ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΕΓΚΑΤΑΣΤΑΣΕΩΝ & ΣΥΣΤΗΜΑΤΩΝ ΑΠΟΦΑΣΗΣ

Μελέτη Μεθόδων Κατασκευής Σύνθετων Ερωτημάτων σε Σχήματα RDF από Φυσική Γλώσσα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΩΤΗΡΟΠΟΥΛΟΥ ΑΠΟΣΤΟΛΟΥ

Επιβλέπων : Δημήτριος Ασκούνης
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΗΛΕΚΤΡΙΚΩΝ ΒΙΟΜΗΧΑΝΙΚΩΝ ΕΓΚΑΤΑΣΤΑΣΕΩΝ & ΣΥΣΤΗΜΑΤΩΝ
ΑΠΟΦΑΣΗΣ

Μελέτη Μεθόδων Κατασκευής Σύνθετων Ερωτημάτων σε Σχήματα RDF από Φυσική Γλώσσα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΩΤΗΡΟΠΟΥΛΟΥ ΑΠΟΣΤΟΛΟΥ

Επιβλέπων : Δημήτριος Ασκούνης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 20η Ιουλίου 2015.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δημήτριος Ασκούνης Καθηγητής Ε.Μ.Π.	Ιωάννης Ψαρράς Καθηγητής Ε.Μ.Π.	Γρηγόρης Μέντζας Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2015

(Υπογραφή)

.....

ΣΩΤΗΡΟΠΟΥΛΟΣ ΑΠΟΣΤΟΛΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η ανάπτυξη εφαρμογής η οποία θα απαντούσε σε σύνθετα ερωτήματα ανθρώπινης γλώσσας κάνοντας χρήση του σημασιολογικού ιστού μέσω διασυνδεδεμένων δεδομένων. Ζητούμενο ήταν μια εφαρμογή που θα μπορούσε να αντλήσει πληροφορίες από παραπάνω από μια συλλογές δεδομένων ενώ θα μπορούσε να απαντήσει τόσο σε απλά ερωτήματα όσο και σε οποιοδήποτε συνδυασμό αυτών εκφρασμένα ως συνδυαστικές ερωτήσεις χρήστη.

Για το σκοπό αυτό, μελετήθηκαν και αναλύθηκαν υπάρχουσες προσεγγίσεις στην ερευνητική περιοχή της απάντησης ανθρώπινων ερωτημάτων από μηχανή (Question Answering). Για κάθε μια σημειώθηκαν προτερήματα και μειονεκτήματα καθώς και σημεία βελτίωσης. Μια εξ' αυτών διαδραμάτισε σημαντικό ρόλο στην επίτευξη του στόχου. Συγκεκριμένα, για την εφαρμογή Quepy [1] μελετήθηκε και επιβεβαιώθηκε η ικανότητα απάντησης ερωτημάτων ενώ εντοπίστηκε η αδυναμία χειρισμού παραπάνω της μιας συλλογής δεδομένων καθώς και ο συνδυασμός ερωτήσεων για την απάντηση πιο σύνθετων ερωτημάτων.

Ο στόχος επιτεύχθηκε κάνοντας χρήση και επεκτείνοντας το Quepy. Στηριζόμενη σε αυτή, αναπτύχθηκε η εφαρμογή ανοιχτού λογισμικού InterQuepy [2] στη γλώσσα προγραμματισμού Python. Με εκτεταμένη χρήση των αρχών του αντικειμενοστραφούς προγραμματισμού καθώς και απαραίτητων σχεδιαστικών προτύπων, η τελευταία, επιτυγχάνει την απάντηση σύνθετων ερωτημάτων συνδυάζοντας γνώση διαφορετικών συλλογών δεδομένων.

Λέξεις Κλειδιά: απάντηση ερωτημάτων φυσικής γλώσσας, διασυνδεδεμένα δεδομένα, σημασιολογικός ιστός

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The scope of this thesis was to develop an application that would respond to complex human language queries using semantic web through interlinked data. The desired application would draw information from more than one datasets and answer both simple queries and any combination of them expressed as a complicated user question.

For this purpose, we studied and analyzed existing approaches in the research area of answering human queries by machine (Question Answering). For each approach there were advantages and disadvantages as well as improvement points. One of the approaches played an important role in achieving the thesis goal. Specifically, application Quepy [1] was studied and verified as an application able to answer effectively human queries but unable to handle more than one datasets or combine questions to answer more complex questions.

The goal was achieved by using and expanding Quepy. Using it as an initial code base, we developed the open source application InterQuepy [2] in Python programming language. With excessive use of the principles of object oriented programming and design patterns needed, the latter achieves the complex questions answering by combining knowledge of different datasets.

Keywords: question answering, linked data, semantic web, quepy, interquepy, rdf, nlp, natural language processing, python

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 *Εισαγωγή*

1.1 Απάντηση ερωτήσεων ανθρώπινης γλώσσας από μηχανή

Ένα γενικότερο πρόβλημα – πρόκληση που αποκτά ολοένα και μεγαλύτερο ενδιαφέρον σήμερα, είναι η εκμετάλλευση της κατανεμημένης γνώσης που μπορεί να αποθηκεύεται σε υπολογιστικά συστήματα για την εξαγωγή συμπερασμάτων και απάντηση ερωτημάτων χρηστών. Θα θέλαμε ιδανικά να θέτουμε ερωτήματα σε ανθρώπινη γλώσσα και να λαμβάνουμε απαντήσεις σε αυτά από ένα μηχανισμό, ο οποίος θα λάμβανε υπόψη όχι μόνο τις πληροφορίες που είναι αποθηκευμένες στη μνήμη ενός μόνο συστήματος αλλά τη διάσπαρτη γνώση του διαδικτύου.

Προς αυτήν την κατεύθυνση έχει γίνει πληθώρα προτάσεων και προσεγγίσεων. Κάθε μια εξ αυτών, καλείται να αντιμετωπίσει σειρά προβλημάτων που προκύπτουν με φυσικό τρόπο από το πρόβλημα προς εξέταση.

Το πρώτο από αυτά δεν είναι άλλο παρά η επιλογή της πηγής πληροφοριών. Αν επιλέξει κανείς να αντλεί τις πληροφορίες από τη μνήμη συγκεκριμένων συστημάτων / βάσεων δεδομένων, επωμίζεται το κόστος καθώς και την ευθύνη της ανανέωσης και συντήρησης του υλικού που είναι αποθηκευμένο και παίζει το ρόλο της γνώσης. Επίσης ένα τέτοιο σύστημα κληρονομεί όλα τα προβλήματα μιας αποκομμένης από τον υπόλοιπο κόσμο γνώσης που για να συντηρηθεί θα πρέπει το ίδιο να αναζητά πληροφορίες και να εμπλουτίζει τη μνήμη του.

Αν από την άλλη επιλεγεί η χρήση της γνώσης που βρίσκεται αποθηκευμένη όχι στην τοπική μνήμη ενός μηχανήματος ή μιας περιορισμένης ομάδας μηχανημάτων αλλά της γνώσης που βρίσκεται κατανεμημένη στο διαδίκτυο, λύνονται τα παραπάνω προβλήματα αλλά δημιουργούνται άλλα.

Το πρώτο από αυτά αφορά στην μέθοδο εξαγωγής πληροφορίας. Αν η εξαγωγή πληροφορίας βασιστεί σε αναζήτηση πάνω σε λέξεις κλειδιά σε κείμενο που είναι διαθέσιμο στο διαδίκτυο καλούμαστε να αντιμετωπίσουμε προβλήματα όπως η σχετικότητα του αποτελέσματος με αυτό που έψαχνε ο χρήστης ή η εξάρτηση του αποτελέσματος από τον ακριβή όρο αναζήτησης[3].

Για παράδειγμα ας υποθέσουμε ότι αναζητούμε απάντηση στο ερώτημα “Ποιοι ποταμοί διασχίζουν την Βραζιλία;”. Μια μηχανή αναζήτησης που βασίζεται σε εύρεση κλειδιών σε κείμενο, θα έβρισκε πόρους του διαδικτύου που περιέχουν το κείμενο του ερωτήματος ή τμήμα αυτού. Έτσι ευθεία απάντηση στο ερώτημα θα μπορούσαμε να λάβουμε μόνο αν κάποιος ανέφερε τη φράση του ερωτήματος σε κάποιο αναρτημένο άρθρο με τους

ποταμούς της Βραζιλίας, προκειμένου το άρθρο να βρεθεί σαν πιθανότερη απάντηση από την μηχανή αναζήτησης.

Εύκολα γίνεται κατανοητό το τι θα συνέβαινε στην περίπτωση που, το εν λόγω άρθρο υπήρχε και κάποιος χρήστης δοκίμαζε το ερώτημα "Από ποιους ποταμούς διαρρέεται η Βραζιλία". Μια ελαφριά παραλλαγή του ερωτήματος με χρήση συνώνυμων όρων και μια μηχανή αναζήτησης που βασίζεται σε εύρεση κλειδιών σε κείμενο θα αντιστοιχούσε μόνο το κλειδί "Βραζιλία" μεταξύ ερωτήματος χρήστη και διαθέσιμων κειμένων στο διαδίκτυο. Αυτό θα είχε εξαιρετικά δυσάρεστα αποτελέσματα αφού ο όρος είναι πολύ κοινός και περιέχεται σε εκατομμύρια ιστότοπους και πόρους του διαδικτύου. Έτσι το άρθρο με την απάντηση θα ήταν ένα από τα εκατομμύρια που περιέχουν τον όρο και ο αλγόριθμος εύρεσης σχετικών απαντήσεων δεν θα είχε τρόπο να κρίνει ότι είναι πιο σχετικό από άλλα (όπως θα μπορούσε αν π.χ. βρίσκονταν αντιστοιχία περισσότερων όρων). Αλλά το παραπάνω δεν αποτελεί τη μόνη δυσκολία.

Τι συμβαίνει, λόγου χάρη, στην περίπτωση που δεν έχει αναρτηθεί ένα τέτοιο άρθρο αλλά η πληροφορία υπάρχει στο λήμμα Βραζιλία μιας ηλεκτρονικής εγκυκλοπαίδειας; Μια μηχανή αναζήτησης που βασίζεται σε εύρεση κλειδιών σε κείμενο, θα έβρισκε την παρούσα διπλωματική εργασία (εφόσον έχει αναρτηθεί στο διαδίκτυο) καθώς περιέχει τη φράση του ερωτήματος και το λήμμα της Βραζιλίας από την ηλεκτρονική εγκυκλοπαίδεια (εφόσον περιέχει τον όρο Βραζιλία). Έτσι τα αποτελέσματα θα περιείχαν τόσο μη σχετικά κείμενα (η παρούσα εργασία δεν απαντά στο ποια είναι τα ποτάμια της Βραζιλίας) όσο και κείμενα τα οποία απαντούν μεν αλλά όχι ευθέως στο ερώτημα (ένα άρθρο εγκυκλοπαίδειας για την Βραζιλία περιέχει πληθώρα πληροφοριών τις οποίες ο χρήστης πρέπει να φιλτράρει προκειμένου να φτάσει στην απάντηση που αναζητεί).

Αν από την άλλη πλευρά η αναζήτηση απάντησης κάνει χρήση του σημασιολογικού ιστού τότε θα ληφθούν υπόψη μεταδεδομένα της ερώτησης. Ο λόγος γίνεται για δεδομένα που αφορούν σε δεδομένα του ερωτήματος αναζήτησης. Έχοντας στη διάθεση μας μεταδεδομένα για τον όρο "ποταμοί" στην ερώτηση "Ποιοι ποταμοί διασχίζουν την Βραζιλία;" τα παραπάνω προβλήματα θα μπορούσαν αν αντιμετωπιστούν επιτυχώς.

Αυτό γίνεται κατανοητό αν αναλογιστούμε τη μορφή των απαιτούμενων μεταδεδομένων. Για τον όρο "ποταμοί" αρκεί να είναι διαθέσιμη η γνώση του ότι ο όρος αποτελεί κλίση του λεκτικού "ποταμός" και αφορά στην έννοια του ποταμού. Αντίστοιχα όταν υπάρχει στη γνώση η πληροφορία ότι η "Βραζιλία" αποτελεί χώρα καθώς και το ότι κάθε χώρα διαρρέεται από ποτάμια, τα δεδομένα που είναι αποθηκευμένα στη μνήμη κάποιου συστήματος συνοδεύονται πλέον από μεταδεδομένα. Δεδομένα δηλαδή που περιγράφουν τον τύπο τους καθώς και τη σχέση τους με άλλα δεδομένα.

Η ύπαρξη των μεταδεδομένων κάνει εφικτή την αναζήτηση με βάση το περιεχόμενο και όχι το κλειδί σε κείμενο. Όπως είναι αναμενόμενο, φυσικά, και αυτή η προσέγγιση απαιτείται να υπερβεί κάποια εμπόδια.

Το πρώτο από αυτά είναι ότι η χρήση του σημασιολογικού ιστού μέσω της χρήσης συλλογών δεδομένων (datasets) που δρουν ως θύλακες δεδομένων και μεταδεδομένων, έχει περιορισμούς. Συγκεκριμένα κάθε μηχανισμός που επιχειρεί να κάνει χρήση μεταδεδόμενων και χρησιμοποιεί κάποια συλλογή δεδομένων περιορίζεται από τη γνώση που αυτή περιέχει. Το ερώτημα στο οποίο ζητά απάντηση ο χρήστης, είναι δυνατόν να μην μπορεί να απαντηθεί από ένα σύστημα που εκμεταλλεύεται τη γνώση μιας μόνο συλλογής δεδομένων που μπορεί να διαθέσιμη στο διαδίκτυο αλλά περισσότερων.

Ένα ακόμη ευδιάκριτο εμπόδιο αποτελεί η ανάγκη για ακριβή μετάφραση του ερωτήματος του χρήστη σε όρους που μπορεί να χρησιμοποιηθούν από μια μηχανή. Στην περίπτωση της αναζήτησης με λέξεις κλειδιά το πρόβλημα αυτό δεν υπήρχε. Ο χρήστης έπρεπε να επιλέξει κάποια λέξη κλειδί υποθέτοντας ότι αυτή θα βρίσκεται στο κείμενο που περιέχει την απάντηση στο ερώτημά του.

Αντίθετα όταν η αναζήτηση πραγματοποιείται σε φυσική γλώσσα, προαπαιτείται η ύπαρξη ενός σταδίου μετάφρασης της ερώτησης του χρήστη σε όρους που μπορούν να χρησιμοποιηθούν για την ανεύρεση πληροφορίας από ένα σύστημα που λαμβάνει υπόψη όχι μόνο τα δεδομένα στα οποία αναφέρεται η ερώτηση αλλά και δεδομένα που αφορούν σε αυτά, στον τύπο τους, στις σχέσεις μεταξύ τους.

Αν και οι δυο τελευταίες προκλήσεις φέρονται να φέρουν μεγάλη δυσκολία στην αντιμετώπιση, έχουν εντούτοις μια εξαιρετικά σημαντική ευεργετική προσθήκη στη λειτουργικότητα όσων συστημάτων τις υπερπηδούν. Να προσδίδουν στα συστήματα αυτά την ικανότητα να εξάγουν συμπεράσματα (χρησιμοποιώντας τα μεταδεδομένα) και να δίνουν την ψευδαίσθηση τεχνητής νοημοσύνης στους τελικούς χρήστες.

1.2 Αντικείμενο διπλωματικής

Η παρούσα εργασία καταπιάνεται με δυο ζητήματα που αφορούν στην απάντηση ερωτημάτων ανθρώπινης γλώσσας, από μηχανή, κάνοντας χρήση του σημασιολογικού ιστού.

Το πρώτο εξ αυτών είναι η άρση του περιορισμού του πλήθους των συλλογών δεδομένων που μπορεί να εκμεταλλευτεί ένα σύστημα λογισμικού προκειμένου να απαντήσει σε ερωτήματα ανθρώπινης γλώσσας. Απαραίτητος στόχος η απεξάρτηση από μια βάση και η ευελιξία της προσθήκης οσωνδήποτε συλλογών δεδομένων.

