

TDK-dolgozat

Gombos Gergő

Eötvös Loránd Tudományegyetem

Informatikai kar

Információs Rendszerek Tanszék

**SPARQL lekérdezést futtató
WebService
mobil alkalmazások számára**

Szerző:
Gombos Gergő
MSc 2010

Témavezető:
Dr. Vincellér Zoltán
műszaki tanár

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

1. ABSZTRAKT	5
2. BEVEZETÉS	5
3. KORÁBBI MUNKÁK.....	6
4. SZEMANTIKUS WEB	9
4.1. MI AZ A SZEMANTIKUS WEB?	9
4.1.1. Gráf adatmodell	9
4.1.2. URI alapú szókészlet	9
4.1.3. Egyszerű tények kifejezése.....	10
4.2. SPARQL LEKÉRDEZÉS	12
4.2.1. SELECT.....	12
4.2.2. CONSTRUCT.....	12
4.2.3. ASK.....	12
4.2.4. DESCRIBE	12
5. WEBSERVICE.....	13
5.1. WEBSERVICE DEFINÍCIÓ	13
5.2. SOAP	13
5.2.1. Definíció.....	13
5.2.2. Előnyök.....	13
5.2.3. Hátrányok.....	14
5.3. REST.....	14
5.3.1. Definíció.....	14
5.3.2. Konceptió.....	14
5.3.3. Megszorítások	15
6. PROBLÉMA FELVETÉS	17
6.1. RENGETEG ADAT	17
6.2. KÖLTSÉGES SZÁMÍTÁSOK	17
6.3. AKKUMULÁTOR IDŐ	17
7. MEGVALÓSÍTÁS	18
7.1. AZONOSÍTÁS.....	18
7.2. LEKÉRDEZÉSEK.....	19
7.2.1. Lokális adatbázis modell hívás.....	19
7.2.2. Távoli hívás futtatása	20
7.2.3. Távoli hívás összevetése saját adatokkal.....	21
7.3. EREDMÉNYEK TÁROLÁSA, VISSZAADÁSA	21
7.4. AZ ALKALMAZÁS SZEKVENCIA DIAGRAMJA.....	23
8. EREDMÉNYEK	24
8.1. LEKÉRDEZÉS 1 (WINE ONTOLÓGIA)	24
8.2. LEKÉRDEZÉS 2 (2 FOKÚ KEVIN BACON[14]).....	24
8.3. LEKÉRDEZÉS 3 (MÚZEUMOK, MŰALKOTÁSOK, MŰVÉSZEK, IGAZGATÓK)	25
8.4. LEKÉRDEZÉS 4 (HONLAPOK NAGYJÁBÓL BERLIN KÖRNYÉKÉRŐL)	25
8.5. GRAFIKONOK.....	25
8.6. EREDMÉNYEK ÉRTÉKELÉSE	28
9. ÖSSZEFOGLALÁS	29
10. TOVÁBBI TERVEK	31
11. IRODALOMJEGYZÉK	32

1. Absztrakt

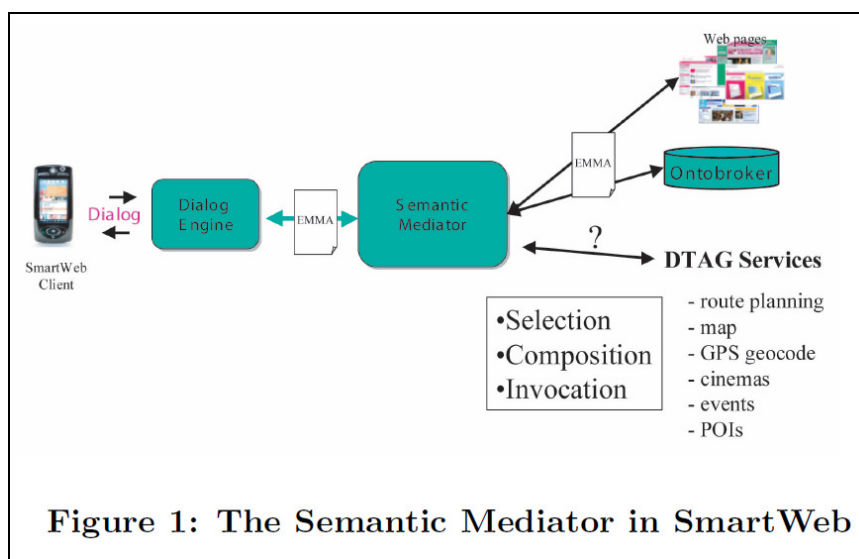
A szemantikus web lényege hogy az adatok rengeteg forrásból jöhetnek. Ezek az adatok nagyon nagy mennyiségűek, és nehezen kezelhetők. Az adatoknak a kezeléséhez komoly erőforrás szükséges, mellyel nem minden kliens rendelkezik. Ilyenek például a mobilkészülékek. A legtöbb mobil készülék már rendelkezik Internet kapcsolattal, így a szemantikus technológia már számukra is elérhető. Az erőforrásaik viszont korlátosak. Erre a problémára készült egy webservice, ami futtatni képes a szemantikus lekérdezéseket, és szolgáltatni tudja a mobil készülékek számára. A webservice egyik erőssége hogy nem a teljes adatmennyiséget küldi át, terhelve így a telefon memóriáját, hanem kisebb adatmennyiséget és a készülék döntheti el, hogy mikor szeretné a további adatokat megkapni.

2. Bevezetés

Az eddig relációs adatbázisokban tárolt adatok, manapság már elérhetőek szemantikus technológiákkal is. A szemantikus technológia alapja, hogy az adatok mindenhol jöhetnek, és mindenhol elérhetőek. Rengeteg publikus adatforrás érhető el világszerte. Az egyik legnagyobb ilyen adatbázis a wikipédia szemantikus megfelelője a dbpedia [1]. Ezen adatokat úgynevezett SPARQL végpontokon [2] tudjuk elérni. Sok alkalmazás használja ezeket a végpontokat. Az végpontokról érkező adatok mennyisége a lekérdezésünktől függően, akár több 10000 sor is lehet. Ezt az adatmennyiséget az alkalmazásoknak tudniuk kell tárolni, kezelni. A mobiltelefonokra és táblagépekre írt alkalmazások figyelnek arra, hogy a készülék erőforrása korlátos, így nem tudnak nagy adatokkal dolgozni. A jelen dolgozatban bemutatott alkalmazás segít a kis erőforrású eszközökre írt alkalmazások számára, hogy egyszerűen tudják kezelni a nagy mennyiségű adatokat. Az alkalmazás alapja hogy nem a telefon erőforrását használja számolásra, tárolásra, hanem az alkalmazás szerverét. Egy szerverben több GB memória található, ami jóval több, mint a telefon pár MB-os memóriája.

3. Korábbi munkák

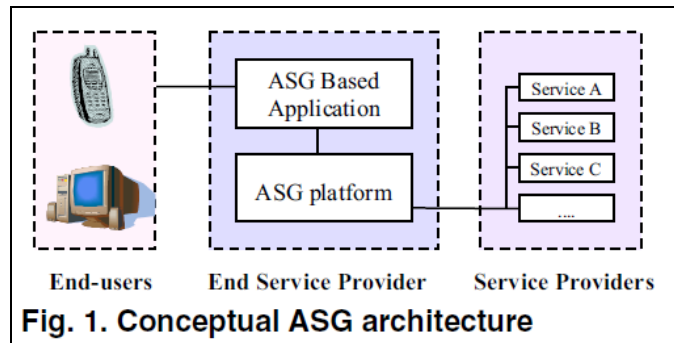
Egy hasonló munka a Karlsruhe-i egyetemen készült Semantic Mediator [8], ami a felhasználói kérdéseket értelmezi, majd ahhoz a webservicehez továbbítja az üzenetet, aki meg tudja válaszolni. A felhasználói kéréseket egy Dialog Engine átalakítja EMMA dokumentummá, ami szemantikus reprezentálja a kérést. Ezt a documentumot fogja feldolgozni a Semantic Mediator, hogy megtalálja a megfelelő webservice-t. A webservicek leírásainak tárolására az Ontobrokert használják.



1. ábra

A technológia a német Telekom-nak készült, ami több mint 50 webservice-szel rendelkezik. Ezek között vannak időjárás előrejelző, útvonal tervező, mozik koordinátáit tároló, mozi műsorokat tároló webservicek.

Egy másik ehhez hasonló munka a finnek által készített ASG rendszer[9]. Az ASG, EU finanszírozásból készült Adaptive Service Grid, aminek célja a service-ek szemantikus információi alapján történő tervekészítés és monitorozás.



2. ábra

Ennek a rendszernek a lényege, hogy összefüggő kéréseket tud kezelni. Ilyen például az utazás, amihez le kell foglalni repülőt, és szállást. A rendszer ezt a kérést fogja fogadni, és lépésről lépésre feldolgozza azt. Például, ha el szeretnék utazni Berlinbe Budapestről tavasszal, akkor a rendszer lefoglal jegyet a repülőre a megfelelő webservice segítségével, majd lefoglal egy hotelt egy másik webservice segítségével.

