Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μετατροπή Φυσικής Γλώσσας σε Οντολογικές Σχέσεις

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ ΣΠΑΝΤΟΥΡΗΣ

Επιβλέπων : Γεώργιος Στάμου
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και
Υπολογιστών

Μετατροπή Φυσικής Γλώσσας σε Οντολογικές Σχέσεις

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΙΩΑΝΝΗΣ ΣΠΑΝΤΟΥΡΗΣ

Επιβλέπων : Γεώργιος Στάμου
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 10η Ιουλίου 2019.

.....	.....	.....
Γεώργιος Στάμου Αναπληρωτής Καθηγητής Ε.Μ.Π.	Ανδρέας-Γεώργιος Σταφυλοπάτης Καθηγητής Ε.Μ.Π.	Κωνσταντίνα Νικήτα Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούλιος 2019

Ιωάννης Σπαντούρης

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

Περίληψη

Οι οντολογίες αποτελούν βασικό κομμάτι της αναπαράστασης της γνώσης και ιδίως οι οντολογικές σχέσεις παίζουν βασικό ρόλο σε ερωτήματα σε δομημένα δεδομένα. Η χρήση τους σε αυτά τα ερωτήματα δεν απαιτεί καμία περαιτέρω γνώση της δομής των δεδομένων ούτε κάποια συγκεκριμένη σύνταξη. Σε αντίθεση με ερωτήματα σε γλώσσες όπως η SQL ή η SPARQL, οι οντολογικές ερωτήσεις έχουν απλή δομή και μπορούν να γίνουν εύκολα κατανοητές από άτομα που δεν είναι εξειδικευμένα πάνω στον τομέα αυτόν.

Σε αυτή την διπλωματική εργασία αναπτύσσουμε ένα Natural Language Interface(NLI), δηλαδή ένα σύστημα που παράγει οντολογικές σχέσεις απο φυσική γλώσσα. Τα συστήματα αυτά ονομάζονται NLIKB δηλαδή Natural Language Interface to Knowledge Base. Το συγκεκριμένο σύστημα που δημιουργούμε είναι πολυγλωσσικό και ανεξάρτητο του γνωσιακού πεδίου της οντολογίας. Για την βέλτιστη αξιοποίηση του αναπτύχθηκε και ένα API με το οποίο μπορεί ο χρήστης να εξάγει τα αποτελέσματα που θέλει γρήγορα, ανεξαρτήτως από που προήλθε το κείμενο εισόδου(πληκτρολόγιο, φωνή κτλ.). Παράλληλα αναπτύχθηκε και ένα σύστημα παραγωγής τυχαίων οντολογικών σχέσεων για την αξιολόγηση των αποτελεσμάτων.

Λέξεις Κλειδιά: Οντολογίες, Natural Language Interface, Βάση Γνώσης, API, Query Generator

Abstract

Ontologies play a decisive role in representing knowledge and they can be used in database queries. Using them in those queries doesn't require any prior knowledge of the way data is structured in the database and also no special syntax has to be used to create them. In contrast with SQL or SPARQL, ontology queries have simple structure and can easily be used by casual users.

In this thesis we create a Natural Language Interface(NLI), a system that takes as input natural language text and translates it into ontology relations. Such systems are called NLIKB(Natural Language Interface to Knowledge Base). This specific system that we tried to create is multilanguage and independent of the knowledge base used. For maximum utilisation of the system we also implemented an API that a client can easily use to query any ontology that he wants independently of the input source(speech, text etc). We created a query generator as well to evaluate the accuracy of the system's results.

Keywords: Ontologies, Natural Language Interface, Knowledge Base, API, Query Generator

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τόσο τον κ. Γεώργιο Στάμου, Αναπληρωτή Καθηγητή Ε.Μ.Π και επιβλέποντα της παρούσας εργασίας, που μου έδωσε την δυνατότητα να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα, όσο και τα μέλη της επιτροπής τον κ. Ανδρέα-Γεώργιο Σταφυλοπάτη και την κα. Κωνσταντίνα Νικήτα. Επίσης, θέλω να ευχαριστήσω τον κ. Enrique Matos Alfonso, που καθ' όλη την διάρκεια εκπόνησης της εργασίας ήταν πάντα διαθέσιμος να με βοηθήσει. Τέλος να ευχαριστήσω την οικογένεια μου και τους φίλους μου που δεν σταμάτησαν να με ενθαρρύνουν και να με στηρίζουν μέχρι την τελευταία στιγμή.

Ιωάννης Σπαντούρης,

Αθήνα, 10η Ιουλίου 2019

Περιεχόμενα

Τμήματα Κώδικα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικά

Οι οντολογίες αποτελούν σημαντικό κομμάτι των γνωσιακών συστημάτων και της αναπαράστασης της γνώσης, αφού μέσω αυτών μπορεί να περιγραφεί με σαφή και κατανοητό τρόπο η πληροφορία μιας γνωσιακής βάσης. Προσφέρουν επίσης έναν τρόπο με τον οποίο μπορούν να συσχετιστούν οι έννοιες της βάσης μεταξύ τους, πράγμα που τις καθιστά σημαντικό εργαλείο για την δημιουργία ερωτήσεων.

Η συνεισφορά των οντολογιών σε ερωτήματα σε δομημένα δεδομένα είναι σημαντική. Με τους παραδοσιακούς τρόπους ερώτησης όπως με την χρήση της SQL απαιτείται από τους χρήστες συγκεκριμένος τρόπος σύνταξης των ερωτήσεων καθώς επίσης και βασικές γνώσεις της δομής των δεδομένων μέσα στην βάση την οποία διερωτάμε. Με τις οντολογίες ο χρήστης μπορεί να εκφράσει τις ερωτήσεις του με σαφή τρόπο και χωρίς να απαιτείται περεταίρω γνώση κάποιας εξεζητημένης σύνταξης.

Η παρούσα διπλωματική ασχολείται με την δημιουργία οντολογικών σχέσεων μέσω φυσικής γλώσσας απλοποιώντας τελείως το συντακτικό που θα χρειαζόταν για να συνταχθεί μια οντολογική σχέση. Ένα παράδειγμα το οποίο προσπαθεί να επιλύσει είναι το παρακάτω:

All airplanes that fly to London and belong to AegeanAir.