Το δεύτερο, είναι η βελτιστοποίηση της διαδικασίας μετατροπής της ανθρώπινης γλώσσας σε όρους που μπορούν άμεσα να καταναλωθούν από το μηχανισμό απάντησης ερωτημάτων.

Για την περάτωση του πρώτου στόχου αναπτύχθηκε λογισμικό το οποίο επιτρέπει και επεκτείνει τον αριθμό των συλλογών δεδομένων που χρησιμοποιεί σύστημα ανοιχτού λογισμικού (Quepy). Η υλοποίηση του λογισμικού σέβεται την προϋπάρχουσα πλατφόρμα και την επεκτείνει όχι μόνο στο να χρησιμοποιεί ανεξάρτητες συλλογές λογισμικού αλλά και να εξάγει συνδυαστική γνώση που βρίσκεται κατανεμημένη σε παραπάνω της μιας συλλογής γνώσης.

Για τον δεύτερο στόχο αναπτύχθηκε λογισμικό που επιτρέπει την αναγνώριση υπερωτήσεων μες στην ερώτηση του χρήστη και η σύνδεση τους με αντικείμενα που μπορούν να χρησιμοποιηθούν άμεσα από το μηχανισμό απάντησης ερωτημάτων (Quepy). Με την μέθοδο αυτή καθίσταται εφικτή η αναγνώριση σύνθετων ερωτήσεων – ερωτήσεων που προκύπτουν ως συνδυασμός άλλων- και η ακριβής αντιστοίχιση τους με τα αντικείμενα που απαιτούνται για την απάντηση ερωτημάτων ανθρώπινης γλώσσας.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε τις διαφορές των τρόπων μηχανικής απάντησης ερωτημάτων φυσικής γλώσσας
2. Μελετήσαμε την προσέγγιση των Saeedeh Shekarpour, Axel-Cyrille Ngonga, Sören Auer
3. Μελετήσαμε την προσέγγιση των Elías Andrawos, Gonzalo García Berrotarán, Rafael Carrascosa, Laura Alonso i Alemany, Horacio Durán (Quepy)
4. Υλοποιήσαμε αναδρομικό αλγόριθμο αναγνώρισης τυχαίου αριθμού υπερωτήσεων εντός ερωτήματος χρήστη με βάση τυχαίου πλήθους συλλογών δεδομένων
5. Υλοποιήσαμε σύστημα λογισμικού που διαχειρίζεται τυχαίο αριθμό συλλογών δεδομένων διαθέσιμων προς ερώτηση με δυνατότητα απάντησης ερωτήσεων σε ερωτήσεις στις οποίες απαιτούνται παραπάνω της μιας συλλογής δεδομένων
6. Ενσωματώσαμε τους αλγόριθμους στο σύστημα Quepy δημιουργώντας την εφαρμογή ανοιχτού λογισμικού InterQuepy.
7. Αποδείξαμε την ορθότητα των αλγορίθμων με χρήση test driven development.

1.3 Οργάνωση κειμένου

Το περιεχόμενο της εργασίες κατανέμεται ως ακολουθεί. Εργασίες σχετικές με το αντικείμενο της διπλωματικής παρουσιάζονται στο Κεφάλαιο 2 . Το Κεφάλαιο 3 χτίζει το θεωρητικό υπόβαθρο που είναι απαραίτητο για την ανάπτυξη του θέματος. Στο Κεφάλαιο 4 καταγράφονται οι απαιτήσεις του συστήματος, ενώ στο Κεφάλαιο 5 φαίνεται η σχεδίαση των κλάσεων μαζί με το διάγραμμα τους. Λεπτομέρειες και σημεία προσοχής της υλοποίησης διαγράφονται στο Κεφάλαιο 7. Το Κεφάλαιο 8 περιέχει τον επίλογο και τέλος στο Κεφάλαιο 9 αναφέρονται βιβλιογραφικές και όχι μόνο αναφορές.

2 *Σχετικές εργασίες*

2.1 Question Answering Over Linked Data από Saeedeh Shekarpour, Axel-Cyrille Ngonga Ngomo, Sören Auer

Εδώ Μια από τις προτεινόμενες μεθόδους φαίνεται στην εργασία των shekarpour et al [4]. Συγκεκριμένα, αν υποθέσουμε ότι τίθεται ένα ερώτημα σε φυσική γλώσσα, ο προτεινόμενος μηχανισμός παράγει από αυτό ένα ερώτημα στη γλώσσα SPARQL.

Το SPARQL ερώτημα που παράγεται λαμβάνει υπόψη του το γεγονός ότι η απάντηση μπορεί να συνδυάζει πληροφορίες που βρίσκονται κατανεμημένες σε διαφορετικά εναποθετήρια δεδομένων, και αν υπάρχει διαθέσιμη διεπαφή για αυτά, τότε αρκεί να την τροφοδοτήσουμε με αυτό για να λάβουμε την απάντηση.

Αναλυτικά, ο μηχανισμός χωρίζεται σε δυο φάσεις.

2.1.1 Φάση Πρώτη – Αποσαφήνιση Πόρων

Έχοντας το ερώτημα του χρήστη δοσμένο σε φυσική γλώσσα, αρχικά το ερώτημα σπάει σε λέξεις. Στη φάση αυτή, επιχειρείται η αντιστοίχιση των λέξεων με υπάρχοντες πόρους, όπως αυτοί έχουν οριστεί στα εναποθετήρια δεδομένων που συγκροτούν μια γνώση. Με τον όρο υπάρχοντες πόροι δεν εννοούνται βέβαια πόροι ενός μόνο εναποθετήριου. Αντίθετα, το sparql query που θα δημιουργηθεί τελικά, αφορά σε διασυνδεδεμένα δεδομένα τα οποία βρίσκονται κατανεμημένα σε διαφορετικές γνώσεις διαφορετικών εναποθετήριων στο διαδίκτυο.

Λόγω της δομής της φυσικής γλώσσας, μια μόνο λέξη -του ερωτήματος του χρήστη- δεν είναι απαραίτητο να αντιστοιχεί σε έναν πόρο. Για παράδειγμα, αν το ερώτημα του χρήστη ήταν “όχι βενζινοκίνητα οχήματα”, δεν θα είχε νόημα να αντιστοιχηθεί η λέξη “όχι” σε κάποιο πόρο, η λέξη “βενζινοκίνητα” σε κάποιον άλλο και ούτω καθεξής. Το πρόβλημα της αντιστοίχησης λέξεων - πόρων λοιπόν χωρίζεται σε δυο τμήματα. Αρχικά την ομαδοποίηση των λέξεων του χρήστη σε υποπροτάσεις (“όχι βενζινοκίνητα”, “οχήματα”) και στη συνέχεια, τη συσχέτιση κάθε μιας υποπρότασης με κάποιο πόρο. Συνοπτικά τα δυο αυτά τμήματα αντιμετωπίζονται ως εξής:

Για την κατάτμηση του αρχικού ερωτήματος σε υποπροτάσεις, δοκιμάζονται μόνο πιθανές κατατμήσεις. Κατατμήσεις στις οποίες δεν μπορεί να βρεθεί ούτε ένας αντιστοιχισμένος πόρος δεν θεωρούνται πιθανές και έτσι μειώνεται ο όγκος των κατατμήσεων που δοκιμάζονται.

Όσο για την συσχέτιση υποπροτάσεων – πόρων, για κάθε υποπρόταση της εξεταζόμενης κατάτμησης βρίσκεται λίστα με πιθανούς πόρους. Κάθε πιθανός πόρος έχει κάποια βαθμολογία (σε σχέση με την ομοιότητα και το πόσο δημοφιλής είναι) καθώς και κάποιο μονοπάτι να τον συνδέει με τους άλλους πόρους άλλων υποπροτάσεων της ίδιας κατάτμησης. Ο τελικός στόχος είναι φυσικά, η εύρεση της καλύτερης αλληλουχίας πόρων. Έχοντας πολλούς πιθανούς πόρους για κάθε κατάτμηση καταλήγουμε σε ένα σύνολο αλληλουχιών εκ των οποίων αρκεί να διαλέξουμε την καλύτερη με βάση τα παραπάνω κριτήρια.

2.1.2 Φάση Δεύτερη – Κατασκευή Ερωτήματος SPARQL

Αφού έχει βρεθεί το σύνολο των αντιστοιχισμένων -στους όρους του χρήστη- πόρων, R, το επόμενο βήμα είναι να καθοριστεί ο γράφος ερωτήματος G'. Για να αποσαφηνιστεί η δομή του, ας υποθέσουμε ότι μοντελοποιούμε τις RDF τριπλέτες με τη μορφή ενός γράφου G ο οποίος περιέχει όλους τους πόρους και τις ιδιότητες των διαθέσιμων εναποθετηρίων. Ο G' θα είναι τότε υπογράφος του G και θα περιέχει όλους τους πόρους του R και όσο το δυνατόν λιγότερους κόμβους και ακμές του G ώστε να παραμένει συνεκτικός. Φυσικά πρέπει να σημειωθεί ότι στα διασυνδεδεμένα δεδομένα, τα περιεχόμενα του συνόλου R μπορεί να ανήκουν σε διαφορετικά εναποθετήρια. Για το λόγο αυτό ο αλγόριθμος κατασκευής του SPARQL ερωτήματος, είναι ικανός να διαχειρίζεται ακμές - ιδιότητες που ενώνουν πόρους διαφορετικών εναποθετηρίων.

Ο πυρήνας των SPARQL ερωτημάτων δεν είναι άλλος από τους γράφους ερωτήματος. Ο αλγόριθμος κατασκευής τους, δέχεται σαν είσοδο το σύνολο R και δίνει σαν έξοδο ένα σύνολο από πρότυπα γράφων καθένα εκ των οποίων αναπαριστά ένα σύνολο ισομορφικών γράφων ερωτήματος (2 γράφοι θεωρούνται ισομορφικοί ως προς τις ακμές αν ο ένας μπορεί να προκύψει από τον άλλο με απλή μετονομασία ακμών).

Ως κριτήρια επιλογής του καλύτερου γράφου ερωτήματος από τους επιστρεφόμενους, χρησιμοποιείται η δεσμευμένη πιθανότητα να είναι σχετικός ο γράφος ερωτήματος όταν είναι σχετικοί οι πόροι που εξετάζονται. Δηλαδή αν QG ένας από τους εξεταζόμενους γράφους ερωτήματος και R ένα εξεταζόμενο σύνολο πόρων στους οποίους

αντιστοιχεί το ερώτημα του χρήστη, τελικά ο QG θα είναι αυτός που θα επιλεχθεί αν P(QG|R) είναι η μεγαλύτερη σε σχέση με όλους τους υπόλοιπους που εξετάστηκαν.

Εδώ έχει αξία να αναφερθεί ότι ως προς τη δομή, ένα γράφος ερωτήματος μπορεί να αναλυθεί σε τριπλέτες (si, pi, oi) όπου κάθε μια αναπαριστά ένα σχήμα κόμβου – ακμής – κόμβου, αν επιλέξουμε τη γραφική αναπαράσταση ή πόρου – ιδιότητα – τιμής/πόρου αν επιλέξουμε την κλασική μοντελοποίηση. Το παραπάνω σχόλιο ξεκαθαρίζει τα εξής. Αν και η P(QG|R) θα εξαρτάται από την πιθανότητα το επιλεγμένο σύνολο R να είναι σχετικό με το ερώτημα του χρήστη, θεωρώντας ότι οι τριπλέτες είναι ανεξάρτητες θα ισχύει P(QG|R) = Πi P(si, pi, oi) (όπου το i = 1... n αν n ο αριθμός των τριπλετών στις οποίες αναλύεται ο QG). Προκειμένου, λοιπόν, να επιλεχθεί ο γράφος ερωτήματος με την μεγαλύτερη P(QG|R), και δεδομένου ότι P(si, pi, oi) < 1 , επιχειρείται να βρεθεί γράφος ερωτήματος με τον ελάχιστο αριθμό από τριπλέτες για να μεγιστοποιηθεί η πιθανότητα. Μια άλλη παράμετρος που επηρεάζει την πιθανότητα αυτή, είναι ο αριθμός των ελεύθερων μεταβλητών – των μεταβλητών δηλαδή που ανήκουν σε κάποια τριπλέτα αλλά δεν έχουν δεθεί με κάποιο πόρο του R. Εδώ επίσης ισχύει ότι περισσότερες ελεύθερες μεταβλητές αυξάνουν την αβεβαιότητα ρίχνοντας την τιμή της πιθανότητας και για αυτό επίσης επιχειρείται η ελαχιστοποίηση τους πλήθους αυτών. Τέλος για την εύρεση του εν λόγω γράφου ερωτήματος, επιβάλλονται τρεις ακόμη περιορισμοί:

- κάθε όρος του R να βρίσκεται σε τουλάχιστον μια τριπλέτα
- κάθε si που αντιστοιχεί σε κόμβο/πόρο της τριπλέτας απαιτείται να βρίσκεται εντός του πεδίου ορισμού της pi ακμής/ιδιότητας της τριπλέτας και
- κάθε oi που αντιστοιχεί σε κόμβο/πόρο/τιμή της τριπλέτας απαιτείται να βρίσκεται εντός του πεδίου τιμών της pi ακμής/ιδιότητας της τριπλέτας

Η σημασία των δυο τελευταίων περιορισμών αναδεικνύεται αν συνυπολογίσουμε το γεγονός ότι λαμβάνονται υπόψη και οι υπερκλάσεις και οι υπεριδιότητες των πόρων και των ιδιοτήτων που συμμετέχουν στις τριπλέτες. Συγκεκριμένα, ορίζουμε ως CT ενός πόρου, το σύνολο των υπερκλάσεων του - δηλαδή κλάσεων που στη βάση της γνώσης σχετίζονται με τον πόρο αυτό μέσω της ιδιότητας rdf:type. Το CT μπορεί να υπολογιστεί εύκολα χρησιμοποιώντας τη μέθοδο forward chaining (βλ. ενότητα 1) στις δηλώσεις rdf:type και rdfs:subClassOf, ενώ ακολουθείται η ίδια προσέγγιση για τις ιδιότητες προκειμένου να αντλήσουμε γνώση για το πεδίο ορισμού και το πεδίο τιμών τους σχηματίζοντας τα αντίστοιχα σύνολα CD (περιέχει τον τύπο που μπορεί αν οριστεί ως υποκείμενο της ιδιότητας και όλους τους υπερτύπους του) και CR (περιέχει τον τύπο που μπορεί αν οριστεί ως

αντικείμενο της ιδιότητας και όλους τους υπερτύπους του). Έτσι, για να βρεθούν οι ιδιότητες που συνδέουν δυο τυχαίους πόρους, έστω A και B, αρκεί να βρεθούν ιδιότητες που το CD τους έχει μη κενή τομή με το CT του A και το CR τους μη κενή τομή με το CT του B ή το αντίστροφο (CR μη κενή τομή με το CT του A και CD μη κενή τομή με το CT του B).

Το γεγονός ότι εξάγεται έμμεση γνώση με την παραπάνω διαδικασία, είναι η πεμπτουσία του προτεινόμενου μηχανισμού και το στοιχείο που δίνει την ψευδαίσθηση της αλληλεπίδρασης με τεχνητή νοημοσύνη, στο χρήστη της εφαρμογής.

2.2 Quepy από Elías Andrawos, Gonzalo García Berrotarán, Rafael Carrascosa, Laura Alonso i Alemany, Horacio Durán

Μια άλλη προσέγγιση προτείνεται από την ομάδα των Elías Andrawos et al χρησιμοποιώντας το Quepy framework [1]. Το Quepy είναι ένα πλαίσιο λογισμικού (framework) ανοιχτού κώδικα γραμμένο στη γλώσσα Python [5]. Χρησιμοποιείται για την μετατροπή ενός ερωτήματος φυσικής γλώσσας σε ερώτημα γλώσσας βάσης δεδομένων. Το πλαίσιο είναι τροποποιήσιμο έτσι ώστε να δέχεται μια γκάμα ερωτήσεων τις οποίες "αντιλαμβάνεται" και για τις οποίες μπορεί να κατασκευάζει ερωτήματα προς τα εναποθετήρια γνώσης με τα οποία είναι να διασυνδεθεί.

