language for RDF, W3C Recommendation
<http://www.w3.org/TR/rdf-sparql-query/>
- [10] Declarative Programming, Wikipedia oldal.
http://en.wikipedia.org/wiki/Declarative_programming
- [11] Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
<http://lod-cloud.net/>
- [12] Jens Lehmann, Lorenz Bühmann: AutoSPARQL: Let Users Query Your Knowledge Base. ESWC (1) 2011: 63-79
- [13] Philipp Heim, Thomas Ertl, Jürgen Ziegler: Facet Graphs: Complex Semantic Querying Made Easy. ESWC (1) 2010: 288-302
- [14] Russell, Alistair and Smart, Paul R. and Braines, Dave and Shadbolt, Nigel R.: NITELIGHT: A Graphical Tool for Semantic Query Construction. Semantic Web User Interaction Workshop (SWUI 2008), 5th April 2008
- [15] Modell-nézet-vezérlő, Wikipedia oldal.
<http://hu.wikipedia.org/wiki/Modell-n%C3%A9zet-vez%C3%A9rl%C5%91>

8. Függelékek

8. 1.Bor ontológia

A példák során többször találkoztunk a W3C által kifejlesztett bor, és a vele szorosan összetartozó food ontológiával. Ezek osztályhierarchiája a következőképpen néz ki:

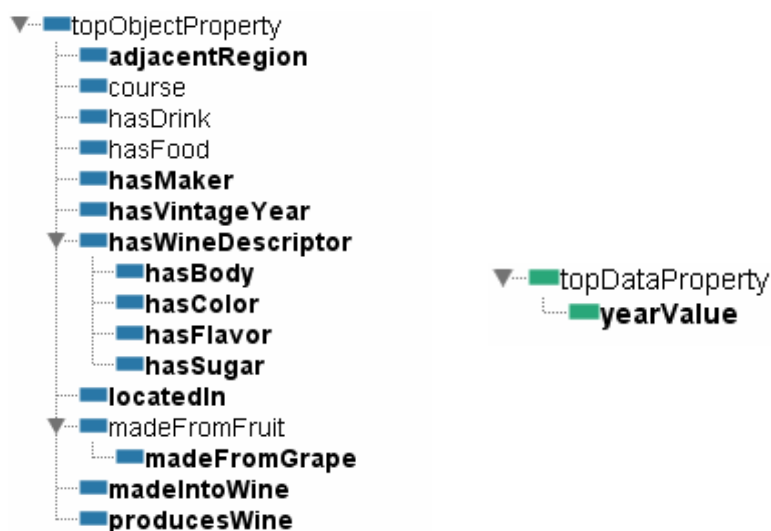


31. ábra Bor ontológia osztályszerkezet

A vastagon szedett osztályok a bor, a többi az étel ontológia része. A képek a Protege ontológiaszerkesztő programból származnak.

Az osztályok többségéhez tartoznak egyedek, és további megszorítások, de a dolgozat szempontjából ezek érdektelenek.

A tulajdonságok (állítványok) szerkezete:



32. ábra Bor ontológia tulajdonságok

8. 2.Futtatáshoz szükséges teendők

A futtatáshoz három fájl szükséges a következő könyvtárszerkezetben:

`./lib/jgraphx.jar`

`./visualquery.jar`

`./index.html`

8. 2. 1.Appletként

Az index.html-be be kell ágyazni az alkalmazást, a következőképpen:

```
<object type="application/x-java-applet"
  name="accessName" width="900" height="670">
  <param name="code" value="visualquery/VisualQuery.class" />
  <param name="archive" value="visualquery.jar" />
  <param name="scriptable" value="true" />
  <param name="mayscript" value="true" />
</object>
```

Ezek után az index.html-t böngészőbe betöltve az alkalmazás elindul.

8. 2. 2.Asztali alkalmazásként

A program asztali alkalmazásként is futtatható, ha a kialakított könyvtárszerkezetben állva kiadjuk a következő parancsot:

```
java -jar visualquery.jar
```

8. 2. 3.Webes elérhetőség

Az alkalmazás kipróbálható a <http://raczg.web.elte.hu/vq/index.html> címen és letölthető a <http://raczg.web.elte.hu/vq/vq.rar> URL-ről.