να έχει αποτέλεσμα την παρακάτω οντολογική σχέση:

$$Airplane(X) \wedge FlyTo(X, London) \wedge BelongTo(X, AegeanAir).$$

Τα συστήματα που χρησιμοποιούν φυσική γλώσσα για ερωτήματα ονομάζονται **Natural Language Interfaces(NLI)** και πιο συγκεκριμένα αυτά που χρησιμοποιούν οντολογίες ονομάζονται **Natural Language Interfaces to Knowledge Bases(NLIKB)**. Υπάρχουν ήδη αρκετές υλοποιήσεις(βλ. 1.2) του δεύτερου και η παρούσα διπλωματική εργασία δημιουργεί και αυτή ένα NLIKB αλλά δεν ασχολείται με την μετατροπή της οντολογικής σχέσης σε μια γλώσσα ερωτήσεων. Οι αναφερθείσες υλοποιήσεις είτε δεν υποστηρίζουν ερωτήσεις σε πολλές γλώσσες, είτε για να πετύχουν πολυγλωσσία χρειάζονται ένα αρκετά μεγάλο δείγμα δεδομένων ή δεν είναι εύκολη η δημιουργία κάποιου λεξικού για να πετύχουν σωστά αποτελέσματα. Στην παρούσα εργασία προσπαθήσαμε να επιλύσουμε τα προβλήματα αυτά. Εκτός από την υλοποίηση του πολυγλωσσικού συστήματος, το οποίο είναι και δυναμικό ως προς την χρησιμοποιούμενη οντολογία(αρκεί να είναι σε OWL), εκτελέσαμε και κάποια παραδείγματα για να διαπιστώσουμε την ακρίβεια των αποτελεσμάτων που παρέχει.

Στην διπλωματική αυτή εργασία, θα ξεκινήσουμε αναφέροντας τα βασικά θεωρητικά μέρη που εμπλέκονται στην υλοποίηση του συστήματος(2), ύστερα θα αναλύσουμε την υλοποίηση και την δομή

του συστήματος(3) και τέλος θα αναφερθούμε στο πως εξάγαμε τα αποτελέσματα και θα τα παρουσιάσουμε(4). Στον επίλογο θα βγάλουμε κάποια συμπεράσματα και θα προσδιορίσουμε κατευθύνσεις για μελλοντική έρευνα(5).

1.2 Παρόμοιες Υλοποιήσεις

Παρακάτω παρουσιάζονται παρόμοιες υλοποιήσεις που έχουν αναπτυχθεί για το συγκεκριμένο πρόβλημα.

1.2.1 CHILL

Το σύστημα CHILL[17] ήταν μια προσπάθεια δημιουργίας ένος parser για προτάσεις και μετατροπή τους σε λογική μορφή. Το σύστημα αυτό περιέχει 5 υποσυστήματα, με κυριότερα ένα για επαγωγική λογική(inductive logic programming) και ένα για shift reduce. Τα υποσυστήματα αυτά βοηθάνε στην εκπαίδευση ενός καινούργιου parser κάθε φορά για μια συγκεκριμένη βάση δεδομένων.

Η διαδικασία που ακολουθούσε το σύστημα ήταν αρχικά να αναλύει τα δεδομένα που χρησιμοποιούνται για την εκπαίδευση του συστήματος για να δημιουργήσει έναν shift reduce parser γενικού σκοπού. Τον parser αυτόν τον χρησιμοποιούσε στην συνέχεια για να εξάγει πληροφορίες από τα ίδια δεδομένα. Ύστερα μέσω του ILP αλγορίθμου το σύστημα μάθαινε διάφορους κανόνες που χαρακτηρίζουν αυτές τις πληροφορίες και τους συνδύαζε μαζί με τον parser γενικού σκοπού για να παράξει τον τελικό parser για την συγκεκριμένη βάση.

Ο parser λειτουργεί στην γλώσσα Prolog και μετατρέπει ερωτήσεις σε φυσική γλώσσα σε ερωτήματα για την συγκεκριμένη βάση στην οποία έχει εκπαιδευτεί. Η αξιολόγηση του συστήματος έγινε σε ένα υπάρχον dataset 800 γεωγραφικών κατηγορημάτων στην Prolog σε Αγγλικά και για μετεφρασμένες προτάσεις στα Ισπανικά. Τα τεστ που πραγματοποιήθηκαν στις 225 ερωτήσεις που χρησιμοποιήθηκαν έδειξαν ότι το σύστημα μάθαινε με πολύ καλούς ρυθμούς και ότι τα ποσοστά σωστών απαντήσεων έφτασαν τα 68% και τα 66% για την Αγγλική και Ισπανική γλώσσα αντίστοιχα. Επίσης το σύστημα αξιολογήθηκε και σε ένα dataset για ερωτήματα σε αγγελίες για εργασίες και με εκπαίδευση πάνω σε 700 παραδείγματα έδωσε ένα ποσοστό επιτυχίας της τάξης του 89.7%.

1.2.2 FOCAL

Το σύστημα FOCAL[4] είναι ένα γενικευμένο σύστημα που αναπτύχθηκε για στρατιωτικούς σκοπούς. Το συγκεκριμένο σύστημα περιέχει πολλές πτυχές, αλλά μας ενδιαφέρουν τα υποσυστήματα του για Natural Language Processing και για Knowledge Representation και Reasoning που περιείχε. Με τα υποσυστήματα αυτά η δοθείσα από τον χρήστη πρόταση(είτε είχε δοθεί από το πληκτρολόγιο ή από ανάλυση ομιλίας) αναλύεται σε μια τυπική μορφή για το γνωσιακό πεδίο και ύστερα η μορφή αυτή απαντιέται από το reasoning υποσήστημα. Οι απαντήσεις του reasoning συστήματος ήταν σε δυαδική λογική πάνω σε ένα στρατιωτικό γνωσιακό τομέα που περιέχει γεωγραφικά, λογιστικά και οργανωτικά στοιχεία.

Το υποσύστημα χρησιμοποιούσε την γλώσσα SHIQ για τις οντολογίες του, μια γλώσσα που σχετίζεται αρκετά με την γλώσσα OWL, η οποία είναι μια γλώσσα περιγραφικής λογικής που χρησιμοποιούταν στο project DAML+OIL. Επίσης χρησιμοποιούσε και το framework RACER σαν reasoner.