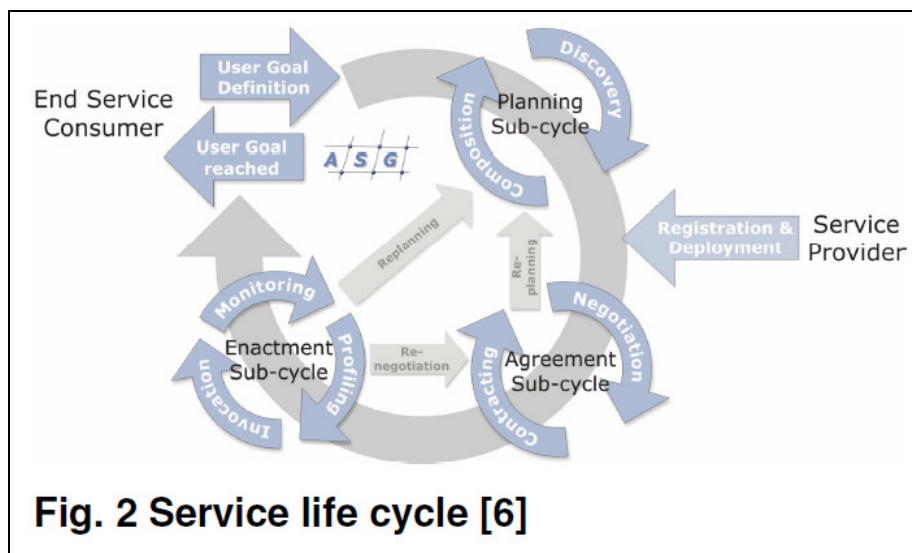
A 2. ábrán látható hogy a rendszer 4 főrészből:

1 – End Service Provider: Ez a rendszer lelke, ez kezeli a felhasználóktól érkezett kéréseket, amiket továbbít az ASG alkalmazásnak, majd a Service-eknek.

2 – End-User: A kérést elindító fél. Ez egy alkalmazás, aki küldi a funkcionális kérését az End Service Providernek hogy dolgozza fel, és válaszoljon.

3 – Service Provider: A különböző webservice-k, amik célzott feladatokat látnak el. Ilyen például a repülőjegy foglalás. Ezeket a serviceket a gyakorlatban aszerint csoportosítják, hogy milyen feladatokat látnak el. A csoportosítás általános, hogy ha esetleg egy service nem elérhető, akkor egy hozzá hasonló, vagy vele megegyező service is el tudja látni a feladatot.

4 – ASG-based application: A kapott kéréseket értelmezi, majd küldi a megfelelő service-eknek, és az azoktól kapott választ szintén értelmezi és küldi vissza az End-User-nek.



3. ábra

A folyamat leírása a 3. ábrán látható.

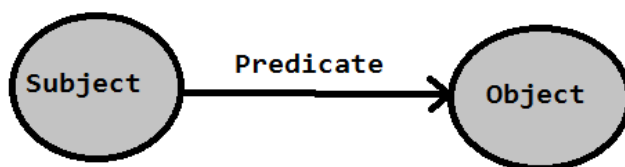
4. Szemantikus web

4.1. *Mi az a szemantikus web?*

Az RDF (Resource Description Framework) egy keretrendszer a weben történő információábrázolás céljára. Az RDF az alábbi fogalmakat használja: gráf adatmodell, URI alapú szókészlet, egyszerű tények kifejezése, következmény.

4.1.1. Gráf adatmodell

Az RDF-ben bármilyen kifejezés alapstruktúrája olyan hármas (egyszerű kijelentő mondatok) gyűjteményéből áll, amelyek mindegyike egy alanyt (subject), egy állítmányt (predicate) és egy tárgyat (object) tartalmaz. Az ilyen hármasok halmazát RDF gráfnak nevezzük. Ennek az irányított gráfnak egy hármasát láthatjuk az 4. ábrán.



4. ábra

Az él iránya szignifikáns, és ennek mindig a tárgyra kell mutatnia. Az RDF gráf csomópontjai a gráf alanyai és tárgyai. Egy RDF gráf állítástartalmát a gráfot alkotó hármasok állításainak összessége adja, vagyis egy RDF gráf jelentése nem más, mint a benne lévő összes hármas által megfogalmazott kijelentés összessége.

4.1.2. URI alapú szókészlet

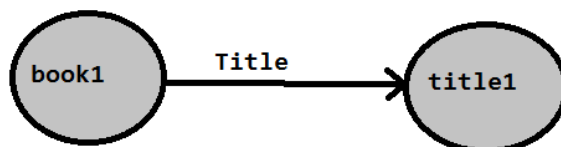
Egy csomópont lehet egy URI egy erőforrásrész-azonosítóval, lehet literál, vagy lehet egy üres csomópont. A tulajdonságok (a gráf-élek) szintén lehetnek URI hivatkozások. Egy állítmányként használt URI hivatkozás nemcsak élként, hanem csomópontként is előfordulhat ugyanabban a gráfban, minthogy a tulajdonságokról, magukról, is tehetünk állításokat. Az üres csomópont egy olyan csomópont, mely nem URI hivatkozás, és nem is literál. Az RDF absztrakt szintaxisában az üres csomópont csak egy speciális segédcsomópont, amelyet egy vagy több RDF kifejezés használ, de amelynek nincs valódi neve.

4.1.3. Egyszerű tények kifejezése

Egy egyszerű tény valamilyen viszonyt fejez ki két dolog között. Egy ilyen tényt egy RDF hármassal fejezhetünk ki, amelyben az állítmány nevezi meg a viszonyt, az alany és a tárgy pedig a két dolgot, amelyre az állítmány vonatkozik. Egy ilyen tény ismerős ábrázolása pl. egy relációs adatbázis két oszlopból álló táblázatának egyik sora, ahol a két oszlop egy RDF hármassal alanyának és tárgyának, a táblázat neve pedig a hármassal állítmányának felel meg. Ezt mutatja a 5. és 6. ábra.

Title (Predicate)		
	Book (Subject)	Title (Object)
1.	book1	title1
2.	book2	title2
..		

5. ábra

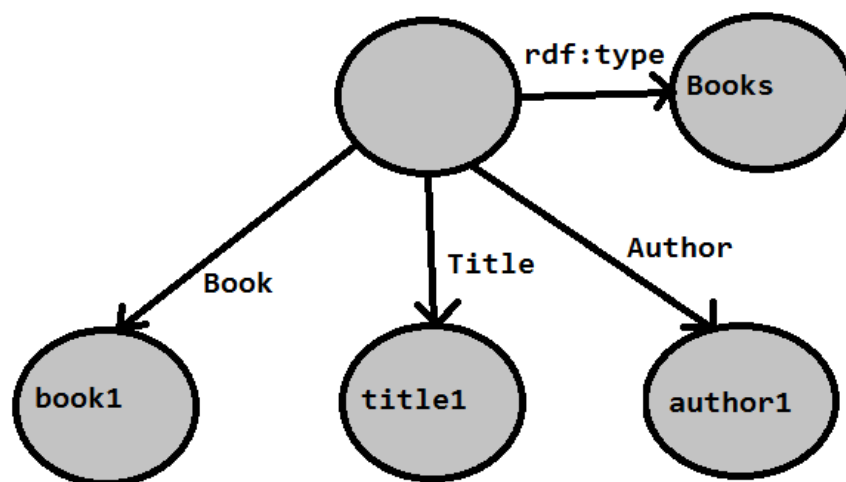


6. ábra

A relációs adatbázisok megengedik, hogy a táblázatuknak tetszőleges számú oszlopa legyen. Ennek legegyszerűbb módja az, hogy bevezetünk egy új, üres csomópontot, mely a sornak felel meg, és egy-egy hármast adunk meg a sor minden cellája számára. Ily módon minden hármassal alanya megfelel az üres csomópontnak, az állítmánya a megfelelő oszlop nevének, a tárgy pedig a cellában lévő értéknek. Az üres csomópontnak lehetne egy `rdf:type` tulajdonsága is, mely a táblázat nevének felelne meg.

Books			
	Book	Title	Author
1.	book1	title1	author1
2.	book2	title2	author2
..			

7. ábra



8. ábra

Láthatjuk tehát, hogy egy összetettebb tényt tudtunk ábrázolni az RDF-ben is. Megjegyzendő azonban, hogy az RDF nem teszi lehetővé a diszjunkció (VAGY kapcsolat) és a negáció (NEM) kifejezését.