Ας υποθέσουμε ότι ενδιαφερόμαστε για ερωτήσεις του τύπου:

Ποιος είναι ο <τάδε> ; π.χ.

Ποιος είναι ο Tom Cruise ; ή

Ποιος είναι ο πρόεδρος Ομπάμια;

Τι είναι <κάτι> ; π.χ.

Τι είναι ένα αυτοκίνητο; ή

Τι είναι η γλώσσα προγραμματισμού Python;

Λίστα με <αντικείμενα> της <φίρμας> π.χ.

Λίστα με λογισμικό της Oracle

Λίστα με αυτοκίνητα της Tesla

Η βασική δομή του πλαισίου λογισμικού είναι η ακόλουθη:

- dbpedia/parsing.py: το αρχείο που ορίζονται οι κανονικές εκφράσεις στις οποίες αντιστοιχούν τα ερωτήματα σε φυσική γλώσσα που παρέχει ο χρήστης και που θα μεταφραστούν σε μια αφηρημένη σημασιολογική αναπαράσταση

- dbpedia/dsl.py: το αρχείο που ορίζεται η γλώσσα τους σχήματος βάσης. Στην περίπτωση χρήσης της SPARQL, σε αυτό καθορίζονται συνήθως αντικείμενα της οντολογίας: ονόματα σχέσεων κ.α.
- dbpedia/settings.py: το αρχείο παραμετροποίησης ορισμένων στοιχείων της εγκατεστημένης εφαρμογής
- main.py: προαιρετικό αρχείο στο οποίο μπορεί να τοποθετηθεί κώδικας ο οποίος θα καλεί την εφαρμογή και θα λειτουργεί σαν διεπαφή με αυτή.

Το πλαίσιο λογισμικού κάνει χρήση της εργαλειοθήκης λογισμικού (toolkit) NLTK. Η πλατφόρμα χρησιμοποιείται για εφαρμογές γραμμένες σε Python που επεξεργάζονται ανθρώπινη γλώσσα. Παρέχει διεπαφές για πάνω από 50 συλλογές από λεκτικούς πόρους (lexical resources) όπως το WordNet [6], μαζί με μια σουίτα από βιβλιοθήκες επεξεργασίας κειμένου (για classification, tokenization, stemming, tagging, parsing, και semantic reasoning) [7]

Επίσης, για τον χειρισμό των κανονικών εκφράσεων (regular expressions) χρησιμοποιείται η βιβλιοθήκη λογισμικού REfO [8] η οποία παρέχει λειτουργικότητα πολύ παρόμοια με αυτή του re module της python αλλά για αυθαίρετες ακολουθίες από αντικείμενα αντί για ακολουθίες από χαρακτήρες (strings). Επιπλέον, είναι δυνατόν να χρησιμοποιηθεί για σύγκριση όχι μόνο η ισότητα μεταξύ αντικειμένων, αλλά μια οποιαδήποτε συνάρτηση (σε Python). Για παράδειγμα για μια ακολουθία ακεραίων, μπορεί να κατασκευαστεί μια κανονική έκφραση που ικανοποιείται για εναν ακέραιο αριθμό τον οποίο ακολουθεί ένας πρώτος τον οποίο ακολουθεί αριθμός που διαιρείται με το 3.

Έχοντας τα παραπάνω διαθέσιμα, προκειμένου η εφαρμογή να "αντιλαμβάνεται" ερωτήσεις του τύπου που προκαθορίσαμε, χρειάζεται να ορίσουμε τις κανονικές εκφράσεις που θα ικανοποιούν οι ερωτήσεις σε φυσική γλώσσα του χρήστη και να τις μετατρέψουμε σε μια αφηρημένη σημασιολογική αναπαράσταση. Στο βήμα αυτό εκτός από το να καθορίζεται τι είδους ερωτήσεις θα μπορεί να χειρίζεται η εφαρμογή, καθορίζεται και το τι θα κάνει με αυτές.

Ένα παράδειγμα για την κανονική έκφραση που χειρίζεται ερωτήσεις της δεύτερης κατηγορίας - Τι είναι … (στην αγγλική What is…) ορίζεται, όπως προαναφέρθηκε στο αρχείο dbpedia/parsing.py:

```
1  from refo import Group, Question
2  from quepy.dsl import HasKeyword
3  from quepy.parsing import Lemma, Pos, QuestionTemplate
4
5  from dsl import IsDefinedIn
6
7  class WhatIs(QuestionTemplate):
8      """
9      Regex for questions like "What is ..."
10     Ex: "What is a car"
11     """
12
13     target = Question(Pos("DT")) + Group(Pos("NN"), "target")
14     regex = Lemma("what") + Lemma("be") + target + Question(Pos("."))
15
16     def interpret(self, match):
17         thing = match.target.tokens
18         target = HasKeyword(thing)
19         definition = IsDefinedIn(target)
20         return definition
```

στο παραπάνω τμήμα κώδικα, αξίζει να σημειωθούν:

Οι χειριστές των κανονικών εκφράσεων όπως το WhatIs είναι υποκλάσεις της QuestionTemplate (γραμμή 7) και θέτουν πάντα σε ένα χαρακτηριστικό κλάσης που ονομάζεται regex μια κανονική έκφραση refo (refo regex) (γραμμή 14).

Η κανονική έκφραση που περιγράφεται στην γραμμή 14, ικανοποιείται από την ερώτηση του τύπου “Τι είναι το X” (στην αγγλική What is X) αλλά σε διαφοροποιήσεις του τύπου “Τι ήταν το X” (στην αγγλική What was X) ή “Τι ήταν τα X” (αντίστοιχα What were X). Και αυτό διότι χρησιμοποιείται η συνάρτηση Lemma για την αντιστοίχηση της κανονικής έκφρασης του ρήματος και όχι το ρήμα αυτό καθαυτό.

Το X ορίζεται από τη μεταβλητή target η οποία καθορίζεται στη γραμμή 13:

```
target = Question(Pos("DT")) + Group(Pos("NN"), "target")
```

και ικανοποιείται από ακολουθίες χαρακτήρων που θα περαστούν στη σημασιολογία για να αποτελέσουν τμήμα του τελικού ερωτήματος βάσης. Στην παραπάνω γραμμή κώδικα, ορίζουμε ότι η μεταβλητή target, αποτελείται προαιρετικά από ένα διαχωριστικό (DT) το οποίο ακολουθεί ένα ουσιαστικό (NN) και λαμβάνει το ψευδώνυμο “target”.

Εδώ να σημειωθεί ότι το quepy έχει πρόσβαση σε διάφορα επίπεδα γλωσσικών πληροφοριών που σχετίζονται με τις λέξεις του ερωτήματος χρήστη, όπως η ετικέτα lemma (λήμμα) και η ετικέτα part of speech (μέρος του λόγου). Η πληροφορία αυτή χρειάζεται για το συσχετισμό με τις ερωτήσεις και βρίσκεται από το τμήμα λογισμικού που λέγεται tagger και που παρέχεται από τη εργαλειοθήκη NLTK[7].

Τέλος αν η κανονική έκφραση ικανοποιείται από το ερώτημα χρήστη, θα κλιθεί η αντίστοιχη μέθοδος Interpret (στο παράδειγμα γραμμές 16 - 22), η οποία καθορίζει τη σημασιολογία της ερώτησης. Στο παραπάνω παράδειγμα, το περιεχόμενο της μεταβλητής target είναι το αποτέλεσμα του κατηγορήματος HasKeyword. Το κατηγόρημα αυτό είναι τμήμα του λεξιλογίου της συγκεκριμένης βάσης που χρησιμοποιείται Αντίθετα, το κατηγόρημα IsDefinedIn είναι τμήμα ενός αφηρημένου σημασιολογικού μέρους το οποίο περιγράφεται παρακάτω.

Το Quepy χρησιμοποιεί μια αφηρημένη σημασιολογία ανεξάρτητη γλώσσας ως αναπαράσταση η οποία αργότερα αντιστοιχίζεται σε ερώτημα βάσης. Αυτή η προσέγγιση επιτρέπει στις ερωτήσεις του χρήστη να αντιστοιχίζονται σε διαφορετικά ερωτήματα βάσης δίδοντας έτσι διαφάνεια και πολυμορφισμό στην προσέγγιση.

Στο παράδειγμα, η γλώσσα που εξαρτάται από το πεδίο, ορίζεται στο αρχείο dbpedia/dsl.py και το κατηρόγρημα IsDefinedIn που χρησιμοποιήθηκε στη γραμμή 21 ορίζεται:

```
from quepy.dsl import FixedRelation

class IsDefinedIn(FixedRelation):
    relation = "rdfs:comment"
    reverse = True
```

Το οποίο σημαίνει ότι τα αντικείμενα της κλάσης IsDefinedIn αποτελούν ιδιότητες όπου το υποκείμενο-πόρος είναι rdf:comment. Κατασκευάζοντας μια κλάση, διασφαλίζεται ένα ακόμη επίπεδο αφαίρεσης το οποίο επιτρέπει την εύκολη ενσωμάτωσή του με κανονικές εκφράσεις.

Κάνοντας χρήση όλων των παραπάνω, και τοποθετώντας τις ακόλουθες γραμμές κώδικα στο main.py:

```
import quepy
dbpedia = quepy.install("dbpedia")
target, query, metadata = dbpedia.get_query("what is a blowtorch?")
print query
```

λαμβάνουμε το ακόλουθο ερώτημα βάσης

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX quepy: <http://www.machinalis.com/quepy#>

