Ατομική Διπλωματική Εργασία

# ΥΛΟΠΟΙΗΣΗ ΕΞΥΠΗΡΕΤΗΤΗ ΣΥΣΤΗΜΑΤΟΣ ΕΠΕΡΩΤΗΣΕΩΝ ΣΕ RDF ΜΕΤΑΔΕΔΟΜΕΝΑ

**Κωνσταντίνος Σοφοκλέους**

**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**Μάιος 2011**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Υλοποίηση εξυπηρετητή συστήματος επερωτήσεων σε RDF μεταδεδομένα

**Κωνσταντίνος Σοφοκλέους**

Επιβλέπων Καθηγητής

Δρ. Μάριος Δικαιάκος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2011

# Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ Μάριο Δικαιάκο για την υποστήριξη,καθοδήγηση και συμβουλές που μου προσέφερε μέχρι και την ολοκλήρωση αυτής της ΑΔΕ.

# Περίληψη

Η παρούσα ΑΔΕ ασχολείται με την υλοποίηση εξυπηρετητή επερωτήσεων σε RDF μεταδεδομένα. Ενός συστήματος το οποίο θα είναι υπεύθυνο για την ανάκτηση δεδομένων που περιγράφουν δεδομένα του Παγκόσμιου Ιστού και τον συνδυασμό τους για εξαγωγή νέων μεταδεδομένων, τα οποία θα παρουσιάζονται στον χρήστη. Τέτοια συστήματα μπορούν να βοηθήσουν σημαντικά στην έρευνα για δεδομένα στον Παγκόσμιο Ιστό.

Έχοντας ήδη ένα τέτοιο σύστημα, έπρεπε να αντικατασταθεί το κατώτερο επίπεδο του (εξυπηρετητής διαχείρισης δεδομένων) με ένα άλλο σύστημα, το οποίο να μπορεί να χρησιμοποιηθεί σε συσκευές (όπως PC) με περιορισμένους πόρους. Γι' αυτό είχε αποφασιστεί αρχικά να υλοποιηθεί στο πλαίσιο OSGI το πλαίσιο βιβλιοθηκών JENA, το οποίο είναι εξειδικευμένο για επεξεργασία τέτοιων μεταδεδομένων.

Αφού διαπιστώθηκε σε σχετικές δοκιμές ότι η πιο πάνω υλοποίηση δεν ήταν αρκετά αποδοτική, αποφασίστηκε η χρήση/ενσωμάτωση εναλλακτικού συστήματος, το οποίο μετά από έρευνα στη βιβλιογραφία (χαρακτηριστικά συστημάτων που υπάρχουν, στοιχεία δοκιμών) αποφασίστηκε ότι θα ήταν το ΣΔΒΔ OpenLink Virtuoso.

Αφού έγινε η ενσωμάτωση αυτού του συστήματος (η οποία εσυνεπάγετο δυστυχώς σημαντικές αλλαγές σε κώδικα ανώτερων επιπέδων του συστήματος, λόγω coupling της υπάρχουσας υλοποίησης με το ΣΔΒΔ Oracle) τροποποιήθηκε ο τρόπος δημιουργίας συνδυασμών επερωτήσεων. Έτσι επιτεύχθηκε καλύτερη επίδοση για τις επερωτήσεις. Έπειτα υλοποιήθηκε μηχανισμός σύνοψης των δεδομένων για βελτιστοποίηση των ενδιάμεσων επερωτήσεων.

Τέλος, έγιναν τελικές δοκιμές για σύγκριση αυτής της υλοποίησης με την υλοποίηση σε Oracle, και εξάχθηκαν κάποια συμπεράσματα και συστάσεις για μελλοντική εργασία.

# Περιεχόμενα

# Κεφάλαιο 1

## Εισαγωγή



### 1.1 Τεχνολογία RDF-Σημασιολογικό ΠΠΠ

Το Παγκόσμιο Πλέγμα Πληροφοριών (World Wide Web) αποτελεί ίσως την μεγαλύτερη προσφορά της επιστήμης της Πληροφορικής στην ανθρωπότητα. Επιτρέπει όχι απλά την πρόσβαση σε πόρους όπως κείμενο και πολυμέσα που βρίσκονται οπουδήποτε, από οπουδήποτε, αλλά και την στιγμιαία επικοινωνία μεταξύ οποιωνδήποτε ατόμων η οργανισμών στον πλανήτη. Αυτό επέφερε επαναστατικές αλλαγές σε όλους τους τομείς της ανθρώπινης ύπαρξης, από τις επιχειρήσεις μέχρι την πολιτική και τις καθημερινές σχέσεις.

Εκτός όμως από ένα μέσο επικοινωνίας και ανταλλαγής πληροφοριών, το ΠΠΠ (ειδικά μετά την έλευση του Web 2.0) αποτελεί και ένα τεράστιο σύνολο εφαρμογών, οι οποίες μπορούν να τρέξουν από τον ίδιο τον φυλλομετρητή η και μέσω επικοινωνίας με έναν εξυπηρετητή, με χρήση τεχνολογιών όπως client/server-side scripting, AJAX, XML, Java Applets.

Παρόλα αυτά, ακόμη και το Web 2.0 είναι προσανατολισμένο (όπως και το Web 1.0) προς την ανθρώπινη κατανόηση και κατανάλωση πόρων. Αυτό συνεπάγεται ότι ανώτερες λειτουργίες (όπως σύνθεση δεδομένων από διάφορες πηγές προς δημιουργία νέων πληροφοριών) δεν μπορούν να εκτελεστούν εύκολα από αυτοματοποιημένα συστήματα, αλλά απαιτούν ανθρώπινη εργασία.

Αυτό συνεπάγεται πάρα πολύ χρόνο χαμένο για έρευνα για συνδυασμούς δεδομένων στο ΠΠΠ, κάτι που αν αυτοματοποιείτο όχι απλά θα βοηθούσε σημαντικά στην έρευνα για δεδομένα στο ΠΠΠ, αλλά και θα άλλαζε τον ίδιο τον χαρακτήρα του ΠΠΠ. Το ΠΠΠ θα ήταν πλέον όχι απλά μια δεξαμένη δεδομένων, αλλά μια τεράστια Βάση Δεδομένων, η οποία θα μπορούσε με απλές επερωτήσεις να μας παρέχει όλες τις πληροφορίες που χρειαζόμαστε (“Web 3.0”) .

Το Semantic Web έχει ως στόχο την απόδοση σημασιολογίας στα δεδομένα στο ΠΠΠ η οποία να είναι κατανοητή και να μπορεί να τύχει επεξεργασίας από αυτόματα συστήματα έτσι ώστε όχι μόνο να διευκολύνεται/αυτοματοποιείται η αναζήτηση πληροφοριών στο ΠΠΠ, αλλά και να μπορούν να εξάγονται αυτόματα νέες πληροφορίες μέσω διασυνδέσεων μεταξύ των προηγούμενων. Είναι ένα «δίκτυο δεδομένων» το οποίο επιτρέπει στις μηχανές να αντιλαμβάνονται την σημασιολογία πληροφοριών στο ΠΠΠ [34]. Στόχος του είναι να καταστεί δυνατό ολόκληρο το ΠΠΠ να μπορεί να επερωτηθεί για οτιδήποτε, όπως μια τεράστια ΒΔ. Για αυτό τον σκοπό χρησιμοποιείται το μοντέλο μεταδεδομένων RDF, το οποίο έχει σχεδιαστεί για να περιγράφει πόρους/δεδομένα στο ΠΠΠ. Το μοντέλο RDF επίσης αποτελεί μια υποδομή που επιτρέπει την κωδικοποίηση, ανταλλαγή και επαναχρησιμοποίηση δομημένων μεταδεδομένων [18] .

Για την επερώτηση σε συνδυασμούς πηγών τέτοιων (μετα)δεδομένων, απαιτούνται (αν λάβουμε υπόψη τους τεράστιους όγκους δεδομένων στο ΠΠΠ) συστήματα επερωτήσεων τα οποία να είναι όχι μόνο εύχρηστα, αλλά και αποδοτικά.

### 1.2 Στόχος αυτής της ΑΔΕ

Ήδη έχουν δημιουργηθεί συστήματα διαμόρφωσης και εκτέλεσης επερωτήσεων σε RDF μεταδεδομένα, ένα από τα οποία είναι το σύστημα τριών επιπέδων του κου Κωνσταντίνου Σαββίδη, το οποίο χρησιμοποιεί το ΣΔΒΔ Oracle 11g για αποθήκευση RDF μεταδεδομένων και εκτέλεση επερωτήσεων πάνω σε αυτά.

Το σύστημα αυτό όμως είχε (όπως εξηγείται σε πάρακατω κεφάλαια) το πρόβλημα των μεγάλων απαιτήσεων σε πόρους συστήματος (κυρίως μνήμη) οι οποίες εδυσχαίρεναν την εγκατάσταση και αποδοτική χρήση του σε συστήματα με περιορισμένους πόρους. Αυτό οφείλετο στις μεγάλες απαιτήσεις του ΣΔΒΔ Oracle 11g.

Επομένως, ο πρώτος στόχος ήταν η υλοποίηση του κατώτερου επιπέδου του συστήματος (επίπεδο διαχείρισης δεδομένων) με χρήση εναλλακτικού μηχανισμού αντί ολοκληρωμένου ΣΔΒΔ και συγκεκριμένα η χρήση της υλοποίησης των βιβλιοθηκών JENA στο πλαίσιο OSGI και η δοκιμή της υλοποίησης αυτής όσο αφορά αποδοτικότητα σε σενάρια πελάτη-εξυπηρετητή.

Επόμενος στόχος (ειδικά αφού η χρήση του πιο πάνω εναλλακτικού μηχανισμού αποδείχθηκε μη ικανοποιητική) ήταν η αντικατάσταση του με ένα άλλο ΣΔΒΔ με λιγότερες απαιτήσεις από το Oracle 11g. Για αυτό τον σκοπό επιλέχθηκε το ΣΔΒΔ OpenLink Virtuoso.

Τέλος, έπρεπε το νέο αυτό ΣΔΒΔ να ενσωματωθεί στο υπάρχον σύστημα με την υλοποίηση όλων των αλλαγών που απαιτούντο και (αφού διαπιστώθηκε χαμηλή επίδοση του συστήματος αρχικά όσο αφορά συνδυασμούς επερωτήσεων σε πολλά επίπεδα-mashups) την εξεύρεση και υλοποίηση εναλλακτικής μεθόδου δημιουργίας επερωτήσεων πολλών επιπέδων, η οποία να προσφέρει καλύτερη επίδοση. Επίσης δοκιμάστηκε η χρήση summaries για σύνοψη των δεδομένων ώστε να διευκολυνθούν κάποιες κοινές ενδιάμεσες επερωτήσεις, οι οποίες χρησιμοποιούνται (λόγω της μη ύπαρξης σχήματος για τα RDF μεταδεδομένα) για την διαμόρφωση των τελικών επερωτήσεων) .

Ακολούθησε τελική δοκιμή του συστήματος και εξαγωγή κάποιων συμπερασμάτων και εισηγήσεων για μελλοντική εργασία πάνω στο θέμα.

# Κεφάλαιο 2

## Ανασκόπηση τεχνολογιών



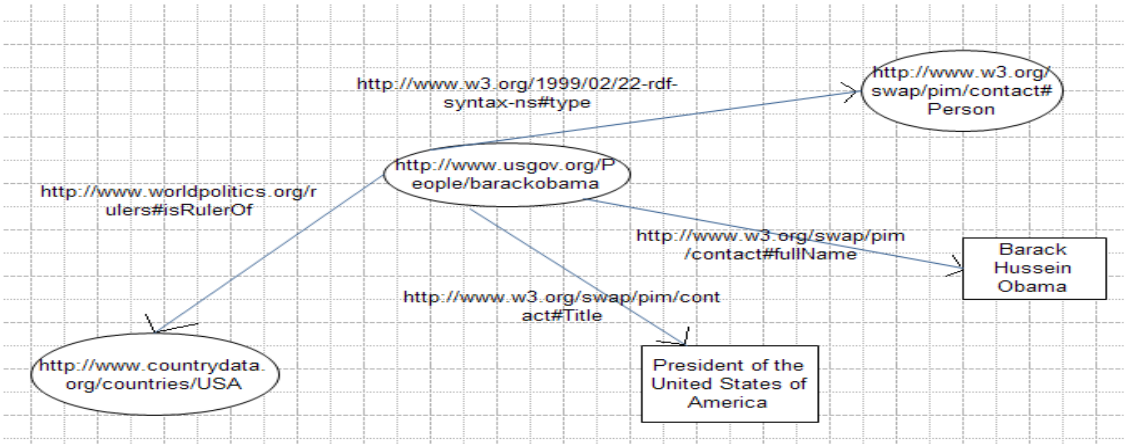
### 2.1 RDF(Resouce Description Framework)

Πριν από τον ερχομό του Web 2.0 η μόνη λειτουργία που θα μπορούσε να επιτελέσει κάποιος χρήστης η σύστημα στο Παγκόσμιο Πλέγμα Πληροφοριών ήταν η ανάκτηση και καταφόρτωση στατικών πληροφοριών. Με την δημιουργία του Web 2.0 το ΠΠΠ προσφέρει πλέον ένα ευρύ φάσμα δυνατοτήτων, όπως κοινωνική δικτύωση, πρόσβαση σε πολυμέσα, εκτέλεση εφαρμογών μέσω του φυλλομετρητή. Παρόλα αυτά, ακόμη και το Web 2.0 είναι προσανατολισμένο (όπως και το Web 1.0) προς την ανθρώπινη κατανόηση και κατανάλωση πόρων, κάτι που συνεπάγεται ότι ανώτερες λειτουργίες (όπως σύνθεση δεδομένων από διάφορες πηγές προς δημιουργία νέων πληροφοριών)

δεν μπορούν να εκτελεστούν εύκολα από αυτοματοποιημένα συστήματα αλλά απαιτούν ανθρώπινη εργασία.

Το Semantic Web έχει ως στόχο την απόδοση σημασιολογίας στα δεδομένα στο ΠΠΠ η οποία να είναι κατανοητή και να μπορεί να τύχει επεξεργασίας από αυτόματα συστήματα ούτως ώστε όχι μόνο να διευκολύνεται/αυτοματοποιείται η αναζήτηση πληροφοριών στο ΠΠΠ αλλά και να μπορούν να εξάγονται αυτόματα νέες πληροφορίες μέσω διασυνδέσεων μεταξύ των προηγούμενων. Είναι δηλαδή ένα «δίκτυο δεδομένων» το οποίο επιτρέπει στις μηχανές να αντιλαμβάνονται την σημασιολογία πληροφοριών στο ΠΠΠ [34]. Στόχος του είναι να καταστεί δυνατό ολόκληρο το ΠΠΠ να μπορεί να επερωτηθεί για οτιδήποτε, όπως μια τεράστια ΒΔ. Για αυτό τον σκοπό χρησιμοποιείται το μοντέλο μεταδεδομένων RDF, το οποίο έχει σχεδιαστεί για να περιγράφει πόρους/δεδομένα παρόντα στο ΠΠΠ. Το μοντέλο RDF επίσης αποτελεί μια υποδομή που επιτρέπει την κωδικοποίηση, ανταλλαγή και επαναχρησιμοποίηση δομημένων μεταδεδομένων [18].

Το μοντέλο RDF βασίζεται στην περιγραφή συσχετίσεων μεταξύ δεδομένων στο ΠΠΠ μέσω τριάδων, οι οποίες αποτελούνται από τρία μέρη: 1) Υποκείμενο (ο πόρος ο οποίος περιγράφεται από την τριάδα, αντιπροσωπεύεται από ένα Μοναδικό Περιγραφέα Πόρου). 2) Κατηγόρημα (η ιδιότητα του πόρου με την περιγραφή της οποίας ασχολείται η συγκεκριμένη τριάδα). 3) Αντικείμενο (η τιμή της περιγραφής, η οποία μπορεί να είναι είτε ένας πόρος αντιπροσωπευόμενος από ένα URI οπότε έχουμε μια συσχέτιση μεταξύ των δύο αυτών πόρων, είτε μια σταθερή τιμή-literal). Έτσι μπορούν όχι μόνο να περιγραφούν ιδιότητες δεδομένων αλλά και συσχετίσεις μεταξύ τους, οι οποίες είναι αυτές που δικαιολογούν την έννοια του σημασιολογικού δικτύου. Οι πόροι οι οποίοι περιγράφονται μπορούν να βρίσκονται οπουδήποτε στο ΠΠΠ.




Σχήμα 2.1 Παράδειγμα τμήματος ενός γράφου RDF. Τα οβάλ αντιπροσωπεύουν πόρους στο ΠΠΠ, τα ορθογώνια σταθερές τιμές, οι ακμές με τα βέλη συσχετίσεις μέσω κατηγορημάτων.

Το μοντέλο περιγραφής μεταδεδομένων RDF μπορεί, όπως ανέφερα πιο πάνω, να χρησιμοποιηθεί και για εξαγωγή νέας πληροφορίας από προηγούμενες. Αυτό γίνεται με την χρήση οντολογιών, μέσω προτύπων όπως OWL (Web Ontology Language), τα οποία αποτελούν σημασιολογική επέκταση του μοντέλου RDF ώστε να μπορούν να εκφραστούν βαθύτερες σημασιολογικές σχέσεις μεταξύ δεδομένων (όπως είναι, ανήκει, αποτελεί υποιδιότητα του). Έτσι δημιουργείται και ένα πιο αντικειμενοστρεφές μοντέλο, που επιτρέπει την εξαγωγή σημασιολογικών σχέσεων μεταξύ δεδομένων με χρήση λογικών συλλογισμών (reasoning) [10].

Μεταδεδομένα RDF μπορούν να εκφραστούν σε δύο μορφές:

1)XML (οπότε μιλάμε για RDF/XML)

```
<Person
rdf:about="http://data.semanticweb.org/person/eugenio-di-
sciascio">
        <swc:affiliation>Politecnico of
Bari</swc:affiliation>
        <rdfs:label>Eugenio Di Sciascio</rdfs:label>
        <family_name>Di Sciascio</family_name>
        <firstName>Eugenio</firstName>
        <homepage
rdf:resource="http://sisinflab.poliba.it/disciascio/"/>

<mbox_sha1sum>3ebb348a486c02477801898c1b4a8da3cd8d8e21</mb
ox_sha1sum>
        <name>Eugenio Di Sciascio</name>
```

```
    </Person>
```

2)Notation 3 (στην οποία βασίζονται οι μορφές Turtle και N-triples)

```
<http://dbpedia.org/resource/Kame%C5%88any>
<http://xmlns.com/foaf/0.1/name> "K\u00F6vi"@hu .

<http://dbpedia.org/resource/Voj%C5%88any>
<http://dbpedia.org/ontology/status> "k\u00F6zs\u00E9g"@hu
.

<http://dbpedia.org/resource/Voj%C5%88any>
<http://dbpedia.org/ontology/postalCode> "059 02"@hu .

<http://dbpedia.org/resource/Voj%C5%88any>
<http://dbpedia.org/ontology/areaCode> "052"@hu .
```

**2.2 Γλώσσα επερωτήσεων SPARQL**

Από την στιγμή που υπάρχουν στο ΠΠΠ μεταδεδομένα σε μορφή RDF φυσικό επακόλουθο ήταν να δημιουργηθούν και τεχνικές για επεξεργασία αυτών των δεδομένων. Ανάμεσα σε αυτές περιλαμβάνεται και η γλώσσα επερωτήσεων SPARQL (Simple Protocol and RDF Query Language), η οποία σχεδιάστηκε για επερωτήσεις σε RDF μεταδεδομένα, όπως σχεδιάστηκαν γλώσσες δομημένων επερωτήσεων όπως την SQL για επερωτήσεις πάνω σε ΒΔ.

Η SPARQL επιτρέπει στους χρήστες να διαμορφώσουν επερωτήσεις, βασισμένοι σε τιμές οι οποίες δεν παρουσιάζουν καμία ασάφεια καθώς όλες οι σταθερές τιμές ( κατηγορήματα) αποτελούν πόρους του ΠΠΠ και επομένως έχουν παντού την ίδια μορφή. Αυτό διευκολύνει στην συνένωση και την συγχώνευση πόρων ολόκληρου του ΠΠΠ, σύμφωνα με την φιλοσοφία της προσπάθειας δημιουργίας του «παγκόσμιου σημασιολογικού δικτύου».

Η SPARQL επιτρέπει τεσσάρων ειδών επερωτήσεις

1)SELECT

Για επιλογή τιμών επιλεγμένων πεδίων από συνδυασμούς συνενώσεων συνόλων δεδομένων RDF βάσει επιλεγμένων περιορισμών

Για παράδειγμα

```
PREFIX examplepref:<http://example.com/exampleOntology#>

SELECT ?city ?street

WHERE{

?X1 examplepref:streetName ?street.

?X1 examplepref:isLocatedInCity ?X2.

?X2 examplepref:cityName ?city.

?X2 examplepref:inCountry examplepref:Cyprus.}
```

Στην πιο πάνω επερώτηση, ζητάμε να μας επιστραφούν τα ονόματα των δρόμων ($1^{η}$ τριάδα επερώτησης) τα οποία βρίσκονται σε πόλεις ($2^{η}$ τριάδα επερώτησης) των οποίων πόλεων τα ονόματα θέλουμε να επιστραφούν ($3^{η}$ τριάδα επερώτησης) και οι οποίες πόλεις εβρίσκονται στην χώρα Κύπρο ($4^{η}$ τριάδα επερώτησης). Το SELECT συμβολίζει τις επιλογές επιστρεφόμενων πεδίων, όπως ακριβώς στην SQL, ενώ τα PREFIX αποτελούν συντομογραφίες πεδίων διευθύνσεων URI, αντιπροσωπεύοντας έναν χώρο ονομάτων (namespace) στο ΠΠΠ. Τα ερωτηματικά συμβολίζουν μεταβλητές, οι οποίες παίρνουν τιμές οι οποίες ταυτίζονται με τις τιμές που εμπεριέχονται στα σύνολα δεδομένων. Οι τελευταίες συσχετίζονται μεταξύ τους μέσω των επιλεγμένων σχέσεων και περιορισμών.

2,3,4) Τα άλλα τρία είδη επερωτήσεων (με τα οποία δεν ασχολείται αυτή η ΑΔΕ αφού δεν χρησιμοποιούνται στα πλαίσια της) είναι: 1)Μέσω CONSTRUCT (λειτουργεί με τον ίδιο τρόπο με την SELECT αλλά επιστρέφει τα αποτελέσματα υπό μορφή RDF). 2) ASK (επιστρέφει αν υπάρχουν ταυτίσεις στα RDF μεταδεδομένα που να ικανοποιούν την επερώτηση). 3) DESCRIBE (επιστρέφει δεδομένα βάσει των περιεχόμενων δεδομένων στον γράφο και επιλεγμένων ρυθμίσεων του συντηρητή της διεπαφής επερωτήσεων RDF).

Το κυρίως πρόβλημα που υπάρχει σχετικά με την δημιουργία επερωτήσεων σε RDF μεταδεδομένα προκύπτει από την ίδια την φύση των μεταδεδομένων RDF, τα οποία αντίθετα με δεδομένα σε ΒΔ, δεν έχουν καθορισμένο σχήμα (που να περιγράφει τις μεταξύ τους σχέσεις) και οι μεταξύ τους σχέσεις πολύ πιθανό να μεταβάλλονται

δυναμικά συνεχώς, η ακόμη και αν υπάρχει σταθερό σχήμα πολύ πιθανό να μην είναι διαθέσιμο. Αυτό συνεπάγεται ότι η γνώση για την δομή των μεταδεδομένων, η οποία είναι απαραίτητη για την διαμόρφωση επερωτήσεων επάνω σε αυτά, πρέπει να ανακτάται δυναμικά, με χρήση ενδιάμεσων επερωτήσεων. Οι οποίες μάλιστα, αφού αποτελούν απλά μέσο για δημιουργία των τελικών επερωτήσεων και πολύ πιθανό να χρειαστούν πολλές ενδιάμεσες επερωτήσεις, οι προηγούμενες για την δημιουργία των επόμενων, πρέπει να εκτελούνται σε ελάχιστο χρόνο, κάτι που για μεγάλα σύνολα δεδομένων είναι ιδιαίτερα προβληματικό.

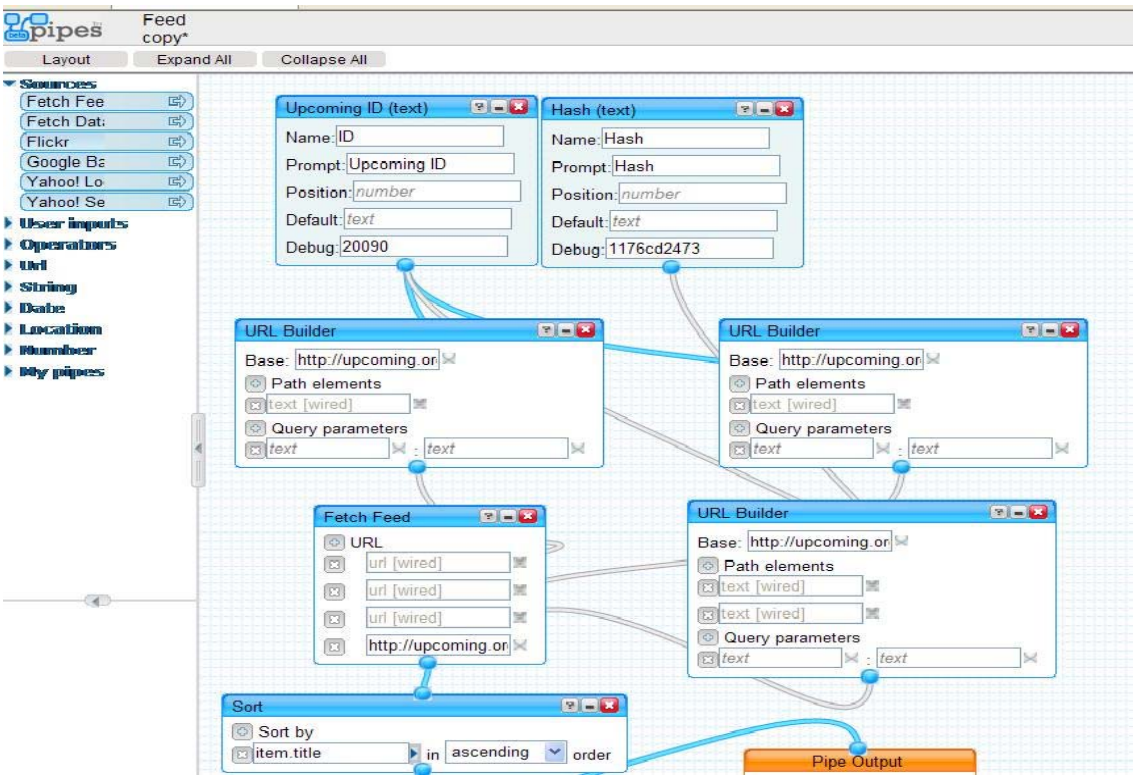
## 2.3 Web Mashups

Ένα mashup είναι ένα σύστημα/ιστοσελίδα/εφαρμογή η οποία συνδυάζει δεδομένα από ένα πλήθος περισσότερων της μίας πηγών ούτως ώστε να παράξει νέα δεδομένα μέσω αυτού του συνδυασμού. Τα νέα παραγόμενα δεδομένα πιθανόν να μην παράγονται κατ'ανάγκη λαμβάνοντας υπόψη τους ίδιους λόγους για τους οποίους χρειάστηκε να παραχθούν τα πηγαία δεδομένα.

Με τον ερχομό και εδραίωση του Web 2.0 και την έμφαση που δίδεται στα παραγόμενα από τους ίδιους τους χρήστες δεδομένα (δηλ ο χρήστης να συνεισφέρει πιο ενεργά και να μην είναι απλά παθητικός καταναλωτής) τα mashups συνεισφέρουν προσφέροντας την δυνατότητα στον απλό χρήστη να παράγει, βάσει δικών του επιλογών, νέα δεδομένα ανάλογα με τις ανάγκες του, χρησιμοποιώντας υπάρχοντα πηγαία δεδομένα.

Για παράδειγμα, ένας χρήστης μπορεί να συνδυάσει δεδομένα από την Google ( Google Maps), την Wikipedia (περιγραφές τοποθεσιών) και ένα ιστότοπο μιας αερογραμμής (τιμές αεροπορικών εισητηρίων) ούτως ώστε να βοηθηθεί στο να επιλέξει προορισμό για τις διακοπές του.

Ήδη υπάρχουν πολλά εργαλεία που το επιτρέπουν αυτό (Mashup Editors) των οποίων ηγείται (κυρίως λόγω της ευκολίας χρήσης του από μη ειδικούς) ο editor Yahoo!Pipes της Yahoo! (πάνω στον οποίο βασίζεται και ο editor του συστήματος που μελετάται σε αυτή την ΑΔΕ).




Σχήμα 2.2 Χρήση Web Mashups στο εργαλείο Yahoo!Pipes

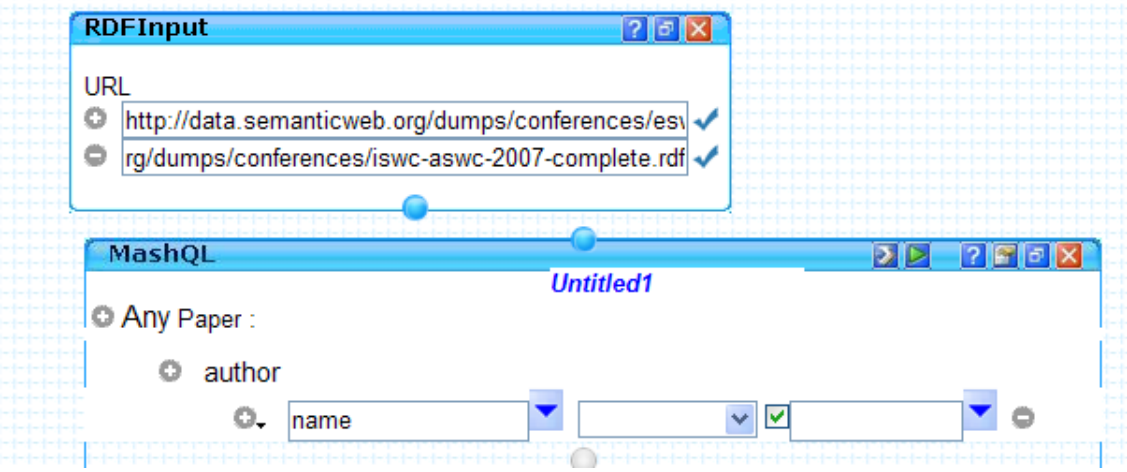
Ειδικά σε ότι αφορά μεταδεδομένα RDF, από την στιγμή που ένας από τους λόγους που δημιουργήθηκε και αναπτύσσεται συνεχώς το σημασιολογικό δίκτυο είναι για να είναι δυνατός ο συνδυασμός δεδομένων από διαφορετικές πηγές ούτως ώστε να διευκολύνεται η έρευνα για δεδομένα και η δημιουργία νέων δεδομένων από προηγούμενα, οι τεχνολογίες mashup έχουν να παίξουν εδώ ένα σημαντικό ρόλο.

## 2.4 MashQL

Η γλώσσα δημιουργίας mashups δεδομένων ΠΠΠ MashQL αναπτύχθηκε από τους Δρ Μάριο Δικαιάκο και Δρ Μουσταφά Τζαράρ προς επίτευξη του πιο πάνω στόχου. Ο οποίος στόχος είναι η δημιουργία τεχνικών συνδυασμού μεταδεδομένων RDF από πολλαπλές πηγές (σύνολα δεδομένων) στο ΠΠΠ με τον ίδιο τρόπο που η SQL μπορεί να συνδυάσει δεδομένα από πολλούς πίνακες μιας ΒΔ [12], με τρόπο που να είναι εύκολος για τον απλό χρήστη και να μην απαιτεί ιδιαίτερες προγραμματιστικές ικανότητες.

Η γλώσσα αυτή βασίζεται στην έννοια του φίλτρου, δηλαδή της διαδοχικής εφαρμογής περιορισμών πάνω σε σύνολα δεδομένων, μια έννοια η οποία μπορεί να γίνει εύκολα

κατανοητή ακόμη και από μη ειδικούς. Αυτά τα φίλτρα εκφράζονται ως περιορισμοί πάνω στο υποκείμενο (ρίζα) της επερώτησης, το οποίο αποτελεί και το σημείο αναφοράς της επερώτησης. Έτσι σχηματίζεται ένα δέντρο για κάθε επερώτηση, τα κλαδιά του οποίου αποτελούν περιορισμούς πάνω στις δυνατές τιμές των πεδίων που επιστρέφονται.




Σχήμα 2.3 Δημιουργία δέντρου επερώτησης MashQL με χρήση της υλοποίησης του συστήματος αυτής της ΑΔΕ

Για παράδειγμα το δέντρο επερώτησης (σε SPARQL) για το πιο πάνω σχήμα είναι

SELECT ?O2

FROM <http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf>
FROM <http://data.semanticweb.org/dumps/conferences/iswc-aswc-2007-complete.rdf>

WHERE {?S :Type :Paper.
?S :author ?O1.
?O1 :name ?O2.}

Ο σχηματισμός του δέντρου αυτού ξεκινά με την επιλογή του υποκειμένου (ρίζα) από τον χρήστη, το οποίο μπορεί να είναι: 1) Ένας τύπος πόρου(οπότε επιλέγονται όλοι οι πόροι-υποκείμενα που είναι δηλωμένοι ότι ανήκουν σε αυτό τον τύπο). 2) Ένας συγκεκριμένος πόρος (υποκείμενο η αντικείμενο). 3) Μια δηλωμένη από τον χρήστη μεταβλητή. Έπειτα ο χρήστης μπορεί να πλοηγηθεί μέσω εκτέλεσης ενδιάμεσων επερωτήσεων στα σύνολα δεδομένων, δημιουργώντας στην πορεία τα κλαδιά του

δέντρου επερώτησης. Αυτό γίνεται με διαδοχική επανάληψη των δύο πιο κάτω βημάτων:

1)Επιλογή ενός κατηγορήματος (βάσει του τρέχοντος κλαδιού επερώτησης μέχρι τώρα) ή δηλωμένης από τον χρήστη μεταβλητής.

2)Προσθήκη ενός φίλτρου πάνω στις επιτρεπόμενες τιμές του επιλεγμένου κστηγορήματος. Εδώ υπάρχουν οι ακόλουθες περιπτώσεις:

i. Επιλογή φίλτρου πάνω στην ιδιότητα που εκφράζει το προηγούμενο κατηγόρημα (πχ αν έχουμε το κατηγόρημα χρόνος δημοσίευσης, το φίλτρο «μεγαλύτερο του 2007») όπως μεγαλύτερο, ίσο, μικρότερο, μικρότερο η ίσο, άνισο, ένα από κοκ.

ii. Επιλογή συγκεκριμένου αντικειμένου (βάσει του τρέχοντος κλαδιού επερώτησης μέχρι τώρα).

iii. Επιλογή δηλωμένης από τον χρήστη μεταβλητής.

Μια πιο λεπτομερής ανάλυση του τρόπου σχηματισμού ενός κλαδιού επερώτησης, καθώς και ο τυπικός ορισμός της MashQL, μπορούν να βρεθούν στο [13].

Επειδή μια επερώτηση MashQL δεν μπορεί να εκτελεστεί απευθείας ως έχει, είναι απαραίτητο πρώτα να μεταφραστεί στην γλώσσα επερωτήσεων SPARQL, βάσει κάποιων μαθηματικών κανόνων μετατροπής, οι οποίοι περιγράφονται στο [13].

### 2.5 OpenLink Virtuoso Universal Server

Όπως αναφέρει και η ονομασία του, το Virtuoso δεν αποτελεί απλά ένα ΣΔΒΔ αλλά μπορεί επίσης να χρησιμοποιηθεί ως σύστημα αποθήκευσης και επεξεργασίας μεταδεδομένων RDF και XML, εξυπηρετητής εφαρμογών ΠΠΠ και εξυπηρετητής αρχείων. Το OpenLink Virtuoso είναι η έκδοση του σε μορφή ανοικτού κώδικα.

Εσωτερικά αποτελείται από μια πολυνηματική διεργασία εξυπηρετητή η οποία υλοποιεί πολλαπλά πρωτόκολλα, οδηγώντας έτσι στον “Universal” χαρακτήρα του. Βασίζεται στο αντικειμενο-σχεσιακό μοντέλο. Συμπεριλαμβάνει δυνατότητα αποθήκευσης δεδομένων τόσο στην κυρίως μνήμη όσο και σε αρχεία.

Η προγραμματιστική επικοινωνία με τον εξυπηρετητή Virtuoso μέσω SQL μπορεί να γίνει μέσω πρόσβασης στο ISQL (Interactive SQL) Endpoint του εξυπηρετητή, μέσω

σύνδεσης με χρήση APIs όπως JDBC, ODBC, ADO.NET και OLE DB. Μπορεί να χρησιμοποιηθεί επίσης το ISQL Endpoint μέσω πρόσβασης σε αυτό από την γραμμή εντολών, καθώς και με χρήση του πρωτοκόλλου HTTP (άρα με χρήση οποιουδήποτε φυλλομετρητή) και πρόσβαση στην διεπαφή Virtuoso Conductor, μέσω της οποίας μπορούν να γίνουν και συνήθεις ρυθμίσεις στο ΣΔΒΔ εκτός από την εκτέλεση επερωτήσεων SQL και λειτουργιών στις ΒΔ.

Χαμηλότερου επιπέδου ρυθμίσεις (χρήσιμες για αύξηση της επίδοσης για μεγάλα σύνολα δεδομένων) στο ΣΔΒΔ μπορούν να γίνουν μέσω της τροποποίησης του αρχείου virtuoso.ini.

Το Virtuoso προσφέρει ενσωματωμένη υποστήριξη για αποθήκευση RDF μεταδεδομένων μέσω της χρήσης γράφων. Ο καθένας τους αντιπροσωπεύει ένα σύνολο δεδομένων RDF, το οποίο μπορεί και να αποτελείται από μεταδεδομένα από διαφορετικά σύνολα δεδομένων [20]. Χρησιμοποιείται ο ειδικός τύπος δεδομένων IRI_ID για αντιπροσώπευση πόρων RDF (εσωτερικά αποθηκευμένος ως ακέραιος 32 δυαδικών ψηφίων η 64 δυαδικών ψηφίων αναλόγως ρυθμίσεων). Προσφέρει ενσωματωμένη υποστήριξη για την γλώσσα επερωτήσεων SPARQL, με τις ακόλουθες διαφορές:

1) Χαρακτήρες Unicode εντός των ονομάτων δεν υποστηρίζονται, ούτε σχόλια εντός κώδικα SPARQL ο οποίος είναι ενσωματωμένος σε SQL.

2) Αντίθετα, υποστηρίζονται κάποιες επεκτάσεις της SPARQL όπως χρήση κανονικών εκφράσεων εντός κώδικα SPARQL, εισαγωγή εξωτερικών παραμέτρων εντός αυτού, υποεπερωτήσεις και συναθροιστικές συναρτήσεις (aggregate functions) ,χαρτογράφηση σχεσιακών δεδομένων σε RDF (με χρήση του εργαλείου Sponger).

Προσφέρεται εξειδικευμένο endpoint για απευθείας εκτέλεση επερωτήσεων SPARQL μέσω φυλλομετρητή και του πρωτοκόλλου HTTP.

Επερωτήσεις SPARQL μπορούν να προωθηθούν προς μια ΒΔ με χρήση κώδικα SPARQL ενσωματωμένου σε SQL και να χρησιμοποιηθούν αποτελέσματα επερωτήσεων SQL και ως πίνακες εισόδου για επερωτήσεις SQL:

```
SELECT TOP 0, 50  P1
```

```
FROM ( SPARQL SELECT ?P1 FROM
<http://data.semanticweb.org/dumps/conferences/eswc-2007-
complete.rdf> FROM
<http://data.semanticweb.org/dumps/conferences/iswc-aswc-
2007-complete.rdf>

WHERE { ?S rdf:type  <http://xmlns.com/foaf/0.1/Person> .
?S ?P1 ?O1  . } ) AS allofthem where P1  LIKE '**'
```

Εδώ εκείνο που θα γίνει είναι ότι πρώτα θα εκτελεστούν τυχόν επερωτήσεις σε SPARQL επιστρέφοντας τα αποτελέσματα στη μορφή πινάκων (όπως με μια επερώτηση SQL). Οι πίνακες μπορούν έπειτα να χρησιμοποιηθούν ακριβώς όπως πίνακες σε μια ΒΔ, εισαγόμενοι στην εξωτερική επερώτηση SQL στα πλαίσια ενός FROM για διαμόρφωση επερωτήσεων SQL.