4.2. SPARQL lekérdezés

A SPARQL az RDF-hez készült lekérdező nyelv. Hasonló az SQL-hez, vagy a XQuery-hez. Feladata a szemantikusan tárolt adatok lekérdezésének biztosítása. A SPARQL lehetőséget biztosít az adatok több forrásból való lekérdezésére, konjunkcióra, diszjunkcióra, valamint pontos egyezésre, ill. opcionális egyezésre a gráfban. Az alábbiakban a SPARQL lekérdezéseket mutatom be.

4.2.1. SELECT

A gráf egy részhalmazát adja vissza, amire teljesül a gráf egyezés.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
WHERE
{ ?x foaf:knows ?y ;
  foaf:name ?nameX .
  ?y foaf:name ?nameY .
  OPTIONAL { ?y foaf:nick ?nickY }
}
```

4.2.2. CONSTRUCT

Egy RDF gráfot ad vissza, amire teljesülnek a feltételek.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }
WHERE      { ?x foaf:name ?name }
```

4.2.3. ASK

Logikai értékkel tér vissza, aszerint hogy a lekérdezés teljesül, vagy nem.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Alice" }
```

4.2.4. DESCRIBE

Visszaad egy olyan RDF gráfot, ami az RDF adat erőforrásait tartalmazza.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE { ?x foaf:mbox <mailto:alice@org> }
```

5. Webservice

5.1. Webservice definíció

A web szolgáltatás [15] vagy angolul webservice alkalmazások közötti adatcserére szolgáló protokollok és szabványok gyűjteménye. Különböző programnyelveken írt és különböző platformokon futó szoftveralkalmazások számítógép-hálózatokon (mint az Internet) keresztültörténő adatcserére használják a web szolgáltatásokat. A web szolgáltatásoknak 2 féle megvalósítása lehetséges. Az egyik a REST-es, a másik a SOAP-os megközelítés.

5.2. SOAP

5.2.1. Definíció

A számítástechnikában a SOAP egy protokoll, amelyet webszolgáltatások hálózaton keresztüli kommunikációjához terveztek. Az üzenetek XML-alapúak és általában más, alkalmazás réteg-béli protokollokra támaszkodva továbbítódnak, például Távoli Eljáráshívás (Remote Procedure Call, RPC) vagy a Hypertext Transport Protocoll (HTTP) protokollokra. A SOAP-nál az autentikáció a SOAP Header-ben van megvalósítva. SOAP-ot általában összetettebb IT rendszerekben használnak, például SOA architektúrában. Ilyenkor egy üzenetközvetítő, ún. message broker gondoskodik az autentikáció ellenőrzéséről, az üzenetek továbbításáról (akár gyorsítótárazásáról is). Az XML formátumra esett a választás, mint standard üzenetformátum, mivel széles körben használt a nagyvállalatoknál és a nyílt forrású fejlesztéseknél. Ezen kívül rengeteg szabadon elérhető eszköz segíti a SOAP alapú megoldásra való áttérést. Az XML szintaxisa lehet előny és hátrány is egyszerre. Az emberi szem számára könnyen olvasható, könnyebb a hibadetektálás és áthidalja az együttműködési problémákat, emellett viszont lassan és nehezen feldolgozható.

5.2.2. Előnyök

A SOAP elég sokoldalú, így különböző szállítási protokollokon keresztül is kommunikálhatunk vele. A szabvány szerint HTML-t használnak, de működik SMTP-vel vagy JMS-el is. Mivel a SOAP jól összefér a HTTP kérés/válasz modelljével, könnyedén átjut a tűzfalakon és proxykon anélkül, hogy a SOAP protokollt módosítani kellene.

5.2.3. Hátrányok

Az XML bőbeszédű formátuma miatt a SOAP lassabb lehet. Csak az egyik fél (kliens) használhatja a másik fél szolgáltatásait. Csak a POST HTTP-igét használják. Emiatt sok létező HTTP képesség nincs kihasználva (pl. gyorsítótárazás, autentikáció).

5.3. REST

5.3.1. Definíció

A REST (Representational State Transfer) egy szoftverarchitektúra típus elosztott hipermédia rendszerek számára, amilyen például a világháló. A Representational State Transfer kifejezést Roy Fielding vezette be és definiálta 2000-ben a doktori disszertációjában. Azokat a rendszereket, amelyek eleget tesznek a REST megköveteléseinek, "RESTful"-nak nevezik. A legnagyobb olyan rendszer, amely eleget tesz a REST szoftverarchitektúra típus követelményeinek az a világháló.

5.3.2. Konceptió

Egy REST típusú architektúra kliensekből és szerverekből áll. A kliensek kéréseket indítanak a szerverek felé; a szerverek kéréseket dolgoznak fel és a megfelelő választ küldik vissza. A kérések és a válaszok erőforrás-reprezentációk szállítása köré épülnek. Az erőforrás lényegében bármilyen koherens és értelmesen címezhető koncepció lehet. Egy erőforrás-reprezentáció tipikusan egy dokumentum, mely rögzíti az erőforrás jelenlegi vagy kívánt állapotát.

Bármely adott pillanatban, egy kliens vagy állapotok közötti átmenetben van, vagy "nyugalmi" állapotban. A nyugalmi állapotban lévő kliens képes interakcióra a felhasználójával, de nem hoz létre terhelést és nem fogyaszt tárolót a szervereken vagy a hálózaton.

Ha a kliens készen áll az átmenetre egy új állapotba, akkor elkezdi küldeni a kéréseit a szerverekhez. Míg legalább egy olyan kérés van, amelyre nem érkezett válasz, a kliens átmeneti állapotban marad. Egyes erőforrás-reprezentációk hivatkozásokat tartalmaznak további erőforrásokra, amelyeket a kliens felhasználhat új állapotba történő átmenetkor.

A REST eredetileg a http keretein belül lett leírva, de nem korlátozódik erre a protokollra. Egy "RESTful" architektúra más alkalmazási réteg protokollra is épülhet, amennyiben az már rendelkezik értelmes erőforrás-reprezentáció átvitelhez szükséges gazdag és egységes szókinccsel. A "RESTful" alkalmazások maximálisan kihasználják a választott hálózati

protokoll már létező, jól kialakított interfészeit és egyéb beépített képességeit, és minimalizálják új alkalmazás-specifikusjellemzők bevezetését.

A HTTP nagyon gazdag szókinccsel rendelkezik igék (vagy "metódusok"), URI-k, média típusok, kérés- és feleletkódok, stb. szempontjából. Egy REST alkalmazás a HTTP protokoll meglévő tulajdonságait használja és így lehetővé teszi a proxyknak és az átjáróknak, hogy együttműködjenek az alkalmazással (például gyorsítótárazás vagy biztonsági funkciók formájában).

A fentiekkel szemben a SOAP RPC protokoll arra ösztönzi az alkalmazásfejlesztőket, hogy új és önkényes főnév és ige szókinccset definiáljanak (például getUsers(), savePurchaseOrder(...)) és csak a POST HTTP-igét használják. Emiatt sok létező HTTP képesség nincs kihasználva (pl. gyorsítótárazás, autentikáció).

5.3.3. Megszorítások

Egy REST architektúra a következő hat megszorításnak kell megfelelnie, miközben az egyes komponensek implementációit hagyja szabadon tervezni:

- **Kliens-szerver architektúra**

A kliensek el vannak különítve a szerverektől egy egységes interfész által. Az érdekeltségek ilyen nemű szétválasztása azt jelenti, például, hogy a kliensek nem foglalkoznak adattárolással, ami a szerver belső ügye marad, és így a kliens kód hordozhatósága megnő. A szerverek nem foglalkoznak a felhasználói felülettel vagy a kliens állapotával, így a szerverek egyszerűbbek és még skálázhatóbbak lehetnek. A szerverek és kliensek áthelyezhetőek és fejleszthetőek külön-külön is, egészen addig, amíg az interfész nem változik meg.

- **Állapotmentesség**

A kliens-szerver kommunikáció tovább korlátozott az által, hogy a szerveren nem tárolják a kliens állapotát a kérések között. Minden egyes kérés bármelyik kienstől tartalmazza az összes szükséges információt a kérés kiszolgálásához, és minden állapotot a kliens tárol. A szerver lehet állapottartó; ez a korlátozás csupán azt követeli meg, hogy a szerver oldali erőforrás-állapotok URL által címezhetőek legyenek. Ez nem csak a szerver felügyeletét teszi lehetővé, de megbízhatóbbá teszi őket a hálózati meghibásodásokkal szemben, valamint tovább fokozza a skálázhatóságot.

- **Gyorsítótárazhatóság**

Mint ahogy a világhálón, a kliensek itt is képesek gyorsítótárazni a válaszokat. A válaszoknak ezért impliciten vagy expliciten tartalmazniuk kell, hogy gyorsítótárazhatóak-e vagy sem. Így elkerülhető, hogy a kliens téves vagy elavult adatokat használjon fel újra. Egy jól menedzselt gyorsítótár lehetővé teszi, hogy teljesen megkerüljünk egyes kliens-szerver interakciókat, továbbá megnöveli a rendszer skálázhatóságát és a teljesítményét.