SELECT DISTINCT ?x1 WHERE {
  ?x0 quepy:Keyword "blowtorch".
  ?x0 rdfs:comment ?x1.
}
```

το οποίο αρκεί να τροφοδοτηθεί σε κάποια τερματική διεπαφή (endpoint) της dbpedia[9] (όπως το virtuoso [10]) για να δώσει αποτελέσματα.

3 *Θεωρητικό υπόβαθρο*

3.1 Σημασιολογικός Ιστός και Απάντηση Ερωτημάτων

Σύμφωνα με άρθρο των Tim Berners-Lee (εφευρέτης του παγκόσμιου ιστού και οραματιστής του σημασιολογικού ιστού), James Hendler και Ora Lassila: "The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users" [11].

Μια ελεύθερη απόδοση του οποίου θα ήταν ότι ο σημασιολογικός ιστός είναι δυνατόν να φέρει δομή στο περιεχόμενο του διαδικτύου δημιουργώντας έτσι γόνιμο έδαφος για την ανάπτυξη εξελιγμένων πρακτόρων λογισμικού. Ποια η χρήση τέτοιων πρακτόρων και πως θα μπορούσαμε να χρησιμοποιήσουμε έναν εξελιγμένο σημασιολογικό ιστό για την απάντηση περίπλοκων ερωτημάτων σε φυσική γλώσσα; Για να απαντηθεί το ερώτημα ας εξετάσουμε αρχικά το τι αποτελεί το σημασιολογικό ιστό.

Ο όρος σημασιολογικός προέρχεται από τη λέξη σημασιολογία και αναφέρεται στην απόδοση νοήματος στα σύμβολα μιας γλώσσας[12]. Ως σημασιολογικός ιστός νοείται ένας ιστός διασυνδεδεμένων δεδομένων στον οποίο τα δεδομένα δεν είναι προσπελάσιμα μόνο από τους ανθρώπους. Αντίθετα πράκτορες λογισμικού μπορούν να "κατανοούν" τα δεδομένα ενός τέτοιου ιστού, να αλληλεπιδρούν και να βγάζουν συμπεράσματα βασιζόμενοι στα δεδομένα και σε συλλογιστικές διαδικασίες.

Στον ορισμό λοιπόν του σημασιολογικού ιστού ξεχωρίζει η έννοια της δομημένης πληροφορίας. Πληροφορία που μπορεί να γίνει αυτόματα κατανοητή από λογισμικό, είτε αυτό έχει τη μορφή διαδικτυακής εφαρμογής, διαδικτυακής υπηρεσίας ή ευφυούς πράκτορα [13]. Με αυτή ανοίγεται πληθώρα δυνατοτήτων και αλλάζει ουσιαστικά η μορφή του παγκόσμιου ιστού, σε έναν ιστό όπου λογισμικό εκμεταλλεύεται έξυπνα τη διάσπαρτη πληροφορία για λογαριασμό των ανθρώπων.

Για την αποσαφήνιση των δυνατοτήτων ενός τέτοιου ιστού, στη συνέχεια ακολουθεί ένα σενάριο χρήσης του:

Δυο πολυάσχολοι φίλοι προσπαθούν να κλείσουν εισιτήρια για έναν αγώνα καλαθοσφαίρισης. Θέλουν να δουν την αγαπημένη τους τοπική ομάδα να παίζει εντός έδρας, αλλά παρατηρούν ότι στους δυο επόμενους αγώνες της, ο καθένας έχει κάποια άλλη δραστηριότητα προγραμματισμένη την ώρα του αγώνα. Αντί να συγκρίνουν τις ατζέντες τους με όλους τους αγώνες, μέχρι να βρεθεί αγώνας που να διεξάγεται κάποια ώρα που δεν έχει

προγραμματίσει κανείς τίποτα, ο πρώτος αναθέτει στον προσωπικό του πράκτορα λογισμικού, έστω πράκτορας-Α, να προτείνει αγώνες.

Ο πράκτορας-Α αναζητά στο σημασιολογικό ιστό τις ημερομηνίες στις οποίες η συγκεκριμένη ομάδα αγωνίζεται εντός έδρας και συμπληρώνει μια λίστα με αυτές. Έχοντας πρόσβαση στο προσωπικό πρόγραμμα του πρώτου φίλου και αφού λάβει έγκριση από τον αντίστοιχο πράκτορα-Β του δεύτερου φίλου, βρίσκει για ποιες από τις ημερομηνίες της λίστας τα δύο προσωπικά προγράμματα είναι κενά και συμπληρώνει μια νέα λίστα με ημερομηνίες. Η νέα λίστα επιστρέφεται σαν αποτέλεσμα στον πρώτο φίλο και αυτός προτείνει τον αγώνα που θα πραγματοποιηθεί νωρίτερα εντός λίστας.

Επειδή ο δεύτερος φίλος ενδιαφέρεται πολύ και για το ποιοι θα είναι οι αντίπαλοι της ομάδας του όμως, αναθέτει στον πράκτορα-Β να βρει τους αντίπαλους από τις προτεινόμενες ημερομηνίες. Αυτός επικοινωνεί με τον πράκτορα-Α και αφού λάβει την λίστα, αναζητά στο σημασιολογικό ιστό τις αντίπαλες ομάδες. Καθώς τις επιστρέφει, ο δεύτερος φίλος προτείνει διαφορετικό αγώνα κρίνοντας πλέον και βάσει των αντιπάλων. Το αποτέλεσμα είναι μια διένεξη από την οποία αποφασίζουν να μη δουν ποτέ πια κανέναν αγώνα μαζί.

Το παραπάνω σενάριο δεν είναι εφικτό με τις υπάρχουσες μεθόδους αναπαράστασης πληροφοριών στον παγκόσμιο ιστό. Ένας ιστός που θα αποτελούσε επέκταση του υπάρχοντος παγκόσμιου ιστού και στον οποίο η πληροφορία θα ήταν κατανοητή σε πράκτορες λογισμικού και όχι μόνο σε ανθρώπους όμως, θα το επέτρεπε. Θα ενισχύονταν η διαλειτουργικότητα της επεξεργασίας πληροφοριών μεταξύ πρακτόρων και εμμέσως θα διευκολύνονταν οι άνθρωποι χρήστες, αφού πιο εξελιγμένοι πράκτορες - πράκτορες που θα φαίνονταν σαν να φέρουν νοημοσύνη - θα αναλάμβαναν την διεκπεραίωση εργασιών που φαίνονται αδύνατες στην υπάρχουσα δομή του ιστού.

Αυτό που στην ουσία μπορεί να παρέχει ο σημασιολογικός ιστός είναι η κατάλληλη υποδομή για την αποδοτικότερη αξιοποίηση των δεδομένων που υπάρχουν ήδη στον παγκόσμιο ιστό[14]. Πληροφορίες που ζουν σε ιστοσελίδες, βάσεις δεδομένων, ηλεκτρονικές υπηρεσίες ή και που ανταλλάσσονται μεταξύ πρακτόρων, έχοντας την κατάλληλη δομή, σε έναν σημασιολογικό ιστό, γίνονται προσβάσιμες στο λογισμικό το οποίο λειτουργεί για τον χρήστη, κατ' απαίτηση του χρήστη. Δίνοντας έτσι τη δυνατότητα σε αυτόν, να βρίσκει τις πληροφορίες που πραγματικά αναζητά, φιλτράροντας τα δεδομένα λαμβάνοντας υπόψη το είδος και τη φύση τους και υπολογίζοντας τη συσχέτιση τους με άλλα δεδομένα.

Μόνο σε έναν τέτοιο ιστό, όπου τα δεδομένα έχουν δομή, με μεταξύ τους ιεραρχίες και σχέσεις που γίνονται αντιληπτές από το λογισμικό μπορεί η ανάκτηση πληροφορίας και η απάντηση ερωτημάτων να γίνει απτή. Και στο σημείο αυτό γεννιέται το ερώτημα, ποια δομή είναι απαραίτητο να έχει η πληροφορία που βρίσκεται στον παγκόσμιο ιστό προκειμένου να είναι προσπελάσιμη.

3.2 Πλαίσιο RDF και Απάντηση Ερωτημάτων

Προκειμένου να γίνουν εφικτά, σενάρια όπως αυτό που προαναφέρθηκε, απαιτείται χρήση μεταδεδομένων, αλληλεπίδραση με οντολογίες και φυσικά δυνατότητα εξαγωγής λογικών συμπερασμάτων από έναν πράκτορα λογισμικού. Αυτό φαίνεται ευθύς αμέσως αν αναρωτηθούμε ποια προβλήματα θα αντιμετώπιζε ένας τέτοιος πράκτορας έχοντας σαν είσοδο μια HTML ιστοσελίδα.

Είναι φανερό ότι ένα αρχείο HTML περιέχει, εκτός από το περιεχόμενό του και πληροφορίες για τη μορφοποίησή του. Δηλαδή ως επί τω πλείστων πληροφορίες που βοηθούν το πρόγραμμα περιήγησης στο να δείχνει στον χρήστη το περιεχόμενο με συγκεκριμένο τρόπο. Δεν είναι όμως αυτές οι πληροφορίες που θα ενδιέφεραν έναν πράκτορα.

Προκειμένου να γίνει εφικτό το όραμα του σημασιολογικού ιστού και να απαντηθούν προκλήσεις όπως η απάντηση ερωτημάτων σε δεδομένα ενός τέτοιου ιστού, χρειάζονται δυο πράγματα. Αφενός να υπάρχει κάπου διαθέσιμη η πληροφορία. Πιο σωστά πρέπει να είναι διαθέσιμη η γνώση που θα περιέχει την απάντηση στο ερώτημά μας. Αφετέρου να εφαρμοστεί πάνω σε αυτή κάποιος αλγόριθμος συλλογιστικής, ο οποίος θα εξάγει την απάντηση στο ερώτημά μας. Και όλα αυτά θα πρέπει να μπορούν να γίνονται για μεγάλα μεγέθη δεδομένων.

Η εξάρτηση των δυο θεμελιωδών αυτών στοιχείων, του αλγόριθμου που εξάγει τις απαντήσεις και του τρόπου με τον οποίο αναπαρίσταται η γνώση είναι προφανώς ισχυρότατη. Αν επιχειρήσουμε να προσεγγίσουμε το ζήτημα ξεκινώντας από το πως θα έμοιαζε η γνώση, μια αρκετά διαδεδομένη και διαισθητική προσέγγιση θα ήταν να χρησιμοποιηθεί ένας γράφος για την αναπαράσταση της.

Ας πάρουμε για παράδειγμα την περίπτωση συστήματος που θέλουμε να γνωρίζει τις ερευνητικές εργασίες ερευνητών που ανήκουν σε συνεργαζόμενα πανεπιστημιακών κοινοτήτων.

Τα μέλη αυτών, έστω ότι καταχωρούν τις εργασίες τους μαζί με κάποιες πληροφορίες για αυτές όπως ο τίτλος, η περίληψη και το θέμα. Ένα ερώτημα που θα μπορούσαμε να θέσουμε στο σύστημα αυτό είναι ποιες εργασίες έχουν θέμα την τεχνητή νοημοσύνη. Αν και τόσο μια εργασία με θέμα το σημασιολογικό ιστό όσο και μια άλλη με θέμα τα νευρωνικά δίκτυα αποτελούν εργασίες με θέμα την τεχνητή νοημοσύνη (και τα δυο θέματα υπάγονται

στην ευρύτερη αυτή ερευνητική περιοχή) η αναζήτηση δεν θα έφερνε αποτελέσματα αν πραγματοποιούνταν με τις σημερινές διαδεδομένες τεχνολογίες.

Όσο έξυπνο και αν ήταν το ερώτημα στη βάση δεδομένων που είναι καταχωρημένες οι εργασίες, η αναζήτηση δεν θα έφερνε αποτελέσματα καθώς η εργασία που αφορά στον σημασιολογικό ιστό θα είχε καταχωρημένο στο θέμα τη φράση "σημασιολογικός ιστός" και όχι τεχνητή νοημοσύνη. Το ίδιο φυσικά θα ίσχυε για την εργασία με θέμα τα νευρωνικά δίκτυα, καθιστώντας την αναζήτηση άκαρπη.

Προκειμένου να αντιμετωπιστεί αυτή η αδυναμία απαιτείται περαιτέρω δόμηση της πληροφορίας. Η πληροφορία που λείπει εδώ είναι σύνδεση της έννοιας "σημασιολογικός ιστός" ως υπο-έννοιας, με την έννοια "τενητή νοημοσύνη" καθώς και η αντίστοιχη σύνδεση της υπο-έννοιας "νευρωνικά δίκτυα" με την "τεχνητή νοημοσύνη".

Το επόμενο ερώτημα προκύπτει με φυσικό τρόπο. Μια είναι η καταλληλότερη υποδομή για να περιγραφεί η παραπάνω συσχέτιση.

3.2.1 Ορισμός RDF

Η RDF είναι ένα πρότυπο μοντέλο για την ανταλλαγή δεδομένων στον Παγκόσμιο Ιστό. RDF έχει χαρακτηριστικά που διευκολύνουν τη συγχώνευση δεδομένων, ακόμη και αν τα υποκείμενα σχήματα διαφέρουν, και υποστηρίζει ειδικότερα την εξέλιξη των σχημάτων με την πάροδο του χρόνου, χωρίς να απαιτείται όλοι όσοι καταναλώνουν τα δεδομένα να πρέπει να αλλάξουν μεθόδους.[15]

Η RDF επεκτείνει τη δομή σύνδεσης του Web ώστε να χρησιμοποιούνται URIs[16] ως τρόπος αναφοράς της σχέσης μεταξύ αντικειμένων, καθώς και των δύο άκρων του συνδέσμου (αυτό συνήθως αναφέρεται ως «τριπλέτα»). Χρησιμοποιώντας αυτήν την απλή δομή, είναι εφικτή χρήση και ο διαμοιρασμός των δεδομένων μεταξύ τελείως διαφορετικών εφαρμογών.

Μια πολύ σημαντική παρατήρηση σε αυτό το σημείο είναι ότι μια RDF τριπλέτα μπορούμε να τη δούμε είτε σαν μια τριάδα που φέρει πληροφορία ή ακόμη και σαν γράφο όπου ο πρώτος πόρος αντιστοιχεί σε κόμβο η ιδιότητα σε ακμή και ο τρίτος πόρος/τιμή επίσης σε κόμβο.

Τέλος, ένας ακόμη λόγος της υιοθέτησης της RDF είναι ότι ακόμη και ο εμπνευστής του σημασιολογικού ιστού Tim Berners-Lee, σε σχετικό άρθρο του το 2006 προτείνει (ανάμεσα σε μια σειρά 4 κανόνων) τη χρήση του μοντέλου αυτού [17].

3.2.2 RDF και Συλλογές Δεδομένων (datasets)

Έχοντας περιγράψει το μοντέλο αναπαράστασης πόρων RDF, κρίνεται αναγκαία η αναφορά στην χρησιμοποίηση του στις συλλογές δεδομένων. Δεν πρόκειται παρά περιγραφές δεδομένων με την έννοια της "τριπλέτας" ή του γράφου (μιας και το μοντέλο μπορούμε να το κατανοήσουμε και με τους δυο αυτούς τρόπους) που μπορεί να θεωρηθεί ως αντίστοιχο μιας βάσης δεδομένων μια και οι δυο προσεγγίσεις αφορούν στην καταχώρηση δεδομένων.

Στο σημείο αυτό πρέπει να σημειωθούν τα εξής. Σύμφωνα με τον οργανισμό W3C που πρότεινε την RDF, η RDF ορίζει την έννοια της RDF συλλογής δεδομένων (RDF dataset), ως μια δομή αποτελούμενη από έναν διακριτό RDF γράφο και κανέναν ή περισσότερους ονοματισμένους γράφους που εννώνονται χρησιμοποιώντας ενα IRI ή έναν κενό κόμβο και έναν RDF γράφο. Αν και οι RDF γράφοι έχουν μοντελο-θεωρητική σημασιολογία που καθορίζει ποιές τοποθετήσεις του κόσμου καθιστούν έναν RDF γράφο έγκυρο, δεν υπάρχει ακόμη συμφωνία για κάποια τυπική σημασιολογία για τις RDF συλλογές δεδομένων[18].

Πριν κλείσει η ενότητα για την RDF, κρίνεται σκόπιμη μια ακόμη σημείωση. Ο λόγος, για την χρήση των RDF από πάροχους πληθώρας δεδομένων όπως η dbpedia[9] και το linkedmdb[19] που καθιστούν διαθέσιμα τα δεδομένα σε μορφή RDF.

Συγκεκριμένα η dbpedia, αποτελεί μια προσπάθεια μελών της κοινότητας της και χρηματοδοτούμενη από το πλήθος, για την εξαγωγή δομημένης πληροφορίαςαπό την Wikipedia και διάθεσης της στο διαδίκτυο. Επιτρέπει σύνθετες ερωτήσεις στα δεδομένα της Wikipedia και σύνδεση διαφορετικών συλλογών δεδομένων του διαδικτύου με τα δεδομένα της Wikipedia. [20]

3.3 Γλώσσα Ερωτημάτων SPARQL

Προκειμένου ο χρήστης να αποκτήσει πρόσβαση στα δεδομένα μιας συλλογής δεδομένων, όπως είναι φυσικό, απαιτείται η χρήση κάποιας γλώσσας που να επιτρέπει την επικοινωνία αυτή. Ο ίδιος οργανισμός (W3C) που πρότεινε το μοντέλο RDF, σε πρόταση του στις 15 Ιανουαρίου του 2008 πρότεινε και την SPARQL [21] ως γλώσσα περιγραφής ερωτημάτων σε δεδομένα που περιγράφονται από το RDF.

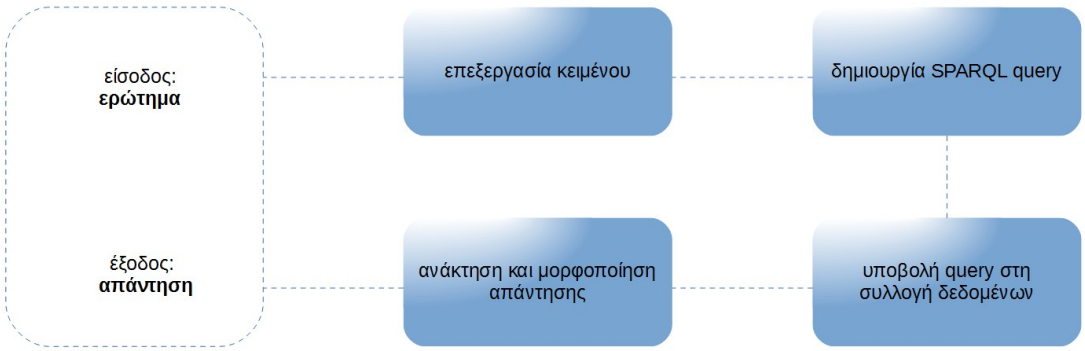
Ο πιο αναλυτικός ορισμός της SPARQL περιλαμβάνει τα ακόλουθα. [22] Η SPARQL (αναδρομικό ακρωνύμιο για το πρωτόκολλο SPARQL και RDF Query Language) είναι μια γλώσσα επερώτησης RDF, δηλαδή, μια σημασιολογική γλώσσα ερωτημάτων για βάσεις δεδομένων, είναι σε θέση να ανακτά δεδομένα που είναι αποθηκευμένα σε Resource Description Framework (RDF) μορφή. Η SPARQL επιτρέπει για ένα ερώτημα να αποτελείται από τριπλέτες, συνενώσεις, διαζεύξεις και προαιρετικά πρότυπα.

Υλοποιήσεις της γλώσσας υπάρχουν για πολλές γλώσσες προγραμματισμού. Υπάρχουν εργαλεία που επιτρέπουν σε κάποιον να συνδεθεί και ημι-αυτόματα να κατασκευάσει ένα ερώτημα SPARQL για τερματικό SPARQL όπως επίσης εργαλεία που μεταφράζουν SPARQL ερωτήματα σε άλλους γλώσσες επερωτήσεων, για παράδειγμα, με τον SQL και να XQuery.

4 *Ανάλυση Απαιτήσεων Συστήματος*

4.1 Αρχιτεκτονική

Η συνολική αρχιτεκτονική του συστήματος συνοψίζεται στο ακόλουθο block diagram:



σε αυτό χρήζουν επισήμανσης τα ακόλουθα:

Καθώς η εφαρμογή InterQuepy πρέπει να επεκτείνει την Quepy δεν κρίνεται σκόπιμο να γραφτούν από την αρχή και οι τέσσερις δομικές λειτουργίες που απεικονίζονται παραπάνω. Αντίθετα, σκοπός είναι η χρησιμοποίηση όσο μεγαλύτερου μέρους του κώδικα του Quepy είναι εφικτό και όχι η κατάργηση των ευέλικτων χαρακτηριστικών του.

Προς αυτήν την κατεύθυνση, και προκειμένου να επιτευχθούν οι δυο ερευνητικοί στόχοι α) της άρσης του περιορισμού του πλήθους των συλλογών δεδομένων που χρησιμοποιούνται και β) της αναγνώρισης υπερωτήσεων σε σύνθετα ερωτήματα χρήστη, απαιτείται η τροποποίηση των πρώτων δυο δομικών λειτουργιών.

Της επεξεργασίας κειμένου αφού ληφθεί το ερώτημα του χρήστη. Και της δημιουργίας του SPARQL ερωτήματος.

4.2 Περιγραφή Επιμέρους Λειτουργιών

Για τις κορυφαίες αυτές λειτουργίες, αναλύονται εδώ διεξοδικά οι απαιτήσεις συνοδευόμενες από παραδείγματα χρήσης.

4.2.1 Επεξεργασία Κειμένου - Είσοδος

Απαιτούμενη είσοδο της επιμέρους αυτής λειτουργίας αποτελεί μια πρόταση που θα περιέχει την ερώτηση του χρήστη. Η πρόταση θα είναι τύπου string και είναι δυνατόν να περιέχει είτε λέξεις κλειδιά είτε ερώτηση σε ανθρώπινη γλώσσα.

4.2.2 Επεξεργασία Κειμένου - Έξοδος

Η απαιτούμενη έξοδος της επιμέρους αυτής λειτουργίας θα διαφοροποιείται αναλόγως με το αν είναι δυνατή η αναγνώριση υπερωτήσεων. Αν δεν είναι δυνατή, τότε η έξοδος αντικατοπτρίζει την είσοδο προκειμένου να ακολουθηθεί η διαδικασία που ακολουθούνταν στην εφαρμογή Quepy πριν τις επεκτάσεις του InterQuepy.

Σε περίπτωση που είναι δυνατή η αναγνώριση υπερωτήσεων, η έξοδος θα πρέπει να είναι μια λίστα. Κάθε όρος της λίστας μια υπερώτηση με τη μορφή λίστας λέξεων.

Η αναγνώριση των υπερωτήσεων είναι απαραίτητο να γίνεται λαμβάνοντας υπόψη τα μεταδεδομένα που έχουν οριστεί στους ορισμούς των αποδεκτών ερωτήσεων.

4.2.3 Επεξεργασία Κειμένου - Παράδειγμα

αν το σύστημα αναγνωρίζει τις υπερωτήσεις:

- “movies about Giant Monsters”
- “movies starring Akira Takarada”

και λάβουμε σαν είσοδο:

“movies about Giant Monsters starring Akira Takarada”

απαιτείται έξοδος:

[['starring', 'Akira', 'Takarada'], ['about', 'Giant', 'Monsters']]

4.2.4 Δημιουργία SPARQL ερωτήματος - Είσοδος

Απαιτούμενη είσοδος της επιμέρους αυτής λειτουργίας αποτελεί μια πρόταση που θα περιέχει την ερώτηση του χρήστη ή μια λίστα υπερωτήσεων. Στην περίπτωση απλής ερώτησης, η πρόταση θα είναι τύπου string ενώ στη περίπτωση λίστας υπερωτήσεων η λίστα θα έχει ως στοιχεία τις υπερωτήσεις. Οι υπερωτήσεις θα είναι σε μορφή λίστας λέξεων.

4.2.5 Δημιουργία SPARQL ερωτήματος - Έξοδος

Η απαιτούμενη έξοδος της επιμέρους αυτής λειτουργίας θα είναι ένα sparql query. Σε περίπτωση που αφορά σε περισσότερες από μια συλλογές δεδομένων είναι απαραίτητο να έχει Federated μορφή καταδεικνύοντας σε ποιο dataset αναφέρεται το κάθε επιμέρους τμήμα του.

4.2.6 Δημιουργία SPARQL ερωτήματος - Παράδειγμα

αν το σύστημα αναγνωρίζει τις υπερωτήσεις:

- “movies about Giant Monsters”
- “movies starring Akira Takarada”
- “sequel of Godzilla vs. the Sea Monster”

και λάβουμε σαν είσοδο:

['starring', 'Akira', 'Takarada'], ['about', 'Giant', 'Monsters'], ['sequel', 'of', 'a', 'movie']

απαιτείται έξοδος:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX quepy: <http://www.machinalis.com/quepy#>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX dbpprop: <http://dbpedia.org/property/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT DISTINCT ?output WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    ?x200 owl:sameAs ?x01.
    ?x200 rdf:type movie:film.
    ?x200 movie:sequel ?result.
    ?result rdfs:label ?output.
  }
  SERVICE <http://dbpedia.org/sparql> {
    ?x00 rdf:type foaf:Person.
    ?x00 rdfs:label "Akira Takarada"@en.
    ?x01 dbpprop:starring ?x00.
    ?x01 rdf:type dbpedia-owl:Film.

    ?x100 rdf:type skos:Concept.
    ?x100 rdfs:label "Giant Monster"@en.
    ?x01 dcterms:subject ?x100.
    ?x01 rdf:type dbpedia-owl:Film.
  }
}
```