Τα frameworks OIL και Racer δεν υποστηρίζουν σχέσεις μεγαλύτερης τάξης από δυο, αλλά οι δημιουργοί του συστήματος δημιούργησαν σχέσεις μεγαλύτερης τάξης χρησιμοποιώντας σαν παραμέτρους δυαδικών σχέσεων άλλες δυαδικές σχέσεις για να το επιτύχουν. Το σύστημα χρησιμοποιούσε NLP(Nuance/Regulus grammar) για την ανάλυση της φυσικής γλώσσας που του δινόταν και έκανε και χρήση συνωνύμων που ήταν ορισμένα στο λεξικό του.

Το σύστημα για να επιτύχει τα αποτελέσματα που ήθελε είχε διάφορους συντελεστές που χωρίζονταν σε τρεις κατηγορίες τους συντελεστές εισαγωγής, τους συντελεστές λογικής και τους συντελεστές εξόδου. Οι πρωτοί ήταν υπεύθυνοι να μετατρέψουν την δοθείσα πρόταση σε ένα Bayesian network που θα χρησιμοποιούταν μετά απο τους συντελεστές λογικής. Οι συντελεστές λογικής δέχονται αυτό το network και προσπαθούν να καταλάβουν ποιά είναι η καλύτερη αναπαράσταση του για το δοθέν πλαίσιο. Οι τελευταίοι συντελεστές αφορούν την μεταφορά της απάντησης από τις δυο προηγούμενες κατηγορίες συντελεστών στον χρήστη είτε σε μορφή HTML(A.1.2) είτε σε μορφή ομιλίας.

1.2.3 ORAKEL

Το σύστημα ORAKEL[13] είναι η προσπάθεια κάποιων ερευνητών για ακόμα πιο δυναμικό τρόπο μετατροπής ερωτήσεων από φυσική γλώσσα σε ερωτήσεις για βάσεις δεδομένων. Το σύστημα που έφτιαξαν αν και μπορεί να απαντήσει μόνο ερωτήσεις που ξεκινάνε από τις λέξεις who, which, what, where και how many ή how σε συνδυασμό με κάποιο επίθετο και μόνο στην αγγλική γλώσσα, είναι τελείως δυναμικό ως προς το γνωσιακό πεδίο της οντολογίας. Επίσης το σύστημα υποστηρίζει οντολογίες σε F-Logic και σε OWL και έχει δοκιμαστεί και για τις δυο.

Για να πετύχουν το σκοπό τους δημιούργησαν μια γραφική διεπαφή που ονομαζόταν FrameMapper με την οποία οποιοσδήποτε μπορεί να δημιουργήσει από το μηδέν χωρίς καθόλου γνώση από οντολογικές σχέσεις διάφορα mappings των σχέσεων που χρειάζονται για τα αποτελέσματα που θέλει να πετύχει. Για παράδειγμα αν έχουμε στοιχεία για πόλεις και για ποτάμια θα δημιουργούσαμε ένα mapping της μορφής river flows_through city και θα αντιστοιχίζαμε το river(ποταμός) σαν υποκείμενο, το city(πόλη) σαν αντικείμενο και θα μπορούσαμε πλέον να ρωτήσουμε ποια ποτάμια περνάνε από μια συγκεκριμένη πόλη.

Το σύστημα που ανέπτυξαν δεν μπορούσε να λειτουργήσει αν υπήρχαν γραμματικά λάθη, αν υπήρχαν άγνωστες λέξεις που δεν υπήρχαν στην οντολογία και αν οι προτάσεις δεν ξεκινούσαν από τις λέξεις που ήδη προαναφέρθηκαν. Τα κύρια όμως χαρακηριστικά που ήθελαν να πετύχουν είναι η εύκολη δημιουργία mappings από τον καθένα και χωρίς γνώσεις, η δυνατότητα συνεχούς βελτίωσης των mappings για να πετύχουν καλύτερα αποτελέσματα και κατά πόσο τα mappings μπορούσαν να αγγίξουν τα επίπεδα ενός ατόμου που είχε γνώσεις πάνω σε οντολογίες.

Το σύστημα ξεκινούσε αναλύοντας την δοθείσα από τον χρήστη πρόταση μέσω ενός Query Intepreter που μετέτρεπε την πρόταση σε λογική μορφή σύμφωνα και με τα mappings που είχε δημιουργήσει για το γνωσιακό πεδίο ο χρήστης του FrameMapper. Αυτή η λογική μορφή είναι λογική μορφή πρώτης τάξης(first order logic) μαζί με στοιχεία για ερωτήσεις, μετρήσεις και αριθμητικές πράξεις. Η λογική αυτή μορφή ύστερα μετατρεπόταν μέσω του Query Converter στην αντίστοιχη γλώσσα ερωτήσεων(query language) και μετά αυτή μπορούσε να εκτελεστεί στην βάση γνώσης. Η γλώσσα ερωτήσεων και η αναπαράσταση της γνώσης μπορούσαν να είναι ανεξάρτητες από την γλώσσα των ερωτήσεων. Όπως αναφέρθηκε είχαν χρησιμοποιηθεί σαν οντολογίες η F-Logic και η OWL και στην πρώτη η γλώσσα ερωτήσεων ήταν αυτή που δινόταν μέσω του συστήματος Ontobroker[16] και για την

γλώσσα OWL ήταν η γλώσσα SPARQL[14].

Για την αξιολόγηση του συστήματος κάλεσαν 25 άτομα που δεν είχαν καμία γνώση του συστήματος και αφού εκπαίδευσαν 2 απο αυτούς για 20 λεπτά στο πως να χρησιμοποιήσουν το FrameMapper για να δημιουργήσουν mappings έφτιαξαν ξεχωρισστά και αυτοί μαζί και ένας μηχανικός οντολογίας 3 διαφορετικά συστήματα τα οποία αξιολόγησαν από τις ερωτήσεις των υπολοίπων ατόμων. Μετά τον πρώτο γύρο δοκιμών τα δυο άτομα είχαν λίγο χρόνο να κάνουν αλλαγές στην οντολογία τους για να μετρήσουν αν τα αποτελέσματα επαναληπτικά θα είχαν καλύτερα ποσοστά. Τα αποτελέσματα που παρήγαγαν έδειξαν ότι ο τρόπος που ακολούθησαν, όντως επαναληπτικά έφερνε καλύτερα αποτελέσματα(διορθώνονταν τα mappigns από τα λάθη που παρουσιάζονταν) και ότι τα ποσοστά των ατόμων που δεν είχαν καμία γνώση άγγιζαν τα ποσοστά του χρήστη που ανέπτυξε το ίδιο το σύστημα που κυμαίνονταν γύρο στο 80%. Το σύστημα αξιολογήθηκε και σε ένα πραγματικό πρόβλημα που αφορούσε βάση για συγκράματα με αποτελέσματα που μετά απο 3 επαναλήψεις διόρθωσης των mappings έφτασαν το 73%.