Η εισαγωγή δεδομένων RDF στο ΣΔΒΔ (συγκεκριμένα σε ένα γράφο) μπορεί να γίνει με τους εξής τρόπους:

1)Χρήση αποθηκευμένων διαδικασιών εντός της ΒΔ (TTLP για N-Triples και N3,RDF_LOAD_RDFXML για RDF/XML και άλλες πιο εξειδικευμένες) στις οποίες εισάγονται: **1.** Τα δεδομένα RDF προς αποθήκευση (για εισαγωγή από URL συνόλου δεδομένων η αρχείο χρησιμοποιούνται συναρτήσεις πρόσβασης όπως file_to_string_output και http_get). **2.** Ο γράφος RDF στον οποίο θα προστεθούν τα εισαγόμενα μεταδεδομένα. **3.** Το URI βάσης το οποίο θα χρησιμοποιηθεί για επίλυση σχετικών URI τα οποία πιθανό να εμφανίζονται εντός των δεδομένων.

2)Χρησιμοποιώντας μηνύματα HTTP, είτε POST και GET (χρησιμοποιώντας το REST API [21] ) είτε PUT.

3)Εντολή SPARQL INSERT.

4) Χρησιμοποιώντας τα εργαλεία WebDav η Virtuoso Crawler (το δεύτερο αποτελεί ένα εργαλείο-αράχνη το οποίο, ξεκινώντας από δοσμένα URL, προσθέτει συνδεδεμένους πόρους αναδρομικά έχοντας και την δυνατότητα μετατροπής δεδομένων από μη-RDF μορφή σε RDF).

5)Χρησαιμοποιώντας το εργαλείο SPONGER.

6)Χρησιμοποιώντας διάφορες άλλες συναρτήσεις διεπαφής.

7)Χρησιμοποιώντας τον bulk loader, ο οποίος επιτρέπει φόρτωση δεδομένων από πολλαπλά σύνολα δεδομένων η/και αρχεία σε ένα γράφο ταυτόχρονα και εκτέλεση πολλών ταυτόχρονων εντολών εισαγόμενων από πριν. Αυτή η μέθοδος εισαγωγής είναι και αυτή που χρησιμοποιείται στα πλαίσια της υλοποίησης του συστήματος σε αυτή την ΑΔΕ και περιγράφεται εκτενέστερα στο κεφάλαιο 6.

### 2.6 JENA Semantic Web Framework

Πρόκειται για ένα πλαίσιο εφαρμογών σε Java σχεδιασμένο για δημιουργία εφαρμογών βασισμένων στο Σημασιολογικό Δίκτυο. Παρέχει ένα προγραμματιστικό περιβάλλον για επεξεργασία μεταδεδομένων RDF, RDFS, OWL, SPARQL και μια μηχανή εξαγωγής λογικών συμπερασμάτων βασισμένη σε κανόνες [14].

Χρησιμοποιεί την έννοια του Μοντέλου, το οποίο είναι μια διεπαφή η οποία ορίζει τον τρόπο αποθήκευσης γράφων RDF και τις μεθόδους οι οποίες μπορούν να καλεστούν επάνω σε αυτό. Μεθόδους για λειτουργίες όπως προσθήκη τριάδων, πόρων και ιδιοτήτων σε πόρους, ανάκτηση τριάδων, πόρων και ιδιοτήτων με χρήση και iterator, γράψιμο μεταδεδομένων RDF στην έξοδο (πχ σε αρχείο RDF/XML) ,προσθήκη μεταδεδομένων σε μοντέλα/γράφους και χειρισμό namespaces.

Αποτελείται από μια προγραμματιστική διεπαφή εφαρμογών η οποία μπορεί να κληθεί με τον ίδιο τρόπο που μπορεί να κληθεί οποιαδήποτε μέθοδος Java, οπότε προσφέρει εξαιρετική ενσωμάτωση και διαλειτουργικότητα με κώδικα γραμμένο σε Java η με χρήση JSP.

Μπορεί να χρησιμοποιηθεί το δομοστοιχείο ARQ της JENA για προώθηση επερωτήσεων SPARQL στο πλαίσιο JENA προς εκτέλεση, με κλήση του είτε από την γραμμή εντολών είτε με κλήση του API της JENA εντός κώδικα Java [17].

Για μόνιμη αποθήκευση RDF μεταδεδομένων μπορούν να χρησιμοποιηθούν δύο υποσυστήματα μόνιμης αποθήκευσης:

1) TDB. Χρησιμοποιεί το τοπικό σύστημα αρχείων για αποθήκευση μοντέλων (γράφων) RDF και μοντέλα αποθηκεύονται σε αυτό μέσω χρήσης της υλοποίησης του Μοντέλου η οποία περιέχεται στο υποσύστημα TDB για δημιουργία μοντέλων τα

οποία συσχετίζονται με μια συγκεκριμένη διεύθυνση του συστήματος αρχείων. Όταν προστίθενται δεδομένα στο μοντέλο, ασύγχρονα ενημερώνεται και η τοποθεσία μόνιμης αποθήκευσης του μοντέλου στο σύστημα αρχείων. Μεθόδοι του μπορούν να καλεστούν είτε από την γραμμή εντολών είτε με χρήση του API σε Java και συμπεριλαμβάνει όλες τις μεθόδους του API της JENA και επιπλέον δυνατότητες διαχείρισης μόνιμης αποθήκευσης. Δεν αποτελεί transactional ΣΔΒΔ και άρα δεν έχει τις δυνατότητες και λειτουργίες ενός τέτοιου συστήματος, κάτι που περιορίζει την λειτουργικότητα που προσφέρει αλλά παράλληλα μειώνει σημαντικά τους πόρους που απαιτούνται από το σύστημα πάνω στο οποίο είναι εγκατεστημένο για την λειτουργία του.

2) SDB. Χρησιμοποιεί μια ΒΔ SQL και υποστηρίζει ACID transactions διασφαλίζοντας ατομικότητα, συνέπεια, απομόνωση και αντοχή, κάτι που στο TDB (το οποίο δεν έχει δυνατότητες ελέγχου transactions) πρέπει να διασφαλίζεται από τον κώδικα που χρησιμοποιεί το υποσύστημα. Έχει όμως το μειονέκτημα ότι απαιτεί πολύ περισσότερους πόρους συστήματος και προσφέρει χαμηλότερη επίδοση λόγω του ότι είναι βασισμένο σε ένα ολοκληρωμένο ΣΔΒΔ.

Παρέχει επίσης δυνατότητα χρήσης και επεξεργασίας οντολογιών μέσω ενσωματωμένης μηχανής εξαγωγής λογικών συμπερασμάτων.

### 2.7 Oracle 11g

Η Oracle 11g ήταν το ΣΔΒΔ που χρησιμοποιούσε η υλοποίηση του κου Κωνσταντίνου Σαββίδη. Είναι ένα αντικειμενο-σχεσιακό ΣΔΒΔ υλοποιημένο σε C και C++ το οποίο παράγεται και διανέμεται από την εταιρεία Oracle.

Μπορεί να διαχειριστεί RDF μεταδεδομένα με χρήση της Oracle Semantic Technology η οποία προσφέρει υποστήριξη για RDF, OWL οντολογίες, εξαγωγή συμπερασμάτων και ενσωμάτωση με υπάρχοντα εργαλεία Σημασιολογικού Ιστού [23].

Στη ΒΔ τα RDF μεταδεδομένα αποθηκεύονται ως κατευθυνόμενοι γράφοι οι οποίοι αποτελούνται από τις τριάδες τους, η κάθε μία αποθηκευμένη ως ο εξειδικευμένος τύπος δεδομένων SDO_RDF_TRIPLE_S [7]. Κάθε γράφος αποθηκεύεται ως ένα Μοντέλο, και όλοι οι γράφοι μαζί συνενωμένοι αποτελούν το Δίκτυο Δεδομένων RDF

της συγκεκριμένης ΒΔ. Τα RDF μεταδεδομένα περιγράφονται εντός ενός σχήματος συστήματος, με χρήση τριών πινάκων και μίας επιπλέον όψης:

1)Πίνακας RDF_MODEL_INTERNAL$. Περιέχει πληροφορίες για όλα τα μοντέλα που είναι αποθηκευμένα στη ΒΔ.

2)Όψη SEMM_<όνομα μοντέλου>. Δημιουργείται μια τέτοια όψη για κάθε μοντέλο που είναι αποθηκευμένο στη ΒΔ και περιέχει πληροφορίες για όλες τις τριάδες του εν λόγω μοντέλου.

3)Πίνακας RDF_VALUE$. Περιέχει πληροφορίες για τα υποκείμενα, ιδιότητες και αντικείμενα που χρησιμοποιούνται για την αναπαράσταση δηλώσεων (τριάδων) RDF. Αποθηκεύει μοναδικά τις τιμές σε μορφή κειμένου (URI πόρων η σταθερές) για τα πιο πάνω [22].

4)Πίνακας RDF_LINKS$. Περιέχει πληροφορίες σχετικά με τις συσχετίσεις μεταξύ πόρων μέσω ιδιοτήτων/κατηγορημάτων.

Για την εκτέλεση επερωτήσεων SPARQL δεν μπορεί να χρησιμοποιηθεί απευθείας προώθηση επερωτήσεων SPARQL ή περιτύλιξη τους σε επερωτήσεις SQL αλλά πρέπει να χρησιμοποιηθεί το API της ΒΔ. Κυρίως η συνάρτηση SEM_MATCH η οποία επιστρέφει ένα πίνακα με τα αποτελέσματα που επιστρέφει η εισαγόμενη ως λίστα παραμέτρων επερώτηση SPARQL ο οποίος μπορεί έπειτα να χρησιμοποιηθεί ως πίνακας ΒΔ στα πλαίσια επερώτησης SQL.

Έστω η παρακάτω επερώτηση από το [22].

```
SELECT x, y
  FROM TABLE(
     SEM_MATCH
     (
        '(?x :grandParentOf ?y) (?x rdf:type :Male)',
         SEM_Models('family'),
         NULL,
```

```
        SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
        NULL
    )
);
```

η οποία επιστρέφει όλους τους παππούδες και τα εγγόνια τους από το μοντέλο το οποίο είναι ορισμένο με το αναγνωριστικό ‘Family’.

Η φόρτωση RDF μεταδεδομένων στη ΒΔ μπορεί να γίνει με έναν εκ τριών τρόπων οι οποίοι περιγράφονται στο [7].

### 2.8 Πλαίσιο OSGI

Το OSGI (Open Services Gateway Initiative) είναι ένα σύστημα δομοστοιχείων και πλατφόρμα υπηρεσιών για την γλώσσα προγραμματισμού Java. Η ανάπτυξη του καθοδηγήθηκε από την ανάγκη για ευκολότερη ενσωμάτωση και διαλειτουργικότητα των δομοστοιχείων και βιβλιοθηκών των εφαρμογών σε Java ούτως ώστε να μπορούν να ανταποκρίνονται στις σημερινές ανάγκες για ραγδαία τροποποίηση και επανασυνδυασμό δομοστοιχείων λογισμικού προς ικανοποίηση συνεχώς μεταλλασσόμενων αναγκών [24].

Η χρήση αρχιτεκτονικής λογισμικού προσανατολισμένης προς την έννοια της υπηρεσίας από την Πλατφόρμα Υπηρεσιών OSGI αντί της έννοιας της στατικής βιβλιοθήκης επιτρέπει την δυναμική τροποποίηση της σύνθεσης δομοστοιχείων ενός συστήματος λογισμικού χωρίς την ανάγκη επανεκκίνησης. Επίσης μειώνει περαιτέρω το κόστος και την πολυπλοκότητα της ανάπτυξης λογισμικού αφού η χρήση δυναμικών υπηρεσιών και η αυτόματη αλληλοανίχνευση παροχέων αυτών των υπηρεσιών επιτρέπει την μείωση του βαθμού σύζευξης (coupling) μεταξύ των δομοστοιχείων ενός συστήματος.

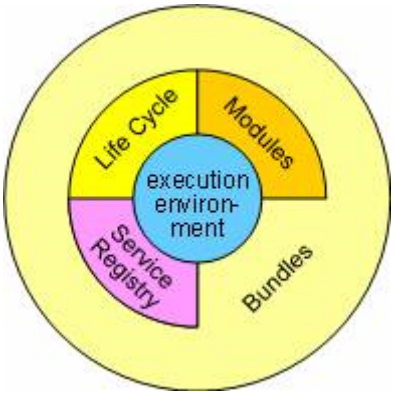
Το πλαίσιο OSGI υποδιαιρείται στα ακόλουθα επίπεδα:

1) Περιβάλλον εκτέλεσης. Το περιβάλλον Java εντός του οποίου τρέχει το πλαίσιο.

2) Δομοστοιχεία. Υπεύθυνο για τις πολιτικές φόρτωσης κλάσεων, οι οποίες πολιτικές βασίζονται στην Java αλλά προσφέρουν μεγαλύτερη δομικότητα σε σχέση με το μοντέλο της Java το οποίο υποστηρίζει κανονικά ένα μοναδικό classpath.

3) Κύκλος ζωής. Υπεύθυνο για την προσθήκη bundles τα οποία μπορούν να εγκατασταθούν, εκκινηθούν, τερματιστούν, ενημερωθούν και αποεγκατασταθούν δυναμικά.

4) Μητρώο υπηρεσιών. Υπεύθυνο για την δυναμική συνεργασία μεταξύ υπηρεσιών προσφερόμενων από bundles μέσω συμβάντων.




Σχήμα 2.4 Σχηματική απεικόνιση των συνιστώντων επιπέδων του πλαισίου OSGI.

Ένα bundle αποτελεί μια ομάδα κλάσεων Java και σχετικών πόρων (όπως ένα συνηθισμένο JAR) με την διαφορά ότι συμπεριλαμβάνει πιο εξειδικευμένη περιγραφή η οποία έχει ως στόχο την παροχή δυνατότητας αξιοποίησης των δυνατοτήτων που προσφέρει το OSGI. Πριν να μπορέσει ένα bundle να προσφέρει υπηρεσίες ανιχνεύσιμες από άλλα δομοστοιχεία πρέπει: 1) Να εγκατασταθεί επιτυχώς. 2) Όλες οι κλάσεις Java επάνω στις οποίες βασίζεται να έχουν επιλυθεί. 3) Να εκκινηθεί και τερματίσει η διαδικασία ενεργοποίησης της υλοποίησης της διεπαφής BundleActivator.start που ισχύει για το bundle. 4) Να έχει ικανοποιηθεί η πολιτική ενεργοποίησης του (ένα σύνολο κανόνων που ορίζει πότε ενεργοποιείται το bundle) . Όταν ένα bundle είναι ενεργό μπορεί να προσφέρει τις υπηρεσίες που ορίζονται στις ρυθμίσεις του.

```java
package com.example.helloworld;

import org.osgi.framework.BundleActivator;

public class Activator implements BundleActivator {

    /*
     * (non-Javadoc)
     * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
     */
    public void start(BundleContext context) throws Exception {
        System.out.println("Hello World!!");
    }

    /*
     * (non-Javadoc)
     * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
     */
    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye World!!");
    }

}
```

Σχήμα 2.5 Παράδειγμα κλάσης η οποία αποτελεί υλοποίηση της διεπαφής BundleActivator.

Το πιο γνωστό παράδειγμα επιτυχούς εφαρμογής του πλαισίου OSGI στην ανάπτυξη λογισμικού μεγάλου μεγέθους είναι το ευρύτατα χρησιμοποιημένο περιβάλλον ανάπτυξης λογισμικού Eclipse IDE.

### 2.9 JSP(Java Server Pages)

Αποτελούν μια τεχνολογία βασισμένη στην γλώσσα προγραμματισμού Java η οποία επιτρέπει την δημιουργία ιστοσελίδων με δυναμικό περιεχόμενο χρησιμοποιώντας HTML και Javascript δημιουργούμενη δυναμικά κατά την ώρα της απάντησης σε μια παραλαμβανόμενη αίτηση από έναν εξυπηρετητή. Βασίζεται σε δεδομένα παραλαμβανόμενα μέσω της αίτησης η/και δεδομένα αποθηκευμένα τοπικά στον εξυπηρετητή πχ σε μια ΒΔ. Αυτό γίνεται με χρήση κώδικα Java ενσωματωμένου εντός κώδικα HTML ο οποίος εκτελείται κατά την ώρα της απάντησης σε μια αίτηση.

Η μεταγλώττιση γίνεται μόνο μία φορά, δημιουργώντας προσωρινά Servlets τα οποία τρέχουν στον εξυπηρετητή όταν παραλαμβάνεται μια αίτηση, δημιουργώντας δυναμικά την ιστοσελίδα η οποία επιστρέφεται πίσω στον πελάτη. Έτσι αποφεύγεται η ανάγκη για επαναμεταγλώττιση του κώδικα για κάθε νέα παραλαμβανόμενη αίτηση. Όπως

ακριβώς τα Servlets τα JSP έχουν πρόσβαση σε όλα τα δεδομένα της παραλαμβανόμενης αίτησης και της εφαρμογής [9].

Για την εγκατάσταση και χρήση τους χρησιμοποιείται ο εξυπηρετητής Apache Tomcat.

Στο σύστημα που περιγράφεται στα πλαίσια αυτής της ΑΔΕ τα JSP χρησιμοποιούνται στον εξυπηρετητή εφαρμογών, κυρίως όπου χρειάζεται απευθείας επικοινωνία με τον εξυπηρετητή ΒΔ και για την διαμόρφωση των ενδιάμεσων επερωτήσεων βάσει παραλαμβανομένων δεδομένων.

### 2.10 Javascript

Η Javascript είναι μια γλώσσα scripting η οποία χρησιμοποιείται για δημιουργία δυναμικών ιστοσελίδων, όπως τα JSP και η PHP, με την διαφορά ότι εκτελείται στον πελάτη μετά την παραλαβή της ιστοσελίδας ως απάντηση σε αίτηση. Δίνει πολύ μεγαλύτερη ελευθερία όσο αφορά την διαχείριση τύπων, μεταβλητών κοκ αφού οι τύποι δεδομένων συσχετίζονται με τιμές και όχι με μεταβλητές, επιτρέποντας δυναμικό καθορισμό τύπου. Παρόλο που βασίζεται μέχρι ενός βαθμού στην έννοια του αντικειμένου δεν αποτελεί πλήρως αντικειμενοστρεφή γλώσσα προγραμματισμού αφού δεν υποστηρίζει πολυμορφισμό ούτε κληρονομικότητα κλάσεων.

Χρήση του Document Object Model, DOM (το οποίο είναι μια δομή δεδομένων η οποία δίδει πρόσβαση στα δομοστοιχεία HTML μιας ιστοσελίδας) επιτρέπει, μέσω χρήσεως της Javascript, την δυναμική τροποποίηση της ιστοσελίδας μέσω της τροποποίησης των στοιχείων του DOM.

Λόγω των πιο πάνω δυνατοτήτων που προσφέρει, καθώς και της δυνατότητας ασύγχρονης επικοινωνίας με τον εξυπηρετητή εφαρμογών μέσω κλήσεων AJAX, χρησιμοποιείται για την υλοποίηση του μέρους του συστήματος που εξετάζει αυτή η ΑΔΕ το οποίο τρέχει στον πελάτη.

### 2.11.Java

Η Java είναι πιθανόν η πιο ευρέως χρησιμοποιούμενη γλώσσα αντικειμενοστρεφούς προγραμματισμού, με ευρεία υποστήριξη κοινότητας και χρήση σχεδόν οπουδήποτε, από εξυπηρετητές εταιρικών δικτύων μέχρι ενσωματωμένα συστήματα. Έχει

δημιουργηθεί από την εταιρεία Sun Microsystems που τώρα είναι θυγατρική της Oracle.

Τα μεγάλα πλεονεκτήματα της Java σε σχέση με άλλες γλώσσες Α/Σ προγραμματισμού είναι:

1) Μπορεί, επειδή γράφεται και μεταφράζεται μόνο μία φορά (παράγοντας αρχεία .class τα οποία εμπεριέχουν μεταφρασμένες κλάσεις) να τρέξει σε οποιαδήποτε συσκευή/πλατφόρμα η οποία συμπεριλαμβάνει Java Virtual Machine χωρίς την ανάγκη επαναπροσαρμογής του κώδικα, οπότε προσφέρει εξαιρετική ευελιξία.

2) Είναι μια σχετικά απλή γλώσσα προγραμματισμού, σχετικά υψηλού επιπέδου, κάτι που μειώνει σημαντικά την προσπάθεια που απαιτείται για την ανάπτυξη λογισμικού, καθώς και την πολυπλοκότητα του παραγόμενου κώδικα (και άρα και την πιθανότητα εμφάνισης σφαλμάτων). Οι μεταγλωττιστές και διερμηνείς της εφαρμόζουν αυστηρούς ελέγχους οι οποίοι βοηθούν επιπλέον στην αποφυγή σφαλμάτων.

3) Συμπεριλαμβάνει ένα μεγάλο αριθμό έτοιμων βιβλιοθηκών για ένα μεγάλο και ποικίλο εύρος αναγκών, συμπεριλαμβανομένης της ανάπτυξης εφαρμογών για το ΠΠΠ, επικοινωνίας μέσω πρωτοκόλλων όπως HTTP και άμεσης διασύνδεσης με ένα μεγάλο εύρος ΣΔΒΔ μέσω χρήσης βιβλιοθηκών όπως JDBC.

4) Τα πιο πάνω οδήγησαν στην ευρεία υιοθέτηση της από όλους, από την ακαδημαική κοινότητα μέχρι την βιομηχανία και την κοινότητα ανοικτού κώδικα, οδηγώντας στην δημιουργία τεράστιας υποστήριξης κοινότητας.

5) Προσφέρει σχετικά εύχρηστους μηχανισμούς υποστήριξης πολυνηματικών εφαρμογών, κάτι που είναι ιδιαίτερα χρήσιμο για εφαρμογές εξυπηρετητών.

Στο σύστημα που αφορά αυτή η ΑΔΕ η Java χρησιμοποιείται στον εξυπηρετητή εφαρμογών για την υλοποίηση του μεταφραστή XML (που αντιπροσωπεύει MashQL) σε SPARQL, του βελτιστοποιητή ενδιάμεσων επερωτήσεων (που περιγράφεται σε επόμενο κεφάλαιο) και του συστήματος καταφόρτωσης συνόλων δεδομένων RDF (που επίσης περιγράφεται σε επόμενο κεφάλαιο).

## 2.12 XML

Αρχικά τα δεδομένα στο ΠΠΠ (εκφρασμένα σε HTML, μια γλώσσα η οποία προορίζεται κυρίως για την παρουσίαση/μορφοποίηση των δεδομένων και όχι την αποθήκευση τους) είχαν μια μεγάλη ασυμβατότητα με τα δεδομένα τα οποία αποθηκεύονταν σε ΒΔ (σχεσιακά σχήματα, διαγράμματα οντοτήτων-συσχετίσεων). Αυτή η ασυμβατότητα οδηγούσε σε δυσκολίες στην ολοένα και πιο συχνά παρουσιαζόμενη περίπτωση που χρειαζόταν δημοσιοποίηση δεδομένων ΒΔ στο ΠΠΠ ή δεδομένα από ιστοσελίδες χρειαζόταν να εισαχθούν (πχ για καταγραφή) σε ΒΔ. Η αποθήκευση των δεδομένων σε μορφή HTML έχει τα μειονεκτήματα της αστάθειας (μικρές αλλαγές στον κώδικα που είναι υπεύθυνος για λεπτομέρειες της εμφάνισης των ιστοσελίδων και όχι την δομή των δεδομένων δημιουργούν προβλήματα σε προγράμματα ανάλυσης) και σπατάλη πόρων λόγω αχρείαστων (όταν θέλουμε απλά «ωμά δεδομένα») μεταδεδομένων σχετικών με την παρουσίαση των δεδομένων.

Για την επίλυση αυτού του προβλήματος δημιουργήθηκε η μετα-γλώσσα XML για κωδικοποίηση πληροφοριών. Αυτή αποτελεί ένα τρόπο-πρότυπο περιγραφής δομημένων δεδομένων. Δεν ασχολείται με την περιγραφή της μορφοποίησης/τρόπου παρουσίασης των δεδομένων αλλά μόνο με την δομή τους και επομένως χαρακτηρίζεται από σταθερότητα και οικονομία όσο αφορά την κατανάλωση χώρου αποθήκευσης.

Η μετα-γλώσσα XML χρησιμοποιήθηκε μέχρι σήμερα για ανάπτυξη πολλών γλωσσών που έχουν να κάνουν με δομή δεδομένων στο ΠΠΠ συμπεριλαμβανομένης της μορφής RDF μεταδεδομένων RDF/XML.

Μπορεί να μεταφραστεί σε μια πιο εύχρηστη μορφή με χρήση σαρωτών XML. Πχ σε μορφή DOM η οποία μπορεί να διαβαστεί προγραμματιστικά με πολύ μεγαλύτερη ευκολία αφού είναι πιο κοντά στο αντικειμενοστρεφές μοντέλο της Java, αφού αποτελείται από αντικείμενα και τις μεταξύ τους σχέσεις και αποτελεί μέρος του API της Java.

Στο σύστημα το οποίο αφορά αυτή η ΑΔΕ η XML χρησιμοποιείται (εκτός από το ότι τα μεταδεδομένα RDF που επεξεργάζεται το σύστημα βασίζονται πάνω σε αυτήν) για μετατροπή σε αυτήν των επερωτήσεων MashQl που δημιουργεί ο χρήστης για αποθήκευση τους στη ΒΔ ή/και προώθηση τους στον εξυπηρετητή εφαρμογών για μετάφραση τους σε SPARQL.

## 2.13 AJAX

Asynchronous Javascript and XML. Αποτελεί μια τεχνική ανάπτυξης εφαρμογών για το ΠΠΠ η οποία επιτρέπει ασύγχρονη επικοινωνία μεταξύ πελάτη και εξυπηρετητή, κάτι το οποίο επιτρέπει συνέχιση της χρήσης μιας ιστοσελίδας από τον χρήστη και δημιουργία νέων αιτήσεων προς τον εξυπηρετητή ενώ οι προηγούμενες αιτήσεις ακόμη τυγχάνουν επεξεργασίας ή/και μεταφέρονται μέσω δικτύων.

Βασίζεται στο αντικείμενο XMLHTTP το οποίο είναι υπεύθυνο για την ανάκτηση και επεξεργασία εξωτερικών ιστοσελίδων από εντός του κώδικα της εφαρμογής η οποία τρέχει.

Η AJAX χρησιμοποιείται στο σύστημα το οποίο αφορά αυτή η ΑΔΕ οπουδήποτε ο πελάτης πρέπει να επικοινωνήσει με τον εξυπηρετητή εφαρμογών ούτως ώστε να επιτρέπει στον χρήστη την εκτέλεση νέων πράξεων και την προώθηση νέων εντολών προς το σύστημα ενώ αναμένεται η επεξεργασία των προηγούμενων (κάτι ιδιαίτερα χρήσιμο όταν έχουμε να κάνουμε με εντολές για επεξεργασία μεγάλων συνόλων δεδομένων οι οποίες παίρνουν περισσότερο χρόνο).

# Κεφάλαιο 3

## Περιγραφή προυπάρχουσας υλοποίησης συστήματος



### 3.1 Εισαγωγή

Ακολουθεί η λεπτομερής περιγραφή του συστήματος πελάτη-εξυπηρετητή για Web Mashups όπως ήταν υλοποιημένο από τον κο Κωνσταντίνο Σαββίδη με βάση την Semantic Technology της Oracle.

### 3.2 Λειτουργίες

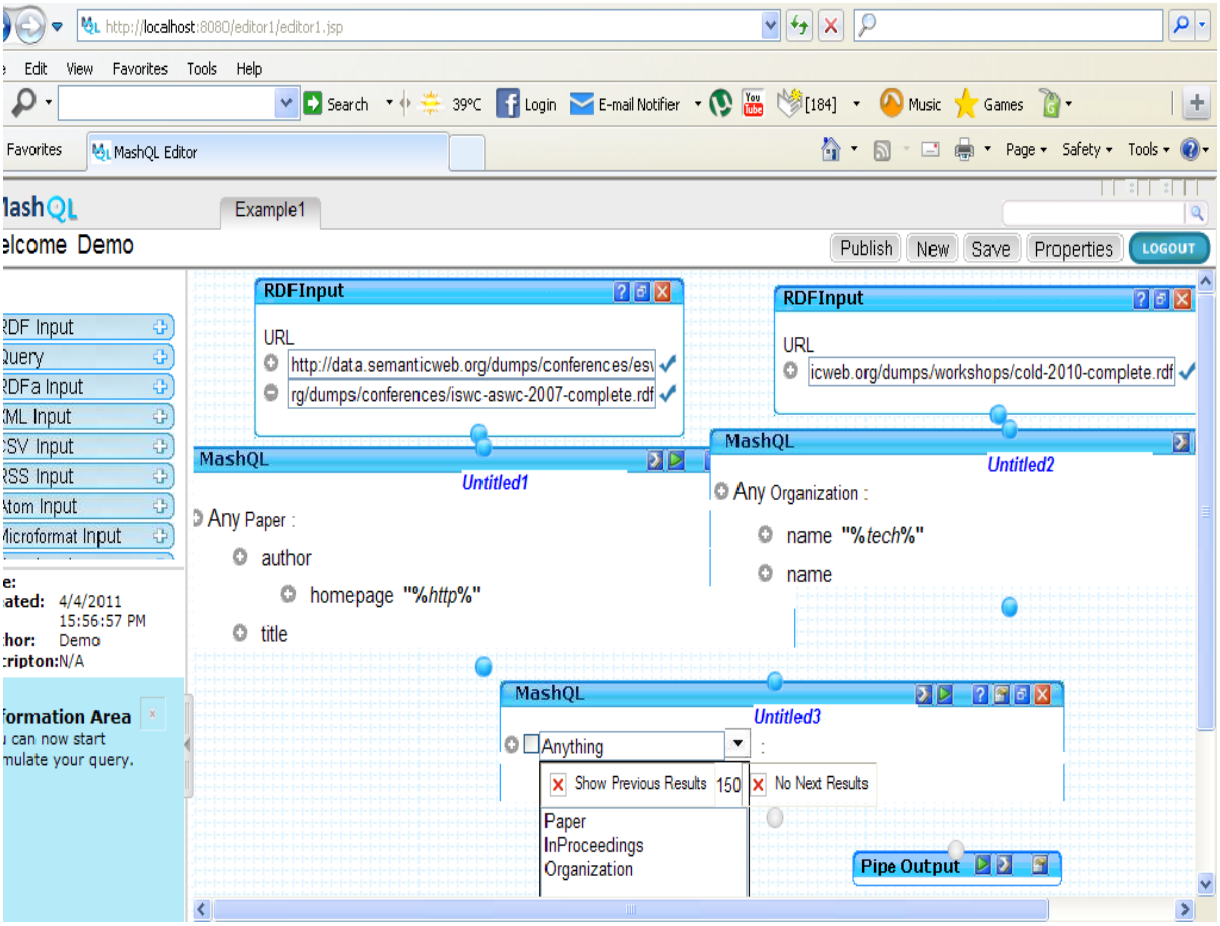
Είναι ένας επεξεργαστής επερωτήσεων SPARQL και Web Mashups στην γλώσσα MashQL διαθέσιμος online και προσβάσιμος μέσω του φυλλομετρητή Internet Explorer [28].

Προσφέρει τις ακόλουθες δυνατότητες:

1) Δημιουργία και εκτέλεση επερωτήσεων SPARQL και σύνθεση Web Mashups μέσω χρήσης τεχνολογίας Web Pipes.

2) Αποθήκευση δημιουργημένων pipes/web Mashups και ανάκτηση τους.

Η κύρια οθόνη διεπαφής είναι όπως πιο κάτω:




Σχήμα 3.1 Κύρια οθόνη διεπαφής

### 3.3 Αρχιτεκτονική συστήματος

Βασίζεται στην αρχιτεκτονική τριών επιπέδων, τα οποία είναι:

Πελάτης

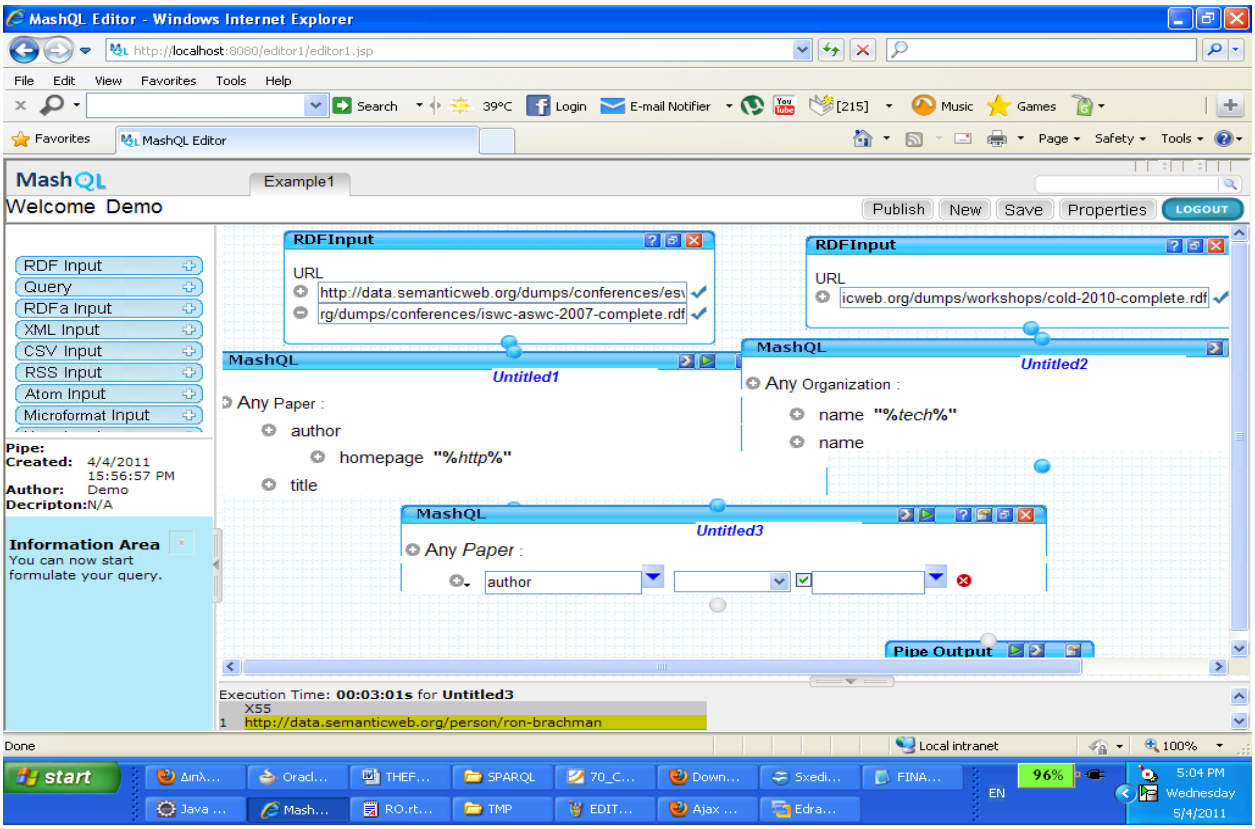
Από εδώ αρχίζει η πρόσβαση στο σύστημα. Στο σύστημα του πελάτη τρέχουν μεθόδοι Javascript οι οποίες είναι υπεύθυνες για:

Α) Παρουσίαση της διεπαφής του συστήματος στον χρήστη.

Β) Προώθηση εντολών προς τον εξυπηρετητή για καταφόρτωση συνόλων μεταδεδομένων από εισαγόμενο URL.

Γ) Δημιουργία τελικών επερωτήσεων στη γλώσσα MashQL μέσω επιλογής από τα αποτελέσματα επερωτήσεων περιβάλλοντος ή εισαγωγής δεδομένων από τον χρήστη.

Δ) Δημιουργία κώδικα XML που αντιπροσωπεύει τις τελικές επερωτήσεις που εισάγονται σε MashQL και προώθηση του στον εξυπηρετητή εφαρμογών σε αυτή την μορφή.




Σχήμα 3.2 Ένα παράδειγμα επερώτησης (Mashup) MashQL όπως μεταφράζεται σε XML από τον πελάτη πριν να προωθηθεί στον εξυπηρετητή εφαρμογών.

```
<?xml version="1.0" encoding="UTF-8"?>
<Pipe ID="0" xsi:noNamespaceSchemaLocation="MashQL.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Meta Content="Title" Name=""></Meta>
<Meta Content="Creator" Name="Demo"></Meta>
<Meta Content="DateCreated" Name="4/4/2011 17:03:38 PM"></Meta>
<Meta Content="DateExecuted" Name=""></Meta>
<Meta Content="UserID" Name=""></Meta>
<RDFInput ID="moduleQuery4" X1="-7" Y1="119" X2="504" Y2="">
```

```
<Source Order="0">
<Ref>http://data.semanticweb.org/dumps/conferences/eswc-2007-
complete.rdf</Ref>
<LastUpload>4/4/2011 17:03:38 PM</LastUpload>
</Source>
<Source Order="1">
<Ref>http://data.semanticweb.org/dumps/conferences/iswc-aswc-2007-
complete.rdf</Ref>
<LastUpload>4/4/2011 17:03:38 PM</LastUpload>
</Source>
</RDFInput>
<RDFInput ID="moduleQuery7" X1="426" Y1="107" X2="504" Y2="">
<Source Order="0">
<Ref>http://data.semanticweb.org/dumps/workshops/cold-2010-
complete.rdf</Ref>
<LastUpload>4/4/2011 17:03:38 PM</LastUpload>
</Source>
</RDFInput>
<Query    ID="moduleQuery8"    X1="207"    X2="530"    Y1="176"    Y2=""
InputModule="moduleQuery4,moduleQuery7"  isConnectToOutput="0">
<Header>
<MetaData Content="Title" Name="Untitled3" ></MetaData>
</Header>
<Body>
<Subject      Name="http://data.semanticweb.org/ns/swc/ontology#Paper"
Type="Constant" isRetrun="false" ></Subject>
<Restriction Prefix="">
<Predicate             Name="http://swrc.ontoware.org/ontology#author"
Type="Constant" isReturn="false" ></Predicate>
<Object Name="X1" Type="Variable" isReturn="true" ></Object>
<ObjectFilter   xsi:type=""   Value=""   Type="Variable"   Language=""
DataType=""></ObjectFilter>
</Restriction>
</Body>
<Footer>
<Order>
<Variable Name="" Direction=""></Variable>
</Order>
<Modifier Limit="" Offset="" Duplication=""></Modifier>
<Output Stylesheet="" Format=""></Output>
```

```
</Footer>
</Query>
</Pipe>
```

Ε) Σύνθεση αλληλοεξαρτούμενων επερωτήσεων σε πολλά επίπεδα για δημιουργία Web Mashups (ώστε το σύστημα να λειτουργεί κάπως ως Web composer) [11].

ΣΤ) Τροποποίηση επερωτήσεων/Mashups στο επίπεδο της γλώσσας SPARQL πριν την εκτέλεση τους.

Ζ) Παρουσίαση αποτελεσμάτων επερωτήσεων περιβάλλοντος και αποτελεσμάτων τελικών επερωτήσεων και μεταδεδομένων σχετικά με την εκτέλεση των τελικών επερωτήσεων (χρόνος εκτέλεσης).

Η) Δημιουργία επιλογών για τις επερωτήσεις περιβάλλοντος βάσει των αποτελεσμάτων προηγούμενων επερωτήσεων περιβάλλοντος και επερωτήσεων που προηγούνται στο Mashup στο οποίο ανήκει η εν λόγω επερώτηση και προώθηση αυτών στον εξυπηρετητή εφαρμογών για δημιουργία απευθείας SPARQL για εκτέλεση.

Θ)Μετατροπή URI πόρων RDF επιστρεφόμενων από την εκτέλεση της επερώτησης σε συντομευμένη μορφή πιο κατανοητή από τον χρήστη (URI normalization).

Ο πελάτης συνδέεται με τον εξυπηρετητή εφαρμογών μέσω AJAX (Asynchronous Javascript and XML) ούτως ώστε να παρέχονται συνδέσεις HTTP οι οποίες να μην μπλοκάρουν (αφού είναι ασύγχρονες) την λειτουργία της υπόλοιπης διεπαφής.

Εξυπηρετητής εφαρμογών

Είναι υπεύθυνος για την εξυπηρέτηση αιτήσεων του πελάτη. Αποτελεί ενδιάμεσο επίπεδο μεταξύ του πελάτη και του εξυπηρετητή ΒΔ. Είναι υλοποιημένος με Java Server Pages (οι οποίες παραλαμβάνουν και επεξεργάζονται τις παραλαμβανόμενες αιτήσεις και επιστρέφουν στον πελάτη τα αποτελέσματα) και κλάσεις Java (όσο αφορά τον μεταφραστή XML σε SPARQL) που καλούνται από JSP.

Α) Λειτουργεί ως μεταφραστής εισαγόμενων τελικών επερωτήσεων από XML (στην οποία διαμορφώνονται από τον πελάτη τα εισαγόμενα/επιλεγμένα δεδομένα του χρήστη και η οποία αντιπροσωπεύει επερωτήσεις στην γλώσσα MashQL) στην γλώσσα επερωτήσεων SPARQL (η οποία αποτελεί την συνηθισμένη γλώσσα επερωτήσεων σε

RDF μεταδεδομένα και την οποία, με κάποιες αλλαγές ώστε να γίνει δεκτή με βάση την σύνταξη της Oracle-Oracle SPARQL-χρησιμοποιεί για προώθηση επερωτήσεων προς την βάση δεδομένων).