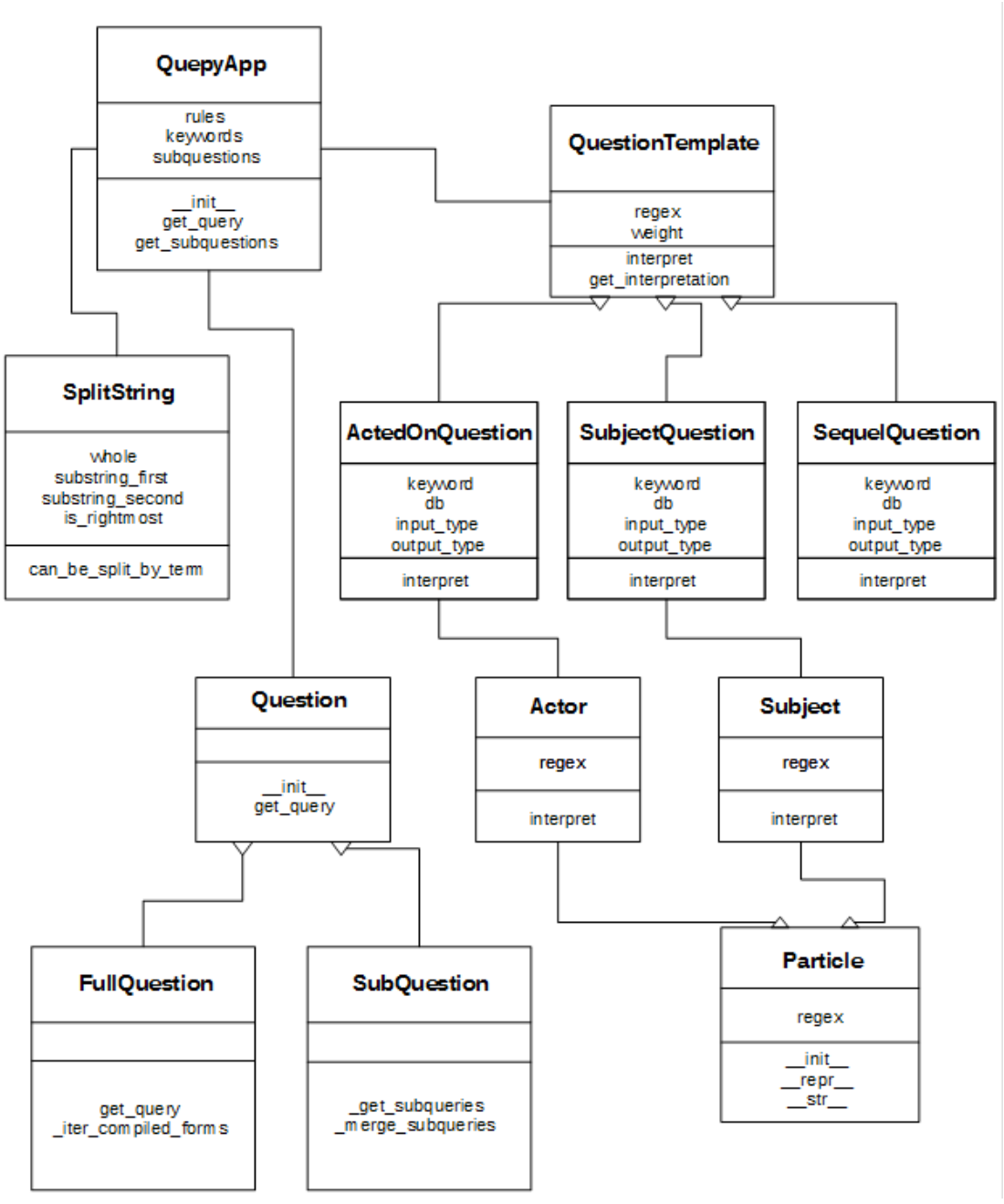
5 *Σχεδίαση Συστήματος*

Στο κεφάλαιο αυτό πραγματοποιείται η σχεδίαση του συστήματος. Περιγράφονται τόσο οι κλάσεις που δημιουργήθηκαν όσο και όσες τροποποιήθηκαν προκειμένου να φιλοξενηθούν οι νέες λειτουργίες.

Το κεφάλαιο ανοίγει η αρχιτεκτονική των υποσυστημάτων που δημιουργήθηκαν ή/και τροποποιήθηκαν. Δεν πρόκειται για την συνολική και αφηρημένη αρχιτεκτονική του κεφαλαίου 4 αλλά για την αρχιτεκτονική που διέπει τα επιμέρους συστήματα.

5.1 Αρχιτεκτονική

Ακολουθεί το block diagram κλάσεων, όπου φαίνεται η κληρονομικότητα και οι συνδέσεις μεταξύ των κλάσεων. Μια κλάση συνδέεται με μια άλλη αν μια μέθοδός της χρησιμοποιεί αντικείμενο από την άλλη ως παράμετρο.



Από το διάγραμμα αξίζει να σημειωθούν δύο θέματα πάνω στον τρόπο λειτουργίας του λογισμικού. Ο λόγος που τα θέματα προς σημείωση είναι δύο είναι επειδή ο τρόπος λειτουργίας χωρίζεται σε δυο μέρη.

5.1.1 Μέρος Πρώτο - Εγκατάσταση

Στο πρώτο μέρος πραγματοποιείται η εγκατάσταση του προγράμματος όπου στο αντικείμενο QuepyApp δημιουργούνται και αποθηκεύονται στο πεδίο rules (με την κλίση της __init__), όλοι οι κανόνες σύμφωνα με τους οποίους γίνεται εφικτή η αντιστοίχηση ερωτήσεων χρήστη με ερωτήματα sparql προς κατανάλωση από κάποιο endpoint.

Οι κανόνες αυτοί αποτελούν στιγμιότυπα αντικειμένων που κληρονομούν ιδιότητες και μεθόδους από την κλάση QuestionTemplate. Αφού δεν κρίνεται σκόπιμο να καταγραφούν στο διάγραμμα όλοι οι κανόνες αντιστοίχισης που θα μπορούσαν να δημιουργηθούν για την απάντηση οποιοδήποτε ερωτήματος, για θέματα σαφήνειας, φαίνονται σε αυτό μόνο οι απαραίτητοι κανόνες για την απάντηση ενός σύνθετου ερωτήματος.

Συγκεκριμένα, ας υποθέσουμε ότι ο χρήστης ενδιαφέρεται για το sequel μια ταινίας της οποίας -αφού δεν θυμάται το όνομα- θυμάται ότι έπαιζε ο ηθοποιός τάδε με θέμα ταινίας το δείνα. Το ερώτημά του λοιπόν “which movie are sequel of a movie about Giant Monsters starring Akira Takarada”, προκειμένου να απαντηθεί είναι απαραίτητο να έχουν δημιουργηθεί και αποθηκευτεί κατά την εγκατάσταση κάποιοι κανόνες.

Ένας εξ αυτών θα είναι ο κανόνας που αναγνωρίζει την φράση “ starring Akira Takarada” μέσα σε άλλες φράσεις. Ο κανόνας που θα είναι στιγμιότυπο της κλάσης ActedOnQuestion, είναι απαραίτητο να περιέχει κάποια μεταδεδομένα τα οποία θα χρησιμοποιηθούν σε μεταγενέστερη φάση, τόσο για τον εντοπισμό τη φράσης μέσα σε άλλες όσο και για την κατασκευή του sparql ερωτήματος.

Συγκεκριμένα, στο πεδίο keyword το στιγμιότυπο του ActedOnQuestion θα έχει την τιμή “starring” προκειμένου να αναγνωριστεί το υπερώτημα εντός συνδιαστικού ερωτήματος. Επίσης, το πεδίο db θα έχει την τιμή “http://dbpedia.org/sparql” αφού το συγκεκριμένο υπερώτημα απαντάται από την συλλογή δεδομένων της dbpedia. Τα δυο ακόμη πεδία που φαίνονται στο γράφημα input_type (θα λάβει τιμή 'foaf:Person') και output_type θα λάβει τιμή ('dbpedia-owl:Film') αναμένεται να χρησιμοποιηθούν επίσης στην κατασκευή του sparql ερωτήματος. Ο λόγος ύπαρξης δεν είναι άλλος από τον καθορισμό της έννοιας της εισόδου

και της εξόδου του υπερωτήματος προκειμένου να συνδεθεί με τα υπόλοιπα υπερωτήματα καν να κατασκευαστεί έτσι το συνδιαστικό ερώτημα.

Αντίστοιχα με τον παραπάνω κανόνα ισχύουν και για τον κανόνα που αναγνωρίζει τη φράση “about Giant Monsters” και αποτελεί στιγμιότυπο της κλάσης SubjectQuestion. Με τελείως ανάλογο σκεπτικό προκύπτει ότι στο στιγμιότυπο το πεδίο db θα έχει την τιμή “http://dbpedia.org/sparql” (το συγκεκριμένο υπερώτημα απαντάται επίσης από την συλλογή δεδομένων της dbpedia) και τα δυο πεδία input_type και output_type τιμές 'skos:Concept' και 'dbpedia-owl:Film' αντίστοιχα.

Το μοτίβο διαφοροποιείται ελαφρώς στον κανόνα αντιστοίχησης της φράσης “sequel of”. Αυτή θα αποτελεί στιγμιότυπο της κλάσης SequelQuestion. Σε αυτό το πεδίο db θα λάβει τιμή 'http://data.linkedmdb.org/sparql' μιας και το υπερώτημα απαντάται στη συλλογή δεδομένων linkedmdb. Κατά τα άλλα, στα υπόλοιπα πεδία των μεταδεδομένων λαμβάνονται αναμενόμενες τιμές keyword = 'sequel', input_type = 'dbpedia-owl:Film', output_type = 'movie:film'.

Πριν ολοκληρωθεί η περιγραφή του πρώτου μέρους του τρόπου λειτουργίας, δηλαδή της εγκατάστασης, αξίζει να σημειωθεί κάτι ακόμη. Το πεδίο κάθε κανόνα αποτελεί εγγραφή στο λεξικό keywords του στιγμιότυπου της κλάσης QuepyApp.

5.1.2 Μέρος Δεύτερο - Εκτέλεση

Όσο αφορά στο δεύτερο μέρος της εκτέλεσης, το στιγμιότυπο της κλάσης QuepyApp καλείται να βρει το sparql query έχοντας την ερώτηση σε ανθρώπινη γλώσσα (και τους κανόνες καθώς και τα keywords στα αντίστοιχα πεδία rules, keywords) με την μέθοδο get_query. Η ίδια αυτή μέθοδος θα λέγαμε ότι περισσότερο κατανέμει τις εργασίες σε άλλες μεθόδους.

Συγκεκριμένα, το πρώτο πράγμα που κάνει είναι να βρίσκει τα υπερωτήματα και τα καταχωρεί στο πεδίο subquestions του στιγμιοτύπου της κλάσης μέσω της μεθόδου get_subquestions. Προκειμένου να επιτευχθεί αυτό είναι απαραίτητη η ύπαρξη μιας ακόμη κλάσης που επιτρέπει τη λειτουργία της μεθόδου.

Ο λόγος γίνεται για την SplitString. Τα αντικείμενα που ανήκουν στην κλάση αυτή “γνωρίζουν” αν περιέχουν υπερώτημα βάση κάποιου keyword με τη βοήθεια της μεθόδου can_be_split_by_term. Έχουν επίσης αποθηκευμένα πεδία στα οποία φαίνεται – σε περίπτωση που υπάρχουν επερωτήματα – ποια είναι αυτά και τη θέση έχουν εντός του

αρχικού ερωτήματος. Τα πεδία αυτά δεν είναι άλλα από τα whole_string, substring_first, substring_second, is_rightmost.

Προκειμένου να ευρεθεί το sparql query, στη συνέχεια δημιουργείται ένα αντικείμενο της κλάσης Question. Η κλάση κληρονομείται από δύο υποκλάσεις καθώς είναι και οι δυο οι περιπτώσεις που είναι αναγκαίο να χειρίζεται ο κώδικας.

Για τα ερωτήματα χρήστη που δεν περιέχουν επερωτήσεις γίνεται χρήση της κλάσης FullQuestion. Αυτή υλοποιεί τις μεθόδους get_query, _iter_compiled_forms, _set_metadata προκειμένου να ευρεθεί το τελικό sparql query. Εδώ σημειώνεται ότι η κλάση αυτή δημιουργείται για να περιέχει μεθόδους που είχαν γραφτεί ήδη στην αρχική υλοποίηση του Quepy και που εφόσον δεν αποτελούν τον μοναδικό τρόπο υπολογισμού του sparql ερωτήματος.

Ο άλλος τρόπος υπολογισμού του sparql ερωτήματος πραγματοποιείται κάνοντας χρήση της κλάσης Subquestion. Στιγμιότυπα αυτής της κλάσης ακολουθούν την εξής απλή λογική για την κατασκευή του τελικού sparql query. Αρχικά υπολογίζονται τα υπερωτήματα με τη μέθοδο _get_subqueries και στη συνέχεια ενοποιούνται στο τελικό sparql query με την _merge_subqueries.

Όπως είναι σύνηθες, πολλοί μέθοδοι χρησιμοποιούν άλλες μεθόδους για την επίτευξη των στόχων τους. Ετσι και η _get_subqueries καλεί τις _get_subquery_by_subquestion_and_rule, _add_offset_to_query_vars, _rename_output_var, _rename_input_var, _find_out_var, _get_final_subquery.