- **Réteges felépítés**

Egy kliens általában nem tudja megmondani, hogy direkt csatlakozott-e a végpont szerverhez, vagy közvetítő segítségével. A közvetítő szerverek megnövelhetik a rendszer skálázhatóságát terheléeloszlás kiegyenlítéssel és megosztott gyorsítótárak használatával.

- **Igényelt kód** (opcionális)

A szerverek képesek időlegesen kiterjeszteni vagy testre szabni egy kliens funkcionalitását, programrészek átadásával, amelyet a kliens képes futtatni képes. Ide tartoznak az előre fordított komponensek (pl. Java appletek) és a kliens oldali szkriptek (pl. JavaScript).

- **Egységes interfész**

Az egységes interfész kliens és szerver között egyszerűsíti és kettéválasztja az architektúrát, és lehetővé teszi, hogy egymástól függetlenül fejlődjenek az egyes részek.

A REST architektúra egyetlen opcionális megszorítása az igényelt kód. Ha egy szolgáltatás sért bármely más megszorítást, azt nem lehet feltétlenül "RESTFull"-nak nevezni.

A webservice amit készítettem, megsérti az állapotmentességet, pont azért, hogy ne a klienst terhelje, így az alkalmazásom nem RESTFull. Az utóbbi időben a két architektúra verseng egymással. A REST-es architektúrának megvan az az előnye, hogy nincs megkötve az üzenetküldés formátuma. A javascriptnek például időigényes az XML adatok feldolgozása. A REST architektúra például megengedi a JSON formátumot is, amit egy javascript könnyebben tud kezelni.

6. Probléma felvetés

6.1. Rengeteg adat

A szemantikus rendszerek egyik nagy problémája, hogy nagy mennyiségű adatot tartalmaz. Egy egyszerű lekérdezés is adhat vissza több ezer, tízezer sort. Az egyes sorokban is az adatok változó méretűek lehetnek. A legtöbb adat URI-ként van reprezentálva, de vannak olyan adatok is, mint például a leírások, amik magukban is sok helyet foglalnak. Ezeket az adatok a kisebb erőforrással rendelkező kliensek nehezen tudják kezelni.

6.2. Költséges számítások

A szemantikus rendszerek másik nagy problémája a lekérdezések, és következtetések számítási költségei. A lekérdezések futhatnak lokálisan egy saját adatbázisban, de akár több SPARQL végponton is. Ezeknek a kezelése is rengeteg költségbe kerül egy telefon számára, még egy szerver ezt könnyebben megoldja.

6.3. Akkumulátor idő

Mindkét esetben nem a telefonon történne a műveletek elvégzése, így nem a telefon erőforrásait használná. Ezzel az akkumulátor idejét is nagyban növelhetjük, hisz kicsi, és könnyen elvégezhető műveleteket kell elvégeznie.

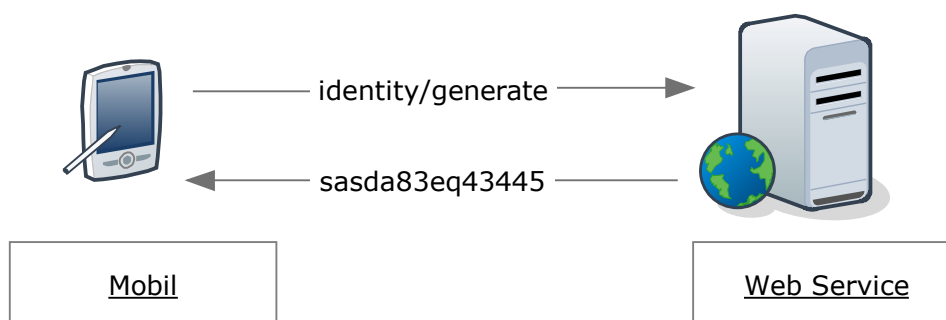
7. Megvalósítás

7.1. Azonosítás

Elérés	/identity/generate
Hívási protokoll	GET
Paraméterek	-
Hívási példa	<code>service.path("identity").path("generate") .accept(MediaType.TEXT_PLAIN) .get(ClientResponse.class);</code>
Válasz eredmény	Szöveges üzenetet ad vissza, ami tartalmazza az azonosítót.

Elérés	/identity/dispose/{azonosító}
Hívási protokoll	GET
Paraméterek	<u>azonosító</u> – a generate-tel generált azonosítónk
Hívási példa	<code>service.path("identity").path("dispose/"+azon) .accept(MediaType.TEXT_PLAIN) .get(ClientResponse.class)</code>
Válasz eredmény	Szöveges üzenet, ami jelzi, hogy a törlés sikeres.

A rendszer lényege hogy minden készüléket egy azonosító jelképez és ezzel az azonosítóval tudnak dolgozni. Minden készüléknek először kérnie kell egy egyedi azonosítót. Ez az azonosító a java secure.SecureRandom-mal generált karakter sorozat. A kliensnek ezt meg kell jegyeznie, és minden hívásnál el kell küldenie a server felé. Ez hasonló a böngészőknél alkalmazott session azonosítókhoz. Ezt az elérést egy GET hívással lehet elérni az /identity/generate alatt. Amikor már nincs szükségünk az erőforrásra, akkor meg is szüntethetjük azt. Ezt szintén egy GET metódushívással történhet, amit a /identity/dispose –ra kell küldeni.



9. ábra

7.2. Lekérdezések

Az alkalmazásban az egyik kihívást a lekérdezések kezelése okozta. Az egyik probléma hogy a szemantikus technológia megenged többféle hívást is. Lehet lekérdezést futtatni egy másik végponton, és lokálisan is. A távoli végpont elérését is több féle képen lehet értelmezni.

Elérés	/query/run
Hívási protokoll	POST
Paraméterek	<u>id</u> - a generate-tel generált azonosítónk, vagy a belépéskor kapott azonosítónk <u>query</u> - a sparql lekérdezésünk <u>model</u> - (opcionális) a lokális modell nevünk, amire a lekérdezést futtatni szeretnénk <u>endpoint</u> - (opcionális) az endpoint amire a lekérdezésünket el szeretnénk küldeni
Hívási példa	<pre>form = new Form(); form.add("id", azon); form.add("query", "select (count(?s) as ?c) „+ „where { ?s ?p ?o }"); // ha modellre szeretnénk futtatni form.add("model", "wine851O"); // ha endpointra szeretnénk futtatni form.add("endpoint", "http://sparql.org/books"); service.path("query").path("run") .type(MediaType.APPLICATION_FORM_URLENCODED) .post(ClientResponse.class, form);</pre>
Válasz eredmény	A lekérdezés eredményének első 10 sorát adja vissza szöveggént.

7.2.1. Lokális adatbázis modellhívás

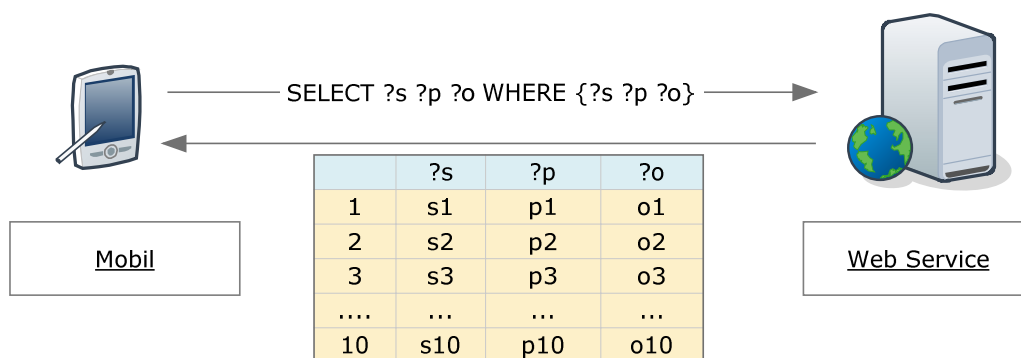
A lokális modellhíváshoz szükségünk van egy szemantikus adatbázisra vagy olyan keretrendszerre, ami szemantikus technológiát tud megvalósítani. A legtöbbet használt keretrendszer a JENA[4], ami java alapú, és sok mindent támogat. Az adatbázisok közül a szemantikus technológiát legjobban támogató az Oracle[5]. A Jena-nak épp ezért van egy változata, ami a Jena Adapter[6], ami kihasználja az Oracle sajátosságait. A másik adatbázis,

amire az alkalmazást tesztelni tudtam egy nyílt forrású adatbázis a PostgreSQL[7]. Erre a sima Jena Api-t használtam.

Hogy is működik a rendszer? Az adatbázishoz való kapcsolódáshoz szükség van JDBC kapcsolatokra, amit az alkalmazás szerveren kell beállítani. Kell egy JDBC kapcsolat, amit az Oracle-höz tudunk használni, ill. kell egy, amit más adatbázisokhoz. Ezek az adatbázisok lehetnek relációsok is, mert a Jena tudja tárolni az adatokat bennük.