Αυτό επιτυγχάνεται με την εξής διαδικασία, η οποία ενεργοποιείται όταν ο εξυπηρετητής παραλαμβάνει αίτηση για το αρχείο JSP ParseQueryToSparql.jsp:

i. Κλήση των κλάσεων Java που είναι υπεύθυνες για την λειτουργία της μετάφρασης.

ii. Parsing από αυτές του παραλαμβανόμενου κώδικα XML με χρήση υλοποίησης του XMLParser της Java και δημιουργία του δέντρου DOM της επερώτησης.

iii. Χρήση κλάσεων Java για δημιουργία επερώτησης σε SPARQL βάσει του δημιουργημένου δέντρου DOM.

iv. Επιστροφή της δημιουργημένης επερώτησης σε SPARQL στον πελάτη.

Περισσότερες λεπτομέρειες σχετικά με την διαδικασία μετατροπής MashQL σε SPARQL μπορούν να βρεθούν στο [28].

Β) Καλεί (μέσω διασύνδεσης με την βάση δεδομένων Oracle) την εκτέλεση της επερώτησης SPARQL που παραλαμβάνεται μέσω αίτησης προς το JSP stmtExecutionConnection/stmtDebugConnection έχοντας ως στόχο είτε την επιστροφή αποτελεσμάτων είτε την δοκιμή (debugging) της εκτέλεσης της επερώτησης και την επιστροφή σχετικών μεταδεδομένων. Έπειτα επιστρέφει τα αποτελέσματα.

Γ) Όταν ληφθεί αίτηση για κλήση επερώτησης περιβάλλοντος (μέσω αίτησης προς το αρχείο callBquery.jsp) δημιουργεί βάσει των εισαγόμενων επιλογών επερώτηση SPARQL η οποία αποστέλλεται στον εξυπηρετητή ΒΔ για εκτέλεση, και επιστρέφει τα αποτελέσματα.

Δ) Καλεί τις αποθηκευμένες διαδικασίες στη ΒΔ για εισαγωγή URL συνόλων δεδομένων προς καταφόρτωση στην ουρά η οποία υπάρχει στην βάση δεδομένων, ούτως ώστε έπειτα να αναληφθούν της καταφόρτωσης τους (αν χρειάζεται) οι υπεύθυνες διαδικασίες της βάσης δεδομένων.

Ε) Είναι υπεύθυνος για προώθηση προς τον εξυπηρετητή ΒΔ αιτήσεων για αποθήκευση MashQl Pipes (δηλωμένων ως κώδικας XML) για ανάκτηση σε επόμενο στάδιο.

Εξυπηρετητής Βάσης Δεδομένων

Αποτελεί το κατώτατο επίπεδο της αρχιτεκτονικής και είναι υπεύθυνος για την διαχείριση σε φυσικό επίπεδο όλων των δεδομένων που σχετίζονται με το σύστημα, δηλαδή:

i. Των συνόλων μεταδεδομένων RDF τα οποία αποθηκεύονται στην ΒΔ Oracle ως ένα δίκτυο γράφων αντιπροσωπευόμενων μέσω πινάκων οι οποίοι περιέχουν τα μεταδεδομένα RDF, καθώς και μετα-μεταδεδομένα όπως το τι μέρος μιας τριάδας αποτελεί κάθε μονάδα δεδομένων, πως συνδέεται με ένα μοντέλο (όπως αντιπροσωπεύεται ένα σύνολο μεταδεδομένων RDF στη βάση) και με άλλες τριάδες. Μέσω αυτών των πινάκων δημιουργείται ένα δίκτυο μοντέλων/γράφων το οποίο αποτελεί τον τρόπο με τον οποίο η Oracle Semantic Technology αποθηκεύει τα μεταδεδομένα RDF και πάνω στο οποίο μπορούν να γίνουν επερωτήσεις μέσω μεθόδων όπως η SEM_MATCH οι οποίες επιστρέφουν μεταδεδομένα από αυτό το δίκτυο βάσει των επιλεγμένων μοντέλων και των περιορισμών επάνω σε αυτά [22].

Η υλοποίηση στην Oracle συμπεριλαμβάνει summaries των μεταδεδομένων RDF τα οποία δημιουργούνται δυναμικά αφού καταφορτωθούν τα σύνολα δεδομένων. Αυτά μειώνουν σημαντικά τους χρόνους εκτέλεσης των ενδιάμεσων επερωτήσεων αφού οι ενδιάμεσες επερωτήσεις γίνονται πάνω σε αυτά αντί πάνω στα αρχικά δεδομένα (τα οποία αυτά συνοψίζουν όπως χρειάζεται για εκτέλεση των ενδιάμεσων επερωτήσεων) οπότε οι χρόνοι πρόσβασης και επεξεργασίας δεδομένων και τα μεγέθη των ενδιάμεσων αποτελεσμάτων μειώνονται σημαντικά.

ii. Των αποθηκευμένων Web Mashup Pipes τα οποία έχει επιλέξει ο χρήστης να αποθηκεύσει, τα οποία είναι αποθηκευμένα ως αρχεία XML στη Βάση Δεδομένων.

Οι λειτουργίες για τις οποίες είναι υπεύθυνη η ΒΔ είναι:

1) Εισαγωγή URL συνόλων δεδομένων από το Διαδίκτυο που πρέπει να καταφορτωθούν μέσα σε μια ουρά (queue) όταν ληφθεί αίτηση για αυτό προερχόμενη

από τον εξυπηρετητή εφαρμογών, από την οποία ουρά άλλες, ασύγχρονα τρέχουσες αποθηκευμένες διαδικασίες στην Βάση Δεδομένων θα τα πάρουν για να καταφορτώσουν τα συσχετιζόμενα σύνολα δεδομένων.

2) Εξαγωγή URL συνόλων δεδομένων προς καταφόρτωση από την ουρά, επαλήθευση τους, έλεγχος για την ύπαρξη τους ήδη στην βάση ως μοντέλα και αν είναι ενημερωμένα. Αν υπάρχουν και είναι ενημερωμένα απλά ενημερώνεται ο εξυπηρετητής εφαρμογών σχετικά. Αν δεν υπάρχουν ή υπάρχουν αλλά δεν είναι ενημερωμένα (ξανα)καταφορτώνονται, μεταφράζονται και προσθέτονται στο δίκτυο μοντέλων της ΒΔ οπότε έπειτα είναι έτοιμα για εκτέλεση επερωτήσεων πάνω σε αυτά [22].

3) Δημιουργία δομών βελτιστοποίησης (summaries) των καταφορτωμένων δεδομένων για μείωση του χρόνου απόκρισης για τις επερωτήσεις περιβάλλοντος.

Συγκεκριμένα, δημιουργούνται summaries για:

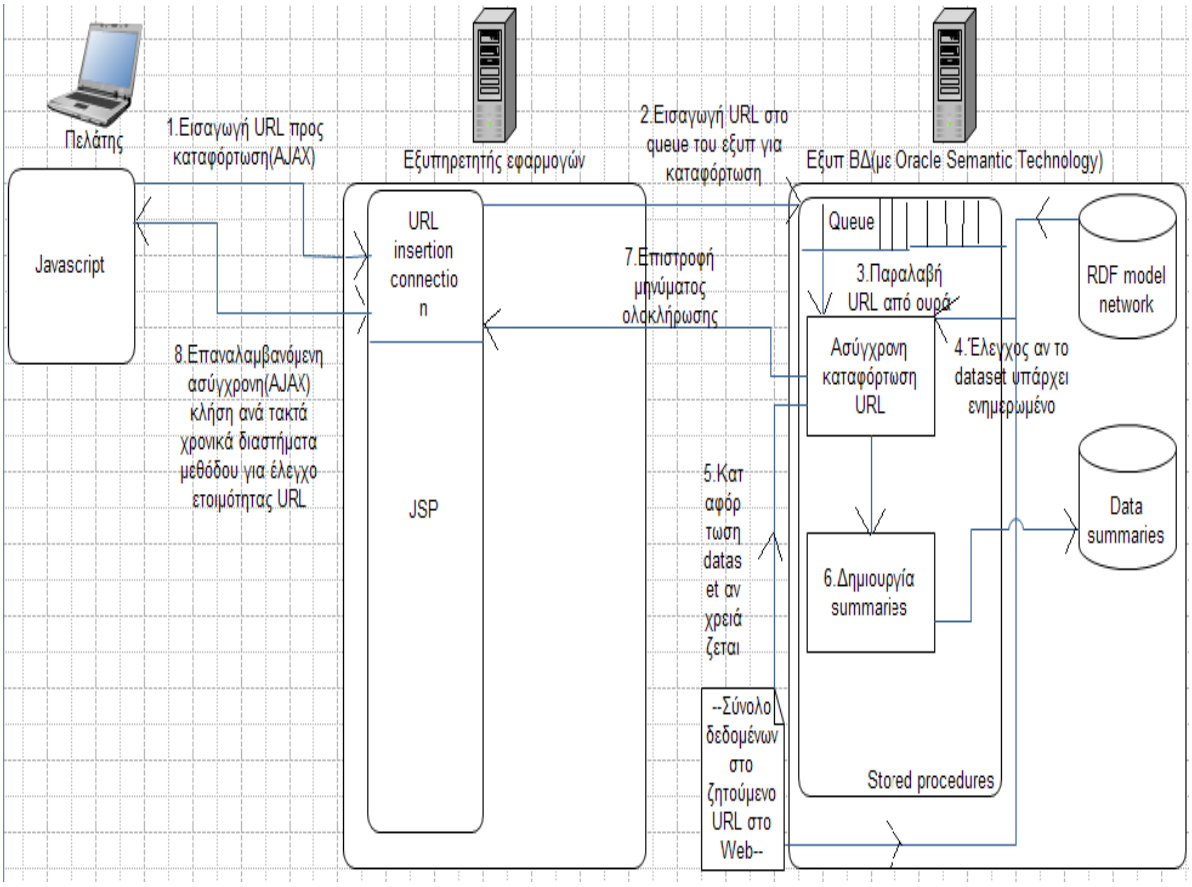
i. Όλα τα διακριτά υποκείμενα, όλα τα διακριτά χαρακτηριστικά/κατηγορήματα, όλα τα διακριτά αντικείμενα τριάδων του συνόλου δεδομένων.

ii. Όλους τους διακριτούς τύπους υποκειμένων και όλα τα διακριτά αντικείμενα που έχουν δηλωμένο τύπο.

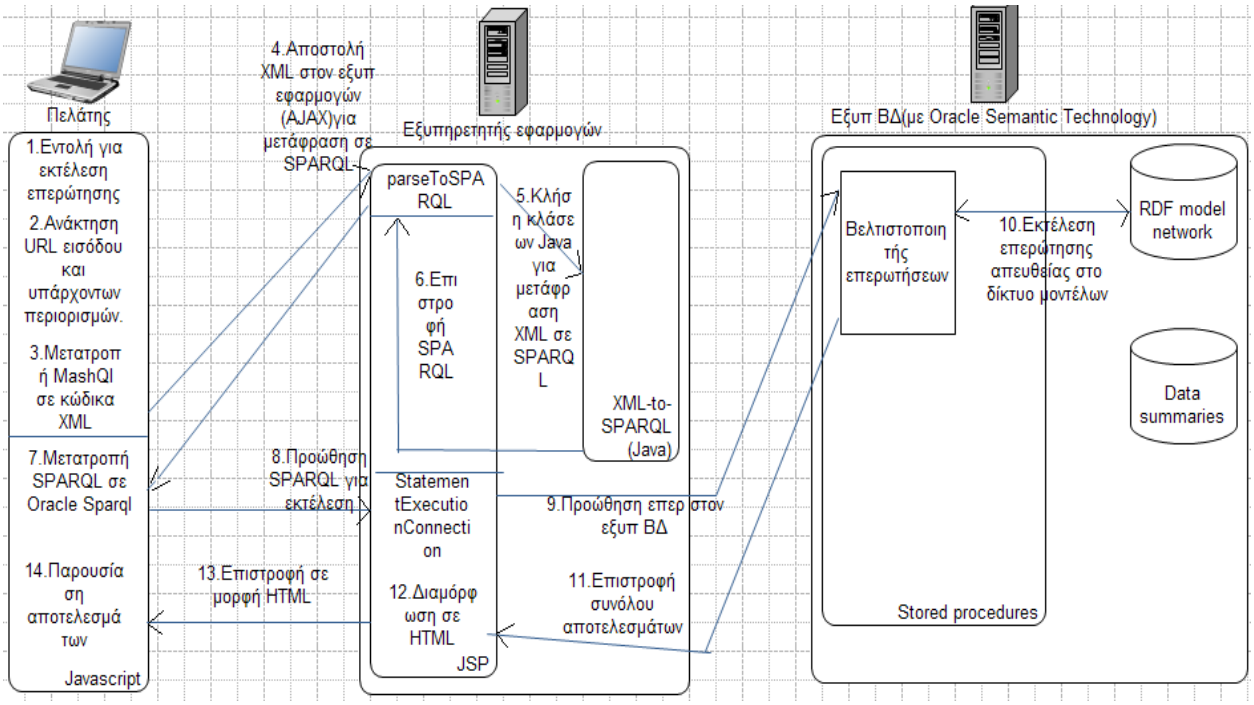
4) Εκτέλεση επερωτήσεων σε SPARQL με χρήση Oracle Semantic Technology (Oracle SPARQL) επάνω σε αποθηκευμένα μοντέλα βάσει εισαγόμενων περιορισμών. Αν πρόκειται για τελικές επερωτήσεις εκτελούνται ως έχουν. Αν όχι, αντί αυτών εκτελούνται επερωτήσεις πάνω στα summaries αναλόγως των παραλαμβανομένων επερωτήσεων, οι οποίες επερωτήσεις επάνω στα summaries αποτελούν προσδέσεις σε έτοιμα templates τα οποία δίνουν τα ίδια ενδιάμεσα αποτελέσματα.

### 3.4 Γενική λειτουργία

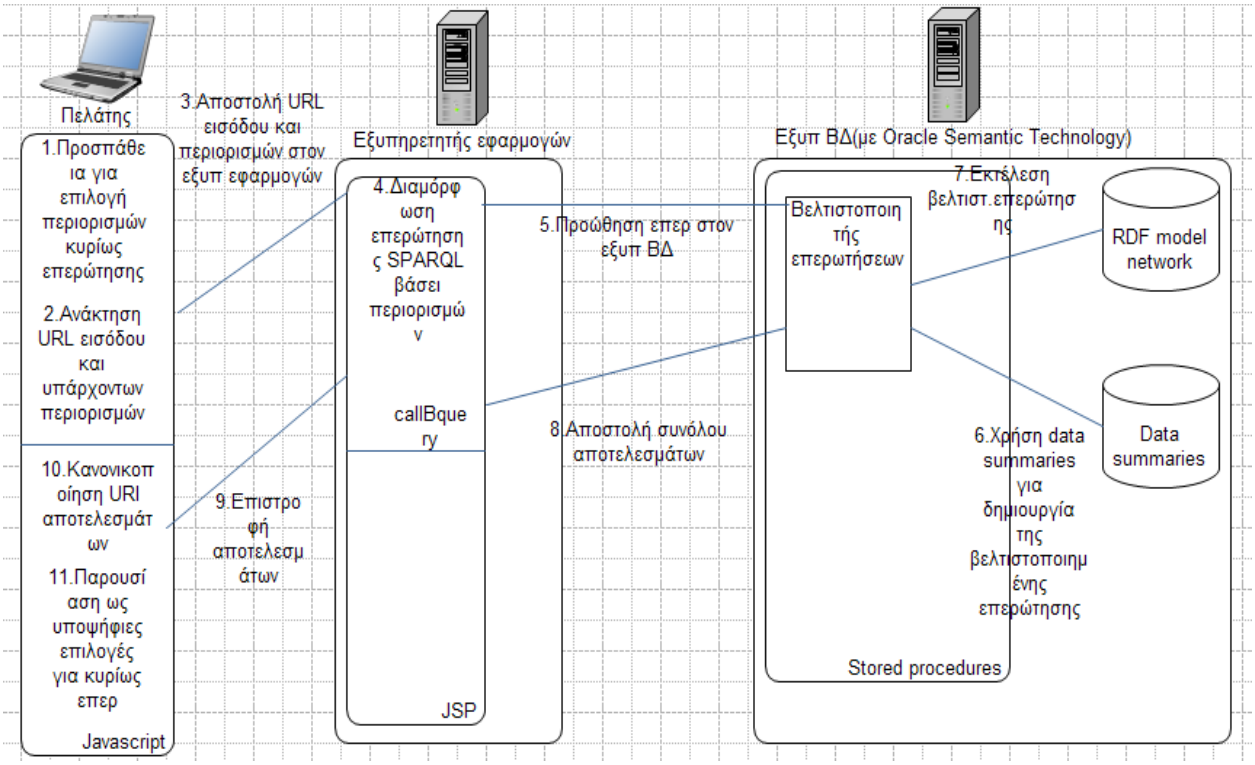
Ακολουθούν σχεδιαγράμματα που δείχνουν την λειτουργία του όλου συστήματος όσο αφορά την καταφόρτωση συνόλου δεδομένων από URL, την απάντηση επερωτήσεων περιβάλλοντος και την απάντηση τελικών επερωτήσεων.




Σχήμα 3.3 Καταφόρτωση συνόλου δεδομένων από URL




Σχήμα 3.4 Απάντηση σε τελική επερώτηση




Σχήμα 3.5 Απάντηση σε ενδιάμεσες επερωτήσεις

### 3.5 Προβλήματα τα οποία οδήγησαν στην προσπάθεια για αντικατάσταση της ΒΔ

Ως ΣΔΒΔ για το κατώτατο επίπεδο (αποθήκευση και επεξεργασία μεταδεδομένων) χρησιμοποιήθηκε η ΒΔ Oracle 11g. Παρόλο που η επίδοση της ήταν ικανοποιητική, ήταν μια «βαρειά» βάση. Βαρειά υπό την έννοια ότι είχε μεγάλες απαιτήσεις σε κυρίως μνήμη (1 GB μόνο για την καθαυτό λειτουργία της ΒΔ). Άρα-αν λάβουμε υπόψη και την ανάγκη για χρήση επιπλέον κυρίως μνήμης για καταφόρτωση και ανάκτηση μεταδεδομένων και επερωτήσεις επάνω σε αυτά (πχ αποθήκευση ενδιάμεσων αποτελεσμάτων)- η υλοποίηση αυτή δεν θα μπορούσε να εγκατασταθεί σε συστήματα με περιορισμένους πόρους όπως πχ έναν προσωπικό υπολογιστή.

Γι αυτό και χρειάστηκε αντικατάσταση της, για την οποία δοκιμάστηκε αρχικά το πλαίσιο βιβλιοθηκών Jena και έπειτα το ΣΔΒΔ OpenLink Virtuoso (το οποίο και αποτέλεσε την λύση που εφαρμόστηκε τελικά και η οποία αναλύεται σε επόμενο κεφάλαιο).

# Κεφάλαιο 4

## Υλοποίηση υποδομής JENA Semantic Web με χρήση πλαισίου OSGI



### 4.1 Λόγος επιλογής JENA ως πρώτης επιλογής προς αντικατάσταση του Oracle Server Semantic Technology

Το κύριο πρόβλημα με την υλοποίηση του εξυπηρετητή για RDF Web Mashups με χρήση του εξυπηρετητή της Oracle ήταν ότι ο εξυπηρετητής αυτός αποτελούσε ένα «βαρύ», με μεγάλες απαιτήσεις σε μνήμη σύστημα διαχείρησης βάσεων δεδομένων. Απαιτούσε 1GB κυρίως μνήμη απλά και μόνο για την λειτουργία του, το οποίο σήμαινε ότι δεν ήταν κατάλληλος για χρήση ως το κατώτατο επίπεδο (στρώμα αποθήκευσης και επεξεργασίας επερωτήσεων) μιας υλοποίησης τριών επιπέδων του συστήματος η οποία να τρέχει σε μια συνηθισμένη μηχανή με περιορισμένους πόρους.

Επομένως ήταν ανάγκη να αντικαταστήσουμε τον εξυπηρετητή της Oracle με ένα πιο «ελαφρύ» σύστημα και η υποδομή της JENA (με χρήση βιβλιοθηκών Java) φαινόταν ως κατάλληλη. Η JENA, αφού δεν αποτελεί ένα πλήρες σύστημα διαχείρησης βάσεων δεδομένων μα ένα εξειδικευμένο πλαίσιο προορισμένο για επεξεργασία-εκτέλεση επερωτήσεων σε RDF μεταδεδομένα και την μόνιμη αποθήκευση αυτών [14], έχει πολύ μικρότερες ανάγκες σε μνήμη και μεγαλύτερη ευελιξία για χρήση σε

«ελαφρύτερα» συστήματα αφού οι παρεχόμενες λειτουργίες ήταν μόνο ότι χρειαζόμαστε.

### 4.2 Λόγοι για χρήση OSGI για υλοποίηση/ενσωμάτωση

Λόγω της χρήσης της έννοιας της υπηρεσίας (service) αντί των τυπικών αρχείων JAR το πλαίσιο OSGI προσφέρει πολλά πλεονεκτήματα τα οποία αναλύονται σε προηγούμενα κεφάλαια τα οποία αναφέρονται στις τεχνολογίες που έχουν χρησιμοποιηθεί. Κάποια από τα οποία είναι μειωμένη πολυπλοκότητα, ευκολία επαναχρησιμοποίησης, μεγαλύτερη ευελιξία σε δυναμικά περιβάλλοντα, ευκολία εγκατάστασης και δυναμικής αναβάθμισης και ευκολότερος έλεγχος εκδόσεων [1].

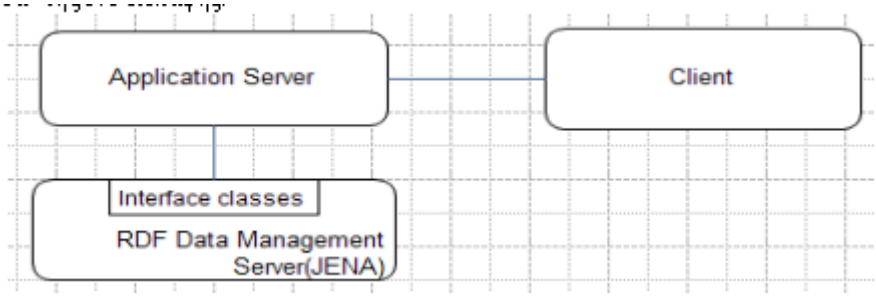
### 4.3 Γενικά

Το πλαίσιο JENA αποτελείται από πολλά μέρη με πολλές αλληλοεξαρτήσεις (όπως περιγράφεται στην ανάλυση τεχνολογιών) τα οποία έπρεπε να ενσωματωθούν στο πλαίσιο OSGI λαμβάνοντας υπόψη αυτές τις αλληλοεξαρτήσεις.

### 4.4 Αρχιτεκτονική συστήματος

Η υλοποίηση της JENA στο OSGI σχεδιάστηκε ως το κατώτατο επίπεδο (επίπεδο διαχείρησης δεδομένων) της αρχιτεκτονικής τριών επιπέδων που χρησιμοποιείται από το σύστημα πελάτη-εξυπηρετητή για RDF επερωτήσεις και mashups, το οποίο σύστημα είχε υλοποιηθεί από τον κο Κωνσταντίνο Σαββίδη. Στόχος της ήταν να αντικαταστήσει το DBMS Oracle το οποίο χρησιμοποιόταν για αυτό τον σκοπό [28].

Για να διευκολυνθεί η ομαλή διεπαφή του εξυπηρετητή εφαρμογών με την JENA δημιούργησα ένα ενδιάμεσο επίπεδο το οποίο αναλαμβάνει ακριβώς τον ρόλο αυτής της διεπαφής. Σε αυτή την υλοποίηση η επικοινωνία μεταξύ των τριών επιπέδων του συστήματος γινόταν με χρήση αιτήσεων μέσω του πρωτοκόλλου HTTP.



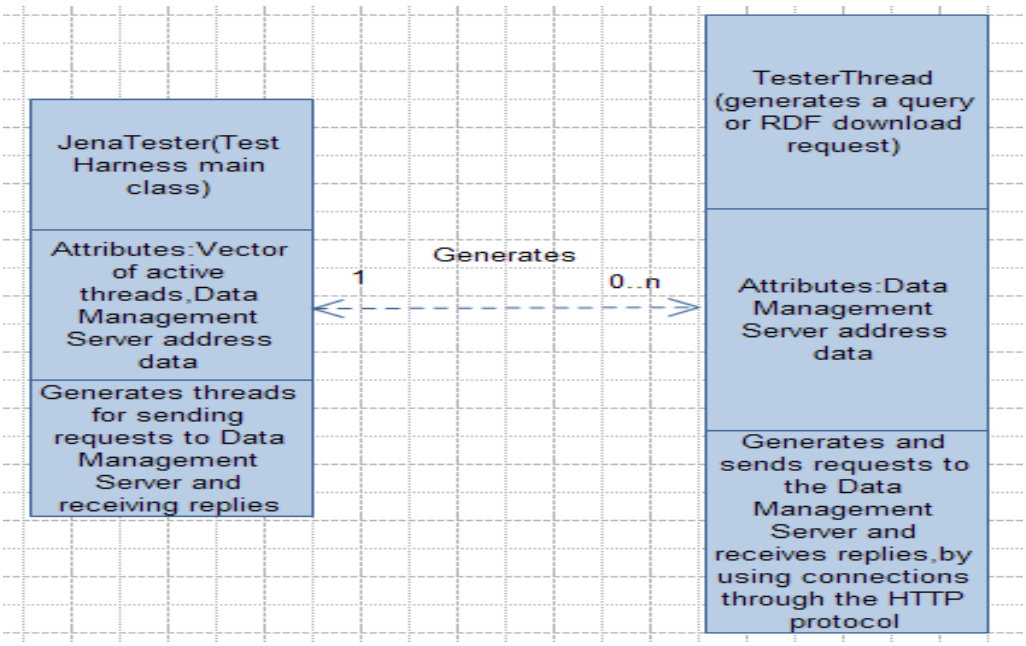
Σχήμα 4.1 Διασύνδεση μεταξύ συστατικών μερών συστήματος

Πελάτης-εξυπηρετητής εφαρμογών

Αποτελούν τα δύο ανώτερα (όσο αφορά απόσταση από την διεπαφή χρήστη) επίπεδα του συστήματος. Δεν έγινε οποιαδήποτε αλλαγή στα δύο αυτά επίπεδα κατά την διάρκεια της υλοποίησης της JENA στο περιβάλλον OSGI και γι' αυτό δεν περιγράφονται εδώ στην τελική τους μορφή αλλά παρακάτω, στην περιγραφή της τελικής υλοποίησης του συστήματος με χρήση του DBMS Virtuoso.

Για την δοκιμή των κατώτερων αυτού επιπέδων (εξυπηρετητής διαχείρησης δεδομένων, ενδιάμεσο επίπεδο διεπαφής) χρησιμοποίησα ένα test harness με κλάσεις Java (το οποίο και είχε προσωρινά τον ρόλο του εξυπηρετητή εφαρμογών). Αυτό ουσιαστικά αποτελείται από δύο κλάσεις στο ενδιάμεσο επίπεδο που επικοινωνεί με τον εξυπηρετητή διαχείρισης δεδομένων οι οποίες απλά έχουν ως ρόλο να δημιουργούν και να αποστέλλουν ανά τακτά χρονικά διαστήματα αιτήσεις είτε για καταφόρτωση συνόλων RDF μεταδεδομένων είτε για επερωτήσεις σε ήδη καταφορτωμένα σύνολα.

Αυτό υλοποιείται με πολυνηματική υλοποίηση η οποία δημιουργεί ένα νήμα για κάθε διαφορετική αίτηση, το οποίο αποστέλλει την αίτηση ανά τακτά χρονικά διαστήματα.



Σχήμα 4.2 Σχεδιάγραμμα που δείχνει την αλληλεξάρτηση μεταξύ των κλάσεων του test harness.

Οι επερωτήσεις σε RDF μεταδεδομένα ακολουθούν το πρότυπο της γλώσσας επερωτήσεων SPARQL:

F:\\WORKSPACE_DIPL8\\TDBData\\tdb-1| SELECT  ?bo

WHERE {

?a <http://www.snee.com/ns/epinfluences> ?c ;

?bp ?bo .

FILTER regex(?bo, \"LOOKING\")

}

Εδώ η διεύθυνση πριν το SELECT αντιπροσωπεύει το μονοπάτι προς την διεύθυνση του μοντέλου προς το οποίο ο πελάτης προωθεί την επερώτηση σε SPARQL που ακολουθεί. Είναι η διεύθυνση στην οποία ο εξυπηρετητής διαχείρησης δεδομένων είχε αποθηκεύσει το σύνολο μεταδεδομένων για το οποίο του είχε προηγουμένως στείλει εντολή ο εξυπηρετητής εφαρμογών (εδώ το test harness) να καταφορτώσει και η οποία διεύθυνση είχε επιστραφεί στον πελάτη ως απάντηση στο αίτημα για καταφόρτωση του εν λόγω πόρου RDF.

Έπειτα ακολουθεί η επερώτηση SPARQL (εδώ η επερώτηση ζητά να επιλεγεί και επιστραφεί κάθε τιμή αντικειμένου/χαρακτηριστικού με το οποίο συνδέεται οποιοσδήποτε κόμβος-υποκείμενο ο οποίος συνδέεται μέσω της ιδιότητας http://www.snee.com/ns/epinfluences με κάποιον κόμβο και η οποία τιμή προς επιστροφή περιέχει την λέξη «LOOKING»).

Αφού επιστραφούν στο test harness τα αποτελέσματα, αυτό τα παρουσιάζει στην έξοδο.

Οι αιτήσεις για καταφόρτωση συνόλων RDF μεταδεδομένων γίνονται από το test harness με αποστολή προς το ενδιάμεσο επίπεδο του URL του ζητούμενου αρχείου.

Είναι υπεύθυνος για:

1) Καταφόρτωση δεδομένων RDF από το Διαδίκτυο.

Μπορούν να καλεστούν μεθόδοι των κλάσεων της βιβλιοθήκης com.hp.hpl.jena.rdf.Model (βιβλιοθήκη υπεύθυνη για την δημιουργία και διαχείρηση μοντέλων που αντιπροσωπεύουν σύνολα από RDF μεταδεδομένα και την διαχείρηση αυτών) για γέμισμα των μοντέλων που αντιπροσωπεύουν αυτά τα μεταδεδομένα, αφού προηγουμένως δημιουργηθούν τα μοντέλα αυτά.

Παρ'όλα αυτά προτιμήθηκε να μην καταφορτώνονται τα μεταδεδομένα απευθείας από το Διαδίκτυο πριν να εισάγονται στα μοντέλα, αφού έτσι δεν δίνεται δυνατότητα εισαγωγής μεταδεδομένων που βρίσκονται αποθηκευμένα στο Διαδίκτυο σε συμπιεσμένη μορφή (όπως .gz, .tar, .bz2) -κάτι που συμβαίνει πολύ συχνά για μεγάλα σύνολα μεταδεδομένων RDF αποθηκευμένα στο Διαδίκτυο. Γι αυτό εισάγονται τα μεταδεδομένα στα μοντέλα της JENA (τα οποία χρησιμοποιούν το υποσύστημα μόνιμης αποθήκευσης TDB) αφού καταφορτωθούν και αποσυμπιεστούν (σε προσωρινά αρχεία) αν χρειάζεται στο επίπεδο της διασύνδεσης μεταξύ εξυπηρετητή εφαρμογών και εξυπηρετητή διαχείρισης δεδομένων.

2) Μόνιμη αποθήκευση μεταδεδομένων (μεταδεδομένων RDF υπό μορφή μοντέλων, σε αρχεία με εξειδικευμένη δομή)

Όταν καταφορτωθούν τα μεταδεδομένα και κληθούν οι μεθόδοι της διεπαφής της JENA που είναι υπεύθυνες για την εισαγωγή των μεταδεδομένων στα μοντέλα η υλοποίηση των interfaces των μοντέλων η οποία εμπεριέχεται στις βιβλιοθήκες του υποσυστήματος μόνιμης αποθήκευσης TDB αυτόματα δημιουργεί αρχεία μόνιμης αποθήκευσης στον εισαγόμενο κατάλογο στο σύστημα αρχείων του εξυπηρετητή διαχείρησης δεδομένων.

3)Ανάκτηση μεταδεδομένων και εκτέλεση επερωτήσεων

Η ανάκτηση γίνεται με χρήση των κλάσεων Query, QueryFactory και QueryExecutionFactory οι οποίες είναι υπεύθυνες για την δημιουργία και εκτέλεση επερωτήσεων πάνω σε υπάρχοντα μοντέλα που ήδη περιέχουν δεδομένα και την

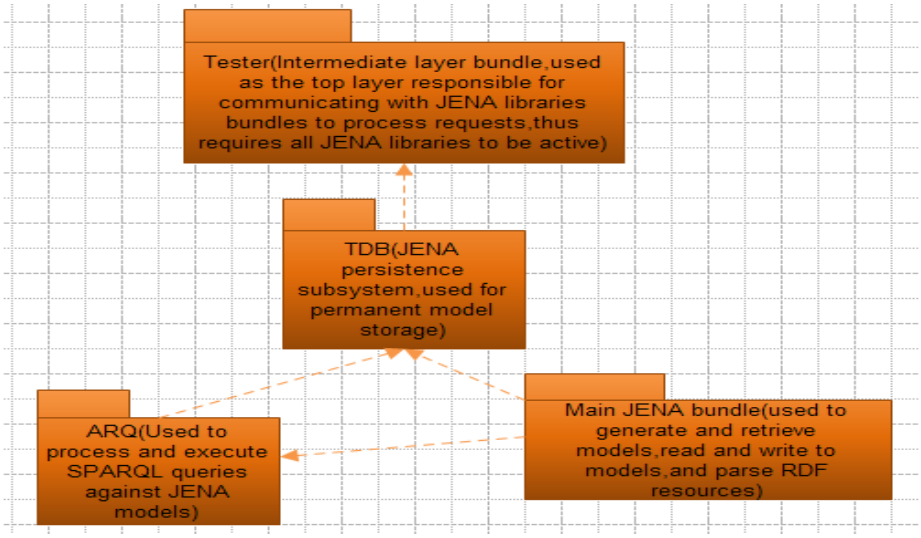
επιστροφή των αποτελεσμάτων ως σύνολα, τα οποία σύνολα με χρήση της κλάσης ResultSetFormatter μετατρέπονται σε κείμενο.

Ενδιάμεσο επίπεδο διασύνδεσης μεταξύ εξυπηρετητή εφαρμογών -εξυπηρετητή διαχείρισης δεδομένων

Χρησιμοποιείται για την επικοινωνία μεταξύ των δύο αυτών επιπέδων.

Είναι υλοποιημένος ως κλάσεις Java, συγκεκριμένα ως ένα σύστημα πολυνηματικής εκτέλεσης το οποίο είναι επίσης ενσωματωμένο στο πλαίσιο OSGI όπως ακριβώς και οι βιβλιοθήκες της JENA.

Το πακέτο που περιέχει τις κλάσεις του ενδ. επιπέδου περιέχει την κυρίως κλάση του ενδιάμεσου επιπέδου η οποία εκτός από υπεύθυνη για την παρακολούθηση των ports του εξυπηρετητή, την ανίχνευση εισερχόμενων αιτήσεων και την δημιουργία νημάτων για διαχείριση αυτών των αιτήσεων και την επιστροφή των αποτελεσμάτων αποτελεί επίσης τον ενεργοποιητή του ενδιάμεσου επιπέδου στο OSGI. Αποτελεί το «σημείο εισόδου» το οποίο όταν ενεργοποιηθεί προκαλεί την ενεργοποίηση του bundle του ενδιάμεσου επιπέδου το οποίο έπειτα θα χρησιμοποιήσει τα (ενεργοποιημένα εξαρχής) bundles τα οποία περιέχουν τις βιβλιοθήκες της JENA (μέσω των υπηρεσιών που αυτά προσφέρουν) για να τρέξει τον κώδικα του. Ο ενεργοποιητής αυτός ενεργοποιείται αυτόματα όταν εκκινήσει το πλαίσιο OSGI αφού έχουν ήδη ενεργοποιηθεί τα bundles από τα οποία εξαρτάται, δηλαδή όταν ενεργοποιηθεί το bundle της JENA με τις κυρίως κλάσεις, το υποσύστημα επεξεργασίας επερωτήσεων SPARQL ARQ καθώς και το υποσύστημα ελέγχου μόνιμης αποθήκευσης μοντέλων TDB.



Σχήμα 4.3 Σχεδιάγραμμα που δείχνει τις εξαρτήσεις (στα υψηλότερα επίπεδα) μεταξύ των bundles της JENA ( εννοείται ότι το κάθε ένα από αυτά έχει και εξαρτήσεις από κάποιες μικρότερες επιμέρους βιβλιοθήκες οι οποίες και προστίθενται στατικά).

Όταν λαμβάνεται αίτηση η κυρίως κλάση δημιουργεί ένα νήμα το οποίο είναι υπεύθυνο για την επεξεργασία της παραλαμβανόμενης αίτησης.

Το νήμα αυτό κατ'αρχάς ελέγχει αν πρόκειται για αίτηση για καταφόρτωση πόρου RDF από URL ή για αίτηση για εκτέλεση επερώτησης σε RDF μεταδεδομένα ήδη καταφορτωμένα.

Στην πρώτη περίπτωση καλείται μέθοδος η οποία είναι υπεύθυνη για την καταφόρτωση των μεταδεδομένων από τον πόρο στο εισαγόμενο URL και την αποθήκευση τους σε ένα προσωρινό αρχείο. Έπειτα καλείται μέθοδος η οποία χρησιμοποιεί κλήση συστήματος (syscall) για εκτέλεση ενός προγράμματος το οποίο καταφόρτωσα από το Διαδίκτυο, το rdf2rdf, το οποίο μπορεί να αποσυμπιέσει συμπιεσμένα αρχεία RDF καθώς και να μετατρέψει μία μορφή μεταδεδομένων RDF σε άλλη [25]. Αυτή η κλήση χρειάζεται για δύο λόγους:

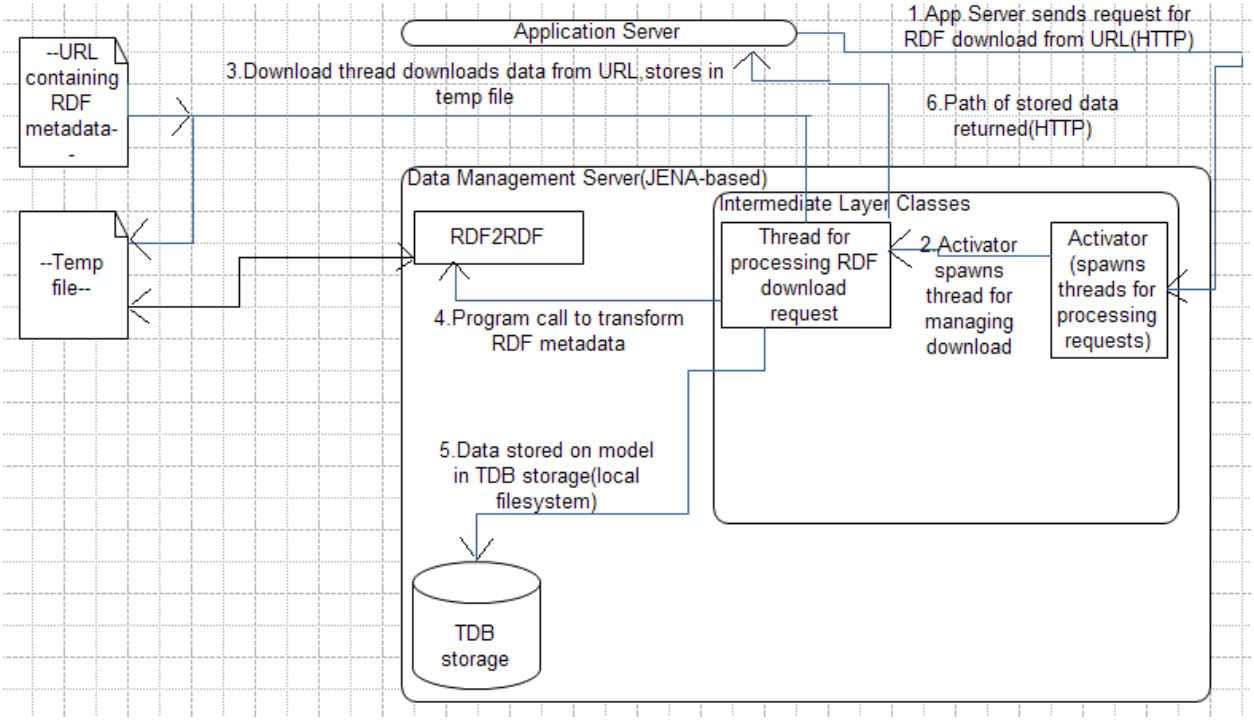
1) Αποσυμπίεση καταφορτωμένων αρχείων RDF τα οποία είναι σε συμπιεσμένη μορφή.