5.2 Περιγραφή Κλάσεων

Εδώ περιγράφονται συνοπτικά τα κυριότερα πεδία και μέθοδοι των κλάσεων.

5.2.1 QuepyApp

rules :

κανόνες που είναι υπεύθυνοι για την αναγνώριση ερωτήματος χρήστη.

keywords :

λέξεις κλειδιά με τα οποία γίνεται η αναγνώριση υπερωτήσεων.

subquestions :

υπερωτήσεις εντός του ερωτήματος χρήστη.

get_query :

μέθοδος υπεύθυνη για την εύρεση sparql query από το ερώτημα χρήστη.

get_subquestions :

μέθοδος υπεύθυνη για την εύρεση υπερωτημάτων εντός του ερωτήματος χρήστη.

5.2.2 QuestionTemplate

regex :

κανονική έκφραση (regular expression) που μπορεί να αντιστοιχιστεί στο ερώτημα χρήστη.

weight :

βάρος κανόνα προκειμένου να είναι δυνατή η διαφοροποίηση της σημαντικότητας.

interpret :

μέθοδος υπεύθυνη για την μετάφραση του κανόνα σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί για την εύρεση sparql query. Απαιτείται η υπερκάλυψη της από τις κλάσεις που την κληρονομούν.

get_interpret :

μέθοδος υπεύθυνη για την κλίση της μετάφρασης των κανόνων σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί για την εύρεση sparql query.

5.2.3 ActedOnQuestion

keyword :

λέξη κλειδί με την οποία γίνεται η αναγνώριση της υπερώτησης που αντιστοιχεί στον συγκεκριμένο κανόνα.

db :

η βάση στην οποία θα γίνει το ερώτημα που αντιστοιχεί στον συγκεκριμένο κανόνα

input_type :

ο τύπος που αποτελεί την είσοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

output_type :

ο τύπος που αποτελεί την έξοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

5.2.4 SubjectQuestion

keyword :

λέξη κλειδί με την οποία γίνεται η αναγνώριση της υπερώτησης που αντιστοιχεί στον συγκεκριμένο κανόνα.

db :

η βάση στην οποία θα γίνει το ερώτημα που αντιστοιχεί στον συγκεκριμένο κανόνα

input_type :

ο τύπος που αποτελεί την είσοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

output_type :

ο τύπος που αποτελεί την έξοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

5.2.5 SequelQuestion

keyword :

λέξη κλειδί με την οποία γίνεται η αναγνώριση της υπερώτησης που αντιστοιχεί στον συγκεκριμένο κανόνα.

db :

η βάση στην οποία θα γίνει το ερώτημα που αντιστοιχεί στον συγκεκριμένο κανόνα.

input_type :

ο τύπος που αποτελεί την είσοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

output_type :

ο τύπος που αποτελεί την έξοδο στο sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

5.2.6 Actor

regex :

κανονική έκφραση (regular expression) που μπορεί να αντιστοιχιστεί στο τμήμα ερώτησης του συγκεκριμένου κανόνα χρήστη.

interpret :

μέθοδος υπεύθυνη για την μετάφραση του κανόνα σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί σε συνδυασμό με άλλα για την εύρεση sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

5.2.7 Subject

regex :

κανονική έκφραση (regular expression) που μπορεί να αντιστοιχιστεί στο τμήμα ερώτησης του συγκεκριμένου κανόνα χρήστη.

interpret :

μέθοδος υπεύθυνη για την μετάφραση του κανόνα σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί σε συνδυασμό με άλλα για την εύρεση sparql query που αντιστοιχεί στον συγκεκριμένο κανόνα.

5.2.8 Particle

regex :

κανονική έκφραση (regular expression). Απαιτείται η υπερκάλυψη του από κλάσεις που την κληρονομούν.

interpret :

μέθοδος υπεύθυνη για την μετάφραση του κανόνα σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί σε συνδυασμό με άλλα για την εύρεση sparql query. Απαιτείται η υπερκάληψη της μεθόδου από κλάσεις που την κληρονομούν.

5.2.9 SplitString

regex :

κανονική έκφραση (regular expression). Απαιτείται η υπερκάλυψη του από κλάσεις που την κληρονομούν.

interpret :

μέθοδος υπεύθυνη για την μετάφραση του κανόνα σε είδος αντικειμένου το οποίο θα χρησιμοποιηθεί σε συνδυασμό με άλλα για την εύρεση sparql query. Απαιτείται η υπερκάληψη της μεθόδου από κλάσεις που την κληρονομούν.

5.2.10 Question

__init__ :

(μέθοδος που καλείται κατά την δημηιουργία στιγμιότυπου της κλάσης) υπεύθυνη για την καταχώρηση των ορισμάτων (subquestions, rules, keywords) στη τοπική μνήμη του ανικειμένου (μέσω τοπικών μεταβλητών)

get_query :

μέθοδος υπεύθυνη για την εύρεση του τελικού sparql query λαμβάνοντας υπόψη το γεγονός ότι το ερώτημα μπορεί να είναι απλό ή συνδιαστικό (περιέχει υπερωτήσεις εντός).

5.2.11 FullQuestion

get_query :

μέθοδος υπεύθυνη για την εύρεση του τελικού sparql query όπως το έβρισκε η εφαρμογή Quepy πριν την τροποποίηση της.

_iter_compiled_forms :

μέθοδος υπεύθυνη να επιστρέφει την μετάφραση του κανόνα που αντιστοιχεί στο ερώτημα χρήστη όπως το έβρισκε η εφαρμογή Quepy πριν την τροποποίηση της.

5.2.12 SubQuestion

get_query :

μέθοδος υπεύθυνη για την εύρεση του τελικού sparql query χρησιμοποιώντας αλγόριθμο υπολογισμού sparql query από υπερωτήσεις

_get_subqueries :

μέθοδος υπεύθυνη να βρει τα sparql που αντιστοιχεί σε κάθε υπερώτημα, τη συλλογή δεδομένων στην οποία στοχεύει και να επιστρέψει λίστα με αυτά.

_merge_subqueries :

μέθοδος υπεύθυνη να συνθέσει το τελικό sparql query από μια λίστα με τα sparql που αντιστοιχεί σε κάθε υπερώτημα και τη συλλογή δεδομένων στο οποίο στοχεύουν.

6 *Υλοποίηση*

6.1 Λεπτομέρειες υλοποίησης

Επικεντρώνοντας το ενδιαφέρον στους δυο στόχους της παρούσας εργασίας α) της άρσης του περιορισμού του πλήθους των συλλογών δεδομένων που χρησιμοποιούνται και β) της αναγνώρισης υπερωτήσεων σε σύνθετα ερωτήματα χρήστη, αναλύονται στις ακόλουθες υποενότητες τα τμήματα του κώδικα που τους αφορούν άμεσα.

Αναφορικά με τον πρώτο στόχο, γίνεται λόγος για τον αναδρομικό αλγόριθμο αναγνώρισης υπερωτήσεων. Ο αλγόριθμος υλοποιείται στη μέθοδο get_subquestions της κλάσης QuepyApp.

Όσο αφορά στον δεύτερο στόχο, δυο είναι τα κύρια στοιχεία που επιτρέπουν την χρησιμοποίηση περισσότερες της μιας συλλογών δεδομένων. Αφενός η εισαγωγή μεταδεδομένων κατά την σχεδίαση των κανόνων που αναγνωρίζουν ερωτήματα. Αφετέρου η κατασκευή υπερωτημάτων σε sparql και η ενοποίηση τους στην συνολική federated μορφή του τελικού sparql ερωτήματος.

Για την εισαγωγή μεταδεδομένων δεν θα γίνει ιδιαίτερη μνεια σε αυτό το σημείο. Η τοποθέτηση τους καλύφθηκε στο κεφάλαιο 5. Εδώ ενδεικτικά αναφέρουμε ότι η τοποθέτηση τους απαιτείται να γίνεται στις κλάσεις που κληρονομούν την QuestionTemplate. Παραδείγματα τέτοιων κλάσεων είναι οι ActedOnQuestion, SubjectQuestion καθώς και SequelQuestion.

Όσο για την ατασκευή υπερωτημάτων σε sparql και η ενοποίηση τους στην συνολική federated μορφή του τελικού sparql ερωτήματος η υλοποίηση χωρίζεται σε δυο μεθόδους. Πρόκειται για τις get_query και _merge_subqueries της κλάσης Question.

Στις επόμενες υποενότητες αναλύονται λοιπόν μόνο ο αναδρομικός αλγόριθμος αναγνώρισης υπερωτημάτων (ενότητα 6.1.1), η κατασκευή υπερωτημάτων σε sparql (ενότητα 6.1.2) καθώς και η ενοποίηση επερωτημάτων sparql σε federated μορφή (ενότητα 6.1.3).

6.1.1 Αναδρομικός Αλγόριθμος Αναγνώρισης Υπερωτημάτων

Ο κώδικας και μια σύντομη περιγραφή ακολουθούν.

```
158     def get_subquestions(self, words, keywords):
159         """ input:
160                 ['hello', 'keyword1', 'is', 'foo', 'keyword2', 'is', 'bar'] and
161                 ['keyword1', 'keyword2']
162             output:
163                 [['keyword1', 'is', 'foo'], ['keyword2', 'is', 'bar']]"""
164 
165         sstring = SplitString(words) if type(words) is list else words
166 
167         if len(keywords) == 0:
168             return 42
169         keywords_copy = deepcopy(keywords)
170         keyword = keywords_copy.pop()
171 
172         if sstring.can_be_split_by_term(keyword) :
173             second_substring = sstring.substring_second
174             if self.get_subquestions(second_substring, keywords_copy) == 42:
175                 self.subquestions.append(second_substring.whole_string)
176 
177             first_substring = sstring.substring_first
178             if self.get_subquestions(first_substring, keywords_copy) == 42 and \
179                 first_substring.is_rightmost:
180                     self.subquestions.append(first_substring.whole_string)
181         else :
182             return self.get_subquestions(words, keywords_copy)
```

Αν υπάρχει υπερώτηση με το ζητούμενο keyword το ερώτημα σπάει σε επερωτήματα και ακολουθείται ανα δρομικά η διαδικασία για τα υπερωτήματα. Η αναδρομή σταματά όταν δεν υπάρχουν άλλες λέξεις κλειδιά οπότε δίνεται και το σήμα τερματισμού (αριθμός 42).

Αν στη μέθοδο αυτή δοθεί ως είσοδος (πρώτο όρισμα με όνομα words) η πρόταση ως λίστα λέξεων:

['hello', 'keyword1', 'is', 'foo', 'keyword2', 'is', 'bar']

καθώς και οι λέξεις κλειδιά (δεύτερο όρισμα με όνομα keywords):

['keyword1', 'keyword2']

τότε η έξοδος θα είναι μια λίστα προτάσεων (κάθε πρόταση αποτελεί μια λίστα λέξεων):

[['keyword1', 'is', 'foo'], ['keyword2', 'is', 'bar']]

6.1.2 Κατασκευή Υπερωτημάτων σε SPARQL

Ο κώδικας και μια σύντομη περιγραφή ακολουθούν.

```
85      def _get_subqueries(self):
86          subqueries = []
87          counter = 0
88          var_and_type = {}
89
90          for subquestion in self.subquestions:
91              rule_matched = self.rules[self.keywords[subquestion[0]]]
92
93              subquery = self._get_subquery_by_subquestion_and_rule(\
94                                                      subquestion, rule_matched)
95              subquery = self._add_offset_to_query_vars(subquery, counter * 10)
96              subquery = self._rename_output_var(subquery)
97              subquery = self._rename_input_var(subquery, var_and_type, \
98                                                rule_matched.input_type)
99
100             out_var = self._find_out_var(subquery, rule_matched.output_type)
101             if rule_matched.output_type in var_and_type:
102                 new_var = var_and_type[rule_matched.output_type]
103                 subquery = subquery.replace(out_var, new_var)
104             else:
105                 last_output_type = rule_matched.output_type
106                 last_output_db = rule_matched.db
107                 var_and_type[last_output_type] = out_var
108
109             subqueries.append({'db': rule_matched.db, 'query': subquery})
110
111             counter += 1
112
113         final_subquery = self._get_final_subquery(last_output_type, \
114                                                   var_and_type)
115         subqueries.append({'db': last_output_db, 'query': final_subquery})
116
117         return subqueries
```

Για κάθε υπερώτηση βρίσκεται ο κανόνας που έχει αντιστοιχηθεί (γραμμή 91).

Στη συνέχεια με βάση τον κανόνα αυτό βρίσκεται ο πυρήνας του sparql query για την συγκεκριμένη υπερώτηση με χρήση των αρχικών μεθόδων της εφαρμογής Quepy (γραμμή 93).

Ακολούθως οι μεταβλητές του μετονομάζονται έτσι ώστε να αποφεύγονται οι διενέξεις λόγω κοινών ονομάτων (conflicts) με άλλα sparql queries (γραμμές 95 εως 99).

Στη συνέχεια οι μεταβλητές εισόδου ή / και εξόδου του sparql query μετονομάζονται έτσι ώστε να έχουν κοινό όνομα με άλλα sparql queries που αναφέρονται στους ίδους πόρους (γραμμές 100 εως 107).

Κάθε επιμέρους sparql query προστίθεται στη λίστα αποτελεσμάτων (γραμμή 109) και στη λίστα προστίθεται και το τελικό τμήμα sparql που καθορίζει το όνομα της μεταβλητής εξόδου (γραμμή 113).

Η μέθοδος αυτή, όταν καλείται από κάποιο αντικείμενο που έχει αρχικοποιηθεί με τις υπερωτήσεις (πεδίο subquestions):

['starring', 'Akira', 'Takarada'], ['about', 'Giant', 'Monsters'], ['sequel', 'of', 'a', 'movie']

παράγει έξοδο ένα λεξικό το οποίο έχει μια εγγραφή για κάθε υπερώτηση. Η εγγραφή αυτή περιλαμβάνει το sparql query καθώς και της συλλογή δεδομένων στο οποίο αφορά:

'query' :

'?x0 rdf:type foaf:Person.

?x0 rdfs:label "Akira Takarada"@en.

?x1 rdf:type dbpedia-owl:Film.

?x1 dbpprop:starring ?x0.',

'db':

'http://dbpedia.org/sparql',

'query' :

?x100 rdf:type skos:Concept .

?x100 rdfs:label "Giant monster films"@en .

?x1 rdf:type dbpedia-owl:Film .

?x1 dcterms:subject ?x100 .

'db':

'http://dbpedia.org/sparql',

'query' :

?x200 owl:sameAs ?x1.

?x200 movie:sequel ?x201 .

'db':

'http://data.linkedmdb.org/sparql'

Για πληρότητα παραθέτονται εδώ και η μέθοδοι που καλεί η _get_subqueries

```
119     def _get_subquery_by_subquestion_and_rule(self, subquestion, rule):
120         question = encoding_flexible_conversion(' '.join(subquestion))
121         tagger = get_tagger()
122         words = list(tagger(question))
123         subquery_expression, meta = rule.get_interpretation(words)
124         return get_core_sparql_expression(subquery_expression)
125 
126     def _add_offset_to_query_vars(self, query, offset):
127         return query.replace(u'?x', u'?x' + str(offset))
128 
129     def _rename_output_var(self, query):
130         return query.replace('"output_var"@en', '?result')
131 
132     def _rename_input_var(self, query, var_and_type, input_type):
133         if '"input_var"@en' in query:
134             old_var = '"input_var"@en'
135             new_var = var_and_type[input_type]
136             return query.replace(old_var, new_var)
137         return query
138 
139     def _find_out_var(self, query, type):
140         for sentence in query.split('.'):
141             if type in sentence:
142                 var_name = [w for w in sentence.split() if w.startswith('?')]
143                 var_name = '?result' if '?result' in query else var_name[0]
144                 return var_name
145 
146     def _get_final_subquery(self, last_output_type, var_and_type):
147         input_var = var_and_type[last_output_type]
148         return u'\n' + input_var + u' rdfs:label ?output .'
```

6.1.3 Ενοποίηση Επερωτησεων SPARQL

Ο κώδικας και μια σύντομη περιγραφή ακολουθούν.

```
150     def _merge_subqueries(self, subqueries):
151         grouped_subqueries = {}
152         for subquery in subqueries:
153             subquery, db = subquery.values()
154             if db in grouped_subqueries:
155                 grouped_subqueries[db] += subquery
156             else:
157                 grouped_subqueries[db] = subquery
158 
159         template = u"" + settings.SPARQL_PREAMBLE + "\n" +\
160                    u"SELECT DISTINCT ?output WHERE {\n"
161 
162         for db, subquery in grouped_subqueries.items():
163             template += u"SERVICE <" + db + "> {\n"
164             template += u"" + subquery + "}\n"
165 
166         template += u"}"
167 
168         return template
```

Για κάθε επερώτηση ελέγχεται αν η συλλογή δεδομένων στην οποία αφορά είναι κοινή με τη συλλογή δεδομένων άλλων υπερωτήσεων ώστε να γίνονται οι απαραίτητες ομαδοποιήσεις (γραμμή 154 έως157).