A folyamat a következő, hogy ha bejön egy kérés, ami lokális modellhez akar csatlakozni, akkor megpróbálunk kapcsolódni az Oracle-hez, majd ha ez nem sikerült, akkor a postgres-hez. Ez azért van így kitalálva, hogy ha valaki használni szeretné az alkalmazást saját rendszerén, akkor kevés beállítással tudja alkalmazni bármilyen adatbázisra.

Mi kell ahhoz, hogy a hívásom a lokális adatbázison fusson le? A hívásnál vannak opcionális paraméterek. Ezek a paraméterek döntenek el, hogy a lekérdezés hol fog lefutni. Ha van modell, akkor a lekérdezésünk a JDBC-en keresztül a saját adatbázisunkban fog lefutni.



10. ábra

7.2.2. Távoli hívás futtatása

Ha nem a saját adatbázisunkra akarjuk futtatni a lekérdezést, akkor lehetőségünk van megadni sparql végpontot, amire csatlakozni szeretnénk. Ekkor a webservice felé küldött kérés nem a webservice-n fut le, hanem tovább küldődik a végpont felé. Ehhez az kell, hogy a paraméterek között szerepeljen egy endpoint is, amire szeretnénk a lekérdezésünket küldeni. Ha a paraméterek között szerepel modell és endpoint is, akkor a rendszer figyelmeztet, hisz így nem tudja eldönteni mit is szeretett volna a készülék. Erre megoldás a következő részben van tárgyalva.

7.2.3. Távoli hívás összevetése saját adatokkal

Ha a saját modellünket szeretnénk összevetni, egy más végponton elérhető modellel, akkor erre a SPARQL biztosít lehetőséget még hozzá a SERVICE kulcsszóval. A SERVICE-ban megadott lekérdezés, a megadott végponton fog lefutni. Ebben az esetben nem kell megadni endpoint paramétert a hívásnál, elég csak a lokális modellt. Ha nem adunk meg se modellt, se endpoint paramétert a hívásnál, akkor a lekérdezés egy üres memória modellen fog lefutni. Így ha a lekérdezésünk SERVICE-os, akkor se fog ürest visszaadni.

Példa SERVICE-es lekérdezésre:

```
SELECT ?s ?p ?o WHERE { SERVICE <http://sparql.org/books> { ?s ?p ?o } }
```

Az eredmény visszaadja a sparql.org/book végponton található könyveket.

7.3. Eredmények tárolása, visszaadása

Elérés	/query/next/{azonosító}
Hívási protokoll	GET
Paraméterek	<u>azonosító</u> – a generate-tel generált azonosítónk
Hívási példa	service.path("query").path("next/"+azon) .accept(MediaType.TEXT_PLAIN) .get(ClientResponse.class)
Válasz eredmény	Az eredmény következő 10 sorát adja vissza szövegesen.

Az egyik legfontosabb része a programnak az eredmények tárolása. Ez azért fontos, mert egy-egy lekérdezés nagy eredmény halmazt képes visszaadni. Ezt az eredményhalmazt a vevő készüléknek (telefonnak) kell kezelnie. A készülék kapacitása legtöbbször korlátos. Ezért az ötlet az, hogy az eredményt a webservice tárolja, és mindig csak egy kis része jut el a telefonnak. Ha szükséges, akkor a telefon kérheti a következő részét az eredménynek. Ennek megvalósításához el kellett készíteni egy adattárolót. A legtöbb lekérdezés Java-ban úgy működik, hogy egy ResultSet-et ad vissza, ami egy iterátorként végig megy az adathalmazon. Mivel a Jena is egy saját ResultSet-et ad vissza, így kézenfekvő volt, hogy én is ezt használjam. A probléma az, hogy ha megszűnik az adatbázis kapcsolatunk, akkor nem tudunk végig iterálni az adatokon. A ResultSet-ekben az újítás, hogy nem iterál végig az adaton, hanem tömbben tárolja az adatokat, így ezeknek lehet kérni a részhalmazait, anélkül hogy lenne adatbázis kapcsolatunk.

Mivel az alkalmazás szerver memóriája is véges, így a lekérdezések úgy lettek lefuttatva hogy LIMIT és OFFSET paraméterek kerülnek bele a lekérdezésbe.

Példa:

Az eredeti query:

```
SELECT ?s ?p ?o WHERE {  
    SERVICE <http://sparql.org/books> {?s ?p ?o}  
}
```

Az átalakított query pedig:

```
SELECT ?s ?p ?o WHERE {  
    SERVICE <http://sparql.org/books> {?s ?p ?o}  
} LIMIT 1000 OFFSET 0
```

Ha az eredmények elérik az 1000 határt, akkor fut újra a lekérdezés. Ekkor már LIMIT 1000 és OFFSET 1000 lesz a paraméter.

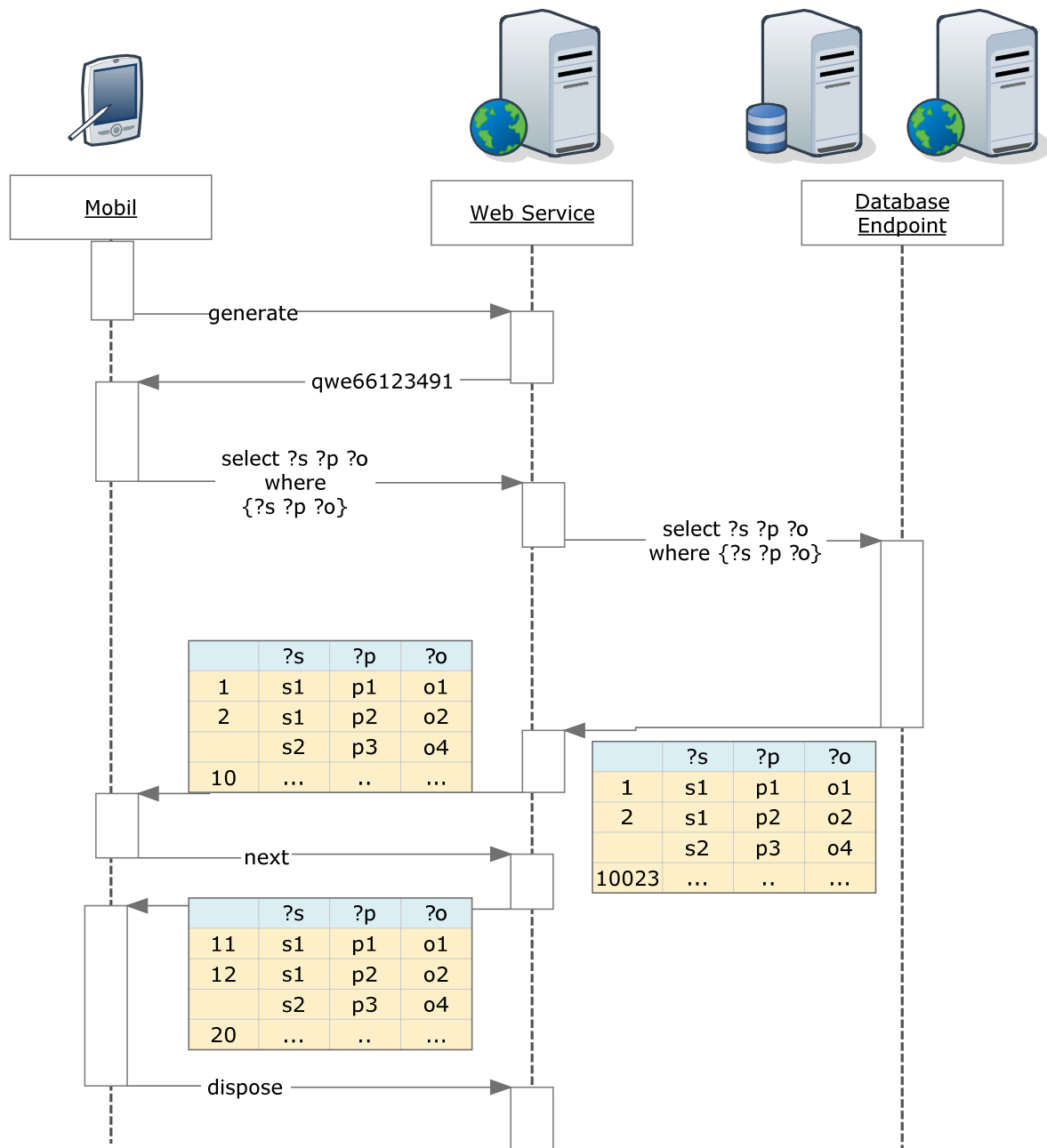
Ha a lekérdezésben szerepel LIMIT akkor az alkalmazás megnézi hogy ez kisebb-e mint a limit határunk, ha igen akkor nem változtat a lekérdezésen, hisz kevesebb eredmény lesz mint amennyit megengedünk, ha viszont nagyobb akkor lecseréli a limit határra.