2) Μετατροπή των μεταδεδομένων σε μορφή .rdf αφού δεν μπορεί να χρησιμοποιηθεί στην πράξη η JENA στο OSGI για parsing και εισαγωγή μεταδεδομένων σε μοντέλα τα οποία βρίσκονται σε άλλη μορφή και μετέπειτα επερωτήσεις SPARQL σε αυτά τα μοντέλα (πχ n-triples). Αυτό οφείλεται στο ότι υπάρχει κυκλική εξάρτηση μεταξύ μιας κλάσης του υποσυστήματος ARQ για ερμηνεία και εκτέλεση επερωτήσεων SPARQL η οποία είναι υπεύθυνη για πράξεις σε μοντέλα και της κυρίως βιβλιοθήκης της JENA η οποία είναι υπεύθυνη για την δημιουργία των μοντέλων και την εισαγωγή δεδομένων σε αυτά σε μορφές άλλες πλην .rdf. Αυτό, παρόλο που στην υλοποίηση με JARs δεν δημιουργούσε προβλήματα, στην υλοποίηση σε OSGI (όπου για να

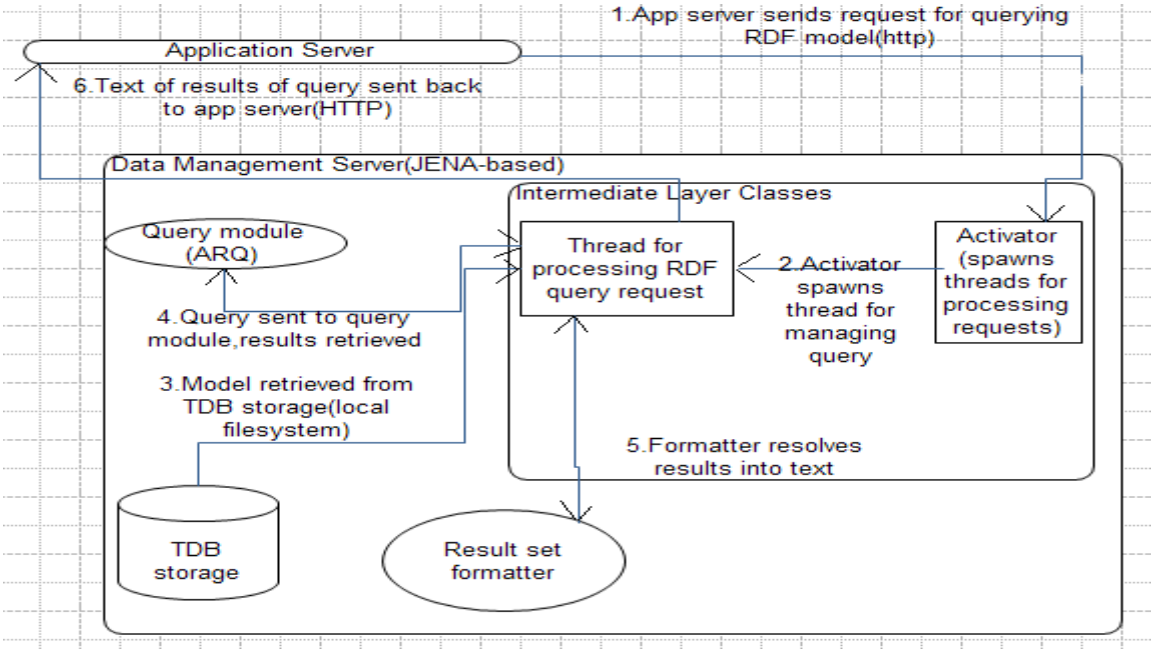
ενεργοποιηθεί ένα bundle πρέπει πρώτα να ενεργοποιηθούν οι εξαρτήσεις του) οδηγεί σε αδιέξοδο και έτσι ήμουν αναγκασμένος να αφαιρέσω την επίμαχη κλάση της κυρίως βιβλιοθήκης της JENA και άρα να μετατρέπω τα μεταδεδομένα σε μορφή .rdf.

Έπειτα (συνέχεια της περίπτωσης αίτησης καταφόρτωσης μεταδεδομένων) δημιουργείται ένα νέο μοντέλο με χρήση της υλοποίησης του interface του μοντέλου στη βιβλιοθήκη TDB (ώστε να έχουμε μοντέλο που να χρησιμοποιεί το σύστημα αρχείων για μόνιμη αποθήκευση των μεταδεδομένων) και προστίθεται το νέο μοντέλο σε ένα αρχείο το οποίο αντιστοιχίζει τα μοντέλα (περιγραφόμενα από το URL τους) με το μονοπάτι στο σύστημα αρχείων στο οποίο το TDB έχει αποθηκεύσει το μοντέλο. Το μονοπάτι αυτό επιστρέφεται στον εξυπηρετητή εφαρμογών.

Στην περίπτωση επερώτησης σε ήδη καταφορτωμένο σύνολο δεδομένων RDF απλά δημιουργείται (με χρήση των βιβλιοθηκών ARQ της JENA) επερώτηση προς εκτέλεση, τα αποτελέσματα της οποίας επιστρέφονται στον εξυπηρετητή εφαρμογών αφού διαμορφωθούν ως κείμενο.




Σχήμα 4.4 Σχεδιάγραμμα το οποίο δείχνει την ολική σύνθεση του ενδιάμεσου επιπέδου μαζί με τα συστήματα των βιβλιοθηκών της JENA καθώς και την σειρά των δράσεων που ακολουθούνται για καταφόρτωση ενός συνόλου δεδομένων RDF.




Σχήμα 4.5 Σχεδιάγραμμα το οποίο δείχνει την ολική σύνθεση του ενδιάμεσου επιπέδου μαζί με τα συστήματα των βιβιοθηκών της JENA καθώς και την σειρά των δράσεων που ακολουθούνται για εκτέλεση επερώτησης σε ένα σύνολο δεδομένων RDF.

### 4.5.Δοκιμές αποδοτικότητας-κλιμακωσιμότητας υλοποίησης

Στόχος αυτών των δοκιμών ήταν να διαπιστωθεί σε ποιο βαθμό ο εξυπηρετητής διαχείρισης μεταδεδομένων RDF βασισμένος στην υλοποίηση της JENA στο OSGI μπορούσε να ανταποκριθεί σε ικανοποιητικό χρόνο στις αιτήσεις καταφόρτωσης συνόλων RDF μεταδεδομένων και εκτέλεσης επερωτήσεων πάνω σε αυτά καθώς το πλήθος ταυτόχρονων αιτήσεων προς εξυπηρέτηση και το μέγεθος των μεταδεδομένων πολλαπλασιάζεται. Γι' αυτό τον σκοπό χρησιμοποιήθηκαν διαφορετικές επαναλαμβανόμενες επερωτήσεις προς την βάση ανά τακτά χρονικά διαστήματα.

Μέθοδος αξιολόγησης

Χρησιμοποίησα τρία σύνολα δεδομένων από την DBPedia σε μορφή n-triples, τα οποία αποτελούν μερικές αποθήκες περιεχομένου της Wikipedia.

Προσπάθησα να χρησιμοποιήσω σύνολα δεδομένων με τριάδες στην ίδια μορφή καθώς έτσι αφαιρείται ένας παράγοντας που επηρεάζει την απόδοση των επερωτήσεων, ο

οποίος είναι η πολυπλοκότητα των συνόλων δεδομένων. Αυτό θα διευκόλυνε επίσης την δημιουργία παρόμοιων -αν όχι ταυτόσημων- σειρών δοκιμαστικών επερωτήσεων παρόμοιας πολυπλοκότητας για όλα τα εμπλεκόμενα σύνολα δεδομένων, έτσι επίσης αφαιρώντας τον παράγοντα της σχετικής πολυπλοκότητας των επερωτήσεων. Έτσι έγινε πιο εύκολο να αξιολογήσω την επίδραση του μεγέθους των συνόλων δεδομένων (όσο αφορά τόσο καθαρό μέγεθος, όσο και τον αριθμό των τριάδων) στην απόδοση για τις επερωτήσεις αλλά και για την καταφόρτωση των δεδομένων και την εισαγωγή στο υποσύστημα μόνιμης αποθήκευσης TDB. Όπως επίσης και την επίδραση του πλήθους των επερωτήσεων που παραλαμβάνονται από τον εξυπηρετητή διαχείρισης δεδομένων ανά δευτερόλεπτο στην απόδοση.

Τα χρησιμοποιημένα σύνολα δεδομένων ήταν τα ακόλουθα:

| **Σύνολο δεδομένων** | **Πλήθος τριάδων(χιλιάδες)** | **Μέγεθος(KB)** |
|---|---|---|
| 1)Short abstracts(Swedish) | 213.2 | 96 124(~96MB) |
| 1)Short abstracts(French) | 544.1 | 212 255(~212 MB) |
| 3)Simple Knowledge Organization System | 2 200 | 369 955(~369 MB) |

Πίνακας 4.1 Σύνολα δεδομένων που χρησιμοποιήθηκαν για τις δοκιμές

Αξίζει να αναφερθεί ότι ακόμη κι αν το τρίτο σύνολο δεδομένων περιείχε 1.5 φορές το μέγεθος δεδομένων του δεύτερου περιείχε περισσότερο από τετραπλάσιο αριθμό τριάδων και άρα αποτελούσε πολύ μεγαλύτερη πρόκληση.

Αρχικά δοκιμάστηκε ο χρόνος που χρειάστηκε για την καταφόρτωση κάθε συνόλου δεδομένων και την εισαγωγή του στο υποσύστημα TDB στην σχετική διεύθυνση.Αυτός ο χρόνος περιελάμβανε  τον χρόνο που χρειαζόταν για

1) Την καταφόρτωση του περιεχομένου των αρχείων .nt από το Διαδίκτυο και την εισαγωγή τους σε ένα προσωρινό αρχείο.

2) Την χρήση του προγράμματος μετατροπής οποιασδήποτε μορφής RDF μεταδεδομένων στη μορφή .rdf.

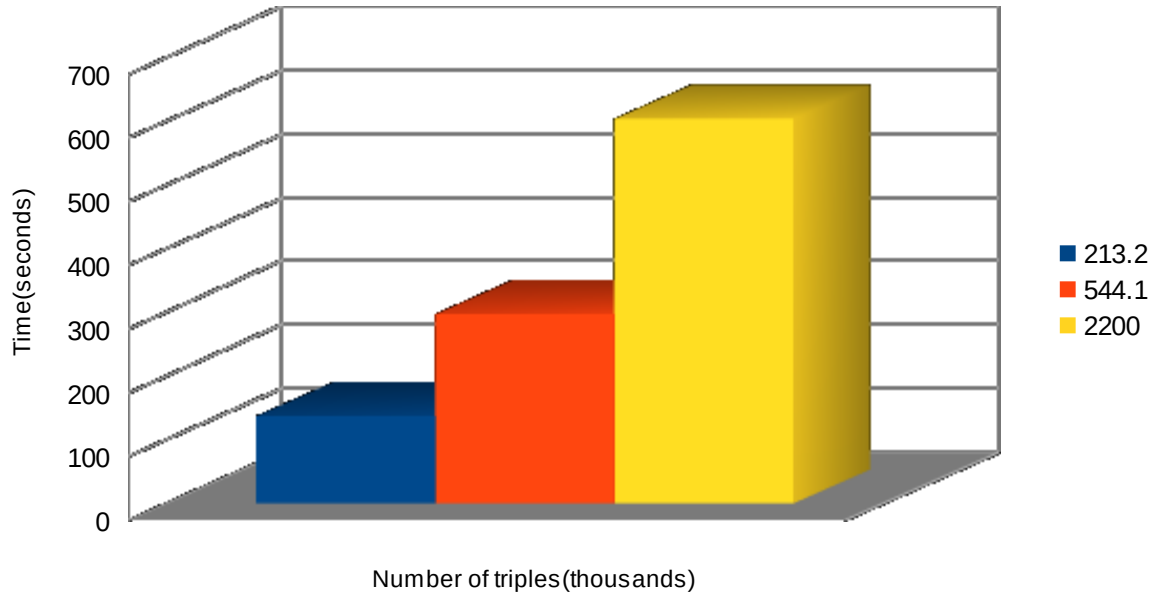
3) Την εισαγωγή των δεδομένων RDF από το προσωρινό αρχείο .rdf σε μια διεύθυνση σχετιζόμενη με το υποσύστημα μόνιμης αποθήκευσης TDB.

Έπειτα σειρές επερωτήσεων στάληκαν για το καθένα από τα τρία σύνολα δεδομένων.

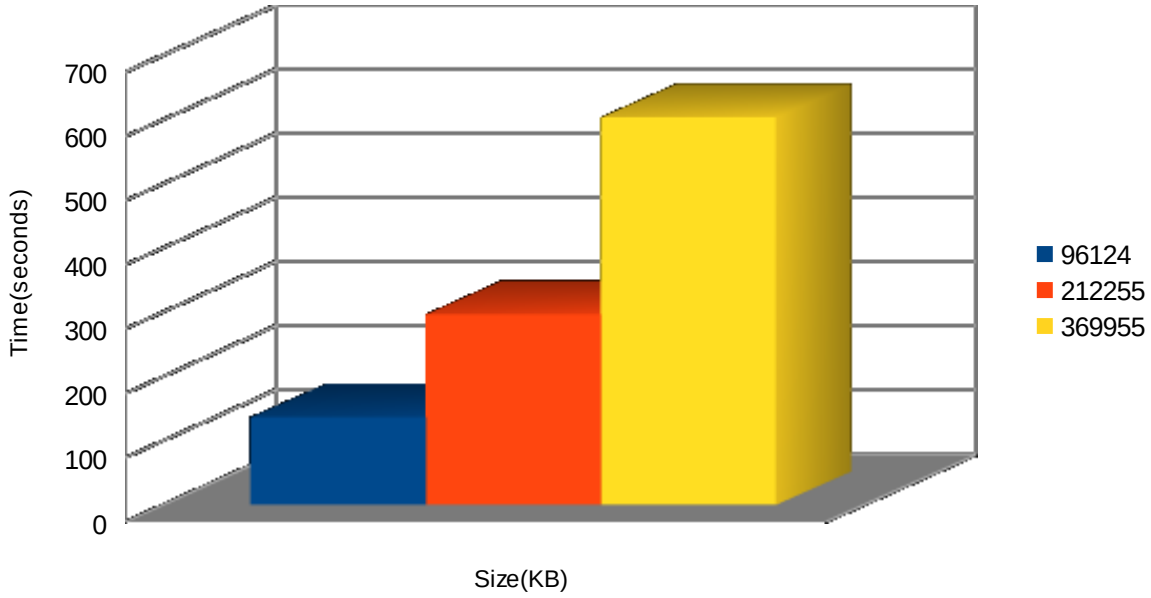
Για κάθε σύνολο δεδομένων στάληκαν επαναλήψεις της ίδιας αίτησης 1, 2, 4, 8 και 16 φορές ανά δευτερόλεπτο έχοντας συνολικά 3, 6, 12, 24 και 48 επερωτήσεις αντίστοιχα για κάθε επερώτηση ξεχωριστά, ώστε να μετρήσω την απόδοση της υλοποίησης για κάθε επερώτηση ξεχωριστά για κάθε ρυθμό παραλαμβανόμενων επερωτήσεων.

Αποδοτικότητα εισαγωγής περιεχομένου

Κατά μέσο όρο παρατηρήθηκαν οι ακόλουθοι χρόνοι.




Σχήμα 4.6 Χρόνος που απαιτήθηκε για την καταφόρτωση κάθε συνόλου δεδομένων συναρτήσει του πλήθους των τριάδων κάθε συνόλου δεδομένων




Σχήμα 4.7 Χρόνος που απαιτήθηκε για την καταφόρτωση κάθε συνόλου δεδομένων συναρτήσει του μεγέθους κάθε συνόλου δεδομένων

Όπως δείχνουν τα αποτελέσματα, για σύνολα δεδομένων με μέγεθος μέχρι 370MB και 2.2 εκατομμύρια τριάδες το σύστημα είναι κλιμακώσιμο, με τον χρόνο για καταφόρτωση και εισαγωγή να είναι σχεδόν ανάλογος με το πλήθος των τριάδων και το μέγεθος του συνόλου δεδομένων αν η αναλογία μεταξύ του μέσου μεγέθους δεδομένων κάθε τριάδας για όλα τα σύνολα δεδομένων είναι σχεδόν η ίδια.Αυτό φαίνεται από τον λόγο του χρόνου που χρειάστηκε για το δεύτερο σύνολο δεδομένων σε σχέση με τον χρόνο που χρειάστηκε για το πρώτο λαμβάνοντας υπόψη ότι ο λόγος τόσο του πλήθους των τριάδων όσο και του συνολικού μεγέθους δεδομένων μεταξύ του δεύτερου και του πρώτου συνόλου δεδομένων κινούνται στα ίδια επίπεδα.

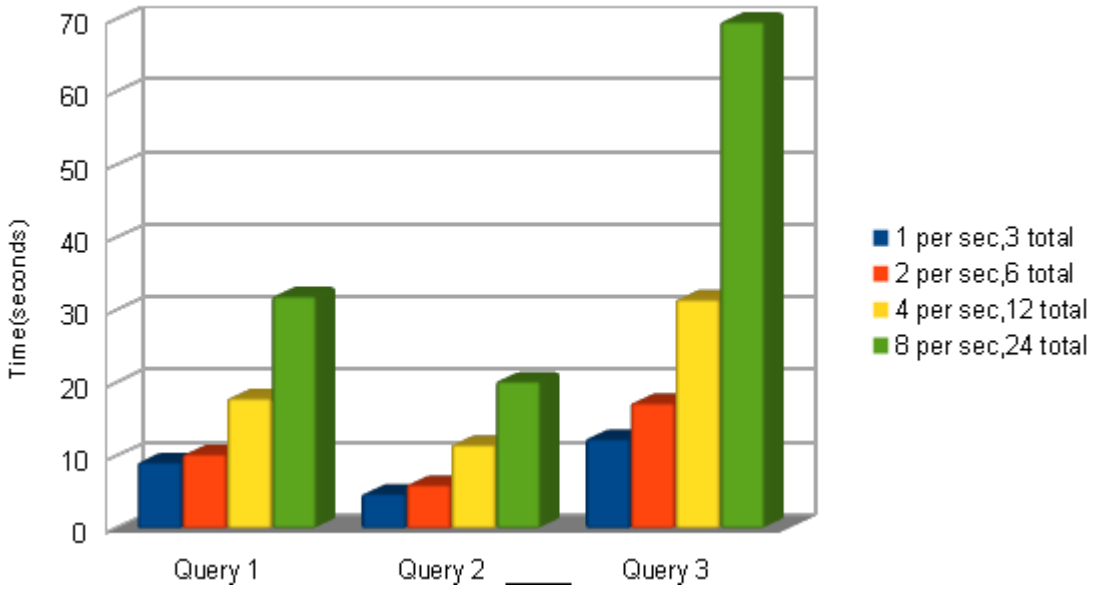
Όμως, για το τρίτο σύνολο δεδομένων ο λόγος του χρόνου που χρειάστηκε σε σχέση με τον χρόνο που χρειάστηκε για το δεύτερο σύνολο δεδομένων ήταν πολύ μεγαλύτερος από τον λόγο του μεγέθους του τρίτου συνόλου δεδομένων σε σχέση με το δεύτερο σύνολο δεδομένων. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι το τρίτο σύνολο δεδομένων περιείχε περισσότερο από τέσσερις φορές των αριθμό των τριάδων του δεύτερου συνόλου δεδομένων, κάτι που συνεπάγεται πολύ περισσότερο χρόνο για

δημιουργία ευρετηρίων και άλλων δομών δεδομένων προς χρησιμοποίηση σχετικά με το μοντέλο του συνόλου στο TDB.

Αποδοτικότητα επερωτήσεων

(συναρτήσει ρυθμού παραλαμβανομένων επερωτήσεων, για κάθε επερώτηση, για κάθε σύνολο δεδομένων)

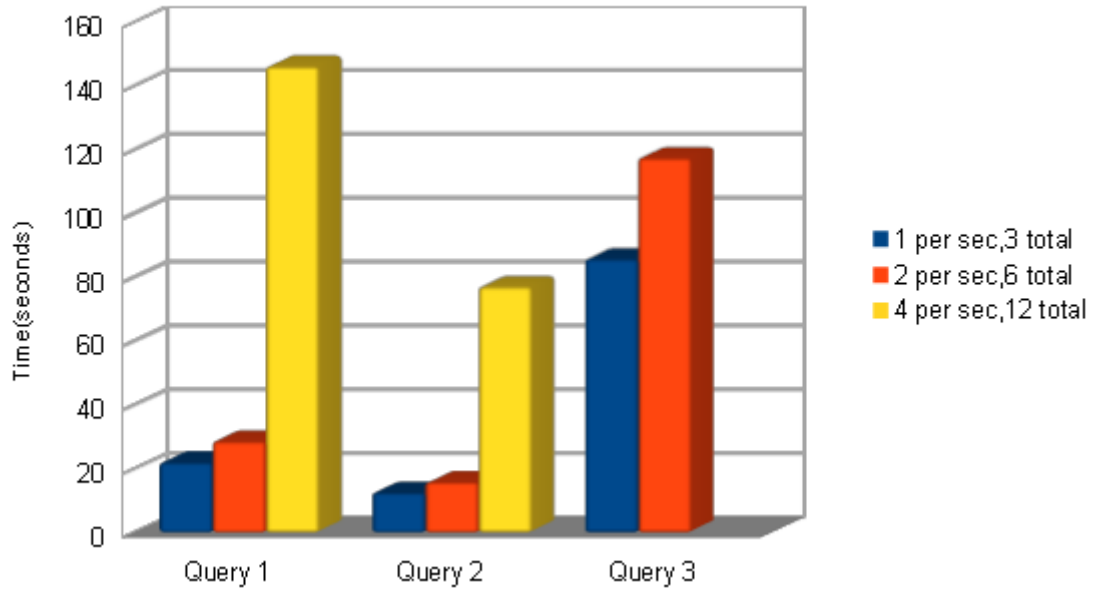
Σύνολο δεδομένων 1 (Short abstracts (Swedish), 213.2 χιλιάδες τριάδες, 96 124 KB (~96MB) )




Σχήμα 4.8 Απαιτούμενος χρόνος για ολοκλήρωση επερωτήσεων για το σύνολο δεδομένων 1 για κάθε εξεταζόμενο ρυθμό παραλαβής αιτήσεων.

Η δοκιμή για 16 επερωτήσεις ανά δευτερόλεπτο σε σύνολο 48 επερωτήσεων απέτυχε να ολοκληρώσει και στις 3 επερωτήσεις.

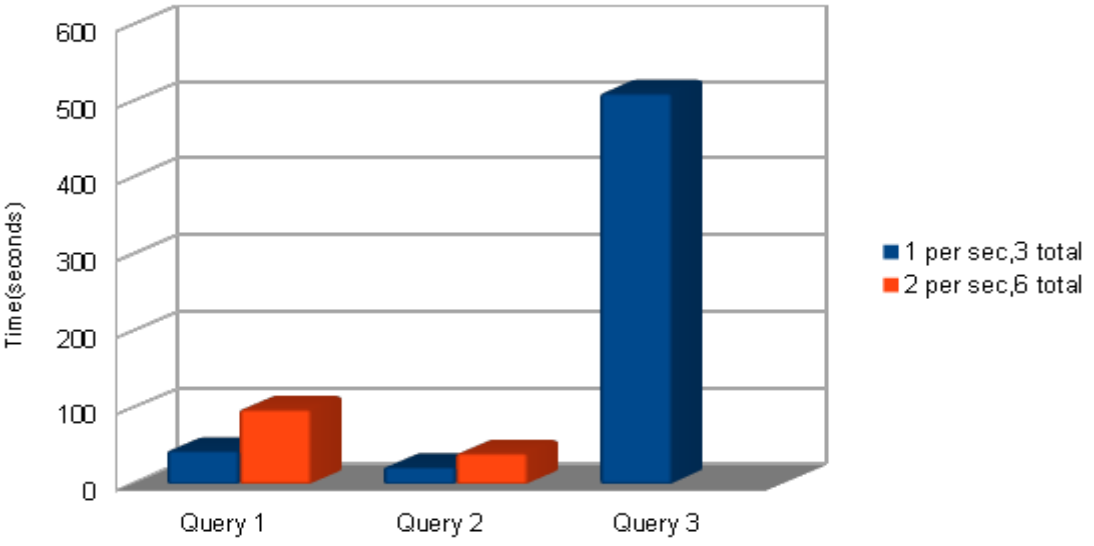
Σύνολο δεδομένων 2 (Short abstracts (French), 544.1 χιλιάδες τριάδες, 212 255 KB (~212 MB) )




Σχήμα 4.9 Απαιτούμενος χρόνος για ολοκλήρωση επερωτήσεων για το σύνολο δεδομένων 2 για κάθε εξεταζόμενο ρυθμό παραλαβής αιτήσεων.

Οι δοκιμές για 16 επερωτήσεις ανά δευτερόλεπτο σε σύνολο 48 επερωτήσεων, καθώς και για 8 σε σύνολο 24 δεν έγιναν, καθώς για 4 σε σύνολο 12 τα αποτελέσματα ήταν οριακά (σχεδόν αποτυχία). Επίσης για την 3η επερώτηση, για 4 επερωτήσεις ανά δευτερόλεπτο σε σύνολο 12 επερωτήσεων παρατηρήθηκε αποτυχία.

Σύνολο δεδομένων 3 (Simple Knowledge Organization System, 2 200 χιλιάδες τριάδες, 369 955 KB (~369 MB)



Σχήμα 4.10 Απαιτούμενος χρόνος για ολοκλήρωση επερωτήσεων για το σύνολο δεδομένων 3 για κάθε εξεταζόμενο ρυθμό παραλαβής αιτήσεων.

Οι δοκιμές για 16 επερωτήσεις ανά δευτερόλεπτο σε σύνολο 48 επερωτήσεων καθώς και για 8 σε σύνολο 24 δεν έγιναν, καθώς για 4 σε σύνολο 12 τα αποτελέσματα ήταν οριακά (σχεδόν αποτυχία) στο προηγούμενο κατά πολύ μικρότερο σύνολο δεδομένων. Επίσης για την $3^{η}$ επερώτηση για 2 επερωτήσεις ανά δευτερόλεπτο σε σύνολο 6 επερωτήσεων παρατηρήθηκε αποτυχία.

Για το πρώτο σύνολο δεδομένων (~96MB και 213.2 χιλιάδες τριάδες) το σύστημα κλιμάκωνε καλά για μέχρι και 8 επερωτήσεις ανά δευτερόλεπτο με μέσο χρόνο επεξεργασίας ανά επερώτηση ο οποίος ήταν σχεδόν ανάλογος του αριθμού των επερωτήσεων που παραλαμβάνονταν ανά δευτερόλεπτο -εκτός μεταξύ της $1^{ης}$ και της $2^{ης}$ επερώτησης, κάτι που μπορεί να εξηγηθεί από τον σταθερό χρόνο που απαιτείται για κάποιες λειτουργίες του συστήματος που είναι ανεξάρτητες από το πλήθος των παραλαμβανομένων επερωτήσεων. Όταν δοκιμάστηκε με 16 επερωτήσεις ανά δευτερόλεπτο το σύστημα τέθηκε εκτός λειτουργίας (καταναλώνοντας περισσότερη μνήμη από τα 1500 MB που ήταν διαθέσιμα στο heap για χρήση από το σύστημα) μετά από 20 λεπτά επεξεργασίας, ακόμη και για την επερώτηση η οποία είχε δώσει τον μικρότερο χρόνο απόκρισης για όλους τους υπόλοιπους ρυθμούς παραλαμβανομένων επερωτήσεων.

Επομένως τα όρια του συστήματος, λαμβάνοντας υπόψη τα όρια της μνήμης η οποία είναι διαθέσιμη σε ένα προσωπικό υπολογιστή (όταν έγινε αυτή η ΑΔΕ) -2GB ολική RAM- είναι κάπου μεταξύ 8 και 16 επερωτήσεων ανά δευτερόλεπτο για σύνολα δεδομένων όμοια του πρώτου στο μέγεθος. Αφού το πρώτο σύνολο δεδομένων είναι και το μικρότερο σε μέγεθος το ίδιο ισχύει και για οποιοδήποτε σύνολο δεδομένων λογικού μεγέθους πέραν των 96 MB.

Για το δεύτερο σύνολο δεδομένων το σύστημα απέδωσε χειρότερα (όπως αναμενόταν, αφού το $2^{ο}$ σύνολο δεδομένων περιείχε περίπου 2.5 φορές το πλήθος τριάδων του $1^{ου}$ και είχε περίπου 2.25 φορές το μέγεθος του όσο αφορά γενικά δεδομένα). Όμως, επίσης έχει δείξει ότι είναι μη κλιμακώσιμο όσο αφορά αύξηση σε μέγεθος δεδομένων

ή πλήθος τριάδων αφού για 4 η περισσότερες αιτήσεις ανά δευτερόλεπτο οι χρόνοι για κάθε επερώτηση πολλαπλασιάζονταν επί παράγοντες πολύ μεγαλύτερους από τον λόγο των πληθών των τριάδων η των μεγεθών των δύο συνόλων δεδομένων, οδηγώντας ακόμη και σε κατάρρευση του συστήματος για επερωτήσεις οι οποίες ήταν έστω και στο ελάχιστο περίπλοκες -όπως ισχύει για την $3^{η}$ επερώτηση. Επίσης, για δύο η περισσότερες επερωτήσεις ανά δευτερόλεπτο η ίδια πολλαπλασιαστική αύξηση παρατηρήθηκε και για την $3^{η}$ επερώτηση, το οποίο μας οδηγεί στο συμπέρασμα ότι ακόμη και δύο ή μία επερώτηση ανά δευτερόλεπτο οδηγούν σε μη αποδεκτούς χρόνους εκτέλεσης για έστω και στο ελάχιστο περίπλοκες επερωτήσεις.

Τέλος, για το $3^{ο}$ σύνολο δεδομένων το σύστημα απέδωσε σε λογικά πλαίσια μόνο για τις πιο απλές επερωτήσεις (όπως την $1^{η}$ και την $2^{η}$) και μόνο για μέχρι 2 επερωτήσεις ανά δευτερόλεπτο.

### 4.6.Συμπεράσματα

Η χρήση του πλαισίου JENA αντί ενός ΣΔΒΔ, ακόμη και με το πλαίσιο ενσωματωμένο στο πλαίσιο OSGI, οδηγεί σε ένα σύστημα μη κλιμακώσιμο για ένα αριθμό επερωτήσεων ανά δευτερόλεπτο μεγαλύτερο του ελάχιστου (δηλαδή για αριθμό επερωτήσεων όπως αναμένεται να μπορεί να διαχειριστεί ένας εξυπηρετητής) η για σύνολα δεδομένων ίσα η μεγαλύτερα από 200 MB. Ακόμη και για επερωτήσεις μέσης πολυπλοκότητας, λαμβάνοντας υπόψη τις δυνατότητες επεξεργαστικής ισχύος και μνήμης ενός τυπικού προσωπικού υπολογιστή. Επομένως, η χρήση της JENA για δημιουργία ενός πιο «ελαφριού» συστήματος -αντί ενός «βαριού» ΣΔΒΔ -για να διευκολύνει λειτουργίες Web Mashups σε «ελαφριά» συστήματα δεν φαίνεται να είναι ικανοποιητική για μεγάλα σύνολα δεδομένων ή χρήση σε εξυπηρετητή.

Επομένως, αποφασίστηκε η έρευνα/σύγκριση για το πιο κατάλληλο ΣΔΒΔ στην κυκλοφορία το οποίο να μπορεί πιθανόν να εξυπηρετήσει τις ανάγκες μας ούτως ώστε να αντικαταστήσει την JENA.

# Κεφάλαιο 5

## Σύγκριση συστημάτων διαχείρισης μεταδεδομένων RDF



### 5.1 Απαιτήσεις από ένα σύστημα διαχείρισης μεταδεδομένων RDF

Ικανότητα διαχείρισης μεγάλων αποθηκών δεδομένων (φόρτωση και εκτέλεση επερωτήσεων από πολλαπλούς πελάτες) σε λογικό χρόνο

Το κύριο πρόβλημα με την JENA ήταν η αδυναμία της όσο αφορά την διαχείριση σεναρίων πολλών πελατών με μέσες η πιο περίπλοκες επερωτήσεις πάνω σε σύνολα δεδομένων τα οποία περιέχουν εκατομμύρια τριάδες. Λόγω αυτού, αυτό είναι το κύριο κριτήριο που χρησιμοποιείται στην σύγκριση μου.

Ευκολία ενσωμάτωσης με το υπάρχον σύστημα (τον εξυπηρετητή εφαρμογών)

Καθώς η εργασία του κου Σαββίδη χρησιμοποιούσε μια αρχιτεκτονική τριών επιπέδων με ένα βαθμό δομικότητας (πελάτης, εξυπηρετητής εφαρμογών, εξυπηρετητής ΒΔ) αυτή η δομικότητα έπρεπε να διατηρηθεί. Επομένως υπάρχει ανάγκη για εύκολη διεπαφή μεταξύ της μονάδας αποθήκευσης και του εξυπηρετητή εφαρμογών, πιθανώς χρησιμοποιώντας μια διεπαφή Java για επικοινωνία με τα JSP του εξυπηρετητή εφαρμογών ως αντικατάσταση μιας ΒΔ με χρήση τεχνολογίας Oracle.

Γι' αυτό και μια μονάδα αποθήκευσης βασισμένη σε Java θα ήταν προτιμότερη οτιδήποτε άλλου καθώς προσφέρει ευκολία ενσωμάτωσης καθώς και την πιθανότητα για αυξημένη δομικότητα και απόδοση με χρήση της υποδομής OSGI. Όμως ΒΔ ή άλλα συστήματα διαχείρισης RDF μεταδεδομένων με καθαρές και εύκολες στη χρήση διεπαφές με την Java μπορούν επίσης να χρησιμοποιηθούν.

### 5.2 Συστήματα διαχείρισης που έχουν εξεταστεί

Α) JENA -αναφέρεται εδώ χάριν σύγκρισης

“A Java framework for building Semantic Web applications” [14].

Η JENA, λόγω της φιλοσοφίας της όσο αφορά την χρήση μόνο της κύριας μνήμης για διαχείριση επερωτήσεων (πχ αποθήκευση ενδιάμεσων αποτελεσμάτων) χαρακτηρίζεται από χαμηλή απόδοση και έλλειψη κλιμακωσιμότητας για μεγάλα σύνολα δεδομένων όταν χρησιμοποιείται σε συστήματα με περιορισμένη μνήμη.

Η JENA χρησιμοποιεί μια προγραμματιστική διεπαφή (API) βασισμένη στην έννοια του Μοντέλου χρησιμοποιώντας μεθόδους Java για φόρτωση RDF μεταδεδομένων από URL στο Διαδίκτυο η τοπικές διευθύνσεις στην κύρια μνήμη και συσχέτιση τους με μοντέλα. Τα μοντέλα αυτά μπορούν έπειτα να επερωτηθούν και να ενημερωθούν μέσω επερωτήσεων SPARQL. Υπάρχουν μηχανισμοί μόνιμης αποθήκευσης οι οποίοι μπορούν να χρησιμοποιηθούν για μόνιμη αποθήκευση και τροποποίηση μοντέλων: Το SDB (χρησιμοποιείται σε συνδυασμό με ένα ΣΔΒΔ που βρίσκεται από κάτω) και το TDB (το οποίο χρησιμοποιεί ένα «ελαφρύ» σύστημα αποθήκευσης βασισμένο στο τοπικό σύστημα αρχείων, το οποίο και χρησιμοποίησα στις δοκιμές στο προηγούμενο κεφάλαιο).

Δυνατότητες

Είναι γραμμένη σε Java, επομένως μπορεί εύκολα να ενσωματωθεί και επίσης να συμπεριληφθεί σε bundles OSGI για καλύτερη απόδοση. Η χρήση του TDB σημαίνει ένα ελαφρύ υποσύστημα μόνιμης αποθήκευσης. Υπάρχει σχετικά ευρεία υποστήριξη από την κοινότητα και εκτενής τεκμηρίωση.

Αδυναμίες

Λόγω της φιλοσοφίας της όσο αφορά χρήση μόνο κύριας μνήμης χαρακτηρίζεται από χαμηλή επίδοση και έλλειψη κλιμακωσιμότητας για μεγάλα σύνολα δεδομένων όταν χρησιμοποιείται σε συστήματα με περιορισμένη μνήμη. Δοκιμές που υπάρχουν στη βιβλιογραφία χρησιμοποιώντας το Berlin SPARQL Benchmark [5] (επικεντρωμένο σε ένα σενάριο ηλεκτρονικού εμπορίου πραγματικού κόσμου) που εμπεριέχεται σε [3], [4] καθώς και το πιο γενικό SP2Bench [29] δείχνουν καθαρά ότι ακόμη και όταν

χρησιμοποιείται σε συστήματα με άφθονους πόρους όσο αφορά μνήμη η JENA είναι κατώτερη σε απόδοση από άλλα συστήματα σε σχεδόν όλα τα εξεταζόμενα σενάρια.

Β) Virtuoso

Το Virtuoso είναι μια υψηλών επιδόσεων αντικειμενο-σχεσιακή ΒΔ SQL [33].

Παρέχει πολλές ενσωματωμένες λειτουργίες, μία εκ των οποίων είναι η δυνατότητα αποθήκευσης και διαχείρισης RDF μεταδεδομένων μέσω μιας αντικειμενο-σχεσιακής ΒΔ. Είναι διαθέσιμο ως λογισμικό ανοικτού κώδικα από τον οργανισμό OpenLink Software.

Δυνατότητες

Χρησιμοποιεί τοπική φυσική μνήμη μόνιμης αποθήκευσης αντί να βασίζεται μόνο στην κυρίως μνήμη επιτρέποντας επεξεργασία δεδομένων πολύ μεγαλύτερου μεγέθους παρά η JENA ή άλλα συστήματα αποθήκευσης που βασίζονται μόνο στην κυρίως μνήμη. Το OpenLink Virtuoso υποστηρίζει SPARQL ενσωματωμένη σε SQL για επερωτήσεις έναντι RDF μεταδεδομένων αποθηκευμένων στη ΒΔ του Virtuoso. Η SPARQL επωφελείται από υποστήριξη σε χαμηλό επίπεδο εντός της ίδιας της μηχανής του Virtuoso, όπως κανόνες μετατροπής τύπων βασισμένοι στην SPARQL και έναν τύπο δεδομένων ειδικά για περιγραφή IRI [33].

Σε δοκιμές πάνω στο testbench SP2Bench -χρησιμοποιώντας σύνολα δεδομένων με μέχρι και 5 εκατομμύρια τριάδες- το Virtuoso επέδειξε πολύ καλύτερη απόδοση όσο αφορά το φόρτωμα του συνόλου δοκιμής αποφεύγοντας την εκθετική αύξηση σχετικά με το μέγεθος η οποία παρουσιάζεται στο Sesame-τον κύριο του αντίπαλο,ο οποίος επίσης δοκιμάζεται στην δοκιμή. Έδωσε πολύ καλύτερους χρόνους, ειδικά για επερωτήσεις οι οποίες χρησιμοποιούν εκφράσεις FILTER -ένα κοινό σενάριο σε καταστάσεις πραγματικού κόσμου- κατά την εκτέλεση των οποίων η χρήση καλά σχεδιασμένων ευρετηρίων μπορεί να βοηθήσει σημαντικά, δίδοντας το πλεονέκτημα στο Virtuoso το οποίο κάνει εκτενή χρήση τους. Το Virtuoso παρουσίασε υψηλότερη μέχρι και σημαντικά (κατά παράγοντα) υψηλότερη επίδοση σε όλα τα σενάρια δοκιμών [29]. Επομένως το Virtuoso έχει δείξει ότι ανταποκρίνεται καλύτερα σε ένα ποικίλο σενάριο το οποίο συμπεριλαμβάνει διάφορα είδη επερωτήσεων. Κάτι το οποίο και αναμένεται από μια γενικού-σκοπού μηχανή επερωτήσεων SPARQL σε RDF

μεταδεδομένα όπως αυτή που ήθελα να αναπτύξω και το οποίο σενάριο συμπεριλαμβάνει σύνολα δεδομένων κάτω των 5 εκατομμυρίων τριάδων,το οποίο είναι στα πλαίσια που αναμένονται από μια μηχανή επερωτήσεων σε RDF η οποία είναι σχεδιασμένη για εγκατάσταση σε συστήματα με σχετικά περιορισμένη επεξεργαστική ισχύ/κυρίως μνήμη.