Αφού ολοκληρωθούν οι ομαδοποιήσεις, δημιουργείται το απαραίτητο κείμενο που θα δείχνει σε ποια συλλογή δεδομένων αφορούν κάθε ομάδα υπερωτήτσεων (γραμμή 162 εως 164).

Στη μέθοδο αυτή αν δοθεί ως είσοδος τα υπερωτήματα εξόδου της προηγούμενης ενότητας λαμβάνουμε ως έξοδο:

```
SELECT DISTINCT ?output WHERE {
  SERVICE <http://data.linkedmdb.org/sparql> {
    ?x200 owl:sameAs ?x01.
    ?x200 rdf:type movie:film.
    ?x200 movie:sequel ?result.
    ?result rdfs:label ?output.
  }
  SERVICE <http://dbpedia.org/sparql> {
    ?x00 rdf:type foaf:Person.
    ?x00 rdfs:label "Akira Takarada"@en.
    ?x01 dbpprop:starring ?x00.
    ?x01 rdf:type dbpedia-owl:Film.

    ?x100 rdf:type skos:Concept.
    ?x100 rdfs:label "Giant Monster"@en.
    ?x01 dcterms:subject ?x100.
    ?x01 rdf:type dbpedia-owl:Film.
  }
}
```

6.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Η συγκεκριμένη υλοποίηση πραγματοποιήθηκε σε προσωπικό υπολογιστή με λειτουργικό σύστημα ανοιχτού λογισμικού Linux[23] και συγκεκριμένα στη διανομή Ubuntu 14.04[24]. Για την ανάπτυξη της επιστρατεύτηκαν εργαλεία επίσης ανοιχτού λογισμικού όπως το Pycharm[25] που αφορά στην ανάπτυξη λογισμικού στη γλώσσα Python.

Για την λειτουργία της απαραίτητη είναι η εγκατάσταση των ακόλουθων πακέτων/βιβλιοθηκών:

- refo [8]
- nltk [7]
- SPARQLWrapper [26]
- graphviz [27]

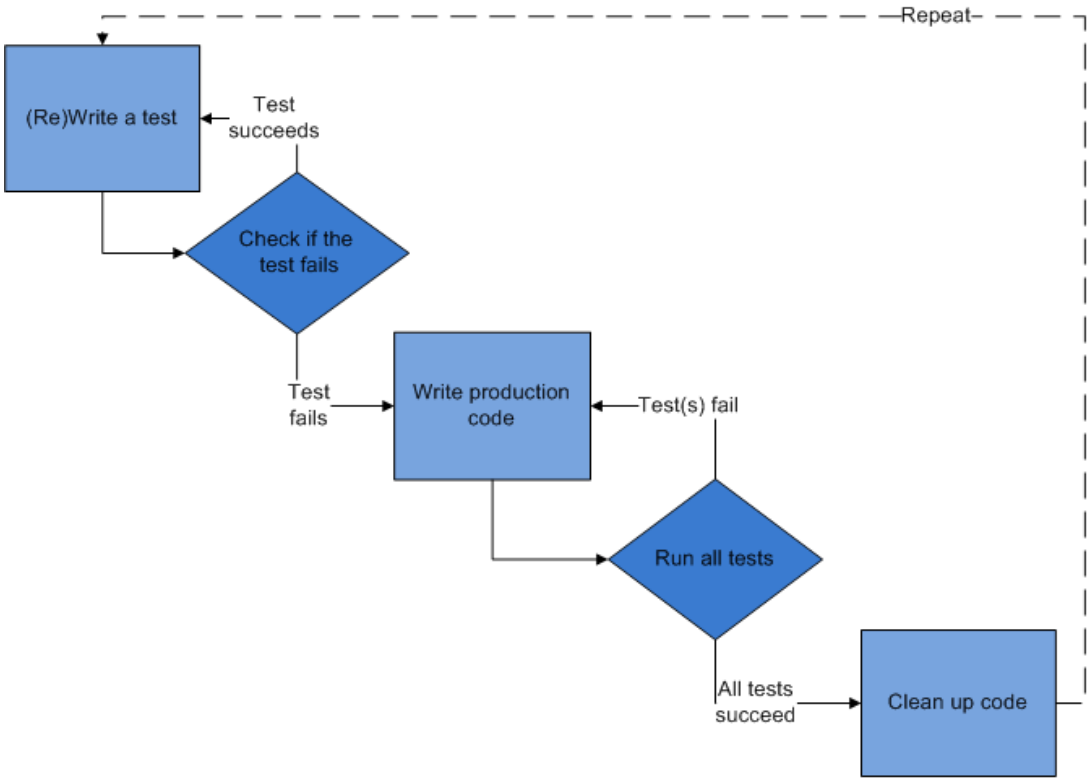
Η διαδικασία εγκατάστασης του κώδικα της διπλωματικής είναι η ακόλουθη (εντολές σε τερματικό linux):

- git clone https://github.com/apostolosSotiropoulos/interQuepy.git
- cd quepy
- sudo python setup.py install
- `quepy nltkdata /some/path/you/find/convenient`

7 *Έλεγχος*

7.1 Μεθοδολογία ελέγχου και Ανάπτυξη Οδηγούμενα από τον Έλεγχο

Σε αντίθεση με κλασικές προσεγγίσεις ανάπτυξης λογισμικού, η εφαρμογή InterQuepy αναπτύχθηκε κάνοντας χρήση της μεθοδολογίας Ανάπτυξης Λογισμικού Οδηγούμενη από τον Έλεγχο (Test Driven Development). Η μεθοδολογία παρουσιάζεται συνοπτικά ακολούθως.



7.1.1 Βήμα 1ο - προσθήκη σεναρίου ελέγχου

Στην ανάπτυξη λογισμικού οδηγούμενη από τον έλεγχο, κάθε νέο χαρακτηριστικό σε μια εφαρμογή ξεκινά με τη σύνταξη ενός σεναρίου ελέγχου για αυτό. Προκειμένου να γραφτεί ένα τέτοιο σενάριο ελέγχου, ο προγραμματιστής πρέπει να έχει κατανοήσει πλήρως τις προδιαγραφές και τις απαιτήσεις του χαρακτηριστικού που πρόκειται να προστεθεί. Για να

επιτευχθεί αυτό είναι δυνατόν να μελετηθούν περιπτώσεις χρήσης και ιστορίες χρηστών ώστε να καλυφθούν οι απαιτήσεις. Η συγγραφή των σεναρίων ελέγχου μπορεί να πραγματοποιηθεί χρησιμοποιώντας οποιοδήποτε πλαίσιο λογισμικού είναι κατάλληλο για το περιβάλλον της εφαρμογής. Φυσικά, ένα σενάριο ελέγχου δεν είναι απαραίτητο να είναι πάντα καινούριο. Θα μπορούσε, απλά, να είναι μια τροποποιημένη έκδοση ενός υπάρχοντος. Τα παραπάνω, διαφοροποιούν την μέθοδο ανάπτυξης λογισμικού σε σχέση με τις κλασικές προσεγγίσεις όπου ο έλεγχος πραγματοποιούνταν μετά τη συγγραφή κώδικα, αφού απαιτούν από τον προγραμματιστή να εστιάσει στις απαιτήσεις πριν από την συγγραφή του κώδικα.

7.1.2 Βήμα 2ο - εκτέλεση όλων των σεναρίων ελέγχου και απαίτηση αποτυχίας μόνο του καινούριου

Το στάδιο αυτό, επικυρώνει ότι η εξάρτηση του σεναρίου ελέγχου λειτουργεί σωστά, ότι η νέο σενάριο ελέγχου δεν θα περάσει χωρίς να απαιτείται εισαγωγή νέου κώδικα, και ότι το απαιτούμενο χαρακτηριστικό δεν υπάρχει ήδη. Αυτό το βήμα ελέγχει επίσης το ίδιο το σενάριο ελέγχου: αποκλείει την πιθανότητα το νέο σενάριο ελέγχου να περνά πάντα, και ως εκ τούτου είναι άνευ αξίας. Το νέο σενάριο ελέγχου θα πρέπει επίσης να αποτύχει για τον αναμενόμενο λόγο. Τέλος, το βήμα αυτό αυξάνει την εμπιστοσύνη του προγραμματιστή ότι το σενάριο ελέγχου δοκιμάζει τη ζητούμενη λειτουργία, και περνά μόνο στις περιπτώσεις για τις οποίες προορίζεται.

7.1.3 Βήμα 3ο - συγγραφή κώδικα

Το επόμενο βήμα είναι η συγγραφή κώδικα που επιτρέπει στο σενάριο ελέγχου να περάσει. Ο νέος αυτός κώδικας, δεν είναι απαραίτητο να είναι τέλειος και μπορεί, για παράδειγμα, να περάσει τη δοκιμασία με έναν άκομψο τρόπο. Αυτό είναι αποδεκτό, διότι θα πρέπει να βελτιωθεί αργότερα όπως διευκρινίζεται στο Βήμα 5. Σε αυτό το σημείο, ο μοναδικός σκοπός του κώδικα είναι να περάσει το τεστ - καμία περαιτέρω (και ως εκ τούτου δεν δοκιμάστηκε) λειτουργικότητα θα πρέπει να προβλεφθεί, ούτε να «επιτρέπεται» σε οποιοδήποτε στάδιο.

7.1.4 Βήμα 4ο - εκτέλεση όλων των σεναρίων ελέγχου

Αν όλα τα σενάρια ελέγχου περάσουν σε αυτό το στάδιο, ο προγραμματιστής μπορεί να είναι βέβαιος ότι ο νέος κώδικας πληροί τις απαιτήσεις των δοκιμών, και δεν σπάει ούτε υποβαθμίζει τα υπάρχοντα χαρακτηριστικά. Εάν πάλι δεν τα καταφέρουν, ο νέος κώδικας θα πρέπει να προσαρμοστεί μέχρι το κάνουν.

7.1.5 Βήμα 5ο - επαναδιατύπωση κώδικα

Για την περάτωση του σταδίου αυτού και την επίτευξη του θεμιτού αποτελέσματος, λαμβάνονται σοβαρά υπόψη τα εξής:

- Η αυξανόμενη βάση του κώδικα θα πρέπει να καθαρίζεται τακτικά κατά τη διάρκεια της ανάπτυξης λογισμικού με οδηγό τον έλεγχο.
- Νέος κώδικας μπορεί να μετακινηθεί από το σημείο που ήταν βολικό να βρίσκεται σε σημείο που είναι λογικό για να περάσει το σενάριο ελέγχου.
- Η επικάλυψη λειτουργικότητας από πολλές μεθόδους πρέπει να αφαιρείται.
- Αντικείμενα, κλάσεις, μεταβλητές και μέθοδοι θα πρέπει να έχουν ονόματα που να αντιπροσωπεύουν σαφώς το σκοπό και τη χρήση τους, καθώς προστίθεται επιπλέον λειτουργικότητα.
- Καθώς προστίθενται χαρακτηριστικά, μέθοδοι και αντικείμενα μπορεί να αυξάνουν το μέγεθός τους. Ο κατακερματισμός τους σε μικρότερα τμήματα, μόνο ωφέλιμος θα μπορούσε να είναι αφού βελτιώνει την αναγνωσιμότητα και διευκολύνει τη συντήρηση, η οποία θα είναι όλο και πιο πολύτιμη αργότερα στη διάρκεια του κύκλου ζωής του λογισμικού.
- Ιεραρχίες κληρονομικότητας μπορεί να αναδιαμορφωθούν για να είναι πιο λογικές και χρήσιμες, και ίσως να επωφεληθούν από αναγνωρισμένα πρότυπα σχεδιασμού.
- Υπάρχουν ειδικές και γενικές κατευθυντήριες γραμμές για την επαναδιατύπωση κώδικα και για τη δημιουργία καθαρού κώδικα. Με τη συνεχή και εκ νέου εκτέλεση των σεναρίων ελέγχου κατά τη διάρκεια κάθε φάσης επαναδιατύπωσης, ο προγραμματιστής μπορεί να είναι βέβαιος ότι η διαδικασία αυτή δεν μεταβάλλει οποιαδήποτε υπάρχουσα λειτουργικότητα.

7.2 Αναλυτική παρουσίαση ελέγχου

Στην ενότητα αυτή παρουσιάζουμε συνοπτικά τα σενάρια ελέγχου που χρησιμοποιήθηκαν για την ανάπτυξης της εφαρμογής. Σημειώνεται ότι για την εκτέλεση του ελέγχου χρησιμοποιήθηκε η βιβλιοθήκη λογισμικού unittest.

7.2.1 Έλεγχος της μεθόδου get_query της κλάσης QuepyApp

ελέγχεται ότι το τελικό sparql query έχει την απαιτούμενη μορφή

```
10      def test_query_generated_when_actor(self):
11          # nl_question = 'Which movie is starring Akira Takarada'
12          nl_question = 'starring Akira Takarada'
13          expected_query = \
14              u"""
15              PREFIX owl: <http://www.w3.org/2002/07/owl#>
16              PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
17              PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
18              PREFIX foaf: <http://xmlns.com/foaf/0.1/>
19              PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
20              PREFIX quepy: <http://www.machinalis.com/quepy#>
21              PREFIX dbpedia: <http://dbpedia.org/ontology/>
22              PREFIX dbpprop: <http://dbpedia.org/property/>
23              PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
24              PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
25              PREFIX dcterms: <http://purl.org/dc/terms/>
26
27              SELECT DISTINCT ?x2 WHERE {
28              SERVICE <http://dbpedia.org/sparql> {
29                ?x0 rdf:type dbpedia-owl:Film.
30                ?x0 dbpprop:starring ?x1.
31                ?x0 foaf:name ?x2.
32                ?x1 rdf:type foaf:Person.
33                ?x1 rdfs:label "Akira Takarada"@en.
34              }
35              }
36              """
37
38          target, query, userdata = self.app.get_query(nl_question)
39          self.assertEqual(expected_query, query)
```

7.2.2 Έλεγχος της μεθόδου _get_subqueries της κλάσης Question

ελέγχεται ότι το επιμέρους sparql queries έχουν την απαιτούμενη μορφή

```
41      def test_get_subqueries_when_starring_question(self):
42          expected_query = \
43          u"""
44          ?x0 rdf:type foaf:Person.
45          ?x0 rdfs:label "Akira Takarada"@en.
46          ?x1 rdf:type dbpedia-owl:Film.
47          ?x1 dbpprop:starring ?x0.
48          """
49
50          subquestions = [['starring', 'Akira', 'Takarada']]
51          question = Subquestion(subquestions, self.app.rules, self.app.keywords)
52
53          query = question._get_subqueries()
54
55          self.assertEqual(expected_query, query)
56
57      def test_get_subqueries_when_subject_question(self):
58          expected_query = \
59          u"""
60          ?x100 rdf:type skos:Concept .
61          ?x100 rdfs:label "Giant monster films"@en .
62          ?x1 rdf:type dbpedia-owl:Film .
63          ?x1 dcterms:subject ?x100 .
64          """
65
66          subquestions = [['about', 'Giant', 'Monster']]
67          question = Subquestion(subquestions, self.app.rules, self.app.keywords)
68
69          query = question._get_subqueries()
70
71          self.assertEqual(expected_query, query)
72
73      def test_get_subqueries_when_sequel_question(self):
74          expected_query = \
75          u"""
76          ?x200 owl:sameAs ?x1.
77          ?x200 movie:sequel ?x201 .
78          """
79
80          subquestions = [['sequel', 'of', 'a', 'movie']]
81          question = Subquestion(subquestions, self.app.rules, self.app.keywords)
82
83          query = question._get_subqueries()
84
85          self.assertEqual(expected_query, query)
```