1.2.4 FREyA

Το σύστημα FREyA[3] είναι μια προσπάθεια βελτίωσης της προταρχικής δουλειάς που χρειάζεται ένα NLIKB σύστημα ώστε να είναι παραγωγικό για νεες οντολογίες. Οι περισσότερες υλοποιήσεις NLIKB χρειάζονται μια αρχικοποίηση της οντολογίας πριν να μπορεί να χρησιμοποιηθεί απο τον τελικό χρήστη. Οι δημιουργοί λοιπόν του συστήματος αυτού προσπαθώντας να εξαλήψουν αυτό το αρχικό στάδιο χρησιμοποιούσαν κατευθείαν τις οντολογικές έννοιες για να καταλάβουν που αναφέρεται κάθε φορά ο χρήστης. Το σύστημα μπορούσε να χρησιμοποιηθεί από πολλαπλές οντολογίες αλλά μόνο στην Αγγλική γλώσσα.

Η διαδικασία που ακολουθούσε το σύστημα ξεκινούσε με την αναζήτηση στην οντολογία των εννοιών που χρησιμοποιούσε η πρόταση του χρήστη. Αν μια έννοια ήταν ασαφής(ambiguous) τότε είχαν φτιάξει μια διεπαφή με την οποία διερωτούσαν τον χρήστη σε τι αναφέρετε στο συγκεκριμένο σημείο(π.χ. αν για το Mississippi ο χρήστης αναφέρεται στον ποταμό ή στην πολιτεία). Ύστερα με την ανάλυση της πρότασης του χρήστη μέσω του Stanford NLP[10] προσπαθούσαν να αντιστοιχίσουν τις λέξεις που έδωσε ο χρήστης με τις αντίστοιχες έννοιες της οντολογίας που είχαν βρει παραπάνω ή τις συσχέτιζαν με τις έννοιες αυτές. Αν δεν έβρησκαν κάποια αντιστοίχιση της λέξης του χρήστη αλλά το σύστημα θεωρούσε ότι είναι λέξη που χρειάζεται στην οντολογική σχέση τότε παρουσίαζαν στο χρήστη μια λίστα με τις καλύτερες απαντήσεις για την συγκεκριμένη λέξη. Για την λίστα με τις καλύτερες απαντήσεις είχαν αναπτύξει έναν αλγόριθμο βαθμολόγησης των λέξεων, το οποίο το βελτίωναν συνεχώς μέσω Reinforcement Learning (RL)[15] από τα αποτελέσματα των απαντήσεων που έδινε ο χρήστης. Μετά την σωστή αντιστοίχιση των λέξεων μετέτρεπαν τις αντιστοιχίσεις σε SPARQL[14] και την έτρεχαν πάνω σε μια βάση γνώσης.

Η δοκιμή του συστήματος έγινε πάνω σε ένα σετ απο 250 ερωτήσεις(απο το Mooney Geoquery dataset) στις οποίες το σύστημα απάντησε σωστά στις 28.8% με την πρώτη προσπάθεια ενώ στις προτάσεις με μια ερώτηση στον χρήστη με ποσοστό 50.8%. Επίσης δοκίμασαν και το σύστημα τους να δουν αν βελτιώνεται μέσω του RL που πραγματοποιούσαν και από 103 τυχαίες ερωτήσεις έδειξαν ότι τα αποτελέσματα βελτιώθηκαν κατά 6%.

1.2.5 SWSNL

Το σύστημα SWSNL[8] αποτελεί ουσιστικά μια μηχανή αναζήτησης πληροφοριών σε βάσεις γνώσεις από ερωτήσεις σε φυσική γλώσσα. Οι δημιουργοί του συστήματος θέλανε να επεκτείνουν τις ήδη υπάρχουσες υλοποιήσεις που είχαν γίνει ώστε να δημιουργήσουν ένα σύστημα που είναι κάτι παραπάνω από ένα σύστημα απάντησης ερωτήσεων. Για να πετύχουν κάτι τέτοιο επέτρεπαν στους χρήστες τους να κάνουν αναζητήσεις με περισσότερες από μια προτάσεις. Το σύστημα που ανέπτυξαν είχε επίσης ένα στοχαστικό μοντέλο για να κάνει parse τις προτάσεις των χρηστών και ήταν επίσης ανεξάρτητο γλώσσας αφού εκπαιδευόταν σε σχολιασμένες οντολογίες ανάλογα με κάθε διαφορετικό γνωσιακό πεδίο. Τέλος το σύστημα τους χρησιμοποιούσε διαφορετική οντολογία για την περιγραφή των δεδομένων της φυσικής γλώσσας και διαφορετική βάση γνώσης, οπότε μπορούσε να αλλαχτεί κάθε φορά η βάση γνώσης χωρίς να χρειάζεται ξανά εκπαίδευση το μοντέλο.

Το σύστημα τους δεχόταν μια πρόταση σε φυσική γλώσσα, την οποία μετέτρεπε σε μια σημασιολογική αναπαράσταση ανεξάρτητη της βάσης γνώσης μέσω ενός στατιστικού μοντέλου που έχει εκπαιδευτή πάνω σε μια οντολογία, η οποία είχε υποστεί σχολιασμό για κάθε στοιχείο της. Ύστερα η σημασιολογική αναπαράσταση μετατρεπόταν σε μια γλώσσα ερωτήσεων(SPARQL[14]) και εκτελούταν στην αντίστοιχη βάση γνώσης.

Για να διευκολύνουν την διαδικασία του σχολιασμού της οντολογίας είχαν δημιουργήσει ένα εργαλείο σχολιασμού που έκρυβε μερική από την πολυπλοκότητα του συστήματος. Για την εκπαίδευση του μοντέλου χρησιμοποιούσαν ένα στατιστικό μοντέλο και εποπτική μάθηση μέσω Named Entity Recognition με τέσσερα διαφορετικά εργαλεία για να πετύχουν τα καλύτερα αποτελέσματα.