Ebből a nagy adathalmazból a telefon kezdetben, csak 10 sort láthat, majd ha kéri, kapja a következő tízet. Így nem kell sok adatot megjelenítenie.

A lekérdezések eredményeit a készülékhez tartozó azonosítóval tudjuk elérni, egy ConcurrentHashMap-ben. Ez a ConcurrentHashMap tárolja a készülékek azonosítóját, eredményeiket, valamint a limit és offset paraméterek aktuális értékeit.

Bár az alkalmazás fel van készítve a kilépésre, mégis a legtöbb felhasználó, programozó (és ez által az alkalmazások is) nem lépnek ki. Így az eredmények benn maradhatnak a Hash-ben, ami foglalná a memóriát. Erre megoldásként bevezettem, egy idő értéket, amit minden azonosítóhoz tárolok. Minden hívásnál beállítódik ez az érték az aktuálisra. Azokat az erőforrásokat, amiket már nem használtak több mint fél órája, az alkalmazás felszabadítja.

7.4. Az alkalmazás szekvencia diagramja



11. ábra

8. Eredmények

Az eredmények teszteléséhez 4 lekérdezést használtam. Ezek közül 1 a saját adatbázisunkba megtalálható modellre futott, 3 pedig a dbpedia végpontján. Az első lekérdezésben található modell a W3C[10] által készített példa ontológia a Wine ontology[11]. A többi 3 pedig Benchmark lekérdezés a DBpedia-hoz[12]. Minden lekérdezés 4 attribútummal rendelkezik. A lekérdezések változói a limit és a lekérdezett attribútumok száma. A méréshez egy JAVA alkalmazást használtam [13], amit a Java Specialists folyóiratban jelent meg. Ez az alkalmazás mérni tudja a létrejött Objektumok méretét. A teszttem egy String objektumba tárolja az eredményt, és ennek a méretét láthatjuk a táblázatokban, grafikonokon.

8.1. Lekérdezés 1 (Wine Ontológia)

```
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 ?wine ?body ?sugar ?color WHERE {
    ?wine rdf:type wine:Wine .
    OPTIONAL { ?wine wine:hasBody ?body . } .
    OPTIONAL { ?wine wine:hasSugar ?sugar . } .
    ?wine wine:hasColor ?color .
    FILTER ((?color = wine:Red) || (?color = wine:White)) .
}
```

8.2. Lekérdezés 2 (2 fokú Kevin Bacon[14])

```
PREFIX p: <http://dbpedia.org/property/>
SELECT ?film1 ?actor1 ?film2 ?actor2
WHERE {
    ?film1 p:starring <http://dbpedia.org/resource/Kevin_Bacon> .
    ?film1 p:starring ?actor1 .
    ?film2 p:starring ?actor1 .
    ?film2 p:starring ?actor2 .
}
```

8.3. Lekérdezés 3 (Múzeumok, műalkotások, művészek, igazgatók)

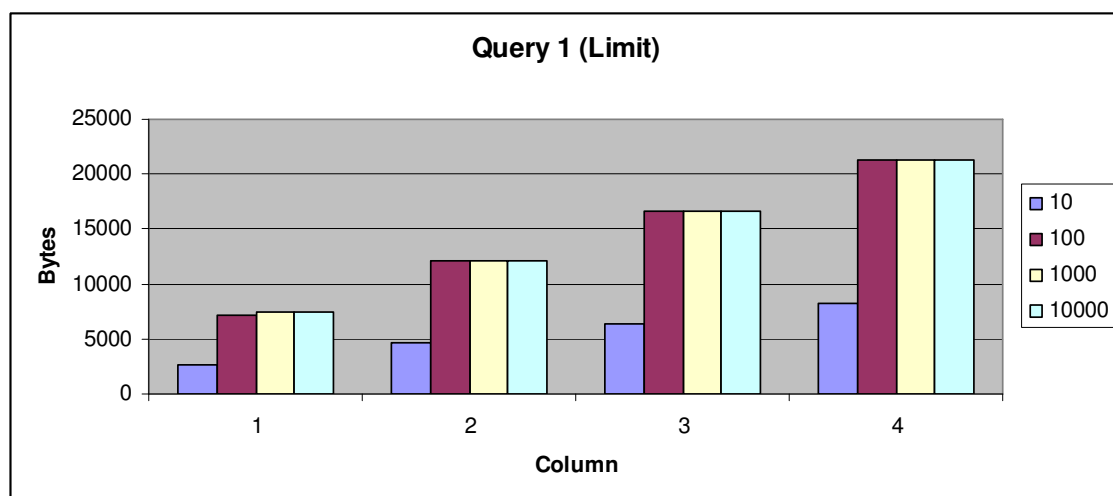
```
PREFIX p: <http://dbpedia.org/property/>
SELECT ?artist ?artwork ?museum ?director
WHERE {
    ?artwork p:artist ?artist .
    ?artwork p:museum ?museum .
    ?museum p:director ?director
}
```

8.4. Lekérdezés 4 (Honlapok nagyjából Berlin környékéről)

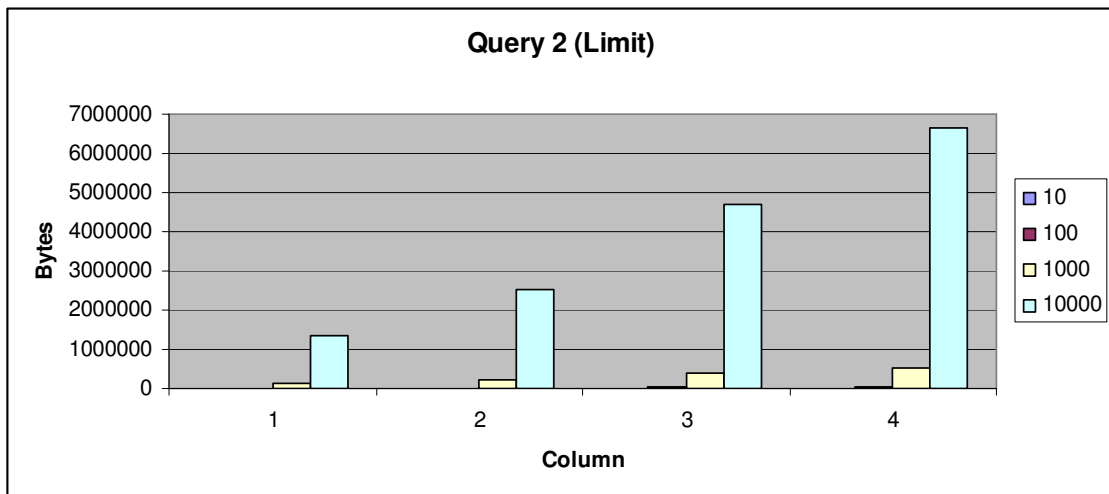
```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?homepage ?abstract ?long WHERE {
    <http://dbpedia.org/resource/Berlin> geo:lat ?berlinLat .
    <http://dbpedia.org/resource/Berlin> geo:long ?berlinLong .
    ?s <http://dbpedia.org/ontology/abstract> ?abstract .
    ?s geo:lat ?lat .
    ?s geo:long ?long .
    ?s foaf:homepage ?homepage .
    FILTER (
        ?lat <= ?berlinLat + 0.03190235436 &&
        ?long >= ?berlinLong - 0.08679199218 &&
        ?lat >= ?berlinLat - 0.03190235436 &&
        ?long <= ?berlinLong + 0.08679199218)
}
```