7.2.3 Έλεγωος της μεθόδου _merge_subqueries της κλάσης Question

ελέγχεται ότι τα επιμέρους sparql queries ενοποιούνται σε ένα με την απαιτούμενη μορφή:

```
class TestSubquestion(unittest.TestCase):
    def setUp(self):
        multi_dbs = quepy.install("testapp")
        self.subqueries = [
            {'db': 'http://dbpedia.org/sparql',
             'query': u"""\
                ?x0 rdf:type foaf:Person.
                ?x0 rdfs:label "Akira Takarada"@en.
                ?movie rdf:type dbpedia-owl:Film.
                ?movie dbpprop:starring ?x0.
            """},
            {'db': 'http://dbpedia.org/sparql',
             'query': u"""\
                ?x20 rdf:type skos:Concept .
                ?x20 rdfs:label "Giant monster films"@en .
                ?movie rdf:type dbpedia-owl:Film .
                ?movie dcterms:subject ?subject .
            """},
            {'db': 'http://data.linkedmdb.org/sparql',
             'query': u"""
                ?x30 owl:sameAs ?movie.
                ?x30 movie:sequel ?movie2 .
            """},
            {'db': 'http://data.linkedmdb.org/sparql',
             'query': u"""
                ?movie2 rdfs:label ?output.
            """}]

        self.subquestion = \
            Subquestion(self.subqueries, multi_dbs.rules)
```

```
    def test_merge_subqueries(self):
        expected_query = u"""
            PREFIX owl: <http://www.w3.org/2002/07/owl#>
            PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
            PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
            PREFIX foaf: <http://xmlns.com/foaf/0.1/>
            PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
            PREFIX quepy: <http://www.machinalis.com/quepy#>
            PREFIX dbpedia: <http://dbpedia.org/ontology/>
            PREFIX dbpprop: <http://dbpedia.org/property/>
            PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
            PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
            PREFIX dcterms: <http://purl.org/dc/terms/>


            SELECT ?output ?subject WHERE {
                SERVICE <http://dbpedia.org/sparql> {
                    ?x0 rdf:type foaf:Person.
                    ?x0 rdfs:label "Akira Takarada"@en.
                    ?movie rdf:type dbpedia-owl:Film.
                    ?movie dbpprop:starring ?x0.
                    ?x20 rdf:type skos:Concept .
                    ?x20 rdfs:label "Giant monster films"@en .
                    ?movie rdf:type dbpedia-owl:Film .
                    ?movie dcterms:subject ?subject .
                }
                SERVICE <http://data.linkedmdb.org/sparql> {
                    ?x30 owl:sameAs ?movie.
                    ?x30 movie:sequel ?movie2 .
                    ?movie2 rdfs:label ?output.
                }
            }"""

        query = self.subquestion._merge_subqueries(self.subqueries)
        self.assertEqual(expected_query, query)
```

7.2.4 Έλεγχος της μεθόδου _get_subquestions της κλάσης QuepyApp

ελέγχεται ότι για οποιαδήποτε μετάθεση στη σειρά των υπερωτήσεων, αυτές αναγνωρίζονται και ταξινομούνται με όμοιο τρόπο:

```
67      def test_get_subquestions_when_keywords_in_all_possible_orders(self):
68          k1 = 'sequel'
69          k2 = 'about'
70          k3 = 'staring'
71
72          self._get_subquestions_when_keywords_in_order(k1, k2, k3)
73          self._get_subquestions_when_keywords_in_order(k1, k3, k2)
74          self._get_subquestions_when_keywords_in_order(k2, k1, k3)
75          self._get_subquestions_when_keywords_in_order(k2, k3, k1)
76          self._get_subquestions_when_keywords_in_order(k3, k1, k2)
77          self._get_subquestions_when_keywords_in_order(k3, k2, k1)
78
79      def _get_subquestions_when_keywords_in_order(self, k1, k2, k3):
80          question = 'what is the sequel of a movie about giant monster staring' \
81                     ' Akira Takarada'
82          keywords = ['keyword1', k1, 'keyword2', k2, 'keyword3',
83                      k3,'keyword4']
84
85          expected_subquestions = [
86              'staring Akira Takarada'.split(),
87              'about giant monster'.split(),
88              'sequel of a movie'.split()
89          ]
90          self.app.get_subquestions(question.split(), keywords)
91          self.assertEqual(expected_subquestions, self.app.subquestions)
92          self.app.subquestions = []
```

8 *Επίλογος*

8.1 Σύνοψη και συμπεράσματα

Κατά τη μελέτη των υπαρχουσών προσεγγίσεων στην απάντηση ερωτημάτων φυσικής γλώσσας από μηχανή προέκυψαν χρήσιμα αρκετά χρήσιμα συμπεράσματα. Τόσο αναφορικά με την δύναμη της χρήσης δομημένων δεδομένων όσο και για αδυναμίες των προσεγγίσεων σε ακρίβεια και απόδοση.

Από τη μία πλευρά, έγινε ξεκάθαρη η δύναμη του σημασιολογικού ιστού. Χρησιμοποιώντας δομημένα δεδομένα αποθηκευμένα σε συλλογές δεδομένων RDF πραγματοποιείται κάτι αδιανόητο για υλοποιήσεις πάνω σε δεδομένα χωρίς δομή. Ο λόγος; Η δυνατότητα εξαγωγή συμπερασμάτων και η ανεξαρτησία των απαντήσεων από το που είναι αποθηκευμένες.

Στην πρώτη προσέγγιση που εξετάστηκε (Question Answering Over Linked Data από Saeedeh Shekarpour, Axel-Cyrille Ngonga Ngomo, Sören Auer), η προσπέλαση των κόμβων του rdf γράφου επέτρεπε την εξαγωγή συμπερασμάτων λόγω του αλγόριθμου εύρεσης συνεκτικού δένδρου. Αν δυο όροι είχαν σχέση πατέρα-παιδιού και ο χρήστης αναζητούσε πληροφορία χρησιμοποιώντας τον όρο του παιδιού και ιδιότητα του πατέρα, χάρη στη συνεκτικότητα του γράφου ο αλγόριθμος θα έβρισκε απάντηση, εξάγοντας με αυτόν τον τρόπο συμπέρασμα για τον όρο που χρησιμοποίησε ο χρήστης.

Αλλά και στην δεύτερη προσέγγιση η χρήση δεδομένων RDF ανέδειξε τη δύναμή της. Χρησιμοποιώντας πόρους που περιγράφονται από URIs -που προσφέρει μόνο η RDF- ήταν δυνατή η εκμετάλλευση τους από όπου και αν προέρχονταν και οτιδήποτε και αν περιέγραφαν.

Από την άλλη πλευρά, ελλείψεις στα περιεχόμενα μια συλλογής δεδομένων μπορούν πολύ εύκολα να οδηγήσουν σε αδυναμία απάντησης. Η αδυναμία αυτή δεν θα ήταν τόσο άσχημη αν δεν αναλογιζόμασταν και το χρόνο απάντησης. Συγκριτικά με τις κλασικές μεθόδους αναζήτησης πληροφορίας που πραγματοποιείται αναζήτηση με κλειδί, η χρόνοι είναι αποτρεπτικοί. Στην πραγματικότητα αυτό που "πληρώνει" κανείς σε χρόνο όταν χρησιμοποιεί αναζήτηση σε δομημένα δεδομένα είναι η ακρίβεια των αποτελεσμάτων. Για αυτόν ακριβώς το λόγο είναι μεγαλύτερη και η απογοήτευση της αδυναμίας απάντησης. Ο χρήστης περιμένει περισσότερο και όταν δεν υπάρχει η απαραίτητη πληροφορία εντός της συλλογής δεδομένων απογοητεύεται και περισσότερο.

Επιστρέφοντας στους στόχους της παρούσας εργασίας μπορούμε να πούμε ότι πραγματοποιήθηκε σημαντική πρόοδος για την επίλυση τους. Ο λόγος γίνεται για την άρση

του περιορισμού του πλήθους των συλλογών δεδομένων που μπορεί να εκμεταλλευτεί ένα σύστημα λογισμικού προκειμένου να απαντήσει σε ερωτήματα ανθρώπινης γλώσσας, καθώς και τη βελτιστοποίηση της διαδικασίας μετατροπής της ανθρώπινης γλώσσας σε όρους που μπορούν άμεσα να καταναλωθούν από το σύστημα αυτό.

Συγκεκριμένα, για το ζήτημα του περιορισμού του πλήθους των συλλογών δεδομένων, η εφαρμογή InterQuepy που αναπτύχθηκε, το αντιμετωπίζει με μια σειρά ενεργειών. Συνοπτικά, η προσέγγιση επιλέγει την κατασκευή όχι ενός αλλά πολλών επιμέρους πυρήνων ερωτημάτων SPARQL για κάθε υπερώτηση που περιέχεται στο ερώτημα χρήστη. Από τα μεταδεδόμενα γίνεται αντιληπτό στην εφαρμογή, το πως θα συνδεθούν οι πυρήνες αυτοί και κατασκευάζεται το συνολικό εκτελέσιμο SPARQL ερώτημα που αφορά σε περισσότερες της μιας συλλογής δεδομένων.

Αντίστοιχα για τον έτερο στόχο, η InterQuepy βελτιστοποιεί τη διαδικασία αντιστοίχησης ανθρώπινων όρων με όρους "κατανοητούς" από τη μηχανή. Χρησιμοποιώντας πάλι μεταδεδομένα, αναγνωρίζει και κατακερματίζει την αρχική ερώτηση του χρήστη σε μικρότερες. Η προσέγγιση αυτή πολλαπλασιάζει τις δυνατές προτάσεις που μπορούν να απαντηθούν από το σύστημα, αφού πλέον μπορεί να απαντήσει οποιοδήποτε συνδιασμό υποπροτάσεων.

Φυσικά και η προσέγγιση αυτή έχει περιορισμούς. Ο κυριότερος είναι ότι οι κανόνες σύμφωνα με τους οποίους αναγνωρίζεται μια ερώτηση και επιχειρείται η απάντηση της, - παρά την ευελιξία των κανονικών εκφράσεων και της εξελιγμένης βιβλιοθήκης refo που χρησιμοποιήθηκε – είναι περιορισμένοι. Ο περιορισμός έγκειται στο ότι εισάγονται από τον προγραμματιστή. Αν και δεν είναι ιδιαίτερα δύσκολη η εισαγωγή τους, εντούτοις απαιτείται εξοικείωση με τον προγραμματισμό και τη γλώσσα γραφφής της εφαρμογής (Python).

Στους περιορισμούς προστίθενται ένα ακόμη στοιχείο. Η αναφορά είναι στην εμπειρία χρήστη. Κατά την ανάπτυξη της εφαρμογής δεν δόθηκε ιδιαίτερο βάρος στο πως ο χρήστης θα αλληλεπιδρά με τη μηχανή αλλά πως θεωρητικά και υπολογιστικά μπορεί να βρεθεί η απάντηση. Έτσι ενώ τα αποτελέσματα παράγονται με εξακριβωμένη ορθότητα, ο χρήστης λαμβάνει σαν έξοδο ένα SPARQL ερώτημα. Στη συνέχει πρέπει ο ίδιος να πάρει το ερώτημα αυτό και να το χρησιμοποιήσει σε ένα script του Jena framework [28] προκειμένου να λάβει την απάντηση που ζητά. Γεγονός που δυσχεραίνει την εμπειρία αλληλεπίδρασης με τη μηχανή.

8.2 Μελλοντικές επεκτάσεις

Στην προηγούμενη ενότητα καταδείχθηκαν περιορισμοί της εφαρμογής. Προκειμένου να αρθούν είναι δυνατόν να πραγματοποιηθούν μελλοντικές επεκτάσεις που αναφέρονται περιγραφικά εδώ.

Η πρώτη αφορά στην ενσωμάτωση του μηχανισμού που εκτελεί federated ερωτήματα (ενώ αυτή τη στιγμή μόνο τα παράγει). Για την ενσωμάτωση απαραίτητος είναι ο μηχανισμός που παρέχει το πλαίσιο ανοιχτού λογισμικού Jena. [28]

Η δεύτερη παρουσιάζει μεγαλύτερο ερευνητικό ενδιαφέρον. Θα ήταν δυνατόν, οι πιθανές ερωτήσεις που μπορούν να απαντηθούν από την εφαρμογή να εξάγονται αυτόματα από τις συλλογές δεδομένων RDF στις οποίες αφορά.

Μια προσέγγιση για να πραγματοποιηθεί αυτό θα ήταν με χρήση μια γνώσης αντιστοίχησης πόρων με όρους αναζήτησης. Αν το όνομα κάθε πόρου αντιστοιχίζονταν σε ένα λεκτικό που χρησιμοποιούνταν στη γλώσσα του χρήστη η λύση θα μπορούσε να είναι εφικτή. Φυσικά τα λεκτικά στα οποία θα γίνονταν η σύνδεση θα έπρεπε να μπορούν να συνδεθούν μεταξύ με κάποια σχέση συνωνυμίας, προκειμένου να αποφεύγεται το γνωστό πρόβλημα που αντιμετωπίζουν οι προσεγγίσεις εύρεσης με κλειδί με τα συνώνυμα.

9 *Αναφορές-Βιβλιογραφία*

[1] Quepy, http://quepy.machinalis.com/

[2] InterQuepy, https://github.com/apostolosSotiropoulos/interQuepy

[3] G.Antoniou, F. Harmelen, Εισαγωγή στον Σημασιολογικό Ιστό. εκδόσεις Κλειδάριθμος, 2009

[4] S.Shekarpour, A-C.Ngonga, S.Auer, Question Answering on Interlinked Data in *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 1145–1156.

[5] Python https://wiki.python.org/moin/

[6] Wordnet https://wordnet.princeton.edu/

[7] NLTK framework http://www.nltk.org/

[8] REFO https://github.com/machinalis/refo/blob/master/README.md

[9] dbpedia http://wiki.dbpedia.org/about

[10] Virtuoso http://dbpedia.org/sparql

[11] T. Berners-Lee, J. Hendler, and O. Lassila. 2001. The Semantic Web. Scientific American Magazine: 34:43

[12] Ι.Βλαχάβας, Π.Γκιούρδας, Ν.Βασιλειάδης, Φ. Κόκκορας, Η.Σακελλαρίου, εκδόσεις Γκιούρδας, 2006

[13] S.Russel, P.Norvig, Τεχνητή Νοημοσύνη Μια σύγχρονη προσέγγιση, εκδόσεις Κλειδάριθμος, 2005

[14] N. Balani. 2005. "The future of the Web is Semantic", http://www.ibm.com/developerworks/web/library/wa-semweb/

[15] RDF http://www.w3.org/RDF/

[16] T. Berners-Lee, R. T. Fielding, and L. Masinter. 2005. Uniform Resource Identifier (URI): Generic Syntax,

[17] Tim Berners-Lee 2006-07-27 http://www.w3.org/DesignIssues/LinkedData.html

[18] RDF datasets http://www.w3.org/TR/rdf11-datasets/

[19] Linkedmdb http://data.linkedmdb.org/sparql

[20] Wikipedia https://en.wikipedia.org/wiki/Main_Page

[21] SPARQL http://www.w3.org/TR/rdf-sparql-query/

[22] SPARQL https://en.wikipedia.org/wiki/SPARQL

[23] Linux https://en.wikipedia.org/wiki/Linux

[24] Ubuntu http://www.ubuntu.com/

[25] Pycharm https://www.jetbrains.com/pycharm/

[26] SparqlWrapper https://pypi.python.org/pypi/SPARQLWrapper

[27] Graphviz http://www.graphviz.org/

[28] Jena Framework https://jena.apache.org/