Το σύστημα αξιολογήθηκε στην τσέχικη γλώσσα αρχικά σε ένα dataset που αφορά καταλύματα για διακοπές. Επειδή τα δεδομένα εκπαίδευσης τους δεν ήταν αρκετά προσπάθησαν να προσομοιώσουν τα αποτελέσματα που θα παράγονταν αν είχαν ένα μεγαλύτερο dataset. Τα αποτελέσματα τους θα έφταναν το 73% αν και με τα δεδομένα που είχαν μέχρι στιγμής η ακρίβεια ήταν στο 25%. Επίσης το σύστημα αξιολογήθηκε σε ένα τσέχικο dataset για μέσα μεταφοράς, για να αποδείξουν ότι μπορεί να χρησιμοποιηθεί σε άλλα γνωσιακά πεδία, με τα αποτελέσματα να αγγίζουν το 90%. Τέλος δοκιμάστηκε και σε ένα μεγαλύτερο dataset στην αγγλική γλώσσα, για να αποδείξουν ότι μπορεί να είναι πολυγλωσσικό, που αφορούσε πτήσεις και για ένα υποσύνολο από 348 προτάσεις τα αποτελέσματα έφταναν το 75%.

1.2.6 Another approach

Το σύστημα αυτό[11] ήθελε να μειώσει συνολικά την προσπάθεια που χρειάζεται από τον τελικό χρήστη να χτίσει σωστά ένα NLIKB σύστημα. Στόχος του συστήματος ήταν να δίνεται σε αυτό μόνο η οντολογία του γνωσιακού πεδίου και από αυτή ο χρήστης να μπορεί να ρωτήσει μέσω λέξεων κλειδιών ή μέσω μιας πρότασης ερωτήσης σε μια βάση γνώσης.

Το σύστημα ξεκινούσε δημιουργώντας από την οντολογία του πεδίου ένα λεξικό μέσω της περιγραφής των κλάσεων, των σχέσεων και των τύπων της οντολογίας, τα οποία ύστερα τα χρησιμοποιούσε για να κάνει αναζήτηση πάνω σε αυτά. Στα πεδία που δεν είχαν περιγραφή στο συγκεκριμένο πεδίο που χρειαζόταν, το σύστημα λάμβανε υπόψη του το όνομα του αντικειμένου και χρησιμοποιούσε αυτό. Ύστερα μέσω τεχνικών επεξεργασίας φυσικής γλώσσας μετέτρεπε όλες τις πληροφορίες της πρότασης σε ένα μοντέλο του συστήματος που ονομαζόταν Question Model. Με αυτό το question model γινόταν η μετατροπή σε SPARQL ερώτηση σύμφωνα με το γνωσιακό πεδίο και τελικά αυτή η ερώτηση

εκτελούνταν στην βάση γνώση.

Μέσω του NLP πραγματοποιούσαν την μετατροπή στο γενικό Question Model. Για να βελτιώσουν τα αποτελέσματα της διαδικασίας αντιστοίχισης με το γνωσιακό πεδίο χρησιμοποιούσαν συνώνυμα από το WordNet[5] και επίσης έκαναν αναζήτηση πιθανόν αντιστοιχίσεων με οντολογικές κλάσεις και σχέσεις με string similarity μέσω του αλγόριθμου Levenshtein[9]. Επίσης πραγματοποιούσαν και Named Entity Recognition για να βρίσκουν Named Entities[6] και να διαπυστώσουν αν μια λέξη ανήκει σε μια συγκεκριμένη κατηγορία π.χ. εταιρεία, τοποθεσία κοκ. Τέλος χρησιμοποιούσαν μια κατηγοριοποίηση για να διαπυστώσουν τον τύπο της απάντησης που θα πρέπει να απαντήσει το σύστημα και έτσι κατάφερναν να μειώσουν το εύρος της αναζήτησης στην βάση γνώσης.

Το σύστημα δοκιμάστηκε στο MusicBrainz[1] το οποίο αποτελεί μια αγγλική εγκυκλοπαίδια που συλλέγει δεδομένα από τραγούδια. Για την αξιολόγηση χρησιμοποιήθηκε ένα γκρούπ ανθρώπων που αποτελούταν απο άτομα που δεν είχαν καμία σχέση με οντολογίες ή την SPARQL. Οι προτάσεις που τους ζητήθηκε να γράψουν δεν έπρεπε να έχουν ορθογραφικά λάθη, δεν επιτρέπονταν ερωτήσης δυαδικής απάντησης ούτε ερωτήσης επιλογής. Τα αποτελέσματα έδειξαν ότι το σύστημα μπορούσε να απαντήσει σωστά με ποσοστό επιτυχίας 78% στις 100 ερωτήσεις στις οποίες αξιολογήθηκε. Τα κυριότερα προβλήματα που αντιμετώπισαν αφορούσαν την δυσκολία αντιστοίχισης των λέξεων με κάποια οντολογική έννοια και προβλήματα ασάφειας.

Κεφάλαιο 2

Θεωρητικό Μέρος

Παρακάτω παρουσιάζονται τα θεωρητικά μέρη που χρειάζονται για την κατανόηση των βασικών λειτουργιών της εφαρμογής.

2.1 Οντολογίες

Η ετυμολογία της λέξης οντολογία προέρχεται από την ένωση της ελληνικής λέξη ον και της λέξης λογία. Την πρώτη εμφάνιση της την έκανε στον τομέα της φιλοσοφίας και με αυτή γίνεται η αναζήτηση της αρχής της υπάρξης του όντος, η συγκρότηση του καθώς επίσης και η μελέτη της ουσίας του.

Όσον αφορά στην επιστήμη των υπολογιστών, η έννοια της οντολογίας άρχισε να εμφανίζεται προς την δεκαετία του 70, με την οποία οι ερευνητές που ασχολιόντουσαν με την Τεχνητή Νοημοσύνη αναφέρθηκαν σε αυτή ως το κλειδί για την δημιουργία ίσχυρων συστημάτων τεχνητής νοημοσύνης.

Ένας ορίσμος που δόθηκε για αυτήν στις αρχές του 1990 από τον Tom Gruber αναφέρει ότι ”Οντολογία είναι μια περιγραφή των εννοιών και των σχέσεων που μπορούν τυπικά να υπάρχουν για έναν παραγόντα ή μια κοινότητα παραγόντων.”[7].

Επομένως στην πληροφορική με τον όρο Οντολογία αναφερόμαστε στην αναπαράσταση και στον ορισμό κατηγοριών, ιδιοτήτων και σχέσεων μεταξύ εννοιών, δεδομένων και οντοτήτων που τεκμηριώνουν έναν ή πολλούς τομείς. Οντολογίες χρησιμοποιούνται εκτός απο την τεχνητή νοημοσύνη σε εφαρμογές για την υγεία, την γεωγραφία κ.α.