Σε δοκιμές έναντι στο (βασισμένο σε σενάριο πραγματικού κόσμου) Berlin SPARQL Benchmark για μεγάλα σύνολα δεδομένων (100/200 εκατομμύρια τριάδες) -το οποίο είναι ποικιλοτρόπως διαφορετικό από το SP2Bench, όπως πχ όσο αφορά μεγέθη συνόλων δεδομένων και την ποικιλομορφία των επερωτήσεων, το Virtuoso σε γενικές γραμμές ξεπέρασε τους άλλους δύο υποψηφίους (JENA και BigOWLIM) σε αριθμό διάφορων επερωτήσεων οι οποίες απαντήθηκαν σε χρόνο που είχε τεθεί όταν περίπλοκες επερωτήσεις είχαν συμπεριληφθεί στο σύνολο. Επέδειξε και καλύτερη επίδοση για κάποιες επερωτήσεις και χειρότερη για κάποιες άλλες (σε σύγκριση με την BigOWLIM) όταν οι πιο περίπλοκες επερωτήσεις δεν είχαν συμπεριληφθεί, με το BigOWLIM να έχει ένα καθαρό πλεονέκτημα στην ολική απόδοση του συνόλου των επερωτήσεων για το σύνολο δεδομένων με 100 εκατομμύρια τριάδες όταν οι πιο περίπλοκες επερωτήσεις είχαν εξαιρεθεί. Στα πιο πάνω το Virtuoso ξεπερνά το BigOWLIM και ξεπερνά την JENA με μεγάλη διαφορά όσο αφορά εξίσου σενάρια με έναν πελάτη η πολλούς (4) πελάτες για το σύνολο επερωτήσεων το οποίο περιείχε τις πιο περίπλοκες επερωτήσεις. Επομένως, τουλάχιστο όσο αφορά αυτό το συγκεκριμένο σενάριο χρήσης -ένα περιβάλλον ηλεκτρονικού εμπορίου- το Virtuoso αποδίδει το ίδιο η καλύτερα από τον πιο κοντινό του αντίπαλο περαιτέρω ενισχύοντας την άποψη ότι το Virtuoso αποδίδει καλύτερα από τα περισσότερα από τα υπόλοιπα συστήματα.

Σε άλλες δοκιμές στις οποίες συγκρίνεται το Virtuoso με άλλα συστήματα (κυρίως με το σύστημα Sesame) σε ένα σενάριο ενός πελάτη [3] το Virtuoso απέδωσε το ίδιο καλά για μικρότερα σύνολα δεδομένων (250 χιλιάδες, 1 εκατομμύριο τριάδες) και υπερίσχυσε για μεγαλύτερα (25/100 εκατομμύρια) για περίπλοκες επερωτήσεις και επίσης για μεγάλα σύνολα δεδομένων με πιο απλές επερωτήσεις. Σε οποιοδήποτε σενάριο με περισσότερους του ενός πελάτη το Virtuoso υπερισχύει καθαρά.

Γενικά το Virtuoso αποδίδει καλύτερα από άλλα συστήματα όσο αφορά φόρτωμα δεδομένων, αποδίδει καλύτερα σε ποικίλα σενάρια και καθαρά υπερισχύει για περίπλοκες επερωτήσεις σε μεγάλα σύνολα δεδομένων ή/και με πολλούς πελάτες.

Μπορεί να χρησιμοποιήσει την διεπαφή JDBC για να επικοινωνήσει με την Java.

Αδυναμίες

Όταν δεν υπάρχουν περίπλοκες επερωτήσεις, ειδικά για μικρότερα σύνολα δεδομένων (μικρότερα του ενός εκατομμυρίου τριάδων) είναι προτιμότερα άλλα συστήματα όπως το Sesame και το BigOWLIM, όπως έδειξαν οι δοκιμές της βιβλιογραφίας. Το οποίο είναι σημαντικό καθώς οι περισσότερες αιτήσεις που αποστέλλονται σε έναν μικρό εξυπηρετητή επερωτήσεων RDF πολύ πιθανόν να είναι σχετικά απλές επερωτήσεις σε σχετικά μικρά σύνολα δεδομένων.

Επίσης έχει χαμηλές δυνατότητες όσο αφορά εξαγωγή δεδομένων μέσω χρήσης inferencing/reasoning όπως δείχνουν οι δοκιμές στο [31].

Γ)OWLIM

Το OWLIM είναι μια ομάδα συστημάτων διαχείρισης RDF ΒΔ τα οποία χρησιμοποιούν native μηχανές RDF υλοποιημένες σε Java και συμβατές με το σύστημα αποθήκευσης Sesame (χρησιμοποιώντας τις βιβλιοθήκες του Sesame).

Είναι κατασκευασμένο ως ένα Storage and Inference Layer (SAIL) για την υποδομή RDF του Sesame. Βασίζεται στο TRREE -μια native RDF ΒΔ και μηχανή λογικής συνεπαγωγής κανόνων η οποία έχει αναπτυχθεί από τον οργανισμό Ontotext.

Μπορεί να χρησιμοποιηθεί ως ένα στρώμα κατώτερης αποθήκευσης για το Sesame, ως ένα plugin SAIL στη θέση της συνηθισμένης υλοποίησης του στρώματος αποθήκευσης του Sesame.

Δυνατότητες

Σύμφωνα με αναφορές από τον κατασκευαστή η «ελαφριά» του έκδοση (swiftOWLIM) είναι κλιμακώσιμη για μέχρι και 10 εκατομμύρια τριάδες χρησιμοποιώντας 1.6 GB RAM, χρησιμοποιώντας μόνο κυρίως μνήμη (το οποίο συνεπάγεται πιο αποδοτική ανάκτηση δεδομένων και επεξεργασία επερωτήσεων).

Μπορεί να φορτώσει το Lehigh University Benchmark (50,0) το οποίο περιέχει 7 εκατομμύρια τριάδες [8] σε 2 λεπτά και να επιτύχει ταχύτητες φορτώματος σε επίπεδα δεκάδων χιλιάδων τριάδων ανά δευτερόλεπτο για σύνολα δεδομένων με εκατομμύρια τριάδες. Σε δοκιμές με χρήση του City Benchmark διαχειρίστηκε εκατομμύρια τριάδες χρησιμοποιώντας 1GB RAM, εκτελώντας επερωτήσεις σε γραμμικό χρόνο.

Σε μια άλλη σειρά δοκιμών που έγινε από ανεξάρτητο τρίτο μέρος [27] χρησιμοποιώντας το LUBM, το BigOWLIM (η έκδοση του OWLIM για μεγάλα σύνολα δεδομένων) απέδωσε πολύ καλύτερα από τα υπόλοιπα συστήματα τα οποία είχαν δοκιμαστεί, συμπεριλαμβανομένου του Sesame, στις περισσότερες περίπλοκες επερωτήσεις. Όμως οι δοκιμές αυτές είχαν γίνει σε μεγάλα σύνολα δεδομένων με χρήση του BigOWLIM το οποίο έχει κάποιες βελτιστοποιήσεις σχετικά με λειτουργίες σε μεγάλα σύνολα δεδομένων σε σχέση με το swiftOWLIM και είναι αβέβαιο αν αυτά τα συμπεράσματα ισχύουν και για το swiftOWLIM το οποίο είναι ο υποψήφιος που θα ήταν κατάλληλος για έναν μικρό εξυπηρετητή επερωτήσεων RDF όπως αυτόν που ήθελα να υλοποιήσω.

Μια τρίτη σειρά δοκιμών είχε εξεταστεί η οποία χρησιμοποιούσε το benchmark UOBM για συστήματα OWL [16] καθώς επίσης και το ελαφρύτερο σύνολο δεδομένων PA, το οποίο έχει ως σενάριο μια εμπορική εγκατάσταση. Σε αυτή την σειρά δοκιμών το BigOWLIM συγκρίθηκε έναντι των JENA(TDB), Virtuoso, Allegro, Sesame και της Σημασιολογικής Τεχνολογίας της Oracle (που ήταν το ΣΔΒΔ το οποίο είχε χρησιμοποιηθεί στην προηγούμενη υλοποίηση του συστήματος του κου Σαββίδη) [31] λαμβάνοντας σοβαρά υπόψη δυνατότητες εξαγωγής λογικών συμπερασμάτων/προτάσεων (inferencing/reasoning). Σε αυτές τις δοκιμές το BigOWLIM επέδειξε χρόνο φόρτωσης σχεδόν ανάλογο του Sesame για όλες τις περιπτώσεις όσο αφορά το UOBM φορτώνοντας όλα τα σύνολα δεδομένων σε γραμμικό χρόνο και πολύ καλύτερο της Oracle, παρόλο που άλλα συστήματα όπως το Virtuoso η το AllegroGraph αποδείκτηκαν ακόμη ισχυρότερα σε αυτό τον τομέα. Όμως, όσο αφορά την φόρτωση του συνόλου δεδομένων PA το οποίο ήταν λιγότερο προσανατολισμένο προς δυνατότητες inferencing το BigOWLIM είχε ξεπεραστεί από το AllegroGraph, το Sesame, ακόμη και την Jena.

Όσο αφορά τις επερωτήσεις, μόνο το BigOWLIM κατάφερε να απαντήσει σχεδόν όλες τις 15 επερωτήσεις του UOBM όμως αυτό πιθανό να μην λέει πολλά αν δεν ενδιαφερόμαστε για reasoning/inferencing καθώς οι επερωτήσεις εσχετίζοντο με reasoning/inferencing. Η Oracle είχε την χειρότερη απόδοση όσο αφορά το φόρτωμα των συνόλων δεδομένων του UOBM και την δεύτερη χειρότερη όσο αφορά τις επερωτήσεις του UOBM, μη μπορώντας να εκτελέσει σχεδόν καμία από αυτές αν και στις δοκιμές είχε χρησιμοποιηθεί μια προηγούμενη έκδοση της Oracle και επομένως πιο πρόσφατες εκδόσεις πιθανόν να παρουσιάζουν σημαντική βελτίωση. Επίσης πρέπει να σημειωθεί ότι το Virtuoso δεν κατάφερε να απαντήσει καμία επερώτηση του UOBM εκτός της $1^{ης}$ σε αυτό το βασισμένο στο inferencing σενάριο.

Στις επερωτήσεις του (πιο ελαφριού όσο αφορά inferencing) συνόλου δεδομένων PA το Virtuoso επίσης είχε την χειρότερη επίδοση, με το BigOWLIM να επιτυγχάνει την καλύτερη απαντώντας όλες τις επερωτήσεις σε χρόνους πολύ καλύτερους των υπολοίπων.

Πρέπει να σημειωθεί εδώ ότι ο χρόνος που απαιτείται για inferencing πιθανό να είχε επηρεάσει τα αποτελέσματα του τρίτου συνόλου επερωτήσεων.

Από τα πιο πάνω μπορούμε να συμπεράνουμε ότι το BigOWLIM (δεν μπόρεσα να βρω συγκρίσεις από ανεξάρτητες πηγές που να συμπεριλαμβάνουν το swiftOWLIM) είναι καλύτερο σε καταστάσεις με μεγάλες απαιτήσεις για inferencing/reasoning, αλλά σε καταστάσεις που έχουν λιγότερες απαιτήσεις reasoning, άλλα, όπως το Virtuoso, αποδίδουν καλύτερα.

Το OWLIM χρησιμοποιείται σε πολλά ερευνητικά προγράμματα χρηματοδοτούμενα από την ΕΕ και σε συνεργασία με τις SAP, IBM, Wikimedia, Google Labs, BT, Telefonica, KT και δεκάδες ευρωπαικά πανεπιστήμια.

Η πλήρης υλοποίηση σε Java προσφέρει ευκολία ενσωμάτωσης.

Αδυναμίες

Το BigOWLIM αποδίδει χειρότερα από το Virtuoso και άλλα συστήματα σε σενάρια που απαιτούν σε λιγότερο βαθμό δυνατότητες reasoning όπως φαίνεται από τις ανεξάρτητες δοκιμές που έχουν εξεταστεί. Επίσης αποδίδει χειρότερα όσο αφορά

επίδοση φόρτωσης δεδομένων. Δεν υπάρχουν επαρκή δεδομένα από ανεξάρτητες δοκιμές όσο αφορά το swiftOWLIM το οποίο θα ήταν καλύτερο για ένα ελαφρύ σύστημα.

Δ) Sesame με χρήση της φυσικής του μονάδας αποθήκευσης ή ΒΔ MySQL

Το Sesame είναι ένα πλαίσιο Java διαθέσιμο ως λογισμικό ανοικτού κώδικα για αποθήκευση, επερώτηση και reasoning σε RDF μεταδεδομένα και σχήματα RDF μεταδεδομένων. Μπορεί να χρησιμοποιηθεί ως ΒΔ για RDF και RDF Schema η ως μια βιβλιοθήκη Java για εφαρμογές που χρειάζονται εσωτερική επεξεργασία RDF μεταδεδομένων [30].

Το Sesame είναι μια αρχιτεκτονική για αποδοτική αποθήκευση και εκφραστικές επερωτήσεις πάνω σε μεγάλες ποσότητες μεταδεδομένων σε RDF και σχήματα RDF. Μπορεί να χρησιμοποιηθεί με ένα μη-ιθαγενές σύστημα αποθήκευσης RDF στο κατώτερο της επίπεδο και μπορεί επίσης να χρησιμοποιηθεί βάσει του ιθαγενούς της συστήματος αποθήκευσης. Το σύστημα αποθήκευσης OWLIM που περιγράφεται πιο πάνω χρησιμοποιείται ως plugin για το Sesame. Εδώ εξετάζω αποτελέσματα δοκιμών από την βιβλιογραφία με χρήση του ιθαγενούς συστήματος αποθήκευσης του Sesame η της ΒΔ ανοικτού κώδικα MySQL.

Δυνατότητες

Στις δοκιμές που παρουσιάζονται στο [31] (χρησιμοποιώντας το benchmark UOBM για συστήματα OWL [16] καθώς και το ελαφρύτερο σύνολο δεδομένων PA, σε ένα σενάριο που απαιτεί σε σημαντικό βαθμό δυνατότητες inferencing όσο αφορά το UOBM) το Sesame απέδωσε εξίσου καλά με το BigOWLIM όσο αφορά το φόρτωμα του συνόλου δεδομένων UOBM (αν και είχε ξεπεραστεί σημαντικά από το Virtuoso) και πολύ καλύτερα από την έκδοση της Oracle που είχε χρησιμοποιηθεί στις δοκιμές. Πέτυχε καλύτερους χρόνους από την JENA, το Virtuoso και το BigOWLIM όσο αφορά το φόρτωμα του συνόλου δεδομένων PA.

Στην δοκιμή SIMILE για κλιμακωσιμότητα συστημάτων διαχείρισης RDF μεταδεδομένων [15] (όπου ένα πολύ ελαφρύ σύνολο δεδομένων με 27.4 MB και περίπου 279 χιλιάδες τριάδες είχε χρησιμοποιηθεί ως benchmark) το Sesame - χρησιμοποιώντας MySQL3- πέτυχε τους δεύτερους καλύτερους χρόνους όσο αφορά

ρυθμίσεις, εμφάνιση δεδομένων και φυλλομέτρηση για μια υπηρεσία ΠΠΠ. Η υπηρεσία βασιζόταν στην επερώτηση μεταδεδομένων RDF από όλα τα συστήματα απομακρυσμένων ΒΔ τα οποία είχαν δοκιμαστεί, οδηγώντας στην παραδοχή ότι το Sesame πιθανώς να είναι καλύτερο για μικρότερα σύνολα δεδομένων.

Στο [29] το Sesame (χρησιμοποιώντας το ιθαγενές του σύστημα αποθήκευσης) κατάφερε να ολοκληρώσει επιτυχώς περισσότερες επερωτήσεις από τα άλλα δοκιμαζόμενα συστήματα (συμπεριλαμβανομένου του Virtuoso) δίδοντας τους καλύτερους αριθμητικούς μέσους χρόνων εκτέλεσης. Επίσης είχε πολύ χαμηλότερη κατανάλωση μνήμης για την δοκιμή στο ιθαγενές σύστημα αποθήκευσης, κάτι που έπρεπε να ληφθεί υπόψη κατά την επιλογή ενός συστήματος διαχείρισης μεταδεδομένων RDF για ένα «ελαφρύ» σύστημα.

Στις δοκιμές που περιγράφονται στο [3] στο benchmark BSBM το Sesame απέδωσε καλύτερα από όλα τα συστήματα για τα μικρά σύνολα δεδομένων όταν οι πιο περίπλοκες επερωτήσεις είχαν εξαιρεθεί.

Από τα πιο πάνω μπορούμε να συμπεράνουμε ότι το Sesame είναι καλύτερο για μικρά σύνολα δεδομένων μέχρι και ενός εκατομμυρίου τριάδων (το οποίο αναμένεται να ισχύει για την μεγάλη πλειοψηφία τον περιπτώσεων χρήσης του συστήματος μας).

Πλήρης υλοποίηση σε Java προσφέρει ευκολία ενσωμάτωσης.

Αδυναμίες

Σε δοκιμές που έγιναν στο benchmark BSBM [3] το Sesame χρειάστηκε πολύ περισσότερο χρόνο για την φόρτωση των συνόλων δεδομένων από όλα τα υπόλοιπα συστήματα -ειδικά όσο αφορά σύνολα δεδομένων με περισσότερα του ενός εκατομμυρίου τριάδων. Αυτό φαίνεται να αντιφάσκει με τα αποτελέσματα των δοκιμών στο [31] σχετικά με το Sesame, αλλά πιθανό να σχετίζεται με το γεγονός ότι οι δοκιμές στο [31] συμπεριλάμβαναν reasoning/inferencing ενώ οι δοκιμές στο [3] δεν συμπεριλάμβαναν reasoning/inferencing.

Όμως, στις ίδιες δοκιμές το Sesame είχε την καλύτερη επίδοση από όλα τα συστήματα για τα μικρά σύνολα δεδομένων. Αυτό ίσως να οφείλεται στο ότι το Sesame δημιουργεί περισσότερες δομές βελτιστοποίησης (όπως ευρετήρια) κατά την φόρτωση των

συνόλων δεδομένων (άρα απαιτώντας περισσότερο χρόνο κατά το φόρτωμα) οι οποίες επιτρέπουν πιο βελτιστοποιημένη εκτέλεση επερωτήσεων.

Σε δοκιμές στο [19] το Sesame απέτυχε να φορτώσει οποιοδήποτε από τα σύνολα δεδομένων σε λογικό χρόνο η να απαντήσει οποιαδήποτε από τις επερωτήσεις. Χρησιμοποιήθηκε ένα σύστημα με 2 GB κυρίως μνήμη, τυπικό για έναν ελαφρύ προσωπικό υπολογιστή όπως αυτόν στον οποίο το σύστημα μας προορίζεται να χρησιμοποιηθεί, χρησιμοποιώντας ως σύνολα δεδομένων τα Barton, Yago και μια εξαγωγή δεδομένων του LibraryThing (έχοντας 51.5 εκατομμύρια τριάδες/4.1GB, 34 εκατομμύρια τριάδες/3.1GB, 36 εκατομμύρια τριάδες/1.8GB αντίστοιχα). Αυτό υποδηλώνει ότι για σχετικά μεγάλα σύνολα δεδομένων το Sesame δεν είναι κλιμακώσιμο σε ελαφριά συστήματα.

Ε) 3Store

Το 3Store συντηρείται από μια ομάδα ανάπτυξης στο Πανεπιστήμιο του Southampton και χρησιμοποιείται στο project Advanced Knowledge Technologies (AKT). Το 3Store "is a core C library that uses MySQL to store its raw RDF data and caches" και βασίζεται στο Redland, ένα έργο διεπαφής RDF βασισμένο στην βιβλιοθήκη C η οποία το αποτελεί, η οποία έχει αναπτυχθεί από τον Dave Beckett του Bristol University's Institute for Learning and Research Technology [15].

Δυνατότητες

Το 3Store χρησιμοποιεί μια βελτιστοποίηση τριών επιπέδων, στα επίπεδα όχι μόνο της σύνταξης RDF και της αντιπροσώπευσης RDF αλλά και στο επίπεδο του ΣΔΒΔ πάνω στο οποίο βασίζεται, αφήνοντας το ΣΔΒΔ να βελτιστοποιήσει επιπλέον τις επερωτήσεις χρησιμοποιώντας τους μηχανισμούς βελτιστοποίησης του.

Στις δοκιμές SIMILE για κλιμακωσιμότητα συστημάτων διαχείρισης RDF μεταδεδομένων [15] (όπου ένα πολύ μικρό σύνολο δεδομένων με μέγεθος 27.4MB και περίπου 279 χιλιάδες τριάδες χρησιμοποιήθηκε ως benchmark) το 3Store πέτυχε τους καλύτερους χρόνους για ρύθμιση, εμφάνιση και φυλλομέτρηση για μια υπηρεσία ΠΠΠ βασισμένη στις επερωτήσεις RDF μεταδεδομένων από όλα τα συστήματα απομακρυσμένων βάσεων δεδομένων τα οποία είχαν δοκιμαστεί. Αυτό μας οδηγεί στην παραδοχή ότι το 3Store ίσως να είναι καλύτερο για μικρότερα σύνολα δεδομένων.

Αδυναμίες

Δεν μπόρεσα να εντοπίσω δεδομένα και συγκρίσεις του 3Store έναντι άλλων συστημάτων για μεγάλα σύνολα δεδομένων.

ΣΤ) RDF-3X

Το RDF-3X είναι "an implementation of SPARQL that achieves excellent performance by pursuing a RISC-style architecture with a streamlined architecture and carefully designed, puristic data structures and operations" [19]. Χρησιμοποιεί μια πολυδιάστατη τακτική πολλαπλών ευρετηρίων για να βελτιστοποιήσει λειτουργίες βάσεων δεδομένων χωρίς της ανάγκη ρυθμίσεων στο επίπεδο του φυσικού σχεδιασμού. Χρησιμοποιεί ένα επεξεργαστή επερωτήσεων βασισμένο στην αρχιτεκτονική RISC [6] ο οποίος βασίζεται σε υψηλά-βελτιστοποιημένα merge joins και έναν βελτιστοποιητή επερωτήσεων βασισμένο στην χρήση αλγορίθμων δυναμικού προγραμματισμού και υπολογισμών επιδράσεων σειράς εκτέλεσης join ο οποίος βασίζεται σε στατιστικές αξιολογήσεις των RDF μεταδεδομένων.

Δυνάμεις

Αντί να βασίζεται σε ένα γενικής-χρήσεως σχεσιακό ΣΔΒΔ για αποθήκευση σε χαμηλό επίπεδο το RDF-3X χρησιμοποιεί τις δικές του δομές αποθήκευσης οι οποίες προσφέρουν βελτιστοποιημένη αποθήκευση μεταδεδομένων RDF. Η τακτική πολλαπλών ευρετηρίων αφαιρεί την ανάγκη για ρυθμίσεις στο επίπεδο του φυσικού σχεδιασμού. Η χρήση στατιστικών αξιολογήσεων στα RDF μεταδεδομένα στο πλαίσιο μιας υλοποίησης ειδικά σχεδιασμένης για RDF μεταδεδομένα για βελτιστοποίηση επερωτήσεων έχει ως αποτέλεσμα έναν απλό αλλά ακριβή βελτιστοποιητή επερωτήσεων.

Σε δοκιμές οι οποίες έγιναν στο [19] χρησιμοποιώντας ένα σύστημα με 2 GB κυρίως μνήμη, τυπικό για ένα ελαφρύ προσωπικό υπολογιστή όπως αυτόν στον οποίο το σύστημα μας προορίζεται να χρησιμοποιηθεί, χρησιμοποιώντας ως σύνολα δεδομένων τα Barton, Yago και μια εξαγωγή δεδομένων του LibraryThing (έχοντας 51.5 εκατομμύρια τριάδες/4.1GB, 34 εκατομμύρια τριάδες/3.1GB, 36 εκατομμύρια τριάδες/1.8GB αντίστοιχα), το RDF-3X ξεπέρασε σημαντικά σε επίδοση τα PostgreSQL και MonetDB. Στις περισσότερες επερωτήσεις είχε υποπολλαπλάσιους

χρόνους, ειδικά όσο αφορά τα σύνολα δεδομένων YAGO και LibraryThing (τα οποία έχουν περισσότερο πολύπλοκες σχέσεις και περισσότερο «θόρυβο») -με το RDF-3X να είναι το μόνο σύστημα που έδινε λογικούς χρόνους για το LibraryThing.

Πρέπει επίσης να σημειωθεί εδώ ότι στις πιο πάνω δοκιμές τα συστήματα JENA, Yars2 και Sesame είχαν επίσης δοκιμαστεί αλλά απέτυχαν καθώς είτε απέτυχαν να φορτώσουν τα σύνολα δεδομένων είτε τα φόρτωσαν σε μη λογικό χρόνο και κατέρρευσαν κατά την εκτέλεση των πρώτων επερωτήσεων.

Τα πιο πάνω δείχνουν ότι το RDF-3X είναι ανώτερο σε επίδοση από άλλα συστήματα όσο αφορά κατανάλωση πόρων συστήματος στο επίπεδο που αναμένουμε να έχουμε σε συστήματα για τα οποία προορίζεται το λογισμικό μας και για σχετικά μεγάλα σύνολα δεδομένων.

Αδυναμίες

Ακόμη βρίσκεται σε πειραματικό στάδιο και δεν χρησιμοποιείται ευρέως για εμπορικές εφαρμογές. Έχει πολύ μικρή κοινότητα χρηστών και άρα και αυξημένες πιθανότητες για σφάλματα και αδυναμίες τα οποία δεν έχουν ακόμη εντοπιστεί.

Είναι υλοποιημένο σε C++ επομένως η ενσωμάτωση με την Java και το υπόλοιπο σύστημα μας θα ήταν κάπως πιο δύσκολη από τα πιο πάνω εξεταζόμενα συστήματα τα οποία είναι υλοποιημένα σε Java.

### 5.3 Γενική αξιολόγηση-επιλογή σε πρώτη φάση

Από τα πιο πάνω μπορούμε να συμπεράνουμε ότι

1) Το OWLIM είναι πιθανόν το καλύτερο από τα πιο πάνω για σενάρια όπου απαιτούνται δυνατότητες reasoning [31] (από τα δεδομένα που έχουμε μέχρι τώρα) αλλά υπερτερούν έναντι αυτού άλλα συστήματα σε σενάρια που δεν σχετίζονται με reasoning (όπως δείχνουν ανεξάρτητες δοκιμές έναντι των δοκιμών που παρουσιάζονται από τον παροχέα). Αυτό ισχύει για σενάρια που σχετίζονται με το λογισμικό μας. Επομένως το OWLIM μπορεί να εξαιρεθεί.

2) Παρόλο που το 3Store έχει δείξει σε μια δοκιμή ότι αποδίδει καλύτερα από άλλα σε πολύ μικρά σύνολα δεδομένων [15] αυτό δεν μας παρέχει επαρκή δεδομένα προς

υποστήριξη του ισχυρισμού ότι προσφέρει καλύτερη επίδοση, ειδικά για σύνολα δεδομένων λογικά μεγάλου μεγέθους. Επομένως το 3Store μπορεί να εξαιρεθεί.

3) Το Virtuoso αποδίδει καλύτερα σε σενάρια πολλών πελατών, περίπλοκων επερωτήσεων και συνόλων δεδομένων με περισσότερες των ενός εκατομμυρίου τριάδων από τα υπόλοιπα [3][4] ακόμη και σε συστήματα με περιορισμένους πόρους [29], το οποίο σημαίνει ότι χρησιμοποιώντας το Virtuoso το λογισμικό μας θα είναι πιο κλιμακώσιμο για μεγάλα σύνολα δεδομένων και πολλούς πελάτες και πιο περίπλοκες επερωτήσεις. Λαμβάνοντας υπόψη την ραγδαία αύξηση των μεγεθών των συνόλων δεδομένων RDF, το Virtuoso σίγουρα ήταν μια από τις επιλογές που έπρεπε να ληφθούν υπόψη, λαμβάνοντας επίσης υπόψη ότι συμπεριλαμβάνει μια ικανοποιητική διεπαφή Java.

4) Το Sesame φαίνεται ότι γενικά υπερτερεί όσο αφορά μικρότερα σύνολα δεδομένων, πιο απλές επερωτήσεις και σενάρια ενός πελάτη σε σχέση με το Virtuoso και όλα τα άλλα εξεταζόμενα συστήματα, όμως αφού αναμένουμε το λογισμικό μας να χρησιμοποιηθεί ως ένας εξυπηρετητής που να εξυπηρετεί πολλούς πελάτες με συνεχώς αυξανόμενα μεγέθη συνόλων δεδομένων το Virtuoso είναι προτιμότερο.

5) Παρόλο που είναι κάπως πιο δύσκολο όσο αφορά την ενσωμάτωση λόγω της χρήσης C++ αντί της Java καθώς επίσης και του ότι είναι πολύ λιγότερο δοκιμασμένο, το RDF-3X φαίνεται να αποδίδει καλύτερα από το Sesame για τις περισσότερες επερωτήσεις όπως δείχνουν οι δοκιμές στο [2] όπως επίσης και η αποτυχία του Sesame να ανταποκριθεί στις απαιτήσεις για τις δοκιμασίες σε σενάρια περιορισμένων πόρων στο [19] (όπου το RDF-3X εξεπέρασε τα υπόλοιπα).

Στις δοκιμές στο [26] όπου χρησιμοποιήθηκε το Wikipedia3 Wikimedia RDF dump (διαθέσιμο στο http://labs.systemone.net /wikipedia3) το οποίο περιείχε περίπου 47 εκατομμύρια τριάδες, στις οποίες το Virtuoso είχε συγκριθεί μαζί με το RDF-3X, το Virtuoso απέδωσε καλύτερα σε κάποιες επερωτήσεις και χειρότερα σε κάποιες άλλες. Επομένως τα δύο συστήματα δείχνουν να είναι περίπου ισοδύναμα λαμβάνοντας υπόψη τους γενικούς χρόνους επερωτήσεων. Όμως η απόδοση επερωτήσεων καθώς ο αριθμός των τριάδων αυξάνεται μειώνεται πολύ περισσότερο για περισσότερες επερωτήσεις για το Virtuoso παρά για το RDF-3X όπως δείχνουν οι δοκιμές

κλιμακωσιμότητας στο ίδιο έγγραφο, δείχνοντας ότι το RDF-3X είναι πιο πιθανό να είναι κλιμακώσιμο από το Virtuoso σε συστήματα με περιορισμένους πόρους.

Για τους πιο πάνω λόγους είχα αποφασίσει πως το RDF-3X ήταν προτιμότερο του Virtuoso, με τα OWLIM και 3Store να μην είναι ικανοποιητικές λύσεις όσο αφορά τις ανάγκες μας και το Sesame σχετικά κατώτερο όσο αφορά φόρτωση και επερωτήσεις σε μεγάλα σύνολα δεδομένων σε συστήματα με περιορισμένους πόρους.

Οι μόνες δυσκολίες τις οποίες είχα προβλέψει ότι θα υπήρχαν όσο αφορά το RDF-3X εντοπίζονταν στο γεγονός ότι δεν ήταν το ίδιο ευρέως δοκιμασμένο όσο τα υπόλοιπα συστήματα και ότι η υλοποίηση του σε C++ σημαίνει ότι πιθανό να χρειαζόταν να χρησιμοποιηθεί μια επιπλέον διεπαφή χρησιμοποιώντας ιθαγενείς μεθόδους Java .

## 5.4 Προβλήματα με την δοκιμή του RDF-3X και επιλογή του Virtuoso εναλλακτικά

Μετά από κάποιες εξερευνητικές δοκιμές ούτως ώστε να επιβεβαιώσω την λειτουργικότητα του RDF-3X θέτοντας προς εκτέλεση κάποιες επερωτήσεις ανακαλύφθηκε ότι το RDF-3X είχε σφάλματα όσο αφορά την εκτέλεση επερωτήσεων SPARQL. Αυτό αφορά επερωτήσεις οι οποίες περιλάμβαναν φιλτράρισμα των επιστρεφόμενων αποτελεσμάτων βάσει ταύτισης τιμών δεσμευόμενων μεταβλητών με μια συγκεκριμένη κανονική έκφραση.

Για παράδειγμα, έχουμε την επερώτηση

BASE <http://example.org/>

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

# This is a relative URI to BASE above

PREFIX ex: <properties/1.0#>

SELECT DISTINCT $person ?name $age

FROM <http://rdf.example.org/personA.rdf>

```
FROM <http://rdf.example.org/personB.rdf>
WHERE {  $person a foaf:Person ;
            foaf:name ?name.
      OPTIONAL { $person ex:age $age } .
      FILTER (!REGEX(?name, "Bob"))
     }
ORDER BY ASC(?name) LIMIT 10 OFFSET 20
```

Στην πιο πάνω επερώτηση ζητείται να επιστραφούν μόνο τα αποτελέσματα στα οποία οι τιμές με τις οποίες δεσμεύεται η μεταβλητή name περιέχουν την γραμματοσειρά «Bob».

Σενάρια όπως αυτό είναι κοινά όσο αφορά επερωτήσεις SPARQL, ειδικά όσο αφορά το σύστημα μας στο οποίο οι περιορισμοί των παρουσιαζόμενων αποτελεσμάτων που επιστρέφονται από τις επερωτήσεις περιβάλλοντος (background queries) βασίζονται σε γραμματοσειρές που εισάγει ο χρήστης στο ανάλογο πεδίο. Επίσης και οι τελικές επερωτήσεις χρησιμοποιούν περιορισμούς βάσει γραμματοσειρών που πρέπει να υπάρχουν στα επιστρεφόμενα αποτελέσματα (όπως επεξηγείται και στην περιγραφή της λειτουργίας της προηγούμενης υλοποίησης του συστήματος).

Από τα πιο πάνω συνεπάγεται ότι είτε θα έπρεπε να δαπανήσουμε επιπλέον χρόνο και προσπάθεια αυξάνοντας και την πολυπλοκότητα του συστήματος με ότι αυτό συνεπάγεται (και πάλι χωρίς εγγύηση επιτυχίας) για να ξεπεράσουμε αυτό το πρόβλημα, είτε θα έπρεπε να επιλέξουμε μια εναλλακτική λύση για υλοποίηση του επιπέδου διαχείρισης δεδομένων RDF στην αρχιτεκτονική του συστήματος μας.

Τελικά επέλεξα το δεύτερο και έτσι στράφηκα στην χρήση της δεύτερης επιλογής που είχα επιλέξει, δηλαδή του ΣΔΒΔ Virtuoso, το οποίο και χρησιμοποιήθηκε στην τελική υλοποίηση που παρουσιάζεται στο επόμενο κεφάλαιο, χρησιμοποιώντας τον εξυπηρετητή Apache Tomcat/6.0.32 ως εξυπηρετητή εφαρμογών.

# Κεφάλαιο 6

## Περιγραφή νέας υλοποίησης συστήματος με χρήση του ΣΔΒΔ OpenLink Virtuoso ως αντικατάσταση της Oracle



### 6.1 Σύστημα καταφόρτωσης συνόλων δεδομένων

Γενικά

Η υλοποίηση του Κου Σαββίδη στην Oracle συμπεριλάμβανε ένα σύστημα καταφόρτωσης το οποίο και αναλύεται σε πιο πάνω κεφάλαιο. Ένα τέτοιο σύστημα έπρεπε να αναπτυχθεί και για την νέα υλοποίηση.

Για αυτή την υλοποίηση αποφάσισα να υλοποιήσω το σύστημα ως πίνακες που διατηρούνται στη Βάση Δεδομένων και ένα ανεξάρτητο πρόγραμμα σε Java με λειτουργία πολυνηματικής εκτέλεσης.

Λειτουργία -αρχιτεκτονική

Το σύστημα αυτό αποτελείται (ανάμεσα σε άλλα) από μια κυρίως κλάση η οποία «αφουγκράζεται» συνεχώς την ουρά της ΒΔ, η οποία ουρά δέχεται URL συνόλων δεδομένων προς καταφόρτωση προωθώντας ανά τακτά χρονικά διαστήματα επερωτήσεις προς την ΒΔ προς ανάκτηση του αρχαιότερου από τα URL που υπάρχουν στην ουρά (δηλαδή η ουρά αποτελεί δομή δεδομένων First-Come First-Served). Τα URL εισάγονται στην ουρά από σχετικό JSP του εξυπηρετητή εφαρμογών, το οποίο JSP τώρα αντί να προωθεί αίτηση προς εσωτερική διαδικασία της ΒΔ η οποία να τοποθετεί το URL στην ουρά (όπως ίσχυε στην υλοποίηση στην Oracle) προωθεί επερώτηση για ενημέρωση του πίνακα ο οποίος αποτελεί την ουρά.

Ο πίνακας αυτός έχει τα εξής πεδία.

RDF_DB.DBA.URLS

| **ID(**INTEGER,PRIMARY KEY,AUTOINCREMENT) | URL(VARCHAR) |
|---|---|

Πίνακας 6.1 Πίνακας ΒΔ RDF_DB.DBA.URLS

Όταν προωθηθεί στη ΒΔ επερώτηση για εισαγωγή ενός URL (που πιθανό να υπάρχει ήδη στη βάση) το URL αυτό εισάγεται και του δίνεται ένας αριθμός ταυτότητας ο οποίος είναι αύξων. Με χρήση του MIN(ID) γίνεται από την κυρίως κλάση του συστήματος καταφόρτωσης ανάκτηση του αρχαιότερου URL το οποίο και αφαιρείται από τον πίνακα (αν υπάρχει URL στον πίνακα, διαφορετικά η κυρίως δεργασία αναστέλλει την λειτουργία της για σύντομο χρονικό διάστημα και ξαναελέγχει).

Όταν βρεθεί URL στον πίνακα (και ενόσω υπάρχουν ακόμη URL στον πίνακα) η κυρίως διεργασία δημιουργεί νήματα τα οποία είναι υπεύθυνα για την διαχείριση της αίτησης καταφόρτωσης, ένα για κάθε URL.

Κάθε νήμα καλεί την κλάση Loader η οποία και εκτελεί την κυρίως εργασία της διαχείρισης του ελέγχου και αν χρειάζεται καταφόρτωσης του συνόλου δεδομένων το οποίο βρίσκεται στο ζητούμενο URL. Συγκεκριμένα, από αυτή την κλάση εκτελούνται τα ακόλουθα:

Α) Επερώτηση προς την ΒΔ για έλεγχο αν αίτηση για τον πόρο αυτού του URL τυγχάνει αυτή την στιγμή επεξεργασίας από άλλο νήμα

Αυτός ο έλεγχος απαιτείται αφού από την στιγμή που έχουμε έναν εξυπηρετητή ο οποίος πιθανό να εξυπηρετεί την ίδια στιγμή πολλούς πελάτες, κάποιοι εκ των οποίων πιθανό να αιτούνται την ίδια στιγμή η σε κοντινές χρονικές στιγμές τον ίδιο πόρο, πρέπει να γίνεται έλεγχος ούτως ώστε να μην καταφορτώνεται (σε περίπτωση που δεν υπάρχει ήδη από πριν στη ΒΔ) ο πόρος του ίδιου URL εις διπλούν σπαταλώντας πόρους του συστήματος και εύρος ζώνης.

Αυτό γίνεται δυνατό με την χρήση του βοηθητικού πίνακα της ΒΔ TEMP_URLS ο οποίος αποτελείται από μόνο ένα πεδίο και κρατά τα URLs τα οποία δέχονται επεξεργασία κάθε χρονική στιγμή. Αν μια αίτηση εγκριθεί για επεξεργασία το URL το οποίο δίδεται προς επεξεργασία εισάγεται στον εν λόγω πίνακα και αφαιρείται όταν

τερματιστεί η επεξεργασία. Αν κατά την διάρκεια της επεξεργασίας του URL αυτού (οπότε -τότε και μόνον τότε- θα υπάρχει το URL στον πίνακα TEMP_URLS) φτάσει μια άλλη αίτηση για το ίδιο URL, η αίτηση αυτή δεν τυγχάνει επεξεργασίας μα αντίθετα η συνάρτηση απλά επιστρέφει.

Β) Επερώτηση προς την ΒΔ για έλεγχο αν υπάρχει ήδη στη ΒΔ ο ζητούμενος πόρος και αν ναι επιστροφή της ημερομηνίας τελευταίας καταφόρτωσης του

Αυτό γίνεται εφικτό μέσω του κυρίως πίνακα του συστήματος καταφόρτωσης, του URL_META, στον οποίο αποθηκεύονται τα URLs που τα σύνολα δεδομένων τα οποία αντιπροσωπεύουν υπάρχουν ήδη φορτωμένα στη ΒΔ, συνοδευόμενα από την ημερομηνία καταφόρτωσης του καθενός. Για URL η τελευταία προσπάθεια φόρτωσης των οποίων ήταν αποτυχημένη υπάρχει ως τιμή του πεδίου ημερομηνίας το έτος 1970 ώστε να αναγκαστεί το σύστημα να ξαναδοκιμάσει την καταφόρτωση τους.

RDF_DB.DBA.URL_META

| **URL**(VARCHAR,PRIMARY KEY) | LASTUPDATE(DATETIME) |
|---|---|

Πίνακας 6.2 Πίνακας ΒΔ RDF_DB.DBA.URL_META

Αν βρεθεί ότι το ζητούμενο URL δεν υπάρχει ήδη στη ΒΔ το σύστημα προχωρά στο βήμα Δ.