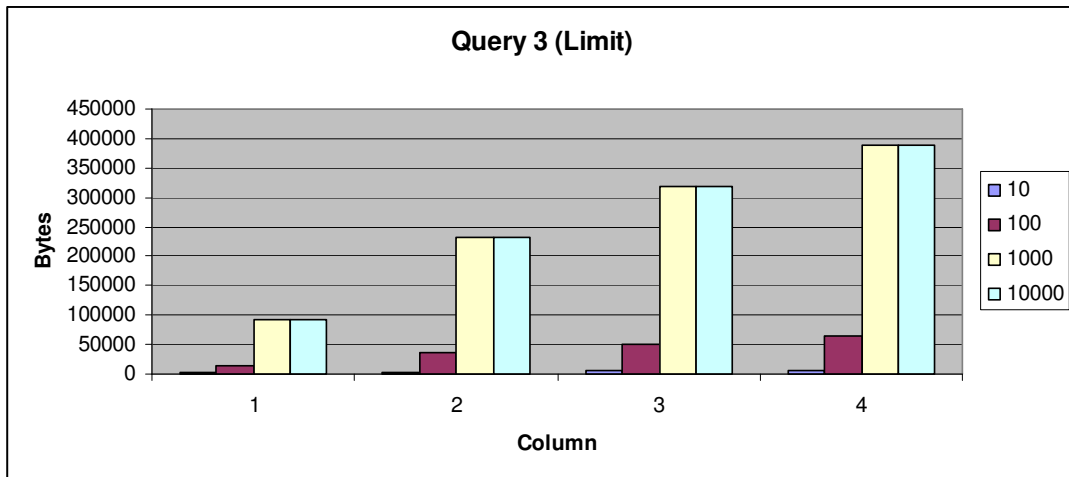
8.5. Grafikonok



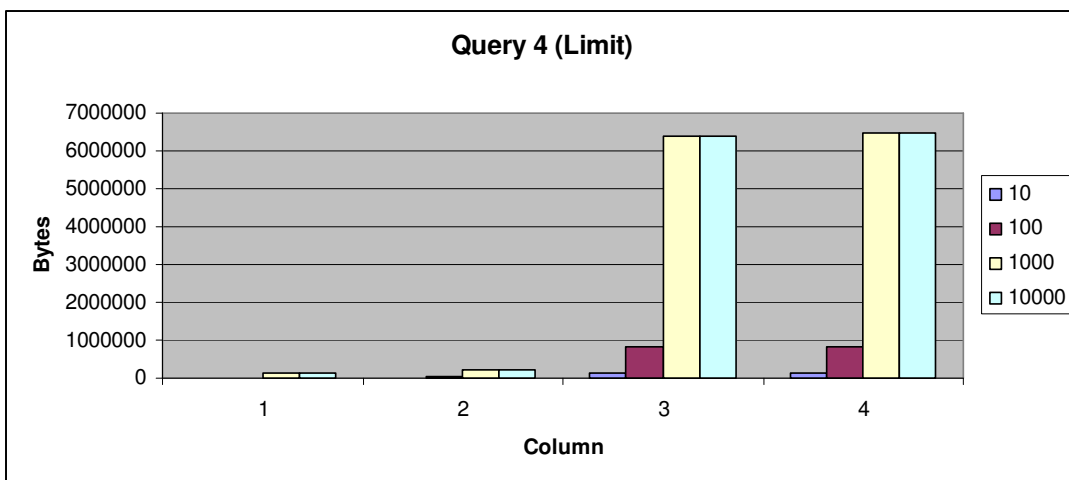
12. ábra



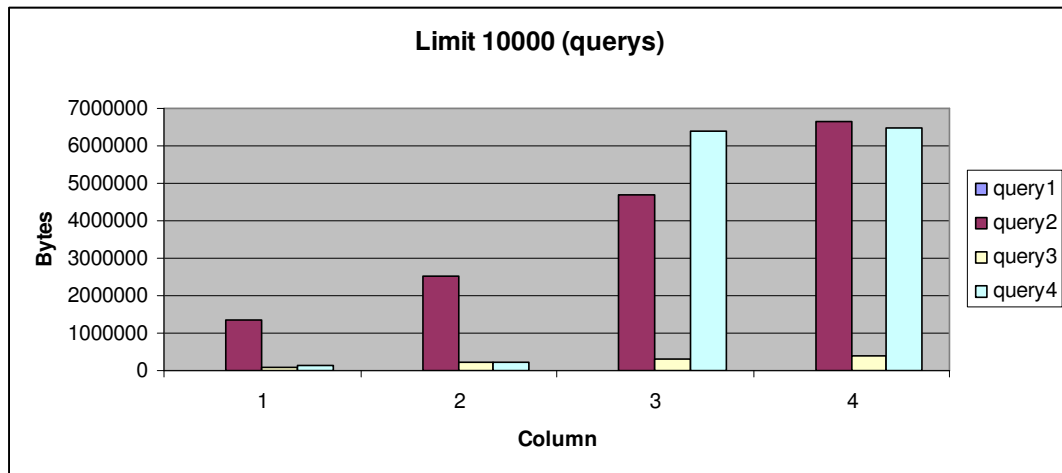
13. ábra



14. ábra



15. ábra



16. ábra

8.6. Eredmények értékelése

Az eredmények értékeléséhez szükségünk van arra, hogy miért is jó ez nekünk. Az alábbi táblázatban láthatunk pár mai Android-os [15] készüléket, és a RAM memóriájukat. A telefonok bővíthetők memóriakártyával, de azok csak háttértárként üzemelnek, és nem a futó processek számára nyújtanak helyet.

telefon	memória
move by T-mobile	150 MB
ZTE Arizona	256 MB
Samsung Galaxy Mini	150 MB
Sony Ericsson Xperia x8	128 MB

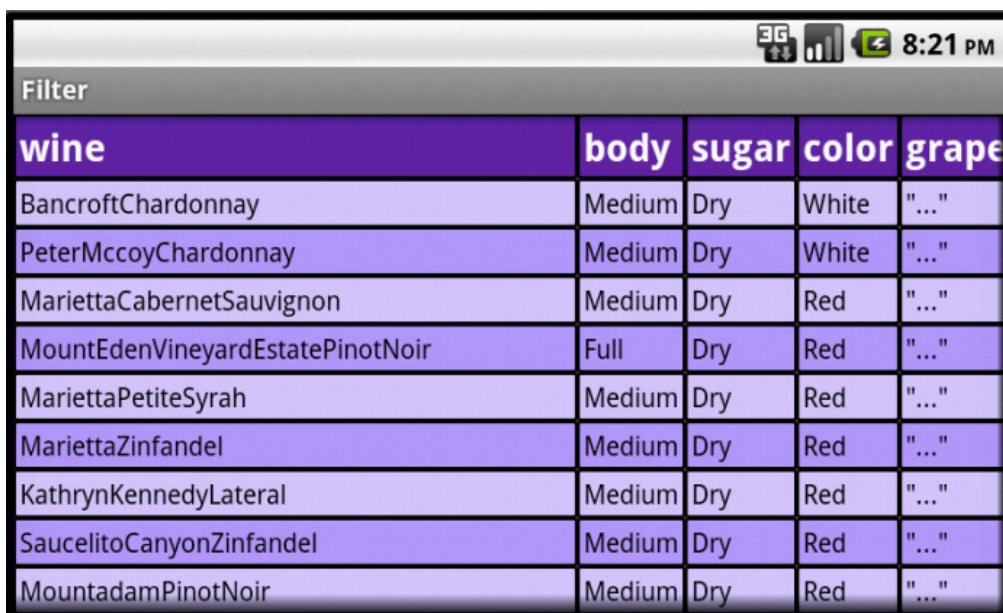
17. ábra

A 12-15. ábrákon láthatjuk, hogy az eredményül kapott string mennyi helyett foglal a kliensen. A 12. ábrán látható a wine ontológia lekérdezésre kapott eredmények 1,2,3,4 adat tagra. Látható hogy a helyfoglalás kicsit több mint 20 KB. Ez nem okozna problémát még a kis memóriával rendelkező készülékeknek sem. Viszont a 13. ábrán a Kevin Bacon-os lekérdezés már nagy mennyiségű adatokat ad vissza. Itt a 10000-es limittel már több mint 6 MB-ot kell kezelni a készüléknek. Ha vesszük a legkisebb memóriával rendelkező készüléket, akkor ez a 6MB csak az 5%-a a memóriának. Legtöbbször a készülékeken más alkalmazások is futnak, így ez az 5% is jelentős, hisz a tesztesetekben csak 4 oszlopot kérdeztünk le. A lekérdezett oszlopokban is legtöbbször URI-k szerepeltek. A 4. lekérdezésnél, ami a 15-ös ábrán szerepel, hozzávettünk egy nagyobb adattal rendelkező adattagot, az abstract-ot. Az abstractban egy leírás szerepel az adott URI-ról. Látható a diagramon, hogy ez az 1 oszlop hogy megnöveli az adatmennyiséget. Ha több ilyen adatunk van, és azokat egyszerre akarjuk lekérdezni, akkor máris több memóriája lesz szükségünk. A 15 ábra azt mutatja, hogy az adatok nem csak a sorok számától függenek, hanem az oszlopok számától is. Míg az előbbi diagramokon a limitet változtattuk, itt a maximális limittel vettjük össze a különböző lekérdezések eredményét, az oszlop számok függvényében. Itt is jól látható, ahogy növekszik a 2. lekérdezés és hogy mennyire megugrik a 4. lekérdezés.