2.1.1 Βασικά Συστατικά Οντολογίας

Τα βασικά συστατικά της οντολογίας, που θα χρησιμοποιηθούν στην παρούσα διπλωματική, είναι οι κλάσεις(Classes), οι σχέσεις(Relations) και τα άτομα(Individuals).

- **Κλάσεις(Classes):** Κλάσεις μπορεί να είναι έννοιες, συλλογές, αντικείμενα, σετ κ.α. π.χ. Male
- **Άτομα(Individuals):** Τα άτομα αποτελούν μεμονωμένες περιπτώσεις της οντολογίας. π.χ. George.
- **Σχέσεις (Relations):** Οι σχέσεις με τις οποίες μπορούν οι κλάσεις και τα άτομα να συνδέονται μεταξύ τους. π.χ. FatherOf(X,Y) υποδηλώνει ότι ο X είναι πατέρας του Y(Τα X και Y θα μπορούσαν να είναι άτομα στην οντολογία).

2.2 Περιγραφική Λογική

Αποτελεί μια ομάδα γνωσιακών περιγραφικών γλωσσών και τα βασικά προβλήματα που λύνει είναι συνήθως προβλήματα απόφασης. Υπάρχουν πολλές περιγραφικές λογικές με κυριότερη την $\mathcal{AL}$.

Χρησιμοποιείται στην τεχνητή νοημοσύνη για την περιγραφή και την αιτιολόγηση των σχετικών εννοιών ενός πεδίου εφαρμογής. Έχει μεγάλη σημασία για τις οντολογίες και για τον σημασιολογικό ιστό(Semantic Web), αφού τους παρέχει έναν τρόπο να περιγραφούν λογικά και να αναπαρασταθούν.

2.2.1 Ερωτήσεις σε περιγραφική λογική

Στις περιγραφικές λογικές μπορούν να συνταχθούν ερωτήσεις, οι οποίες αν εκτελεστούν σε μια βάση γνώσης μπορούν να επιστρέψουν αποτελέσματα είτε δυαδικής λογικής:

$$Male(George)$$

που η απάντηση είναι ναι ή όχι, είτε να επιστρέψουν τα άτομα που ανταποκρίνονται στην οντολογική σχέση:

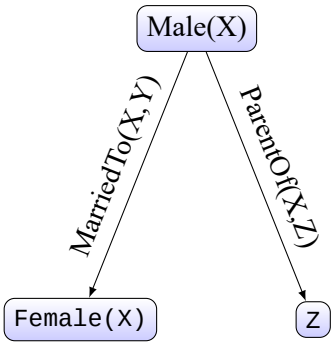
$$Male(X)$$

η οποία θα επιστρέψει όλα τα άτομα που είναι άντρες. Τα δεύτερα απο αυτά ονομάζονται instance queries και θα μπορούσαν να συνδυαστούν και με άλλες σχέσεις και κλάσεις, ώστε να δημιουργήσουν ένα πιο σύνθετο query που ονομάζεται και conjuctive query. Ένα παράδειγμα τέτοιου query είναι το παρακάτω:

$$Male(X) \wedge MarriedTo(X, Y) \wedge Female(Y)$$

το οποίο θα φέρει όλα τα ζευγάρια(άντρας και γυναίκα) που είναι παντρεμένα μεταξύ τους.

Τα ερωτήματα που γίνονται σε περιγραφική λογική μπορούν να αναπαρασταθούν με την μορφή δέντρου, όπου κάθε κόμβος είναι ένα άτομο της οντολογίας, ο οποίος περιγράφεται με διάφορες κλάσεις, και οι ακμές του δέντρου είναι μια σχέση που ενώνει δυο κόμβους μεταξύ τους. Παράδειγμα τέτοιου δέντρου είναι το παρακάτω:



το δέντρο αντιστοιχεί στο παρακάτω query:

$$Male(X) \wedge MarriedTo(X, Y) \wedge Female(Y) \wedge ParentOf(X, Z)$$

2.3 Η γλώσσα OWL

OWL(Web Ontology Language) είναι μια οικογένεια απο γνωσιακές γλώσσες αναπαράστασης για την συγγραφή οντολογιών. Οι γλώσσες OWL χαρακτηρίζονται απο τυπική σημασιολογία για την συγγραφή αυτών και έχουν δημιουργηθεί πάνω στο XML standard.

Οι γλώσσες OWL παρουσιάστηκαν πρώτη φορά το 2004 και ύστερα απο διάφορα καινούργια χαρακτηριστικά που προστέθηκαν το 2009 παρουσιάστηκε η OWL2.

Ένα παράδειγμα OWL2 στο οποίο παρουσιάζεται μια κλάση σε XML είναι το παρακάτω:

```
<Ontology ontologyIRI="urn:absolute:persons-ontology" ...>
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
  <Declaration>
    <Class IRI="#Person"/>
  </Declaration>
</Ontology>
```

Κώδικας 2.1: Παράδειγμα OWL

Οι γλώσσες OWL υποστηρίζουν άτομα, κλάσεις, σχέσεις που μας χρειάζονται για την παρούσα διπλωματική. Για καθένα απο αυτά μπορούν να οριστούν και αντίστοιχα σχόλια(comments) σε διαφορετικές γλώσσες. Παρακάτω παρουσιάζεται ένα σχόλιο στα αγγλικά:

```
<AnnotationAssertion>
  <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
  <IRI>#friendof</IRI>
  <Literal xml:lang="en"
  datatypeIRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#PlainLiteral">
  friend
  </Literal>
</AnnotationAssertion>
```

Κώδικας 2.2: Παράδειγμα σχολίου σε OWL

2.4 Απόσταση Levenshtein

Η απόσταση Levenshtein είναι ένας δείκτης που μετρά την διαφορά μεταξύ δυο προτάσεων. Η απόσταση αυτή είναι ο ελάχιστος αριθμός χαρακτήρων που χρειάζονται για την μετατροπή της μιας πρότασης στην άλλη μέσω εισαγωγής, διαγραφής και αντικατάστασης χαρακτήρων. Πήρε το όνομα της απο τον Σοβιετό μαθηματικό Vladimir Levenshtein που την σκέφτηκε πρώτος το 1965(στα αγγλικά πρωτοεμφανίστηκε το 1966)[9].