Γ) Ανάκτηση ημερομηνίας τελευταίας αναβάθμισης του πόρου στο Web τον οποίο αντιπροσωπεύει το ζητούμενο URL

Αυτό γίνεται μέσω της αποστολής HEAD HTTP request προς τον εξυπηρετητή ο οποίος διαχειρίζεται τον ζητούμενο πόρο στο Διαδίκτυο και ανάκτησης του πεδίου last-modified της επιστρεφόμενης επικεφαλίδας HTTP.

Έπειτα γίνεται σύγκριση της τιμής που είχε επιστρέψει ο εξυπηρετητής υπεύθυνος για τον πόρο στο Διαδίκτυο ως ημερομηνία τελευταίας ενημέρωσης με την ημερομηνία τελευταίας καταφόρτωσης του πόρου που έχει ανακτηθεί από την ΒΔ του συστήματος. Αν ο πόρος βρεθεί να είναι έγκυρος (η πρώτη πιο πάνω ημερομηνία προηγείται της δεύτερης) τότε η συνάρτηση επιστρέφει με μήνυμα επιτυχίας -δεν χρειάζεται

ενημέρωση του ήδη υπάρχοντος στη ΒΔ πόρου. Αν όχι το σύστημα προχωρά στο βήμα Δ.

Δ) Καταφόρτωση αιτούμενου πόρου από URL

Η καταφόρτωση γίνεται παράλληλα με χρήση πολλαπλών υπο-νημάτων, καθένα εκ των οποίων είναι υπεύθυνο για την καταφόρτωση ενός μέρους του ζητούμενου πόρου.

Συγκεκριμένα, ο αριθμός των υπο-νημάτων μπορεί να εισαχθεί ως παράμετρος στη γραμμή εντολών κατά την ενεργοποίηση του συστήματος καταφόρτωσης και το σύστημα διαιρεί το μέγεθος του αιτούμενου πόρου (το οποίο παίρνει μέσω HEAD HTTP request προς τον εξυπηρετητή στο Διαδίκτυο που είναι υπεύθυνος για τον πόρο) με τον αριθμό των υπο-νημάτων που έχουν εισαχθεί ως παράμετρος. Έτσι δίνει σε κάθε υπο-νήμα την ευθύνη της καταφόρτωσης μεγέθους συνεχόμενων δεδομένων ίσου με το (στρογγυλοποιημένο) πηλίκο αυτής της διαίρεσης. Η δυνατότητα καταφόρτωσης από ένα υπο-νήμα μόνο ενός μέρους του αρχείου δίδεται από το ίδιο το πρωτόκολλο HTTP, το οποίο προβλέπει για τον καθορισμό του μέρους του αρχείου που θα μεταφερθεί ως απάντηση σε μια αίτηση GET μέσω του πεδίου Range, στο οποίο μπορούν να τεθούν οι τιμές του σημείου αρχής και τέλους.

Πχ αν θέλουμε να καταφορτώσουμε μέσω μιας αίτησης τα bytes από το 100$^{ο}$ μέχρι το 200ό τότε θέτουμε στο πεδίο Range

Bytes = 100 - 200

Κάθε νήμα αποθηκεύει προσωρινά τα δεδομένα τα οποία καταφορτώνει σε ένα αρχείο. Αν ένα υπο-νήμα ενώ καταφορτώνει δεδομένα αντιμετωπίσει σφάλμα σύνδεσης (πχ διακοπεί για οποιονδήποτε λόγο η σύνδεση με τον εξυπηρετητή στο Διαδίκτυο ο οποίος είναι υπεύθυνος για τον ζητούμενο πόρο) τότε το υπο-νήμα ξαναξεκινά την καταφόρτωση του μέρους του αρχείου για το οποίο είναι υπεύθυνο.

Σημειωτέο εδώ ότι σε περίπτωση σφάλματος δεν χρειάζεται η επανεκκίνηση της καταφόρτωσης ολόκληρου του πόρου οπότε το σύστημα μας είναι πιο ανθεκτικό σε απρόβλεπτα συμβάντα, το οποίο είναι και ένας από τους δύο λόγους που χρησιμοποιούμε πολλά ταυτόχρονα υπο-νήματα και ταυτόχρονες αιτήσεις για καταφόρτωση του ζητούμενου πόρου. Ο δεύτερος λόγος είναι επειδή έτσι μπορεί να

γίνεται πιο αποδοτική και πλήρης χρήση της σύνδεσης με το Διαδίκτυο και έτσι να επιταχύνεται η καταφόρτωση.

Το σύστημα αναμένει μέχρι να ολοκληρώσουν την λειτουργία τους όλα τα υπο-νήματα και έπειτα προχωρά στο να συνενώσει τα μέρη του ζητούμενου πόρου που έχουν καταφορτωθεί σε ένα άλλο προσωρινό αρχείο το οποίο περιέχει ολόκληρο τον πόρο.Όλα τα προσωρινά αρχεία, τόσο τα μερικά όσο και το ολικό, είναι αποθηκευμένα στο σύστημα αρχείων του εξυπηρετητή ΒΔ, σε διεύθυνση η οποία ακολουθεί την μορφή του URL το οποίο αντιστοιχεί σε κάθε πόρο. Έτσι διευκολύνεται η διαχείριση πολλών ταυτόχρονων προσωρινών αρχείων από πολλαπλά νήματα (και πολλαπλά υπο-νήματα για κάθε νήμα).

Ε) Φόρτωση των δεδομένων από προσωρινό αρχείο στον εξυπηρετητή ΒΔ

Γίνεται με χρήση της συνάρτησης ld_dir του Virtuoso η οποία προσφέρει την δυνατότητα bulk load. Αποφασίστηκε να χρησιμοποιηθεί αφού σε δοκιμή την οποία εκτέλεσα χρησιμοποιώντας σύνολο δεδομένων με μέγεθος πέραν των 370 MB σε υπολογιστή με κυρίως μνήμη 720 MB και επεξεργαστή (single-core) 1.7 GHz χρησιμοποιώντας τις συναρτήσεις του API του Virtuoso η φόρτωση του συνόλου δεδομένων απέτυχε μετά από 45 λεπτά χρησιμοποιώντας ολόκληρη την κυρίως μνήμη. Ο bulk loader κατάφερε να φορτώσει το ίδιο σύνολο δεδομένων στον ίδιο υπολογιστή σε χρόνο μικρότερο των 30 λεπτών.

Η συνάρτηση αυτή δεν ξεκινά την φόρτωση του πόρου στην βάση η ίδια αλλά προσθέτει το μονοπάτι προς την διεύθυνση των αρχείων που πρέπει να εισαχθούν στη ΒΔ σε ένα πίνακα συστήματος ο οποίος κρατά τα αρχεία τα οποία θα πρέπει να φορτωθούν μαζί στην βάση (bulk load) όταν ενεργοποιηθεί το φόρτωμα.

Ο πίνακας αυτός δημιουργείται ως εξής, μέσω της εκτέλεσης στη ΒΔ ενός script το οποίο δημιουργεί τις απαιτούμενες δομές για χρήση του bulk loader.

```
create table DB.DBA.LOAD_LIST (
  ll_file varchar,
  ll_graph varchar,
  ll_state int default 0, -- 0 not started, 1 going, 2
done
```

```
  ll_started datetime,
  ll_done datetime,
  ll_host int,
  ll_work_time integer,
  ll_error varchar,
  primary key (ll_file))
alter index LOAD_LIST on DB.DBA.LOAD_LIST partition
(ll_file varchar)
create index LL_STATE on DB.DBA.LOAD_LIST (ll_state,
ll_file, ll_graph) partition (ll_state int)
;
```

Ο πίνακας αυτός αποτελείται από τα ακόλουθα πεδία:

i. ll_file: Η διεύθυνση του αρχείου προς φόρτωμα στη βάση δεδομένων.

ii. ll_graph: Το URI του γράφου με τον οποίο θα αντιστοιχιστούν τα δεδομένα που θα εισαχθούν από το εν λόγω αρχείο (θέτοντας το ίδιο URI για πολλά αρχεία επιτρέπει την εισαγωγή στον ίδιο γράφο συνόλων δεδομένων τα οποία περιέχονται σε πολλά αρχεία).

iii. work_time: Χρόνος που απαιτήθηκε για το φόρτωμα του αρχείου στον γράφο.

Τα υπόλοιπα πεδία είναι αυτοεπεξηγηματικά.

Αφού προστεθεί στον πίνακα των αρχείων προς φόρτωση στην ΒΔ η διεύθυνση του προσωρινού αρχείου (το οποίο περιέχει τα δεδομένα του συνόλου δεδομένων που έχουμε καταφορτώσει από το Web), χρησιμοποιούνται πολλαπλά υπο-νήματα(το πλήθος των οποίων μπορεί να εισαχθεί ως παράμετρος από την γραμμή εντολών κατά την διάρκεια της ενεργοποίησης του συστήματος καταφόρτωσης) τα οποία ενεργοποιούν ταυτόχρονα το φόρτωμα των αρχείων στη λίστα του bulk loader. Έτσι έχουμε παράλληλη φόρτωση των αρχείων στη λίστα βελτιώνοντας τον χρόνο εισαγωγής των δεδομένων των συνόλων δεδομένων στη ΒΔ.

ΣΤ) Δημιουργία summaries (δομών βελτιστοποίησης για βελτιστοποίηση των ενδιάμεσων επερωτήσεων)

Περιγράφεται σε λεπτομέρεια στο μεθεπόμενο κεφάλαιο.

Ζ) Τροποποίηση μετα-μεταδεδομένων στη ΒΔ σχετικά με τα σύνολα μεταδεδομένων τα οποία υπάρχουν στη βάση

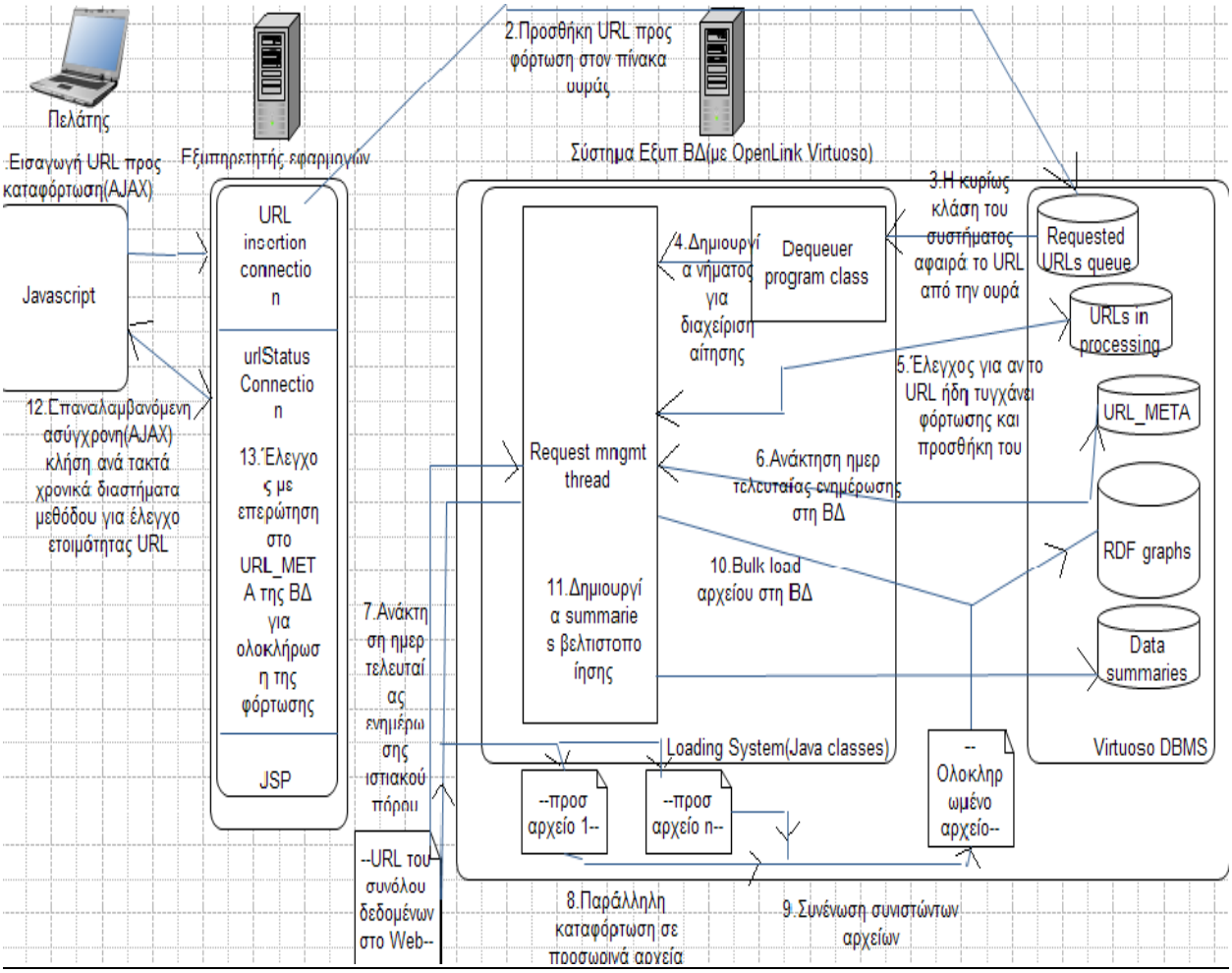
Με προσθήκη του νέου URL και της τρέχουσας ημερομηνίας ως ημερομηνίας τελευταίας ενημέρωσης στον πίνακα URL_META αν πρόκειται για νεοεισαγόμενο σύνολο δεδομένων. Με ενημέρωση του σχετικού πεδίου στον πίνακα για το ζητούμενο URL αν πρόκειται για επαναφόρτωση ήδη εισαγμένου συνόλου δεδομένων.

Η) Διαγραφή προσωρινών αρχείων.

Ασύγχρονος έλεγχος για ολοκλήρωση καταφόρτωσης

Η δυνατότητα που πρέπει να έχει ο χρήστης να συνεχίσει να εισάγει δεδομένα στο σύστημα μέσω της διεπαφής ενώ το σύστημα αναμένει για καταφόρτωση (πιθανώς μεγάλων) συνόλων δεδομένων επιβάλλει την ασύγχρονη υλοποίηση του μηχανισμού καταφόρτωσης.

Πέρα από την ανεξαρτησία των κλάσεων Java που χρησιμοποιούνται για την διαχείριση των εισερχομένων αιτήσεων και την χρήση ουράς FCFS για την υλοποίηση του μηχανισμού διαχείρισης αιτήσεων, η ασυγχρονία του συστήματος εξασφαλίζεται με την χρήση συνδέσεων AJAX για σύνδεση του πελάτη με τον εξυπηρετητή εφαρμογών ανά τακτά χρονικά διαστήματα. Μέσω αυτής της σύνδεσης ο πελάτης αποστέλλει στον εξυπηρετητή αιτήσεις για εκτέλεση επερωτήσεων προς την ΒΔ στον πίνακα URL_META ούτως ώστε να διαπιστωθεί αν έχει ολοκληρωθεί η φόρτωση (πιθανόν και μετά από προηγούμενη αίτηση) του ζητούμενου συνόλου δεδομένων. Αυτές οι αιτήσεις συνεχίζουν να αποστέλλονται στον εξυπηρετητή εφαρμογών μέχρι η ΒΔ να απαντήσει θετικά σχετικά με την ύπαρξη του ζητούμενου URL στον πίνακα URL_META που περιέχει τα URL των φορτωμένων συνόλων δεδομένων.




Σχήμα 6.1 Σχεδιάγραμμα με την περιγραφή της διαδικασίας φόρτωσης συνόλου δεδομένων.

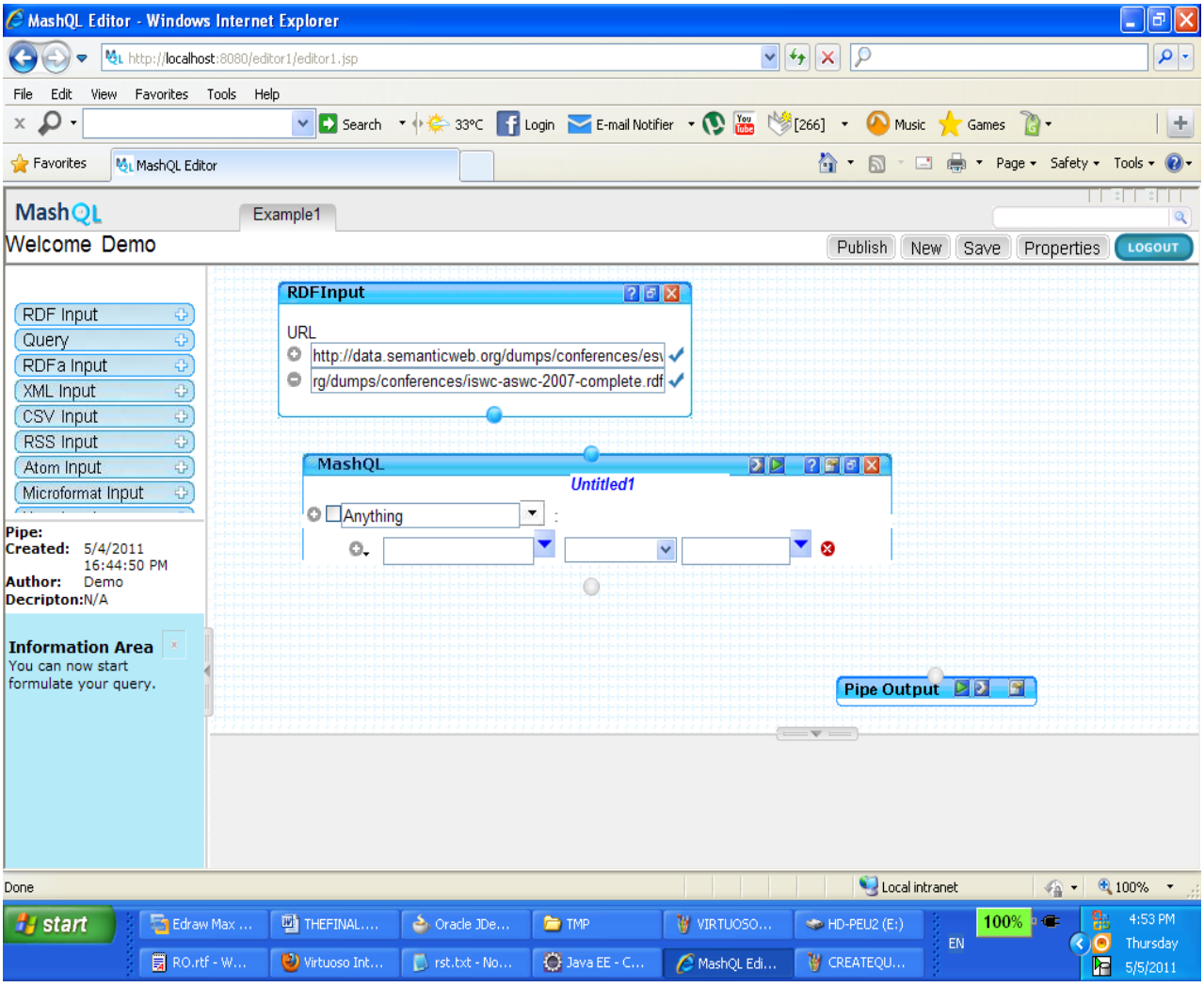
### 6.2 Δημιουργία τελικής επερώτησης -εκτέλεση ενδιάμεσων επερωτήσεων

Επειδή στο μεγαλύτερο μέρος του ο πελάτης δεν έτυχε σημαντικών τροποποιήσεων θα αναφερθώ περισσότερο στις αλλαγές που χρειάστηκε να γίνουν στον εξυπηρετητή εφαρμογών αλλά και στην υλοποίηση των μηχανισμών της ΒΔ οι οποίοι επιτρέπουν την λειτουργία του συστήματος.

Αρχικά, για να δημιουργήσει ο χρήστης μια επερώτηση θα πρέπει να εισάγει τα URL των συνόλων δεδομένων τα οποία θα επερωτηθούν σε ένα δομοστοιχείο RDFInput και να αναμένει μέχρι το σύστημα καταφόρτωσης (το οποίο περιγράφεται πιο πάνω) να ολοκληρώσει την καταφόρτωση τους.

Έπειτα θα πρέπει να δημιουργήσει (είτε με drag-n-drop από το sidebar είτε με πάτημα του ποντικιού στην επιφάνεια εργασίας) ένα νέο δομοστοιχείο Query.

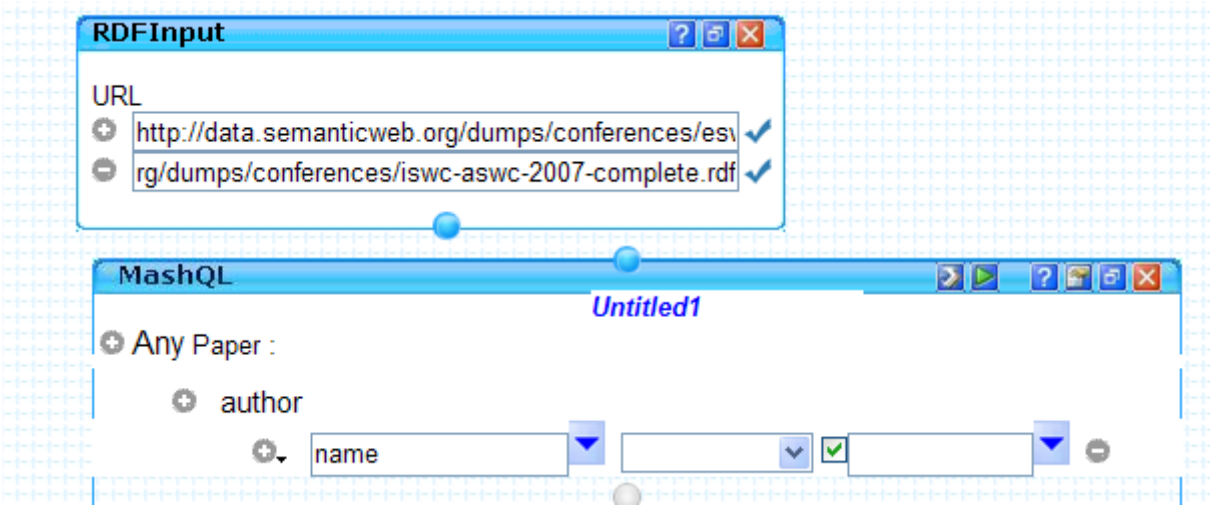
Έπειτα θα πρέπει να ενώσει το δομοστοιχείο RDFInput με το δομοστοιχείο Query. Αυτό θα επιτρέψει την προσθήκη των URL των συνόλων δεδομένων εισόδου προς χρήση στην δημιουργία τόσο των επερωτήσεων περιβάλλοντος όσο και της τελικής επερώτησης. Η προσθήκη των URL που εισάγονται στο δομοστοιχείο RDFInput στον κώδικα της επερώτησης MashQL που θα δημιουργηθεί στο δομοστοιχείο Query γίνεται με χρήση της τεχνολογίας Yahoo!Pipes η οποία περιγράφεται στην περιγραφή τεχνολογιών σε πιο πάνω κεφάλαιο.




Σχήμα 6.2 Δημιουργία νέας επερώτησης

Στο επόμενο βήμα ουσιαστικά εισερχόμαστε στο όλο νόημα της λειτουργίας του editor: Την δημιουργία ενδιάμεσων επερωτήσεων.

Εδώ ο πρώτος τρόπος με τον οποίο μπορεί να ενεργοποιηθεί ο μηχανισμός επερωτήσεων περιβάλλοντος είναι όταν ο χρήστης πατήσει το κουμπί για εμφάνιση της λίστας των υποκειμένων προς επιλογή. Αυτό που πρέπει να γίνει τότε είναι να εμφανιστούν στον χρήστη όλα τα υποκείμενα (πρώτα μέρη σε τριάδες RDF, τα οποία αποτελούν πόρους οι οποίοι περιγράφονται από τις τιμές των ιδιοτήτων/κατηγορημάτων τους) τα οποία εντοπίζονται: 1)Στα σύνολα δεδομένων τα οποία έχουν εισαχθεί από τον χρήστη. Η 2) Στα υποσύνολα αυτών των συνόλων δεδομένων τα οποία παραμένουν αφού εφαρμοστούν πάνω σε αυτά οι περιορισμοί της προηγούμενης συνδεδεμένης επερώτησης στο Mashup, οι οποίοι και προσαρμόζονται και μεταφέρονται-προστίθενται και σε αυτή την επερώτηση. Για αυτό απαιτείται χρήση ενδιάμεσης επερώτησης για εντοπισμό των υποκειμένων των συνόλων δεδομένων εισόδου που ικανοποιούν τους περιορισμούς που ήδη έχουν τεθεί από τον χρήστη σε αυτό το δομοστοιχείο Query. Παρόμοιοι μηχανισμοί ενεργοποιούνται για επιλογή κατηγορήματος, αντικειμένου (τα δύο τελευταία μάλιστα πιθανόν σε πολλά επίπεδα κάτω στην ιεραρχία, πχ αν θέλουμε από το σύνολο δεδομένων του eswc για τα συνέδρια του 2007 τα ονόματα των συγγραφέων που έχουν γράψει άρθρα).




Σχήμα 6.3 Παράδειγμα επερώτησης περιβάλλοντος σε βάθος στην ιεραρχία της επερώτησης MashQL.

Η ενδιάμεση επερώτηση δημιουργείται από τον πελάτη απευθείας σε SPARQL με χρήση μεθόδων Javascript.

Αρχικά δημιουργείται το pattern των τριάδων οι οποίες θα πρέπει να επιστραφούν χρησιμοποιώντας τους υπάρχοντες περιορισμούς οι οποίοι ήδη έχουν εισαχθεί από τον χρήστη, οι οποίοι αποθηκεύονται στον κώδικα HTML του δομοστοιχείου Query. Ταυτόχρονα, ανακτάται η επιλογή, δηλαδή το ποιο μέρος της τρέχουσας τριάδας θα πρέπει να επιστραφεί, αναλόγως αν μιλάμε για υποκείμενο -ισχύει μόνο για την πρώτη τριάδα και αποτελεί την «ρίζα» της επερώτησης, κατηγόρημα η αντικείμενο.

Έπειτα εισάγονται τα σύνολα δεδομένων εισόδου και τυχόν περιορισμοί στα σύνολα δεδομένων οι οποίοι προκύπτουν από επερωτήσεις σε προηγούμενα επίπεδα ενός Mashup. Εδώ θεωρούμε ότι μιλάμε για το πρώτο επίπεδο άρα εδώ δεν θα υπάρχουν. Αν υπήρχαν θα έπρεπε να τροποποιηθεί η επερώτηση ώστε να ληφθούν υπόψη αυτοί οι περιορισμοί, πράγμα που γίνεται στην τελική τροποποίηση της επερώτησης στον εξυπηρετητή εφαρμογών.

Έπειτα προωθούνται τα στοιχεία που ανακτήθηκαν πιο πάνω στον εξυπηρετητή εφαρμογών, όπου και γίνεται η τελική διαμόρφωση της ενδιάμεσης επερώτησης με χρήση του JSP callBquery.jsp.

Εκεί αρχικά ανακτώνται τα URL των συνόλων δεδομένων εισόδου. Έπειτα διαμορφώνονται οι τριάδες της επερώτησης οι οποίες σχετίζονται με τους περιορισμούς οι οποίοι προκύπτουν άμεσα από το τρέχον δομοστοιχείο Query με χρήση των περιορισμών/τριάδων οι οποίοι παραλαμβάνονται από τον πελάτη, με τρόπο που να συμβαδίζει με την SPARQL που χρησιμοποιεί το Virtuoso.

Το τι ακολουθεί εξαρτάται από το αν το τρέχον δομοστοιχείο Query αποτελεί ανώτερο επίπεδο κάποιου Mashup η όχι, αν έχει δηλαδή ενωμένες ως εισόδους του άλλες επερωτήσεις η όχι, καθώς και το αν είναι επιλεγμένη από τον χρήστη η επιστροφή υποκειμένου (και τότε αν πρόκειται για τύπους υποκειμένων η instances αυτών), κατηγορήματος ή αντικειμένου.

Να σημειωθεί ότι στην SPARQL (τα πιο κάτω παραδείγματα είναι απλά χάριν παραδείγματος) προστίθενται επιπλέον τριάδες οι οποίες αντιπροσωπεύουν τυχόν επιπλέον περιορισμούς οι οποίοι πιθανό να υπάρχουν, οπότε θα πρέπει να λαμβάνονται υπόψη. Επίσης σε περίπτωση επιλογής κατηγορήματος η αντικειμένου πιθανό αυτή η επιλογή να γίνεται αναδρομικά έπειτα από πολλά επίπεδα στην ιεραρχία, οπότε θα

επιλεγεί το αντικείμενο/κατηγόρημα της τριάδας στην ιεραρχία στην οποία βρίσκεται ο χρήστης.

1) Περίπτωση επιλογής τύπου υποκειμένων (πχ οι τύποι των υποκειμένων από το http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf)

Σε αυτή την περίπτωση δημιουργείται επερώτηση προς επιστροφή των αντικειμένων στα σύνολα δεδομένων εισόδου τα οποία ανήκουν σε τριάδα που έχει ως κατηγόρημα το rdf:type, το οποίο στο standard του RDF συμβολίζει τον τύπο ενός υποκειμένου που αντιπροσωπεύεται από ένα URI.

Για το πιο πάνω παράδειγμα

```
SELECT DISTINCT TOP 0, 50  O

FROM (

 SPARQL SELECT ?O

FROM   <http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf> WHERE {

{?S rdf:type ?O . }

} ) AS allofthem

WHERE CAST(O  AS VARCHAR) LIKE '**'

AND CAST(O  AS VARCHAR) LIKE '*http:\*'

group by O   order by O  ASC
```

Να σημειωθεί εδώ ότι –όπως περιγράφεται και στην περιγραφή τεχνολογιών- η SPARQL δεν προωθείται απλά απευθείας στην ΒΔ προς εκτέλεση αλλά ενσωματώνεται σε SQL αφού ο εσωτερικός μηχανισμός της ΒΔ μπορεί να δεκτεί (μέσω του JDBC) μόνο επερωτήσεις σε SQL. Η ενσωματωμένη SPARQL εκτελείται κατά την διάρκεια του σταδίου ανάκτησης πόρων εισόδου (δηλαδή του FROM) της εκτέλεσης της SQL δίνοντας έναν πίνακα ως ενδιάμεσο αποτέλεσμα.

2) Περίπτωση επιλογής ατομικών πόρων (individuals) (πχ όλοι οι κόμβοι παρόντες είτε ως υποκείμενα είτε ως αντικείμενα στο http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf)

Σε αυτή την περίπτωση χρειάζεται η επιστροφή όλων των υποκειμένων στις τριάδες των συνόλων δεδομένων εισόδου αλλά και όλων των αντικειμένων τα οποία αποτελούν πόρους RDF (δηλαδή που περιγράφονται μέσω ενός URI). Γι' αυτό η επερώτηση εδώ συνενώνει (UNION) στο επίπεδο της SQL δύο επερωτήσεις.

```
SELECT DISTINCT TOP 0, 50 S

FROM (

SPARQL SELECT ?S

FROM   <http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf> WHERE { {?S ?P1 ?O1 . } }

) AS allofthem

 WHERE S  LIKE '**'

UNION

SELECT DISTINCT TOP 0, 50 O1

FROM (

SPARQL SELECT ?O1 FROM <http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf>

WHERE { {?S ?P1 ?O1 . } } ) AS allofthem

 WHERE CAST(O1 AS VARCHAR) LIKE '**'

 AND CAST(O1 AS VARCHAR) LIKE '*http:\*'

 group by S ,01  order by S  ASC
```

3) Περίπτωση επιλογής αντικειμένων τιμών ενός κατηγορήματος ενός τύπου (πχ για το κατηγόρημα συγγραφέας όλων των αντικειμένων που είναι άρθρα)

Ομοίως με την περίπτωση επιλογής των τύπων υποκειμένων (που και αυτοί αποτελούν αντικείμενα) μόνο που εδώ πρέπει να εξετάζουμε για κάθε αντικείμενο αν αποτελεί RDF πόρο.

```
SELECT DISTINCT TOP 0, 50  O1

FROM (

SPARQL          SELECT          ?O1                    FROM
<http://data.semanticweb.org/dumps/conferences/eswc-2007-
complete.rdf>

WHERE              {              {?S              rdf:type
<http://data.semanticweb.org/ns/swc/ontology#Paper>   .   }
{?S <http://swrc.ontoware.org/ontology#author> ?O1   . } }
)

AS allofthem

WHERE CAST(O1   AS VARCHAR) LIKE '**' AND CAST(O1   AS
VARCHAR) LIKE '*http:\*' group by O1   order by O1  ASC
```

Να σημειωθεί εδώ ότι σε αυτή την επερώτηση υπάρχουν δύο τριάδες/περιορισμοί οι οποίοι αφορούν το ίδιο υποκείμενο αφού θέλουμε

1. Για όλα τα άρθρα

2. Τους συγγραφείς

<u>4) Περίπτωση επιλογής όλων των κατηγορημάτων που εντοπίζονται για τα υποκείμενα που ανήκουν σε ένα είδος</u>

```
SELECT DISTINCT TOP 0, 50  P1  FROM (

SPARQL          SELECT          ?P1                    FROM
<http://data.semanticweb.org/dumps/conferences/eswc-2007-
complete.rdf>
```

```
WHERE { {?S rdf:type <http://data.semanticweb.org/ns/swc/ontology#Paper> . } {?S ?P1 ?O1 . } } ) AS allofthem

WHERE CAST(P1 AS VARCHAR) LIKE '**' AND CAST(P1 AS VARCHAR) LIKE '*http:\*' group by P1 order by P1 ASC
```

Το ίδιο για την περίπτωση που αντί για τύπο υποκειμένου έχουμε συγκεκριμένο υποκείμενο με την διαφορά ότι η μεταβλητή ?S αντικαθίσταται από το υποκείμενο και η τριάδα `{?S rdf:type <http://data.semanticweb.org/ns/swc/ontology#Paper> . }` δεν υφίσταται αφού εδώ θέλουμε συγκεκριμένο υποκείμενο.

Ακολουθεί ένα παράδειγμα επιλογής ενός αντικειμένου σε χαμηλότερα επίπεδα της ιεραρχίας της επερώτησης (για περισσότερα σχετικά με το τι αποτελεί μια ιεραρχία επερώτησης MashQL η οποία έπειτα μεταφράζεται σε SPARQL στο κεφάλαιο της περιγραφής τεχνολογιών).

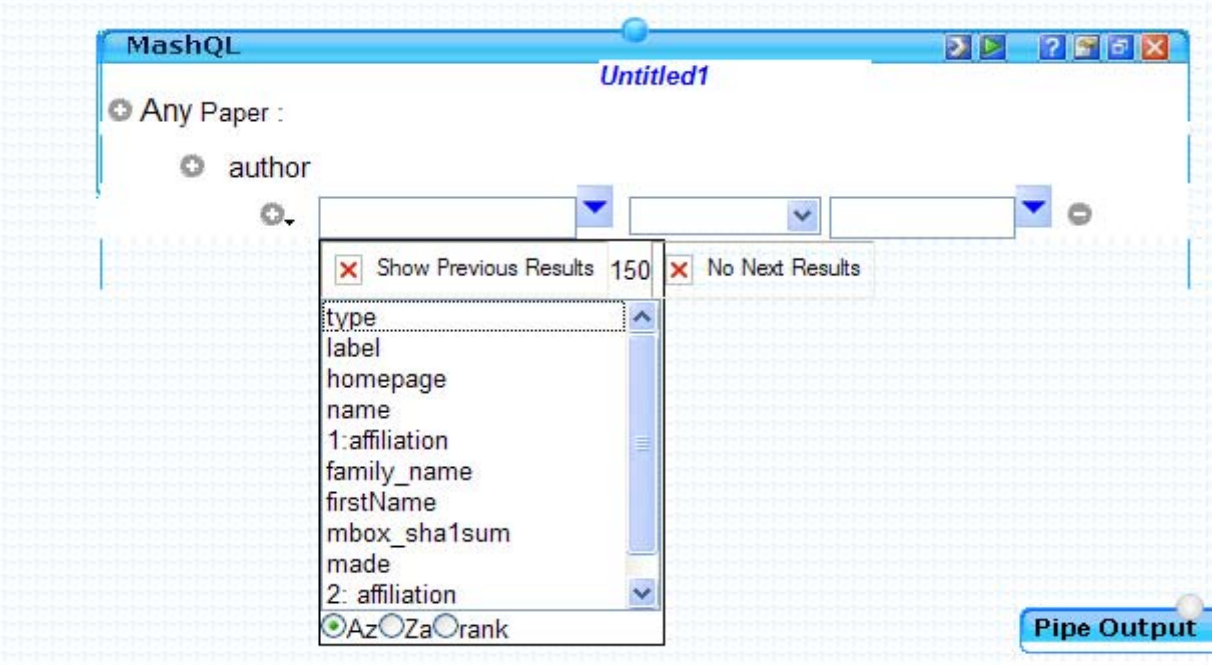
Εδώ θέλουμε τα κατηγορήματα/ιδιότητες των συγγραφέων που έγραψαν κάποιο άρθρο.

```
SELECT DISTINCT TOP 0, 50 P2

FROM (

SPARQL SELECT ?P2 FROM <http://data.semanticweb.org/dumps/conferences/eswc-2007-complete.rdf>

WHERE { {?S rdf:type <http://data.semanticweb.org/ns/swc/ontology#Paper> . } {?S <http://swrc.ontoware.org/ontology#author> ?O1 . }

{ ?O1 ?P2 ?O2 . } } )

AS allofthem WHERE CAST(P2 AS VARCHAR) LIKE '**' AND CAST(P2 AS VARCHAR) LIKE '*http:\*' group by P2 order by P2 ASC
```

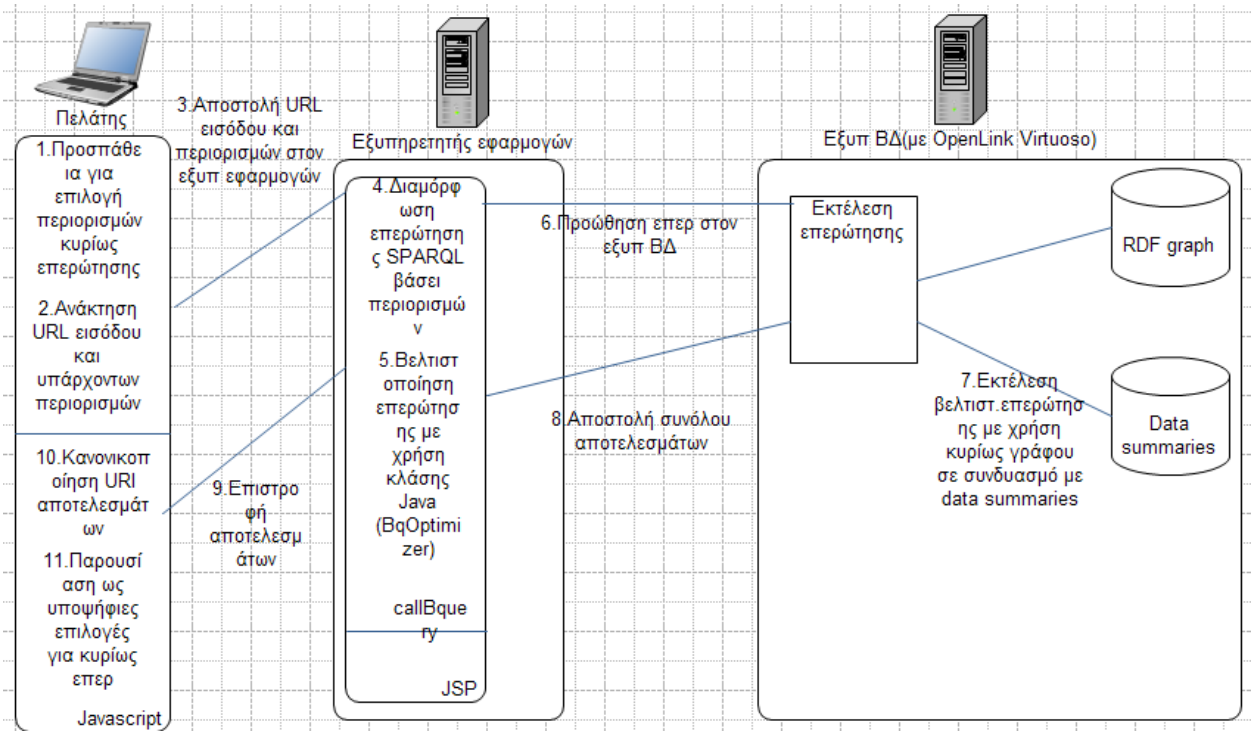
Όπως φαίνεται και από την επερώτηση η επιλογή μας (P2) αποτελεί το κατηγόρημα της $2^{ης}$ τριάδας στην οποία τα υποκείμενα αποτελούν τα αντικείμενα της $1^{ης}$ τριάδας. Επομένως θα πρέπει να ζητηθεί από την ΒΔ να ανακτήσει τις τριάδες που ικανοποιούν την $1^{η}$ τριάδα της επερώτησης και έπειτα να θέσει τα αντικείμενα της ως υποκείμενα της $2^{ης}$. Αυτό μπορεί να ισχύσει αναδρομικά και για περισσότερα των δύο επιπέδων, μέχρι ενός απροσδιόριστου αριθμού.