9. Összefoglalás

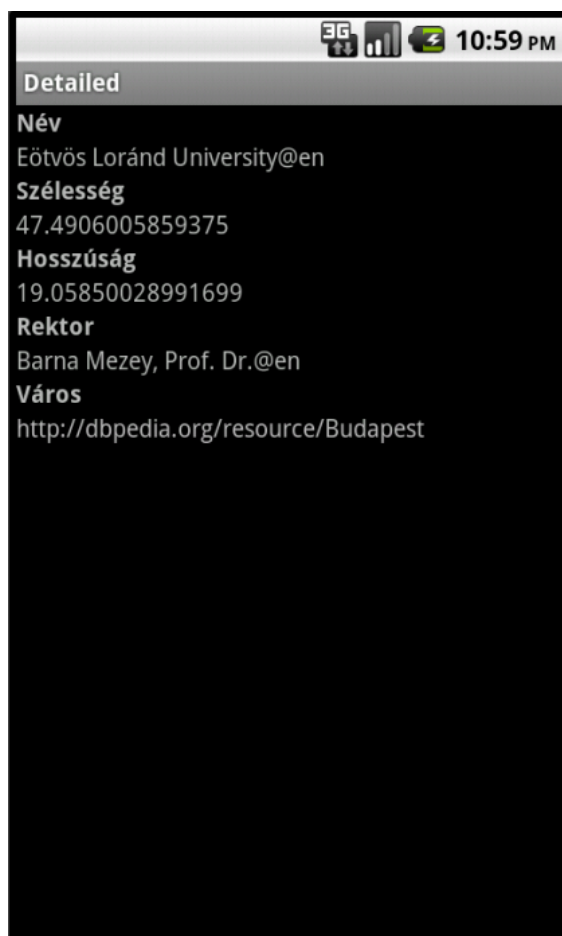
A relációs adatbázisokat egyre több helyen el lehet érni már szemantikusan is. Ezek az adatforrások nagy mennyiségű adatot tartalmaznak, melyet a klienseknek tudniuk kell kezelni. A klienseknek rendelkezniük kell Internet eléréssel, ez a mai világban már a legtöbb mobil készülékre is teljesül, így a szemantikus technológia számukra is elérhető. A probléma a szemantikus technológiával, hogy túl sok adat van és nehéz feldolgozni őket. A feldolgozás időt és erőforrást emészt fel. A kisebb készülékek, mint telefonok, vagy tábla gépek általában kevés erőforrással rendelkeznek. A lekérdezések futtatása éppen ezért elég lassú is lehet. Több korábbi munka is foglalkozott azzal, hogy a kéréseket ne a készülék, hanem egy szerver dolgozza fel. Ezek munkák több web szolgáltatást használtak. Az alkalmazásom egy web szolgáltatás, ami kifejezetten a szemantikus technológia felhasználására készült. Az alkalmazásommal elértem azt, hogy a rendszeremet használó telefonoknak nem kell rengeteg adatot tárolniuk, mindig csak épp annyit, amit meg tudnak jeleníteni. Ezeknek a lekérdezéseknek a futási ideje is kevesebb, mint ha a telefonon futtatnánk őket. A web szolgáltatásom képes más végpontokkal kommunikálni, ill. saját adatbázisokon is futni. Az adatbázisok elérését egyszerűen tudjuk konfigurálni, nincs szükség a kódban változtatásra. Az adatbázis lehet a szemantikus technológiában élenjáró Oracle, vagy más szemantikus technológiával nem rendelkező relációs adatbázis. Az adatbázisokon történő munkához a Jena keretrendszert használtam, ill. az Oracle saját változatát, a Jena Adaptert. Az alkalmazás REST protokollal kommunikál, mert ki tudja használni a http előnyeit, a SOAP rendszerekkel ellentétben. Az alkalmazás egyik kritikus pontja az azonosítás, mellyel a kliens készülékeket tudjuk azonosítani. Ezt a böngészőkben is alkalmazott session azonosítókhoz hasonlóan valósítottam meg. Egy véletlen karaktersorozat, ami azonosítóként szolgál. A kliens mindig ezt az azonosítót küldi fel, ha kérést szeretne intézni a szerver felé. A kliensről tárolt adatok egy concurrentHashMap-ben tárolom, ami száll biztosan képes kezelni az adatokat. Az alkalmazás másik kritikus része az eredmény adatok tárolása. Az eredményeket a Jena egy ResultSet-ben adja vissza, de ezeket az adatokat csak egyszer tudjuk kiolvasni, ezért a Jena Resultset-je alapján készítettem egy saját ResultSet-et, ami rendelkezik, azzal a tulajdonsággal, hogy le tudjuk kérdezni az eredmény egy adott részhalmazát. Az alkalmazást egy ELTE-n futó projekt keretein belül, már alkalmaztuk is kisebb módosításokkal. Ennek a projektnek a célja egy olyan android alkalmazás fejlesztése, mely képes szemantikus technológiák használatára. A készülék egyszerű paraméter adatokat küld fel a szervernek, mely értelmezi azt, és futtatja a megfelelő lekérdezéseket. Ezután az eredményt visszaküldi a

telefonnak, mely megjeleníti azt. Az alábbi képeken (18. és 19. ábra) egy android készülék képernyőjét láthatjuk, ahol megjelenik a wine ontológiára futtatott eredmény, ill az ELTE egy leírása.



Filter				
wine	body	sugar	color	grape
BancroftChardonnay	Medium	Dry	White	"..."
PeterMccoyChardonnay	Medium	Dry	White	"..."
MariettaCabernetSauvignon	Medium	Dry	Red	"..."
MountEdenVineyardEstatePinotNoir	Full	Dry	Red	"..."
MariettaPetiteSyrah	Medium	Dry	Red	"..."
MariettaZinfandel	Medium	Dry	Red	"..."
KathrynKennedyLateral	Medium	Dry	Red	"..."
SaucelitoCanyonZinfandel	Medium	Dry	Red	"..."
MountadamPinotNoir	Medium	Dry	Red	"..."

18. ábra



Detailed	
Név	Eötvös Loránd University@en
Szélesség	47.4906005859375
Hosszúság	19.05850028991699
Rektor	Barna Mezey, Prof. Dr.@en
Város	http://dbpedia.org/resource/Budapest

19. ábra

10. További tervek

Az alkalmazás egyszerű szöveggént adja vissza a lekérdezések eredményét. A szemantikus weben van több elfogadott formátum, melyet az alkalmazások tudnak kezelni. Ilyen formátumok az RDF, N3, Turtle. Az alkalmazásnak biztosítania kellene az ilyen formátumba való eredmény visszaadást.

A szemantikus rendszerek másik jelentős előnye, hogy adatokból, új következtetett adatokat képes előállítani. A jelenrendszerben ez nem paraméterezhető. A probléma ezzel, hogy még egy lekérdezés egyféleképpen tud lefutni, addig egy következtetéshez több féle következtetőt tudunk felhasználni. Az Oracle is támogat több következtető rendszert, de nem mindegyik van még implementálva. Az Oracle-ös megvalósításnál ezt a következtetést az adatbázis készíti el, addig a relációs adatbázisokon futó Jena csak kódból tudja ezt elvégezni. Ezenkívül találhatóak olyan külső következtetők is, amiket fel lehet használni.

Másik hasonló probléma, amit jelen alkalmazás alapján meg lehet oldani, egy dinamikus végpont. A végpontok egyik rossz tulajdonsága, hogy egy modellt lehet rajtuk csak 'kijánlani'. A kijánlást nem lehet program kódból megoldani, hanem csak konfigurációs fájlokból, amik legtöbbször nem dinamikusak. Rengeteg olyan alkalmazással találkoztuk, melyek egy paraméteren keresztül megkapott végponton található modellt tudnak felhasználni. Ha lenne ilyen dinamikus végpontok, akkor az alkalmazás is tudna dinamikusan működni.

11. Irodalomjegyzék

- [1] DBpedia: <http://dbpedia.org/About>
- [2] SPARQL endpoint: http://semanticweb.org/wiki/SPARQL_endpoint
- [3] SPARQL Query Language: <http://www.w3.org/TR/rdf-sparql-query/>
- [4] Jena: <http://jena.sourceforge.net/>
- [5] Oracle: <http://hu.wikipedia.org/wiki/Oracle>
- [6] Jena Adapter:
http://download.oracle.com/docs/cd/E18283_01/appdev.112/e11828/sem_jena.htm
- [7] PostgreSQL: <http://www.postgresql.org/>
- [8] Anupriya Ankolekar, Pascal Hitzler, Holger Lewen, Daniel Oberle and Rudi Studer:
Integrating Semantic Web Services for Mobile Access , University of Karlsruhe, Karlsruhe,
Germany
- [9] Jari Veijalainen¹, Sergiy Nikitin, Ville Törmälä: Ontology-based Semantic Web Service
platform in Mobile Environments, University of Jyväskylä, Finland - Information Technology
Research Institute
- [10] World Wide Web Consortium: <http://hu.wikipedia.org/wiki/W3C>
- [11] Wine Ontology: <http://www.w3.org/TR/owl-guide/wine.rdf>
- [12] RDF Store Benchmarks with DBpedia:
<http://www4.wiwi.fu-berlin.de/benchmarks-200801/>
- [13] Dr. Heinz M. Kabutz: Determining Memory Usage in Java:
<http://www.javaspecialists.eu/archive/Issue029.html>
- [14] Six Degrees of Kevin Bacon: http://en.wikipedia.org/wiki/Six_Degrees_of_Kevin_Bacon
- [15] Webszolgáltatások: <http://hu.wikipedia.org/wiki/Webszol%C3%A1ltat%C3%A1s>