Μαθηματικά, η απόσταση Levenshtein μεταξύ δυο προτάσεων a, b (μεγέθους $|a|$ και $|b|$ αντίστοιχα)

ορίζεται ως:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j) + 1 \\ lev_{a,b}(i,j-1) + 1 \\ lev_{a,b}(i-1,j-1) + 1_{a_i \neq b_i} \end{cases} & \text{otherwise.} \end{cases}$$

όπου το $1_{a_i \neq b_i}$, ισούτε με 0 όταν $a_i = b_i$ αλλιώς ισούτε με 1.

2.5 Επεξεργασία Φυσικής Γλώσσας

Η επεξεργασία φυσική γλώσσας (Natural Language Processing) είναι ένας τομέας της επιστήμης των υπολογιστών που ασχολείται με το πως οι υπολογιστές μπορούν να επεξεργαστούν και να αναλύσουν μεγάλο αριθμό απο δεδομένα φυσικής γλώσσας.

2.5.1 Αναλυτής Εξαρτήσεων και Συντακτικό Δέντρο

Ο αναλυτής εξαρτήσεων (Dependency Parser) είναι ένα εργαλείο φυσικής γλώσσας, το οποίο αναλύει την γραμματική δομή μιας πρότασης και δημιουργεί σχέσεις μεταξύ των λέξεων αυτής.

Ένα ακόμα είδος ανάλυσης που ονομάζεται Constituency Parsing δημιουργεί το συντακτικό δέντρο της πρότασης, χρησιμοποιώντας μια πιθανοτική γραμματική χωρίς περιεχόμενα(Probabilistic Context-Free Grammar).

Και τα δυο είδη ανάλυσης φαίνονται στο παρακάτω παράδειγμα(έχει χρησιμοποιηθεί το Stanford NLP(A.2.3)):

```
(ROOT
  (S
    (NP (DT A) (JJ male))
    (VP (VBN married)
      (PP (TO to)
        (NP (DT a) (NN female))))
    (. .)))
```

Κώδικας 2.3: Constituency Parse

```
root(ROOT-0, married-3)
det(male-2, A-1)
nsubj(married-3, male-2)
case(female-6, to-4)
det(female-6, a-5)
nmod:to(married-3, female-6)
punct(married-3, .-7)
```

Κώδικας 2.4: Dependency Parse

όπου η πρόταση που χρησιμοποιήθηκε είναι **A male married to a female.** και ισχύει ότι:

Μέρος του λόγου	Περιγραφή
S	Sentence
NP	Noun Phrase
DT	Determiner
JJ	Adjective
VP	Verb Phrase
VBN	Verb, past participle
PP	Prepositional Phrase
TO	To
NN	noun, singular or mass

Εξάρτηση	Περιγραφή
root	Root
det	Determiner
nsubj	Nominal Subject
case	Case Marking
nmod	Nominal Modifier
punct	Punctuation

Πίνακας 2.1: Επεξήγηση συντομογραφιών

Μπορούμε να παρατηρήσουμε ότι στο Constituency parse δημιουργείται μια δεντρική δομή στην οποία περιγράφεται και που αναφέρεται κάθε φορά μια λέξη. Για παράδειγμα βλέπουμε ότι το A είναι πριν την λέξη male και ότι είναι στο ίδιο επίπεδο του δέντρου οπότε το A αναφέρεται στο male. Αντίστοιχα μπορούμε να δούμε ότι το married σαν ρήμα δημιουργεί ένα πιο βαθύ κλαδί στο δέντρο, αφού όλα τα επόμενα στοιχεία εξαρτώνται από αυτό. Από την άλλη το Dependency parse δημιουργεί μόνο τις εξαρτήσεις των λέξεων και μπορούμε ξεκάθαρα να δούμε ότι μεταξύ του married και του male έχουμε ότι το δεύτερο αποτελεί υποκείμενο του πρώτου καθώς επίσης και ότι το married - female συνδέονται μεταξύ τους με την λέξη το η οποία αποτελεί ένα modifier για την πρόταση(οι modifiers είναι μελή του λόγου που μπορούν να παραληφθούν χωρίς να επηρεάζουν την γραμματική της πρότασης).

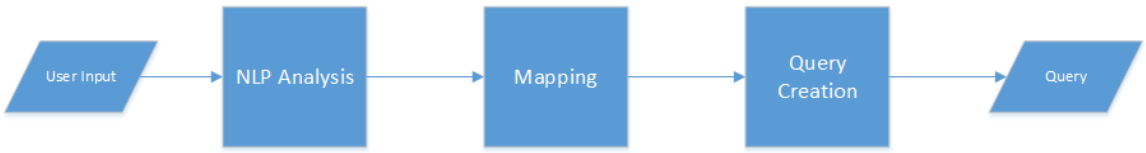
Κεφάλαιο 3

Μεθοδολογία και Υλοποίηση

Στο παρόν κεφάλαιο θα αναλυθεί η μεθολογία που ακολουθήθηκε και θα παρουσιαστεί και κομμάτι της υλοποίησης που έγινε. Στο τέλος του κεφαλαίου θα παρουσιαστούν οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.

3.1 Γενική Περιγραφή

Η διαδικασία που ακολουθήθηκε χωρίζεται σε τρια κύρια κομμάτια που αναλύονται παρακάτω. Αρχικά γίνεται η ανάλυση της πρότασης, που δίνεται απο τον χρήστη, με την χρήση ενός εργαλείου επεξεργασίας φυσικής γλώσσας, μετά η αντιστοίχιση των λέξεων που δίνονται στην πρόταση με τις αντίστοιχες στην οντολογία και τέλος η δημιουργία των οντολογικών σχέσεων και της τελικής ερώτησης. Η διαδικασία φαίνεται και στο παρακάτω σχήμα.



Ξεκινώντας με μια πρόταση της παρακάτω μορφής:

A person that is married and friend with a female.

τελικά θα δημιουργηθεί μια οντολογική σχέση της παρακάτω μορφής:

Person(X0), Marriedto(X0,X1), Friendof(X0,X1), Female(X1)

όπου το Person και Female είναι κλάσεις και τα FriendOf, MarriedTo είναι σχέσεις από την οντολογία. Τα X0 και X1 δημιουργούνται αυτόματα και με την μετατροπή της οντολογικής αυτής σχέσης σε μια γλώσσα ερωτήσεων π.χ. της SPARQL[14] και την εκτέλεση της σε μια βάσης γνώσης θα αναπαριστούσαν άτομα.