Αφού η επερώτηση δημιουργηθεί περνά από τον βελτιστοποιητή επερωτήσεων (ο οποίος μετατρέπει την επερώτηση σε μία που επιστρέφει ισοδύναμα αποτελέσματα χρησιμοποιώντας τα βοηθητικά summaries αντί των κυρίως γράφων των συνόλων δεδομένων εισόδου, όπως περιγράφεται σε επόμενο κεφάλαιο) και προωθείται προς την ΒΔ μέσω JDBC για εκτέλεση.

Τα αποτελέσματα επιστρέφονται στον πελάτη, όπου χρησιμοποιείται ο κανονικοποιητής URL για να τα μετατρέψει σε μορφή πιο κατανοητή από τον χρήστη και έπειτα παρουσιάζονται στην λίστα των επιλογών στην διεπαφή με τον χρήστη.




Σχήμα 6.4 Παρουσίαση αποτελεσμάτων επερώτησης περιβάλλοντος στην διεπαφή.




Σχήμα 6.5 Σχεδιάγραμμα που δείχνει πως εκτελούνται οι επερωτήσεις περιβάλλοντος

### 6.3 Εκτέλεση κυρίως επερώτησης

Αφού ο χρήστης έχει δημιουργήσει την κυρίως επερώτηση με χρήση των επερωτήσεων περιβάλλοντος και τον επιλογών από τα αποτελέσματα αυτών -ή με την εισαγωγή δικών του δεδομένων- μπορεί να εκτελέσει την τελική επερώτηση για το συγκεκριμένο δομοστοιχείο Query πατώντας το κουμπί εκτέλεσης στο άνω δεξιά μέρος του δομοστοιχείου.

Ο τρόπος με τον οποίο εκτελείται η κυρίως επερώτηση (η οποία δεν κάνει χρήση βελτιστοποιητή) είναι παρόμοιος με τον τρόπο που εκτελείτο όταν στην υλοποίηση του Κου Σαββίδη με την διαφορά στην σύνταξη της επερώτησης, η οποία τώρα είναι συμβατή με την SPARQL ενσωματωμένη σε SQL την οποία χρησιμοποιεί το Virtuoso.

### 6.4 Βοηθητικές λειτουργίες

Το σύστημα μπορεί επίσης να αποθηκεύσει στην ΒΔ και να ανακτήσει δημιουργημένα Pipes όπως η προηγούμενη υλοποίηση .

Χρησιμοποιούνται οι παρακάτω πίνακες.

RDF_DB.DBA.PIPE

Αποθηκεύει δεδομένα για ένα αποθηκευμένο Pipe.

| **PID**(INTEGER,PRIMARY KEY,AUTO-INCREMENT) | UUSERID(INTEGER) | TITLE(VARCHAR) | CREATEDATE(DATE) |
|---|---|---|---|

| LASTMOD(DATE) | ISPUBLIC(INTEGER) | DESCRIPTION(VARCHAR) | MARKUP(LONG VARCHAR) |
|---|---|---|---|

Πίνακας 6.3 Πίνακας RDF_DB.DBA.PIPE στη ΒΔ

Συγκεκριμένα

UUSERID: Αναγνωριστικός αριθμός του χρήστη στον οποίο ανήκει η διασωλήνωση.

MARKUP: Ο κώδικας σε XML που αντιπροσωπεύει την διασωλήνωση.

Τα υπόλοιπα πεδία είναι αυτοεπεξηγηματικά.

RDF_DB.DBA.USER1

Αποθηκεύει δεδομένα για έναν χρήστη.

| USERID(INTEGER,PRIMARY KEY,AUTO-INCREMENT) | NAME(VARCHAR) | PSWD(VARCHAR) |
|---|---|---|

Πίνακας 6.4 Πίνακας RDF_DB.DBA.USER1 στη ΒΔ

# Κεφάλαιο 7

## Επεξεργασία Mashups



### 7.1 Προηγούμενη λογική -υλοποίηση

Τα Mashups (όπως περιγράφεται στην περιγραφή τεχνολογιών) επιτρέπουν την σύσμιξη δεδομένων από διάφορες πηγές ώστε μέσω του συνδυασμού τους να προκύψουν νέα δεδομένα. Στις επερωτήσεις σε μεταδεδομένα RDF αυτό σημαίνει την συνένωση αποτελεσμάτων από διαφορετικές επερωτήσεις και την εκτέλεση επερωτήσεων στο σύνολο αυτών των δεδομένων η οποία τα συνενώνει και εξάγει συνολικά αποτελέσματα.

Στην υλοποίηση του Κου Σαββίδη εκείνο που γίνεται είναι ότι όταν επερωτήσεις που έχουν ήδη εκτελεστεί σε δομοστοιχεία Query τεθούν ως εισόδοι σε άλλα δομοστοιχεία ο κώδικας Oracle SPARQL που αποτελεί αυτές τις επερωτήσεις προσαρμόζεται ώστε να μπορεί να χρησιμοποιηθεί ως είσοδος στην επερώτηση του δομοστοιχείου προορισμού. Ο κώδικας αυτός αποτελεί έπειτα ένα σύνολο επιπλέον περιορισμών στα αποτελέσματα της επερώτησης του δομοστοιχείου προορισμού ούτως ώστε να ληφθούν υπόψη οι περιορισμοί της προηγούμενης επερώτησης. Περιορισμοί περισσοτέρων της μίας προηγούμενων επερωτήσεων μπορούν να ληφθούν υπόψη, και το κάθε σύνολο αυτών ισχύει για τα σύνολα δεδομένων τα οποία συσχετίζονται με την επερώτηση στην οποία είχαν τεθεί αρχικά. Επίσης συνδυάζονται τα σύνολα δεδομένων εισόδου.

Αφού τροποποιηθεί ο κώδικας των προηγούμενων επερωτήσεων, όταν δοθεί εντολή για εκτέλεση ενδιάμεσης επερώτησης ή τελικών επερωτήσεων οι περιορισμοί αυτοί (μαζί με τα σύνολα δεδομένων του καθενός) απλά συνενώνονται μέσω τελεστή UNION στο επίπεδο της SQL και τίθενται ως συνολικοί επιπλέον περιορισμοί στα αποτελέσματα της επερώτησης του τρέχοντος δομοστοιχείου.

Πχ αν έχουμε τις επερωτήσεις προηγούμενων δομοστοιχείων

```
SELECT X2
FROM
TABLE(SEM_MATCH('(<http://data.semanticweb.org/ns/swc/onto
logy#Paper>      <http://swrc.ontoware.org/ontology#author>
?X1)(?X1      <http://xmlns.com/foaf/0.1/name>      ?X2)(?X1
:http://xmlns.com/foaf/0.1/name
?X3)',SEM_MODELS('SFSD'),null,null,null))
WHERE X3 = '%a%'
```

Και

```
SELECT X1
FROM   TABLE(SEM_MATCH('(<http://xmlns.com/foaf/0.1/Person>
<http://xmlns.com/foaf/0.1/name> ?X1)
(<http://xmlns.com/foaf/0.1/Person>
<http://xmlns.com/foaf/0.1/name>
?X2)',SEM_MODELS('dgdfg'),null,null,null))
WHERE X2 = '%b%'
```

Θα συνδυαστούν δίνοντας την παρακάτω επερώτηση.

```
SELECT    XXX    FROM    TABLE(SEM_MATCH('(?S    ?P    ?O)(?S
<http://xmlns.com/foaf/0.1/name>                    ?X2)(?O
:http://xmlns.com/foaf/0.1/name
?X3)',SEM_MODELS('SFSD'),null,null,null))
WHERE                          S                          =
<http://data.semanticweb.org/ns/swc/ontology#Paper>  AND  P
```

```
= <http://swrc.ontoware.org/ontology#author> AND X3 =
'%a%' OR P = <http://xmlns.com/foaf/0.1/name>
OR P = 'http://xmlns.com/foaf/0.1/name'

UNION

SELECT XXX FROM TABLE(SEM_MATCH('(?S ?P
?O)(?http://xmlns.com/foaf/0.1/Person>
<http://xmlns.com/foaf/0.1/name>
?X2)',SEM_MODELS('dgdfg'),null,null,null))
WHERE S = <http://xmlns.com/foaf/0.1/Person> AND P =
<http://xmlns.com/foaf/0.1/name> AND X2 = '%b%' OR P =
<http://xmlns.com/foaf/0.1/name>)
```

Οι επιλογές έχουν αφαιρεθεί και αντικατασταθεί από μεταβλητές και το ίδιο τυχόν σταθερές στο πρώτο κλαδί κάθε επερώτησης, με την θέση τους να παίρνουν ισοδύναμα φιλτραρίσματα μέσω WHERE στο τέλος της επεξεργασίας της επερώτησης. Αυτό γίνεται ούτως ώστε να μπορούν να αντικατασταθούν οι επιλογές και οι σταθερές με την επιλογή του χρήστη στην εξωτερική επερώτηση η οποία αποτελεί την τρέχουσα επερώτηση, ώστε να μπορούν έτσι να ενοποιηθούν οι επερωτήσεις.

Τελικά θα προστεθεί η πιο πάνω επερώτηση ως φίλτρο με χρήση <επιλογή> IN <πιο πάνω επερωτήσεις, στις οποίες το XXX θα αντικατασταθεί με την επιλογή της εξωτερικής επερώτησης, υποκείμενο κατηγόρημα η αντικείμενο> στην εξωτερική επερώτηση.

Οπότε τελικά θα έχουμε επερώτηση όπως την πιο κάτω.

```
SELECT P FROM TABLE(SEM_MATCH('(?S ?P ?O)(?S <predicate>
?X2)',SEM_MODELS('SFSDss'),null,null,null))
WHERE P IN(SELECT P FROM TABLE(SEM_MATCH('(?S ?P ?O)(?S
<http://xmlns.com/foaf/0.1/name> ?X2)(?O
:http://xmlns.com/foaf/0.1/name
?X3)',SEM_MODELS('SFSD'),null,null,null))
```

```
WHERE S =
<http://data.semanticweb.org/ns/swc/ontology#Paper> AND P
= <http://swrc.ontoware.org/ontology#author> AND X3 =
'%a%' OR P = <http://xmlns.com/foaf/0.1/name>
OR P = 'http://xmlns.com/foaf/0.1/name'
UNION
SELECT P FROM TABLE(SEM_MATCH('(?S ?P
?O)(?http://xmlns.com/foaf/0.1/Person>
<http://xmlns.com/foaf/0.1/name>
?X2)',SEM_MODELS('dgdfg'),null,null,null))
WHERE S = <http://xmlns.com/foaf/0.1/Person> AND P =
<http://xmlns.com/foaf/0.1/name> AND X2 = '%b%' OR P =
<http://xmlns.com/foaf/0.1/name>)
```

### 7.2 Προβλήματα

Παρόλο που η τεχνική αυτή μέχρι και το δεύτερο επίπεδο ενός Mashup είναι αποτελεσματική και λογική έχει τα εξής προβλήματα:

Α) Σε Mashups με πέραν των δύο επιπέδων οι επερωτήσεις γίνονται εξαιρετικά πολύπλοκες αυξάνοντας την δυνατότητα εμφάνισης σφαλμάτων τα οποία δεν είχαν προηγουμένως εντοπιστεί αφού μιλάμε ουσιαστικά για αναδρομικές επερωτήσεις.

Β) Το κόστος της χρήσης του IN σε μια επερώτηση σε μεγάλο σύνολο δεδομένων όταν συνδυάζεται μέσω αυτού με άλλες επερωτήσεις σε μεγάλα σύνολα δεδομένων είναι μεγάλο. Το ίδιο και το κόστος της αντικατάστασης πολλών σταθερών με μεταβλητές με το φιλτράρισμα να γίνεται στο τέλος της εκτέλεσης των προηγούμενων επερωτήσεων στο επίπεδο της SQL. Λόγω αυτών και του (Α) αυξάνεται σημαντικά ο χρόνος επεξεργασίας. Αυτό ισχύει λόγω του ότι το IN απαιτεί πολύ χρόνο για έλεγχο του συνδυασμού κάθε στοιχείου του ενδιάμεσου αποτελέσματος της εξωτερικής επερώτησης με κάθε στοιχείο της ένωσης των προηγούμενων επερωτήσεων και λόγω του ότι η χρήση μεταβλητών στη θέση σταθερών δημιουργεί πολύ μεγαλύτερα ενδιάμεσα αποτελέσματα.

Το Β παρατηρήθηκε στην πράξη αφού ακόμη και απλές ενδιάμεσες επερωτήσεις στο δεύτερο επίπεδο Mashup (όπως «επέλεξε τους τύπους όλων των στοιχείων που

επιστρέφει η συνένωση των επερωτήσεων του προηγούμενου επιπέδου») έπαιρναν απαράδεκτο χρόνο, περισσότερο από ένα λεπτό.

Γι' αυτό αποφάσισα να βελτιστοποιήσω τον τρόπο που συνδυάζονται οι επερωτήσεις στο πλαίσιο των Mashups προσπαθώντας να απαλείψω τα πιο πάνω προβλήματα.

### 7.3 Λογική -υλοποίηση νέας μεθόδου συνδυασμού

Αντί να προστίθενται οι περιορισμοί των επερωτήσεων προηγούμενων επιπέδων στην επερώτηση του τρέχοντος επιπέδου στο επίπεδο της SQL προσαρμοσμένοι στην τρέχουσα επερώτηση και να χρησιμοποιούνται για το φιλτράρισμα των αποτελεσμάτων της τρέχουσας επερώτησης να γίνεται το εξής: Να προσαρμόζονται οι περιορισμοί της τρέχουσας επερώτησης ξεχωριστά σε κάθε επερώτηση του αμέσως προηγούμενου επιπέδου και να προστίθενται σε κάθε επερώτηση του αμέσως προηγούμενου επιπέδου, η οποία θα εκτελείται βάσει και αυτών των περιορισμών. Τέλος να συνενώνονται τα αποτελέσματα όλων των επερωτήσεων του αμέσως προηγούμενου επιπέδου μέσω UNION ώστε να παίρνουμε τα συνολικά αποτελέσματα για όλα τα σύνολα δεδομένων τα οποία τώρα θα λαμβάνουν υπόψη και τους περιορισμούς της τρέχουσας επερώτησης.

Αυτό υλοποιείται μέσω της αναζήτησης για κάθε προηγούμενη επερώτηση της ρίζας της και της εύρεσης του αντίστοιχου μέρους της πρώτης τριάδας της ρίζας μέσω του οποίου θα συνενωθούν οι περιορισμοί της τρέχουσας επερώτησης με την προηγούμενη (πχ αν η ρίζα της προηγούμενης επερώτησης είναι το S και η πρώτη της τριάδα

{?S ?P1 ?O1}

τότε η συνένωση θα γίνει στο P1 αν η τρέχουσα επερώτηση έχει ως ζητούμενο προς επιστροφή την τιμή κάποιου κατηγορήματος σε κάποιο επίπεδο του τρέχοντος κλαδιού του δέντρου της τρέχουσας επερώτησης). Έπειτα γίνεται συνένωση με τους περιορισμούς της τρέχουσας επερώτησης μέσω: 1. Της αλλαγής των ονομάτων των μεταβλητών της τρέχουσας επερώτησης ώστε να μην υπάρξει σύγκρουση με ονόματα μεταβλητών της προηγούμενης επερώτησης. 2. Της αλλαγής του ονόματος του μέρους της τρέχουσας επερώτησης πάνω στο οποίο θα γίνει η συνένωση ώστε να είναι το ίδιο με αυτό του αντίστοιχου μέρους της προηγούμενης επερώτησης πάνω στο οποίο θα γίνει η συνένωση.

Για παράδειγμα έστω ότι έχουμε στο πρώτο επίπεδο ενός Mashup τις ακόλουθες επερωτήσεις:

```
SELECT X2
  FROM(SPARQL SELECT ?X2
,?X3
 FROM  <http://data.semanticweb.org/dumps/conferences/eswc-
2007-complete.rdf>
 WHERE {
{{?S      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://data.semanticweb.org/ns/swc/ontology#Paper>}}
{{?S <http://swrc.ontoware.org/ontology#author> ?X1}}
{{?X1 <http://xmlns.com/foaf/0.1/name> ?X2}}
{{?X1    <http://xmlns.com/foaf/0.1/name>    ?X3}}})    AS
SELECTION
  WHERE  ( CAST(X3 AS VARCHAR)  LIKE '*a*')
```

Και

```
SELECT X1
  FROM(SPARQL SELECT ?X1
,?X2
 FROM  <http://data.semanticweb.org/dumps/conferences/iswc-
aswc-2007-complete.rdf>
 WHERE {
{{?S      <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person>}}
{{?S <http://xmlns.com/foaf/0.1/name> ?X1}}
{{?S <http://xmlns.com/foaf/0.1/name> ?X2}}}) AS SELECTION
  WHERE  ( CAST(X2 AS VARCHAR)  LIKE '*b*')
```

Και ότι θέλουμε στο δεύτερο επίπεδο του Mashup να μας επιστραφούν μέσω ενδιάμεσης επερώτησης όλα τα υποκείμενα των δεδομένων που επιστρέφει η ένωση των πιο πάνω επερωτήσεων.

Τότε η τελική επερώτηση που θα διαμορφωθεί θα είναι η εξής:

```
SELECT DISTINCT  X56 FROM
(SPARQL SELECT ?X56,  ?S, ?P, ?O, ?X2,?X3
FROM  <http://data.semanticweb.org/dumps/conferences/eswc-
2007-complete.rdf>
WHERE   {{{?S   <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>
<http://data.semanticweb.org/ns/swc/ontology#Paper>}}{{?S
<http://swrc.ontoware.org/ontology#author>      ?X1}}{{?X1
<http://xmlns.com/foaf/0.1/name>                ?X2}}{{?X1
<http://xmlns.com/foaf/0.1/name> ?X3}}{{?S rdf:type ?X55 .
}} {{ ?S ?X56 ?X57  . }} })
AS SELECTION  WHERE  ( CAST(X3 AS VARCHAR)  LIKE '*a*')

UNION

SELECT DISTINCT  X55 FROM
(SPARQL SELECT ?X55,  ?X1,?X2
FROM  <http://data.semanticweb.org/dumps/conferences/iswc-
aswc-2007-complete.rdf>
WHERE   {{{?S   <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>          <http://xmlns.com/foaf/0.1/Person>}}{{?S
<http://xmlns.com/foaf/0.1/name>                ?X1}}{{?S
<http://xmlns.com/foaf/0.1/name> ?X2}}{{?S rdf:type ?X54 .
}} {{ ?S ?X55 ?X56  . }} })
AS SELECTION  WHERE  ( CAST(X2 AS VARCHAR)  LIKE '*b*')
```

### 7.4 Πλεονεκτήματα σε σχέση με προηγούμενη υλοποίηση

Έχουμε τα εξής πλεονεκτήματα σε σχέση με την προηγούμενη υλοποίηση:

Α) Αντί να δημιουργούνται ενδιάμεσα αποτελέσματα μετά την εκτέλεση της τρέχουσας επερώτησης χρησιμοποιώντας μόνο τους τρέχοντες περιορισμούς σε όλα τα χρησιμοποιούμενα από τις προηγούμενες επερωτήσεις σύνολα δεδομένων γίνεται το εξής: Έχουμε για κάθε ομάδα συνόλων δεδομένων κάθε προηγούμενης επερώτησης ενδιάμεσα αποτελέσματα με μικρότερο μέγεθος αφού λαμβάνονται υπόψη και οι

περιορισμοί της προηγούμενης επερώτησης. Έτσι αποφεύγουμε μεγάλα σύνολα ενδιάμεσων αποτελεσμάτων τα οποία μετά θα πρέπει να φιλτραριστούν.

Β) Χρησιμοποιείται τροποποίηση των προηγούμενων επερωτήσεων με προσθήκη νέων περιορισμών απευθείας στο επίπεδο της SPARQL. Έτσι έχουμε σημαντικά μικρότερα μεγέθη ενδιάμεσων αποτελεσμάτων μετά την εκτέλεση του επιπέδου επερώτησης SPARQL τα οποία χρησιμοποιούνται ως εισόδοι στο επίπεδο της επερώτησης SQL η οποία κάνει wrap την επερώτηση SPARQL.

Γ) Αφού πλέον δεν χρειάζεται η χρήση του IN στο επίπεδο της SQL δεν χρειάζεται πλέον η αντικατάσταση σταθερών με μεταβλητές στο επίπεδο των προηγούμενων επερωτήσεων, οπότε οι επερωτήσεις στο επίπεδο της SPARQL γίνονται με χρήση σταθερών οδηγώντας και σε μικρότερο μέγεθος ενδιάμεσων αποτελεσμάτων μετά την εκτέλεση της SPARQL.

# Κεφάλαιο 8

## Summaries-δομές βελτιστοποίησης



### 8.1 Ανάγκη για δημιουργία δομών βελτιστοποίησης

Οι επερωτήσεις περιβάλλοντος υποχρεωτικά θα πρέπει να επιστρέφουν τα ίδια αποτελέσματα που θα περιείχαν αν εκτελούνταν στο σύνολο των δεδομένων των γράφων εισόδου. Αν εκτελούνται στο σύνολο των δεδομένων των γράφων αυτό θα έχει ως αποτέλεσμα ο χρόνος για την εκτέλεση των επερωτήσεων περιβάλλοντος να κινείται στα ίδια επίπεδα με τους χρόνους των τελικών επερωτήσεων. Αυτό είναι ανεπίτρεπτο αφού αυτό θα κατέστρεφε την χρηστικότητα του συστήματος αφού οι επερωτήσεις περιβάλλοντος πρέπει να επιστρέφουν τα αποτελέσματα τους σε ελάχιστο χρόνο, αν είναι δυνατόν σχεδόν στιγμιαία, αφού χρησιμοποιούνται για επιλογή επιλογών από τον χρήστη όσο αφορά την τελική επερώτηση.

Γι’ αυτό και χρειάστηκε η χρήση δομών οι οποίες να συνοψίζουν τα δεδομένα και οι οποίες να έχουν την δυνατότητα να χρησιμοποιηθούν για την απάντηση επερωτήσεων περιβάλλοντος επιστρέφοντας τα ίδια αποτελέσματα. Έτσι έχουμε σημαντικά μειωμένους χρόνους για πρόσβαση και επεξεργασία δεδομένων αλλά και σημαντικά μικρότερα ενδιάμεσα αποτελέσματα κατά την διάρκεια της επεξεργασίας των επερωτήσεων (κάτι που επίσης συνεπάγεται σημαντικά μειωμένο χρόνο επεξεργασίας αλλά και μείωση του φόρτου στην κυρίως μνήμη του εξυπηρετητή ΒΔ).

### 8.2 Επιλογή δομών βελτιστοποίησης-λογική

Στο [7] περιγράφεται ένα σύστημα summaries τα οποία συνοψίζουν τα δεδομένα και επίσης ένας αλγόριθμος για δημιουργία έτοιμων αποτελεσμάτων για επερωτήσεις πολλών επιπέδων σε βάθος.

Συγκεκριμένα, διαχωρίζονται οι επερωτήσεις σε δύο ειδών, με τα summaries να χρησιμοποιούνται για απάντηση του πρώτου είδους (γενικές επερωτήσεις περιβάλλοντος) και ο αλγόριθμος (αλγόριθμος BR) για απάντηση του δεύτερου.

Στα πλαίσια αυτής της ΑΔΕ έχω υλοποιήσει για το OpenLink Virtuoso το σύστημα για τις γενικές επερωτήσεις περιβάλλοντος. Λόγω του ότι η ύπαρξη των Mashups διαμορφώνει κάπως διαφορετικές απαιτήσεις όσο αφορά τις περιπτώσεις γενικών επερωτήσεων περιβάλλοντος που απαιτούνται τα υλοποιημένα ευρετήρια στην υλοποίηση αυτού του συστήματος διαφέρουν σε σχέση με αυτά στο [7].

Γενικές επερωτήσεις περιβάλλοντος

Το ενοποιητικό χαρακτηριστικό τους (που τις διαχωρίζει από τις επερωτήσεις του δεύτερου είδους) είναι ότι δεν απαιτούν επέκταση της επερώτησης σε βάθος στον γράφο.

Πρόκειται για τις εξής:

**Α) Επέλεξε τα υποκείμενα του γράφου τα οποία έχουν τύπο**

SELECT ?S

FROM <GRAPH>

WHERE {?S rdf:type ?O}

**Β) Επέλεξε τα υποκείμενα του γράφου τα οποία έχουν τύπο και τον τύπο τους**

SELECT ?S,?O

FROM <GRAPH>

WHERE {?S rdf:type ?O}

**Γ) Επέλεξε τα υποκείμενα και τα κατηγορήματα του γράφου**

SELECT ?S,?P

FROM <GRAPH>

WHERE {?S ?P ?O}

**Δ) Επέλεξε τα υποκείμενα και τα αντικείμενα του γράφου**

SELECT ?S,?O

FROM <GRAPH>

WHERE {?S ?P ?O}

**Ε) Επέλεξε τα υποκείμενα του γράφου**

SELECT ?S

FROM <GRAPH>

WHERE {?S ?P ?O}

**ΣΤ) Επέλεξε τα κατηγορήματα ενός συγκεκριμένου υποκειμένου**

SELECT ?P

FROM <GRAPH>

WHERE{<SUBJECT> ?P ?O}

**Ζ) Επέλεξε τα αντικείμενα ενός συγκεκριμένου υποκειμένου**

SELECT ?O

FROM <GRAPH>

WHERE{<SUBJECT> ?P ?O}

**Η) Επέλεξε το αντικείμενο ενός συγκεκριμένου υποκειμένου με το οποίο συνδέεται με ένα συγκεκριμένο κατηγόρημα**

SELECT ?O

FROM <GRAPH>

WHERE{<SUBJECT> <PREDICATE> ?O}

Το ίδιο για οποιαδήποτε περίπτωση όπου έχουμε δύο συγκεκριμένα μέλη της τριάδας ως συγκεκριμένους κόμβους και θέλουμε να επιστραφεί ο τρίτος.

**Θ) Επέλεξε όλα τα υποκείμενα που εμπεριέχουν ένα συγκεκριμένο κατηγόρημα και τα αντικείμενα με τα οποία συνδέονται μέσω αυτού του κατηγορήματος.**

SELECT ?S,?O

FROM <GRAPH>

WHERE{?S <PREDICATE> ?O}

**Ι) Επέλεξε όλα τα υποκείμενα που εμπεριέχουν ένα συγκεκριμένο κατηγόρημα.**

SELECT ?S

FROM <GRAPH>

WHERE {?S <PREDICATE> ?O}

Τα προβλήματα με την απόδοση κατά την εκτέλεση αυτών των επερωτήσεων οφείλονται στον τρόπο με τον οποίο δηλώνονται με χρήση SPARQL αφού ενώ χρειάζεται να επιστραφεί μόνο ένα πεδίο του γράφου ο κώδικας SPARQL που τις αποτελεί αναγκάζει την ΒΔ να επεξεργαστεί τα δεδομένα όλων των πεδίων του γράφου και να επιλέξει το πεδίο που πρέπει να επιστραφεί στο τέλος. Αυτό οδηγεί και σε ενδιάμεσα αποτελέσματα πολύ μεγαλύτερα από ότι θα ήταν εφικτό αν χρησιμοποιούνταν ευρετήρια που να συνοψίζουν τους γράφους επιστρέφοντας τα απαιτούμενα και μόνο δεδομένα.

Επομένως δημιουργούνται τα εξής ευρετήρια για κάθε γράφο που εισάγεται στη ΒΔ:

1) Για την απάντηση της επερώτησης Α πίνακας με μόνο τα υποκείμενα του γράφου τα οποία έχουν δηλωμένο τύπο.

RDF_DB.DBA.RDFINDEX_ SUBJECTOFTYPE_<graph URI>

| **S**(VARCHAR,PRIMARY KEY) |
|---|

2) Για την απάντηση της επερώτησης Β πίνακας με τα υποκείμενα του γράφου που έχουν δηλωμένο τύπο και τον τύπο τους.

RDF_DB.DBA.RDFINDEX_ SOOFTYPE_<graph URI>

| **S**(VARCHAR,PRIMARY KEY) | **O**(VARCHAR,PRIMARY KEY) |
|---|---|

3) Για την απάντηση της επερώτησης Γ και της Ι πίνακας με τα υποκείμενα του γράφου και τα κατηγορήματα τους.

RDF_DB.DBA.RDFINDEX_ SP_<graph URI>

| **S**(VARCHAR,PRIMARY KEY) | **P**(VARCHAR,PRIMARY KEY) |
|---|---|

4) Για την απάντηση της επερώτησης Δ πίνακας με τα υποκείμενα του γράφου και τα αντικείμενα τους.

RDF_DB.DBA.RDFINDEX_ SO_<graph URI>

| **S**(VARCHAR,PRIMARY KEY) | **O**(VARCHAR,PRIMARY KEY) |
|---|---|

5) Για την απάντηση της επερώτησης Ε πίνακας με τα υποκείμενα του γράφου.

RDF_DB.DBA.RDFINDEX_ SUBJECT_<graph URI>

| **S**(VARCHAR,PRIMARY KEY) |
|---|

Ομοίως και για τα κατηγορήματα (επερώτηση ΣΤ) και τα αντικείμενα (επερώτηση Ζ).

6) Για τις επερωτήσεις Η και Θ (που χρειάζεται υποχρεωτικά επεξεργασία και των τριών μερών κάθε τριάδας) αναγκαστικά χρησιμοποιείται ο αρχικός γράφος.

### 8.3 Δημιουργία και χρήση summaries

Τα πιο πάνω ευρετήρια δημιουργούνται/ανανεώνονται κάθε φορά που φορτώνεται ένα σύνολο δεδομένων/γράφος στην ΒΔ.

Όταν προωθείται από τον πελάτη προς τον εξυπηρετητή εφαρμογών οδηγία για επερώτηση περιβάλλοντος ο εξυπηρετητής εφαρμογών, αφού διαμορφώσει την αρχική επερώτηση καλεί κλάση Java η οποία είναι υπεύθυνη να διαχωρίσει την επερώτηση

στα συνιστώντα μέρη της. Όταν πρόκειται για Mashup στις επιμέρους επερωτήσεις, και έπειτα κάθε επιμέρους επερώτηση στα «κλαδιά» του δέντρου επερώτησης (βλέπε περιγραφή τεχνολογιών για περιγραφή των δέντρων επερωτήσεων MashQL). Έπειτα βελτιστοποιείται ξεχωριστά κάθε «κλαδί» αντικαθιστώντας την επιμέρους υπο-επερώτηση την οποία αποτελεί με μια βελτιστοποιημένη υπο-επερώτηση.

Παράδειγμα:

Έχουμε την επερώτηση (Mashup)

```
SELECT DISTINCT  X55 FROM
(SPARQL SELECT ?X55,  ?S, ?O, ?X2,?X3
FROM   <http://data.semanticweb.org/dumps/conferences/eswc-
2007-complete.rdf>
WHERE    {{{?S    <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>
<http://data.semanticweb.org/ns/swc/ontology#Paper>}}{{?S
<http://swrc.ontoware.org/ontology#author>       ?X1}}{{?X1
<http://xmlns.com/foaf/0.1/name>                 ?X2}}{{?X1
<http://xmlns.com/foaf/0.1/name> ?X3}}{{?S rdf:type ?X55 .
}} })
AS SELECTION   WHERE   ( CAST(X3 AS VARCHAR)   LIKE '*a*')
AND CAST(X55 AS  VARCHAR)  LIKE '*http:\*'
UNION
SELECT DISTINCT  X54 FROM
(SPARQL SELECT ?X54,  ?X1,?X2
FROM   <http://data.semanticweb.org/dumps/conferences/iswc-
aswc-2007-complete.rdf>
WHERE    {{{?S    <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>          <http://xmlns.com/foaf/0.1/Person>}}{{?S
<http://xmlns.com/foaf/0.1/name>                 ?X1}}{{?S
<http://xmlns.com/foaf/0.1/name> ?X2}}{{?S rdf:type ?X54 .
}} })
AS SELECTION   WHERE   ( CAST(X2 AS VARCHAR)   LIKE '*b*')
AND CAST(X54 AS  VARCHAR)  LIKE '*http:\*'
```

Η επερώτηση αυτή αποτελεί ένα Mashup μέσω του οποίου ζητούμε να επιστραφούν οι τύποι όλων των υποκειμένων (από $2^{ο}$ επίπεδο Mashup) τα οποία έχουν τύπο Paper και έχουν συγγραφέα με όνομα που περιέχει το γράμμα a (από $1^{ο}$ επίπεδο Mashup, $1^{η}$ επερώτηση) μαζί με τους τύπους όλων των υποκειμένων (από $2^{ο}$ επίπεδο Mashup) τα οποία έχουν τύπο Person και τα οποία έχουν όνομα που περιέχει το γράμμα b (από $1^{ο}$ επίπεδο mashup, $2^{η}$ επερώτηση).

Αυτό που γίνεται εδώ είναι το εξής:

1) Διαχωρισμός των δύο επιμέρους επερωτήσεων που αποτελούν το UNION («σπάσιμο» του Mashup).

2) Για την κάθε μία εξ αυτών εξαγωγή των «κλαδιών» του δέντρου επερώτησης -πχ για την $2^{η}$ επερώτηση έχουμε τα «κλαδιά»

i. `{{?S <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person>}}`

ii. `{{?S <http://xmlns.com/foaf/0.1/name> ?X1}}`

iii. `{{?S <http://xmlns.com/foaf/0.1/name> ?X2}}`

iv. `{{?S rdf:type ?X54 . }}`

3) Βελτιστοποίση κάθε κλαδιού ξεχωριστά.

Πχ για το κλαδί ii πιο πάνω θα έχουμε ως βελτιστοποιημένη επερώτηση

```
SELECT S AS S FROM RDF_DB.DBA."RDFINDEX_SP_http://data.semanticweb.org/dumps/conferences/iswc-aswc-2007-complete.rdf" WHERE P = 'http://xmlns.com/foaf/0.1/name')AS SELECTION2
```

4) Συνένωση των βελτιστοποιημένων επερωτήσεων ώστε να επαναδημιουργήσουμε το (βελτιστοποιημένο) δέντρο επερωτήσεων και το Mashup.

Πχ εδώ θα έχουμε τελικά

```
SELECT DISTINCT  X55 FROM
(SELECT S AS S
FROM(SPARQL SELECT ?S
FROM   <http://data.semanticweb.org/dumps/conferences/eswc-
2007-complete.rdf>
WHERE     {?S     <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>
<http://data.semanticweb.org/ns/swc/ontology#Paper>})AS
SELECTSPARQL)AS SELECTION1,
 (SELECT DISTINCT S FROM(SPARQL SELECT ?S
FROM   <http://data.semanticweb.org/dumps/conferences/eswc-
2007-complete.rdf>
WHERE   {{{?S   <http://swrc.ontoware.org/ontology#author>
?X1}}{{?X1    <http://xmlns.com/foaf/0.1/name>    ?X2}}})AS
SELECTSPARQL)AS SELECTION2,
 (SELECT S AS S,O AS X55
FROM
RDF_DB.DBA."RDFINDEX_SOOFTYPE_http://data.semanticweb.org/
dumps/conferences/eswc-2007-complete.rdf" )AS
SELECTION3
WHERE  SELECTION1.S  =  SELECTION2.S  AND  SELECTION2.S  =
SELECTION3.S   AND CAST(X55 AS  VARCHAR)  LIKE '*http:\*'

UNION

SELECT DISTINCT  X54 FROM
(SELECT S AS S
FROM(SPARQL SELECT ?S
 FROM  <http://data.semanticweb.org/dumps/conferences/iswc-
aswc-2007-complete.rdf>
WHERE     {?S     <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>            <http://xmlns.com/foaf/0.1/Person>})AS
SELECTSPARQL)AS
```

```
SELECTION1,
(SELECT S AS S
FROM
RDF_DB.DBA."RDFINDEX_SP_http://data.semanticweb.org/dumps/
conferences/iswc-aswc-2007-complete.rdf"
WHERE P = 'http://xmlns.com/foaf/0.1/name')AS
SELECTION2,
 (SELECT S AS S
FROM
RDF_DB.DBA."RDFINDEX_SP_http://data.semanticweb.org/dumps/
conferences/iswc-aswc-2007-complete.rdf"
WHERE P = 'http://xmlns.com/foaf/0.1/name')AS
 SELECTION3,
 (SELECT S AS S,O AS X54
FROM
RDF_DB.DBA."RDFINDEX_SOOFTYPE_http://data.semanticweb.org/
dumps/conferences/iswc-aswc-2007-complete.rdf" )AS
SELECTION4

WHERE SELECTION1.S = SELECTION2.S AND SELECTION2.S =
SELECTION3.S AND SELECTION3.S = SELECTION4.S AND
CAST(X54 AS VARCHAR) LIKE '*http:\*'
```

Όπως φαίνεται εδώ, κάθε κλαδί κάθε επιμέρους επερώτησης έχει βελτιστοποιηθεί κάνοντας πλέον χρήση των summaries αντί του κυρίως γράφου, όπου αυτό είναι δυνατό.

Ουσιαστικά αυτό που γίνεται είναι ότι στο τέλος τα κλαδιά κάθε επιμέρους επερώτησης επιστρέφουν όχι μόνο τα επιλεγμένα τους προς επιστροφή μέρη αλλά και την ρίζα τους ούτως ώστε τελικά να συνενωθούν τα κλαδιά μεταξύ τους στη βάση των κοινών ριζών. Έτσι επαναδημιουργείται η αρχική επερώτηση (στην βελτιστοποιημένη της μορφή) η οποία αποτελείται από πολλά κλαδιά με την ίδια ρίζα.

# Κεφάλαιο 9

## Τελικές δοκιμές-αποτίμηση συστήματος



### 9.1 Στόχοι δοκιμών

Οι στόχοι μου ήταν να αξιολογήσω τα εξής:

1) Την επίδοση του συστήματος όσο αφορά την απάντηση σε ενδιάμεσες επερωτήσεις έναντι συνόλων δεδομένων. Να συγκρίνω την απόδοση αυτή με την απόδοση που έδειξε η υλοποίηση του κου Κωνσταντίνου Σαββίδη στην Oracle [28] ούτως ώστε να αποφανθώ κατά πόσο η υλοποίηση με χρήση OpenLink Virtuoso (σε συνδυασμό με τα summaries) προσφέρει καλύτερη επίδοση.

2) Την επίδοση του συστήματος όσο αφορά την απάντηση σε ενδιάμεσες επερωτήσεις σε μεσαίου μεγέθους σύνολα δεδομένων (της τάξης των 80 MB).

3) Τον χρόνο που απαιτείται για την φόρτωση ήδη καταφορτωμένων στο τοπικό σύστημα αρχείων συνόλων δεδομένων στη ΒΔ με χρήση του bulk loader. Αποφάσισα να μην αξιολογήσω τον χρόνο για καταφόρτωση των συνόλων δεδομένων από το Διαδίκτυο καθώς αυτός εξαρτάται από παράγοντες που δεν έχουν να κάνουν με το ΣΔΒΔ η με την υλοποίηση μου (πχ σύνδεση με το Διαδίκτυο, συνωστισμός γραμμών).

### 9.2 Περιβάλλον δοκιμών

Χρησιμοποιημένα σύνολα δεδομένων

Για την δοκιμή φορτωσης συνόλων δεδομένων χρησιμοποίησα μέρη του συνόλου δεδομένων DBLP [32] το οποίο αποτελεί περιγραφή της βιβλιογραφίας στο πεδίο της Πληροφορικής.

Συγκεκριμένα, χρησιμοποίησα τέσσερα διαφορετικά μέρη/υποδιαιρέσεις του συνολικού συνόλου δεδομένων συμπεριλαμβανομένου ολόκληρου του συνόλου με

μεγέθη ως ακολούθως

| Πλήθος τριάδων(εκατομμύρια) | ~1.3 | ~2.6 | ~5.2 | ~9.7 |
|---|---|---|---|---|
| Μέγεθος (MB) | ~78 | ~156 | ~310 | ~577.6 |

Πίνακας 9.1 Σύγκριση μεγεθών συνόλων δεδομένων δοκιμής φόρτωσης

Για την δοκιμή χρόνου απόκρισης στις ενδιάμεσες επερωτήσεις χρησιμοποίησα:

1) Για την σύγκριση των επιδόσεων σε σχέση με τις επιδόσεις της υλοποίησης του κου Σαββίδη στο ΣΔΒΔ Oracle 11g σύνολα δεδομένων από τον ιστότοπο του data.semanticweb.org, τα οποία περιγράφουν δραστηριότητα σχετικά με τον τομέα του σημασιολογικού δικτύου. Τα συνδύασα ώστε να χρησιμοποιήσω σύνολο δεδομένων με μέγεθος ~10MB (97054 τριάδες, 10946 μοναδικά υποκείμενα, 104 μοναδικές ιδιότητες, 35463 μοναδικά αντικείμενα). Αυτό είναι εφάμιλλο με το μεγαλύτερο σύνολο που χρησιμοποίησε ο κος Σαββίδης για δοκιμή ενδιάμεσων επερωτήσεων στην Oracle.

2) Για την δοκιμή της επίδοσης του συστήματος για μεσαίου μεγέθους σύνολα δεδομένων το σύνολο δεδομένων με μέγεθος ~78 MB όπως περιγράφεται πιο πάνω, το οποίο περιέχει επίσης ~203 000 μοναδικά υποκείμενα, 18 ιδιότητες και ~439 000 μοναδικά αντικείμενα.

<u>Επερωτήσεις φάσης 1 (σύγκριση με υλοποίηση κου Σαββίδη)</u>

Οι ίδιες που είχε χρησιμοποιήσει ο κος Σαββίδης στο [28].

<u>Επερωτήσεις φάσης 2 (δοκιμή σε σύνολο δεδομένων μεσαίου μεγέθους)</u>

Οι ίδιες που είχαν χρησιμοποιήσει οι Δρ Δικαιάκος και Δρ Τζαράρ στο [11] για δοκιμή στο σύνολο δεδομένων DBLP.

Τεχνικές προδιαγραφές συστήματος δοκιμών

Επεξεργαστής 2GHz Dual Core

Κυρίως μνήμη 2GB

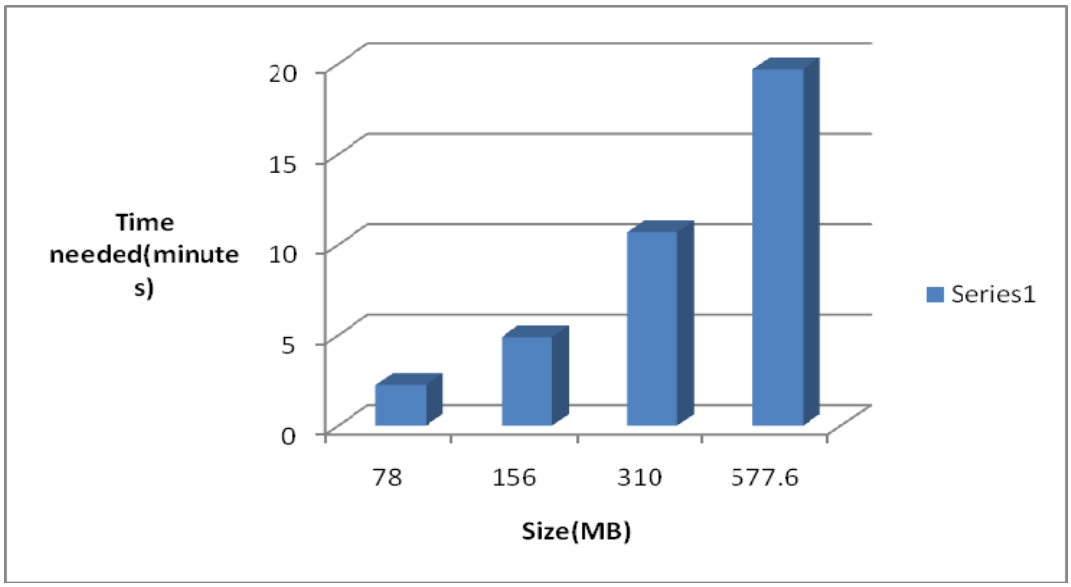
Σκληρός δίσκος (διαθέσιμος χώρος/ολικός χώρος) 71.5 GB/ 111 GB

Λειτουργικό σύστημα Windows 7

Όσο αφορά επεξεργαστές και κυρίως μνήμη το σύστημα είναι εφάμιλλο με το σύστημα επάνω στο οποίο έκανε τις δοκιμές της υλοποίησης του ο κος Σαββίδης με την διαφορά ότι εδώ έχουμε τον εξυπηρετητή εφαρμογών και τον εξυπηρετητή ΒΔ στην ίδια συσκευή, κάτι που όμως δεν διαφοροποιεί το περιβάλλον σημαντικά αφού ο εξυπηρετητής εφαρμογών σπαταλά σχετικά ελάχιστους πόρους σε σχέση με τον εξυπηρετητή ΒΔ.

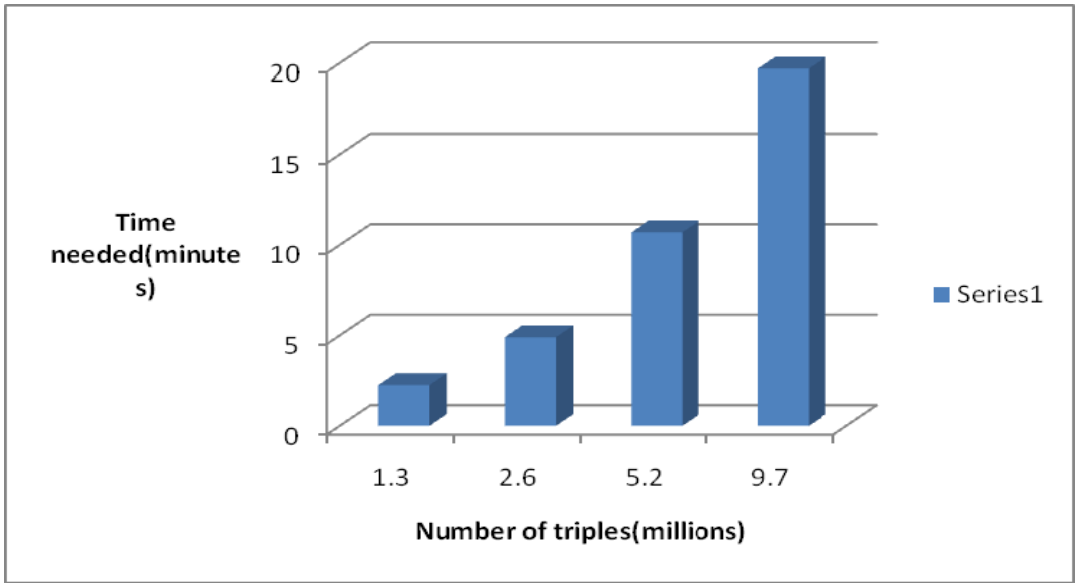
### 9.3 Αποτελέσματα

Φόρτωση συνόλων δεδομένων




Σχήμα 9.1 Χρόνος για φόρτωση συνόλων δεδομένων συναρτήσει του μεγέθους τους

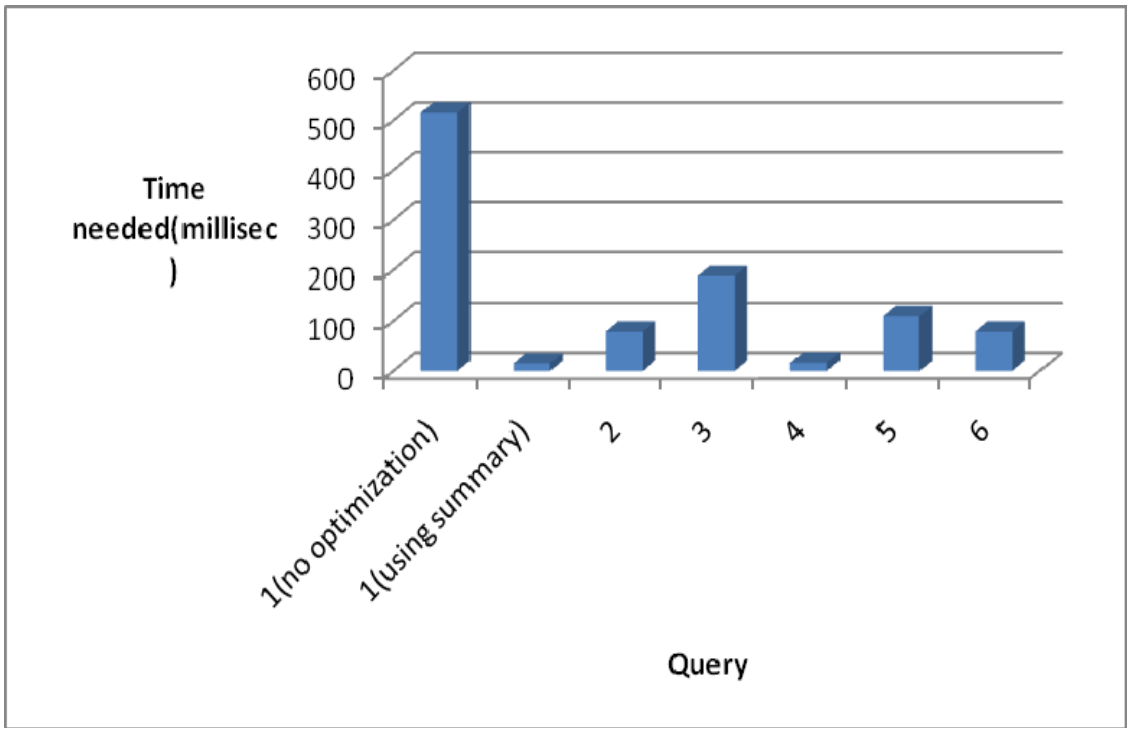
Παρατηρείται ότι για σύνολα δεδομένων με μέγεθος μέχρι και 577.6 MB ο χρόνος φόρτωσης είναι γραμμικός σε σχέση με το μέγεθος των δεδομένων.




Σχήμα 9.2 Χρόνος για φόρτωση συνόλων δεδομένων συναρτήσει του πλήθους των τριάδων τους

Το ίδιο σε σχέση με το πλήθος των τριάδων.

Ενδιάμεσες επερωτήσεις (σύγκριση με υλοποίηση Oracle)



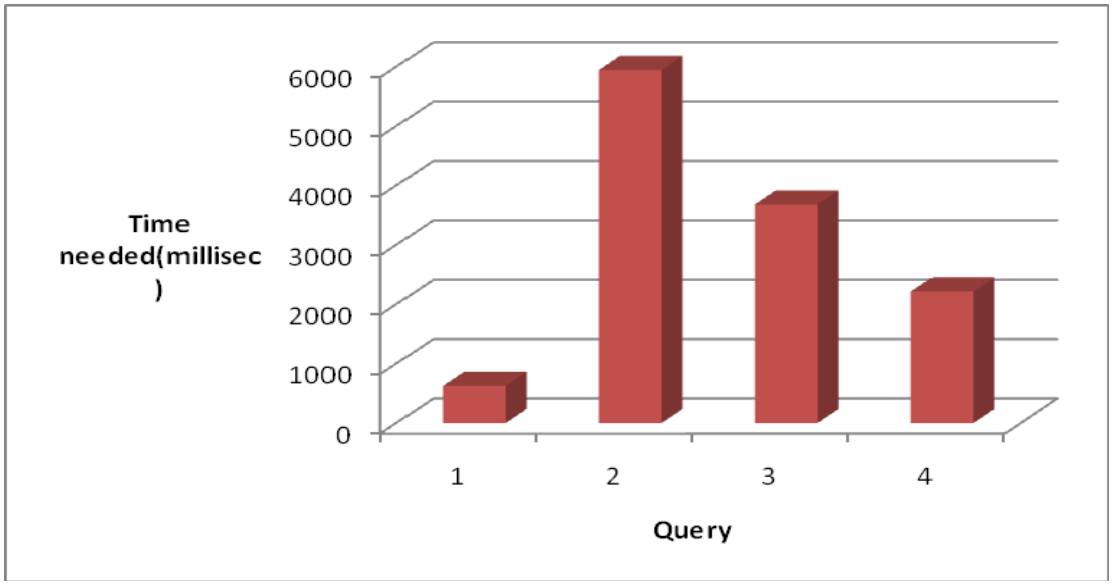
Σχήμα 9.3 Χρόνος απόκρισης ενδιάμεσων επερωτήσεων

Εδώ για την απάντηση στην 1η επερώτηση χρησιμοποιήθηκαν τα ευρετήρια για τις γενικές επερωτήσεις και επίσης έγινε δοκιμή δίχως χρήση ευρετηρίων. Παρατηρείται ότι ο χρόνος εκτέλεσης με χρήση των ευρετηρίων είναι κατά πολύ λιγότερος, τάξει λιγότερος από τον χρόνο εκτέλεσης χωρίς χρήση ευρετηρίων.

Όσο αφορά τις υπόλοιπες επερωτήσεις αλλά και την πρώτη, ο χρόνος για την υλοποίηση με χρήση του Virtuoso είναι πάντοτε μικρότερος του 1 δευτερολέπτου και για όλες πλην της 1ης μικρότερος των 200 milliseconds. Αντίθετα, στην υλοποίηση με χρήση Oracle όλοι οι χρόνοι για το σύνολο δεδομένων των 10 MB (για τις ίδιες επερωτήσεις) ήταν μεγαλύτεροι των 2 δευτερολέπτων. Μάλιστα σε όλες πλην της 1ης μεγαλύτεροι των 6 δευτερολέπτων. Παρατηρείται λοιπόν μια τεράστια διαφορά.

Το τρίτο που παρατηρείται είναι ότι αντίθετα με την υλοποίηση στην Oracle όπου κάθε κατώτερο επίπεδο διακλάδωσης (που συνεπαγόταν περισσότερες τριάδες στους περιορισμούς της επερώτησης) σήμαινε μεγαλύτερο χρόνο εκτέλεσης, με το Virtuoso οι χρόνοι δεν φαίνεται να εξαρτώνται από το επίπεδο διακλάδωσης της επερώτησης αλλά από το είδος της επερώτησης.

Ενδιάμεσες επερωτήσεις (δοκιμή σε μεσαίου μεγέθους σύνολο δεδομένων)




Εδώ παρατηρείται ότι για μεσαίου μεγέθους σύνολα δεδομένων (~78 MB) και μάλιστα χωρίς πολύ «θόρυβο» (το σύνολο δεδομένων DBLP είναι σχετικά ομοιογενές)

απαιτούνται χρόνοι μεγαλύτεροι από ότι είναι αποδεκτό (δηλαδή πέραν του ενός δευτερολέπτου) όσο αφορά επερωτήσεις μετά το πρώτο επίπεδο. Και εδώ παρατηρείται ότι οι χρόνοι δεν εξαρτώνται κατά κύριο λόγο από το βάθος της διακλάδωσης της επερώτησης αλλά από το είδος της επερώτησης.

### 9.4 Συμπεράσματα

Όσο αφορά δυνατότητες φόρτωσης δεδομένων το Virtuoso είναι κλιμακώσιμο για ακόμη και μεγάλα σύνολα δεδομένων (μέχρι ~600 MB) και παρουσιάζει επίδοση κατά πολύ καλύτερη από το ΣΔΒΔ Oracle. Σε δοκιμές στο [28] η Oracle για μικρά σύνολα δεδομένων (<= 10 MB) χρειάστηκε μέχρι και περισσότερο των 10 λεπτών (για το σύνολο δεδομένων με 10 MB) ενώ το Virtuoso χρειάστηκε ανάλογους χρόνους για σύνολα δεδομένων με εκατοντάδες MB. Αυτό ίσως να οφείλεται στον μηχανισμό bulk loader του Virtuoso ο οποίος μπορεί να χρησιμοποιήσει και πολλά νήματα τα οποία να εκτελούν ταυτόχρονη φόρτωση.

Όσο αφορά εκτέλεση ενδιάμεσων επερωτήσεων, το Virtuoso ξανά επέδειξε επίδοση πολύ καλύτερη της Oracle αφού όλες οι ενδιάμεσες επερωτήσεις στο σύνολο δεδομένων των 10 MB απαντήθηκαν σε χρόνο λιγότερο του ενός δευτερολέπτου (χωρίς χρήση βελτιστοποιητή) και όλες πλην της πρώτης σε χρόνο μικρότερο των 200 milliseconds. Αντίθετα, με την βασισμένη σε Oracle υλοποίηση είχαμε χρόνους πολύ πέραν του παραδεκτού (δηλαδή ενός δευτερολέπτου) ακόμη και για σύνολο δεδομένων με το ½ του μεγέθους του συνόλου που συμμετείχε σε αυτή την δοκιμή. Επίσης παρατηρήθηκε ότι ενώ στην υλοποίηση με χρήση Oracle οι χρόνοι αυξάνονταν όσο αυξανόταν το βάθος της διακλάδωσης της επερώτησης αυτό δεν παρατηρήθηκε με το Virtuoso, στο οποίο κάποες επερωτήσεις με μεγαλύτερο βάθος διακλάδωσης απαντήθηκαν σε πολύ λιγότερο χρόνο από κάποιες με μικρότερο βάθος.

Το κλειδί για το γιατί το Virtuoso παρουσιάζει αυτή την τεράστια διαφορά επίδοσης σε σχέση με την Oracle αλλά και γιατί κάποιες πιο «βαθιές» επερωτήσεις παρουσιάζουν χρόνους εκτέλεσης πολύ μικρότερους πιο «ρηχών» ίσως να βρίσκεται στον τρόπο με τον οποίο το Virtuoso βελτιστοποιεί εσωτερικά τις επερωτήσεις ή/και επιλέγει σειρά εκτέλεσης και JOIN. Αυτό στηρίζεται από τα ακόλουθα:

Η επερώτηση 1 (χωρίς βελτιστοποίηση) η οποία προωθείται στη ΒΔ προς εκτέλεση για το σύνολο δεδομένων των 10 MB και η οποία είχε με διαφορά τον χειρότερο χρόνο εκτέλεσης για το Virtuoso είναι η εξής:

SELECT DISTINCT TOP 0, 50 P1

FROM ( SPARQL SELECT ?P1 FROM <http://testing10.org>

WHERE **{ {?S rdf:type ?O . } { ?S ?P1 ?O1 . } }** ) AS allofthem WHERE CAST(P1 AS VARCHAR) LIKE '**' AND CAST(P1 AS VARCHAR) LIKE '*http:*' group by P1 order by P1 ASC

Εδώ δεν υπάρχουν πολλά περιθώρια βελτιστοποίησης, ειδικά όσο αφορά την $2^{η}$ τριάδα της επερώτησης.

Η επερώτηση 6 (η οποία παρά το ότι περιείχε μια βαθιά διακλάδωση 6 επιπέδων είχε πολύ μικρότερο χρόνο από την 1 η οποία είχε μόνο ένα επίπεδο) είναι η εξής:

SELECT DISTINCT TOP 0, 50 P6 FROM ( SPARQL SELECT ?P6 FROM <http://testing1.org> WHERE

{{?S rdf:type ?O . } **{ ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . }** { ?O1 <http://data.semanticweb.org/ns/swc/ontology#hasRelatedDocument> ?O2 . **} { ?O2 <http://www.cs.vu.nl/~mcaklein/onto/swrc_ext/2005/05#authorList> ?O3 . }** { ?O3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2> ?O4 . } { ?O4 <http://xmlns.com/foaf/0.1/made> ?O5 . } { ?O5 ?P6 ?O6 . } } ) AS allofthem WHERE CAST(P6 AS VARCHAR) LIKE '**' AND CAST(P6 AS VARCHAR) LIKE '*http:*' group by P6 order by P6 ASC

Εδώ παρατηρείται ότι υπάρχουν 7 τριάδες, οι περισσότερες εκ των οποίων συμπεριλαμβάνουν σύνδεση μέσω ενός κατηγορήματος που τίθεται ως σταθερά. Μάλιστα οι τριάδες σε μαύρο μέσω κατηγορημάτων που για κάθε υποκείμενο στο οποίο εμφανίζονται δεν επιστρέφουν πολλά αντικείμενα και εμφανίζονται μόνο σε υποκείμενα συγκεκριμένων κατηγοριών(το isSuperEventOf σε γεγονότα και το authorList σε papers κοκ).

Οπότε αν ο εσωτερικός μηχανισμός βελτιστοποίησης σειράς εκτέλεσης JOIN του Virtuoso μπορεί με κάποιο τρόπο να επιλέγει (με χρήση πχ στατιστικών υπολογισμών) να εκτελέσει πρώτα τα JOIN των επιστρεφόμενων ενδιάμεσων αποτελεσμάτων τα οποία επιστρέφουν τα λιγότερα ενδιάμεσα αποτελέσματα, έχοντας ως αποτέλεσμα μικρότερες ανάγκες επεξεργασίας δεδομένων και χρήσης κυρίως μνήμης, με μεγαλύτερη επιτυχία από τον αντίστοιχο μηχανισμό της Oracle, αυτό ίσως να δικαιολογεί τα πιο πάνω. Έτσι δικαιολογείται η κατά πολύ καλύτερη επίδοση του Virtuoso και μάλιστα σε επερωτήσεις σε βάθος (όπου μια αφελής σειρά συνένωσης επιμέρους αποτελεσμάτων θα είχε ως αποτέλεσμα τεράστια ενδιάμεσα αποτελέσματα λόγω συνενώσεων μεταξύ πινάκων με πολλά στοιχεία και ελάχιστες σχετικά διαγραφές ενώ μια βελτιστοποιημένη σειρά πολύ περισσότερες διαγραφές σειρών από τους ενδιάμεσους πίνακες παρά συνενώσεις).

Όσο αφορά επίδοση με και χωρίς χρήση βελτιστοποίησης για τις γενικές επερωτήσεις ($1^{ου}$ επιπέδου) τα αποτελέσματα δείχνουν καθαρά ότι η βελτιστοποίηση με χρήση summaries μπορεί να οδηγήσει σε τάξει καλύτερη επίδοση. Πράγμα λογικό αφού χρησιμοποιούμε στη διαδικασία εκτέλεσης της επερώτησης (και άρα και των συνενώσεων ενδιάμεσων αποτελεσμάτων, πολλές φορές με τον εαυτό τους) πολύ λιγότερα δεδομένα. Αυτό επειδή δεν χρησιμοποιούμε όλα τα πεδία και όλες τις σειρές των συνόλων δεδομένων, κάτι που μειώνει εκθετικά το μέγεθος των ενδιάμεσων αποτελεσμάτων όταν έχουμε self-joins.

Όσο αφορά τις επιδόσεις στο σύνολο δεδομένων των 78 MB, εδώ παρατηρείται ότι η πρώτη επερώτηση μόνο είχε παραδεκτό χρόνο εκτέλεσης (λόγω του ότι δεν εσυμπεριελάμβανε JOINS). Οι υπόλοιπες είχαν ξεπεράσει με σημαντική διαφορά το όριο του ενός δευτερολέπτου. Εδώ φαίνεται ότι παρά τις αισθητά καλύτερες επιδόσεις του Virtuoso σε σχέση με την Oracle, για μεσαία και μεγάλα σύνολα δεδομένων απαιτείται χρήση βελτιστοποιητικών δομών επιπλέον αυτών για τις γενικές επερωτήσεις. Επειδή απαιτείται η ικανοποίηση των επερωτήσεων πολλών επιπέδων οι οποίες συμπεριλαμβάνουν σε βάθος διακλαδώσεις (και άρα πολλά JOIN). Όπως αναπτύσσεται και στο [7] αυτού του είδους οι επερωτήσεις λόγω αυτού του χαρακτηριστικού παρουσιάζουν ιδιαίτερα προβλήματα για την επίλυση των οποίων απαιτούνται εξειδικευμένες δομές βελτιστοποίησης, όπως χρήση του αλγορίθμου BR

[7], κάτι το οποίο πιστεύω και ότι θα πρέπει να είναι η πρώτη προτεραιότητα οποιουδήποτε πιθανό να συνεχίσει από εκεί που αφήνει αυτή η ΑΔΕ.

# Κεφάλαιο 10

## Συμπεράσματα



### 10.1 Επίλογος

Αυτή η διπλωματική είχε ως στόχο την υλοποίηση ενός εξυπηρετητή συστήματος επερωτήσεων/mashups σε RDF μεταδεδομένα ο οποίος να μπορεί να τρέξει σε συστήματα με περιορισμένους πόρους (πχ κυρίως μνήμη).

Μελετήθηκε και υλοποιήθηκε το πλαίσιο βιβλιοθηκών JENA για επεξεργασία σημασιολογικών δεδομένων στο πλαίσιο OSGI και δοκιμάστηκε με μη ικανοποιητικά αποτελέσματα τα οποία κατέδειξαν και στην πράξη τις αδυναμίες του(κυριότερη εκ των οποίων ο περιορισμός του στην χρήση μόνο της κυρίως μνήμης). Έπειτα έγινε έρευνα για την πιο υποσχόμενη εναλλακτική λύση η οποία αποφασίστηκε ότι θα ήταν το ΣΔΒΔ OpenLink Virtuoso.

Δημιουργήθηκε σύστημα ελέγχου ταυτόχρονης καταφόρτωσης συνόλων RDF μεταδεδομένων ούτως ώστε να αντικαταστήσει τις διαδικασίες που ήταν υπεύθυνες γι' αυτό τον τομέα στην υλοποίηση στην Oracle. Έπειτα έγιναν οι αναγκαίες τροποποιήσεις στον πελάτη και τον εξυπηρετητή εφαρμογών και ετοιμάστηκε το ΣΔΒΔ ούτως ώστε να ενσωματωθεί το επιλεγμένο ΣΔΒΔ στα πλαίσια του υπάρχοντος συστήματος αρχιτεκτονικής τριών επιπέδων της εφαρμογής.

Διαπιστώθηκε ότι λόγω μη αποδοτικού τρόπου διαμόρφωσης συνδυασμένων επερωτήσεων στο επίπεδο των mashups η εκτέλεση των ενδιάμεσων επερωτήσεων οι

οποίες αφορούσαν mashups έπαιρναν απαράδεκτο χρόνο ακόμη και για μικρά μεγέθη δεδομένων. Γι' αυτό άλλαξε η όλη λογική της διαμόρφωσης των επόμενων επιπέδων των mashups βάσει των προηγούμενων ώστε να περιορίζονται οι ανάγκες επεξεργασίας δεδομένων και το μέγεθος των ενδιάμεσων αποτελεσμάτων. Αλλαγή που οδήγησε σε τεράστια βελτίωση στον χρόνο εκτέλεσης των ενδιάμεσων επερωτήσεων θέτοντας τον πολύ κάτω από το μέγιστο επιτρεπτό (1 δευτερόλεπτο) για μικρά σύνολα δεδομένων.

Προστέθηκαν summaries τα οποία συνόψιζαν τα δεδομένα των εισαγόμενων στο ΣΔΒΔ γράφων RDF ούτως ώστε να μειωθούν περαιτέρω οι ανάγκες για επεξεργασία δεδομένων και τα μεγέθη των ενδιάμεσων αποτελεσμάτων. Αυτό οδήγησε σε περαιτέρω σημαντικότατη μείωση του χρόνου για εκτέλεση των γενικών ενδιάμεσων επερωτήσεων.

Οι τελικές δοκιμές επιβεβαίωσαν τα πιο πάνω, αφού η υλοποίηση με βάση το Virtuoso πέτυχε πολύ καλύτερους χρόνους από την Oracle όσο αφορά ενδιάμεσες επερωτήσεις. Παρ'όλα αυτά για μεσαία και μεγάλα σύνολα δεδομένων η υπάρχουσα υλοποίηση μπορεί να βελτιωθεί μελλοντικά περαιτέρω αφού οι χρόνοι για μεγαλύτερα σύνολα δεδομένων δεν ήταν ικανοποιητικοί.

### 10.2 Μελλοντική εργασία

Σίγουρα το πρώτο που θα πρέπει να γίνει είναι η υλοποίηση ενός αλγορίθμου όπως ο αλγόριθμος BR [7] για βελτιστοποίηση των επερωτήσεων σε βάθος σε μεγάλα σύνολα δεδομένων.

Επίσης θα μπορούσε να δοκιμαστεί η χρήση της εναλλακτικής μεθόδου διαμόρφωσης επερωτήσεων όσο αφορά το επίπεδο των mashups, τόσο στην Oracle όσο και στο Virtuoso, ώστε να ποσοτικοποιηθεί η βελτίωση που μπορεί να προσφέρει σε κάθε ΣΔΒΔ η χρήση αυτής της μεθόδου.

# Βιβλιογραφία

[1] Benefits of using OSGI,OSGI Alliance.Monday, May 09, 2011 2:48:45 AM.May 9,2011 http://www.osgi.org/About/WhyOSGi

[2] Binna,R.,Gassler,W.,Zangerle,E.,Pacher,D.,Specht,G.:

SpiderStore: Exploiting Main Memory for Efficient

RDF Graph Representation and Fast Querying.In:

Proceedings of the Workshop on Semantic Data Management at VLDB 2010

[3] Bizer,C.,Schultz,A.:

Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints.In:

Proceedings of the 4th International Workshop on Scalable Semantic Web knowledge Base Systems (SSWS2008).

[4] Bizer,C.,Schultz,A.:

BSBM Results for Virtuoso, Jena TDB, BigOWLIM (November 2009).Monday, December 07, 2009 4:18:24 PM.May 10,2011

http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V5/

[5] Bizer,C.,Schultz,A.:

The Berlin SPARQL Benchmark.In:

International Journal On Semantic Web and Information Systems

Special Issue on Scalability and Performance of Semantic Web Systems,pp.1-24(2009)

[6] Chaudhuri,S.,Weikum,G.:

Rethinking database system architecture: Towards a self-tuning risc-style

database system.In:

Proceedings of the 26th International Conference on

Very Large Data Bases (VLDB), pp. 1-10 (2000)

[7] Georgiou,M.:

Challenges in building an efficient relational architecture for MashQl

University of Cyprus,Computer Science Department.

[8] Guo,Y.,Pan,Z.,Heflin,J.:

LUBM: A benchmark for OWL knowledge base systems.In:

Web Semantics: Science, Services and Agents

on the World Wide Web 3,pp.158–182 (2005)

[9] Harms,D.:JSP,Servlets,and MySQL.Hungry Minds,Inc,2001

[10] Horrocks,I:

Ontologies and the Semantic Web.In:

Communications of the ACM,Vol 51,Issue 12 pp.58-67(December 2008)

[11] Jarrar,M.,Dikaiakos,M.:

A Query Formulation Language for the

Data Web.In:

IEEE Transactions on Knowledge and Data Engineering.IEEE Computer Society. (2010, In Press).

[12] Jarrar,M.,Dikaiakos,M.:

MashQL: A Query-by-Diagram Topping SPARQL

Towards Semantic Data Mashups.In:

LDOW2009, April 20, 2009, Madrid, Spain.

[13] Jarrar,M.,Dikaiakos,M.:

Querying the Data Web –the MashQL Approach.In:

IEEE Internet Computing. Volume 14, No. 3. pp.58-670.IEEE Computer Society, ISSN 1089-7801(May 2010)

[14] Jena – A Semantic Web Framework for Java,SourceForge.NET.Friday, December 17, 2010 9:53:32 PM.May 9,2011

http://jena.sourceforge.net

[15] Lee,R.:

Scalability Report on Triple Store Applications.

Tuesday, May 10, 2011 12:30:35 AM.May 10,2011.

http://simile.mit.edu/reports/stores/index.html

July 26, 2004

[16] Ma,L.,Yang,Y.,Qiu,Z.,Xie,G.,Pan,Y.,Liu,S.:

Towards a Complete OWL Ontology Benchmark.In:

Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 125–

139. Springer, Heidelberg (2006)

[17] McCarthy,P.:

Search RDF data with SPARQL:SPARQL and the Jena Toolkit open up the semantic Web.In:

http://www.ibm.com/developerworks/xml/library/j-sparql.Tuesday, May 10, 2011 1:07:36 AM.May 10,2011.

[18] Miller, E.:

An Introduction to the Resource Description Framework.In:

Bulletin of the American Society for Information Science and Technology, Vol. 25 pp.15–19. doi: 10.1002/bult.105 (1998)

[19] Neumann,T.,Weikum,G.:

RDF3X:a RISC-style Engine for RDF.In:

Proceedings of the 34th International Conference on

Very Large Data Bases (VLDB), pp. 647-659 (2008)

[20] OpenLink Virtuoso Universal Server: Documentation.RDF Data Access and Data Management.SPARQL.OpenLink Software.

Tuesday, May 10, 2011 1:03:27 AM.May 10,2011.

http://docs.openlinksw.com/virtuoso/rdfsparql.html

[21] OpenLink Virtuoso Universal Server: Documentation.RDF Data Access and Data Management.RDF Insert Methods in Virtuoso.OpenLink Software.

Tuesday, May 10, 2011 1:06:02 AM.May 10,2011.

http://docs.openlinksw.com/virtuoso/rdfinsertmethods.html

[22] Oracle® Database

Semantic Technologies Developer's Guide

11g Release 2 (11.2)

[23] Oracle Database 11g Semantic Technologies Overview,

Bill Beauregard, Senior Principal Product Manager, Oracle USA.Thursday, September 09, 2010 8:58:44 PM.May 10,2011.

[24] OSGI Technology,OSGI Alliance.Tuesday, May 10, 2011 1:12:24 AM.May 10,2011

http://www.osgi.org/About/Technology

[25] RDF2RDF,Enrico Minack.Friday, May 07, 2010 11:17:44 PM.May 9,2011.

http://www.l3s.de/~minack/rdf2rdf/

[26] Roberto De Virgilio,Pierluigi Del Nostro,Gianforme G,Paolozzi S:

A scalable and extensible framework for query

answering over RDF.In:

World Wide Web Journal,Preprint,14 January 2011

[27] Rohloff,K.,Dean,M.,Emmons,I.,Ryder,D.,Sumner,J.:

An Evaluation of Triple-Store Technologies for Large

Data Stores.In:

OTM'07 Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems

- Volume Part II pp.1105-1114

[28] Savvides,C.:

"MashQl,A step Towards Semantic Web Pipes",University of Cyprus,Computer Science Department, 2010

[29] Schmidt,M.,Hornung,T.,Lausen,G.,Pinkel,C.:

SP2Bench: A SPARQL Performance Benchmark.In:

Proceedings of the 25th International Conference on Data Engineering(ICDE),pp. 222-233 (2007)

[30] Sesame,OpenRDF,org.Tuesday, May 10, 2011 12:31:20 AM.May 10,2011

http://www.openrdf.org/

[31] Thakker,D.,Osman,T.,Gohil,S.,Lakin,P.:

A Pragmatic Approach to Semantic Repositories

Benchmarking.In:

7th Extended Semantic Web Conference, ESWC 2010

Heraklion, Crete, Greece, May 30 –June 3, 2010, Proceedings, Part I

[32] The DBLP Computer Science Bibliography,Universitat Trier.Monday, May 09, 2011 11:26:45 PM.May 10,2011

http://www.informatik.uni-trier.de/~ley/db/

[33] Virtuoso OpenSource Edition Introduction.Tuesday, May 10, 2011 12:25:07 AM.May 10,2011.

http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSIntro

[34]"W3C Semantic Web Frequently Asked Questions". W3C. http://www.w3.org/2001/sw/SW-FAQ.

Thursday, November 12, 2009 12:51:33 PM.May 10,2011

# Παράρτημα Α

## Στο παράρτημα αυτό ακολουθούν οι επερωτήσεις που αποστέλλοντο στην ΒΔ προς εκτέλεση(μετά την όποια βελτιστοποίηση) κατά την τελική δοκιμή της νέας υλοποίησης

### Σύγκριση Virtuoso με ΣΒΔΒ Oracle

Χωρίς χρήση βελτιστοποιητή

Πρώτο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P1 FROM ( SPARQL SELECT ?P1 FROM <http://testing10.org> WHERE { {?S rdf:type ?O . } { ?S ?P1 ?O1 . } } ) AS allofthem WHERE CAST(P1 AS VARCHAR) LIKE '**' AND CAST(P1 AS VARCHAR) LIKE '*http:*' group by P1 order by P1 ASC

Δεύτερο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P2 FROM ( SPARQL SELECT ?P2 FROM <http://testing1.org> WHERE { {?S rdf:type ?O . } { ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . } { ?O1 ?P2 ?O2 . } } ) AS allofthem WHERE CAST(P2 AS VARCHAR) LIKE '**' AND CAST(P2 AS VARCHAR) LIKE '*http:*' group by P2 order by P2 ASC

Τρίτο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P3 FROM ( SPARQL SELECT ?P3 FROM <http://testing1.org> WHERE { {?S rdf:type ?O . } { ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . } { ?O1 <http://data.semanticweb.org/ns/swc/ontology#hasRelatedDocument> ?O2 . } { ?O2 ?P3 ?O3 . } } ) AS allofthem WHERE CAST(P3 AS VARCHAR) LIKE '**' AND CAST(P3 AS VARCHAR) LIKE '*http:*' group by P3 order by P3 ASC

Τέταρτο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P4 FROM ( SPARQL SELECT ?P4 FROM <http://testing1.org> WHERE { {?S rdf:type ?O . } { ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . } { ?O1 <http://data.semanticweb.org/ns/swc/ontology#hasRelatedDocument> ?O2 . } { ?O2 <http://www.cs.vu.nl/~mcaklein/onto/swrc_ext/2005/05#authorList> ?O3 . } { ?O3 ?P4 ?O4 . } } ) AS allofthem WHERE CAST(P4 AS VARCHAR) LIKE '**' AND CAST(P4 AS VARCHAR) LIKE '*http:*' group by P4 order by P4 ASC

Πέμπτο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P5 FROM ( SPARQL SELECT ?P5 FROM <http://testing1.org> WHERE { {?S rdf:type ?O . } { ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . } { ?O1 <http://data.semanticweb.org/ns/swc/ontology#hasRelatedDocument> ?O2 . } { ?O2 <http://www.cs.vu.nl/~mcaklein/onto/swrc_ext/2005/05#authorList> ?O3 . } { ?O3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2> ?O4 . } { ?O4 ?P5 ?O5 . } } ) AS allofthem WHERE CAST(P5 AS VARCHAR) LIKE '**' AND CAST(P5 AS VARCHAR) LIKE '*http:*' group by P5 order by P5 ASC

Έκτο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P6 FROM ( SPARQL SELECT ?P6 FROM <http://testing1.org> WHERE { {?S rdf:type ?O . } { ?S <http://data.semanticweb.org/ns/swc/ontology#isSuperEventOf> ?O1 . } { ?O1 <http://data.semanticweb.org/ns/swc/ontology#hasRelatedDocument> ?O2 . } { ?O2 <http://www.cs.vu.nl/~mcaklein/onto/swrc_ext/2005/05#authorList> ?O3 . } { ?O3 <http://www.w3.org/1999/02/22-rdf-syntax-ns#_2> ?O4 . } { ?O4 <http://xmlns.com/foaf/0.1/made> ?O5 . } { ?O5 ?P6 ?O6 . } } ) AS allofthem WHERE CAST(P6 AS VARCHAR) LIKE '**' AND CAST(P6 AS VARCHAR) LIKE '*http:*' group by P6 order by P6 ASC

Γενική επερώτηση πρώτου επιπέδου διακλάδωσης,με χρήση βελτιστοποιητή

SELECT DISTINCT TOP 0, 50 P1 FROM (SELECT S AS S FROM RDF_DB.DBA."RDFINDEX_SUBJECTOFTYPE_http://testing1.org" )AS SELECTION1, (SELECT S AS S,P AS P1 FROM RDF_DB.DBA."RDFINDEX_SP_http://testing1.org" )AS SELECTION2 WHERE SELECTION1.S = SELECTION2.S AND CAST(P1 AS VARCHAR) LIKE '**' AND CAST(P1 AS VARCHAR) LIKE '*http:*'

**Δοκιμή σε μεσαίου μεγέθους σύνολο δεδομένων**

Πρώτο επίπεδο διακλάδωσης(τύποι)

SELECT DISTINCT TOP 0, 50 O FROM ( SPARQL SELECT ?O FROM <testdblp161> WHERE { {?S rdf:type ?O . } } ) AS allofthem WHERE CAST(O AS VARCHAR) LIKE '**' AND CAST(O AS VARCHAR) LIKE '*http:*' group by O order by O ASC

Πρώτο επίπεδο διακλάδωσης(ιδιότητες)

SELECT DISTINCT TOP 0, 50 P1 FROM ( SPARQL SELECT ?P1 FROM <testdblp161> WHERE { {?S rdf:type <http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article> . } {?S ?P1 ?O1 . } } ) AS allofthem WHERE CAST(P1 AS VARCHAR) LIKE '**' AND CAST(P1 AS VARCHAR) LIKE '*http:*' group by P1 order by P1 ASC

Δεύτερο επίπεδο διακλάδωσης

SELECT DISTINCT TOP 0, 50 P2 FROM ( SPARQL SELECT ?P2 FROM <testdblp161> WHERE { {?S rdf:type <http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article> . } {?S <http://purl.org/dc/elements/1.1/creator> ?O1 . } { ?O1 ?P2 ?O2 . } } ) AS allofthem WHERE CAST(P2 AS VARCHAR) LIKE '**' AND CAST(P2 AS VARCHAR) LIKE '*http:*' group by P2 order by P2 ASC

Δεύτερο επίπεδο διακλάδωσης(αντικείμενα)

SELECT DISTINCT TOP 0, 50 O2 FROM ( SPARQL SELECT ?O2 FROM <testdblp161> WHERE { {?S rdf:type <http://sw.deri.org/~aharth/2004/07/dblp/dblp.owl#Article> . } {?S <http://purl.org/dc/elements/1.1/creator> ?O1 . } { ?O1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?O2 . } } ) AS allofthem WHERE CAST(O2 AS VARCHAR) LIKE '**' AND CAST(O2 AS VARCHAR) LIKE '*http:*' group by O2 order by O2 ASC