Η εφαρμογή που υλοποιήθηκε υποστηρίζει πολλαπλές γλώσσες: Αγγλικά, Γαλλικά, Γερμανικά, Ισπανικά και Πορτογαλικά και επίσης μπορεί να χρησιμοποιηθεί από πολλές διαφορετικές οντολογίες.

3.2 Στάδια Υλοποίησης

Παρακάτω παρουσιάζονται και αναλύονται τα στάδια υλοποίησης που ακολουθήθηκαν.

3.2.1 Εισαγωγή πρότασης απο τον χρήστη

Η πρόταση που θα πρέπει να εισάγει ο χρήστης θα πρέπει να είναι μια και να μην είναι σειρά απο προτάσεις. Επίσης θα πρέπει να έχει ένα υποκείμενο. Από αυτό το υποκείμενο μπορούν να δημιουργούνται σχέσεις με άλλα άτομα και μετά σχέσεις να αναφέρονται σε αυτά κ.ο.κ.

Η εισαγωγή της πρότασης του χρήστη επειδή δεν χρειάζεται να είναι αναγκαστικά από το πληκτρολόγιο, δημιουργήθηκε ένα Web API στο οποίο μπορεί να στέλνεται η πρόταση είτε αυτή έχει δοθεί απο κάποιο πληκτρολόγιο, είτε μέσω φωνής, είτε αυτοματοποιημενα κ.ο.κ. Το Web API και ο τρόπος που μπορεί να χρησιμοποιηθεί παρουσιάζονται παρακάτω(3.3).

3.2.2 Ανάλυση της πρότασης

Χρησιμοποιώντας το Spacy(A.2.2), το οποίο είναι ένα εργαλείο επεξεργασίας φυσικής γλώσσας(2.5) για την γλώσσα προγραμματισμού Python(A.1.1), πραγματοποιείται το Dependency Parsing και το Constituency Parse(2.5.1) που χρειάζεται για να γίνει η σωστή δημιουργία των σχέσεων. Με την χρήση των δυο είναι γνωστό σε ποια άλλη οντολογική κλάση ή άτομο αναφέρεται κάθε φορά η λέξη που πάμε να αναλύσουμε. Στα επόμενα βήματα χρησιμοποιείται η δομή που μας δίνουν οι αναλύσεις αυτές, ώστε να προσπελαστεί η δοθείσα πρόταση κομμάτι κομμάτι και όχι ολόκληρη.

Κατά την ανάπτυξη της παρούσας διπλωματικής, διαπιστώθηκε ότι ο καλύτερος τρόπος για την προσπέλαση της πρότασης είναι μέσω των εξαρτήσεων της και όχι μέσω του συντακτικού δέντρου αυτής. Πολλές φορές το εργαλείο που χρησιμοποιήθηκε συγκεντρώνει με πιο περίεργο τρόπο τα διαφορετικά κομμάτια της πρότασης στο συνακτικό δέντρο με αποτέλεσμα να είναι δύσκολη η επιλογή του κάθε φορά κατάλληλου ατόμου/κλάσης με το οποίο θα πρέπει να συσχετιστεί μια σχέση.

Θα μπορούσε να χρησιμοποιηθεί και η υλοποίηση του Stanford NLP στην παρούσα φάση του προβλήματος, με την χρήση ένος wrapper από την γλώσσα Java για να λειτουργήσει στην γλώσσα Python, αλλά προτιμήθηκε η χρήση του Spacy, γιατί είναι εκ γενετής καλύτερο εργαλείο για την γλώσσα αυτή με εξίσου καλά αποτελέσματα.

Ένα βασικό πρόβλημα που παρατηρήθηκε είναι ότι το Spacy δημιουργεί διαφορετική ανάλυση για δυο προτάσεις που θα έπρεπε να έχουν το ίδιο αποτέλεσμα, οπότε μπορεί τελικά να προκύψουν διαφορετικά αποτελέσματα. Ένα παράδειγμα δυο τέτοιων προτάσεων φαίνεται παρακάτω, μαζί με το αντίστοιχο συντακτικό δέντρο που παράγεται:

Πρόταση	A person has a friend and a parent	A person has a friend and has a parent
Αποτέλεσμα	(has_VBZ (person_NN A_DT) (friend_NN a_DT and_CC (parent_NN a_DT)))	(has_VBZ (person_NN A_DT) (friend_NN a_DT) and_CC (has_VBZ (parent_NN a_DT)))

Πίνακας 3.1: Πρόβλημα ανάλυσης από το Spacy

3.2.3 Αντιστοιχίσεις λέξεων

Το επόμενο στάδιο που ακολουθείται είναι η αντιστοίχιση των δοθέντων λέξεων με τις αντίστοιχες κλάσεις/σχέσεις/άτομα της οντολογίας. Για την διαδικασία αυτή διαπιστώθηκε ότι οι λέξεις που δεν

ανήκουν στα παρακάτω μέρη του λόγου δεν πρέπει να αντιστοιχίζονται. Τα μέρη του λόγου και οι περιγραφές είναι:

Μέρος του λόγου	**Περιγραφή**
Verb	Ρήμα
Noun	Ουσιαστικό
ADJ	Επίθετο
PROPN	Ονόματα

Πίνακας 3.2: Μέρη του λόγου που αντιστοιχίζονται

Ειδική μεταχείριση γίνεται και για τους συνδέσμους και τις προθέσεις που αναπαριστώνται με το **IN** tag(τα tags είναι πιο ειδικά μέρη του λόγου στο Spacy), καθώς επίσης και για τις συνδετικές λέξεις π.χ. And, που αναπαριστώνται με το **CONJ** και το **CCONJ** μέρος του λόγου στο Spacy. Τα συγκεκριμένα μέρη του λόγου θα μας απασχολήσουν παρακάτω στο τελευταίο στάδιο της εφαρμογής, την δημιουργία της οντολογικής σχέσης(3.2.4). Να αναφερθεί εδώ ότι για να έχουμε πρόσβαση στις οντολογικές σχέσεις/κλάσεις/άτομα χρησιμοποιήθηκε η βιβλιοθήκη Owlready2(A.2.1).

Για κάθε λέξη που ανήκει στα μέρη του λόγου που αντιστοιχίζονται, γίνεται σύγκριση με τις οντολογικές σχέσεις, κλάσεις και άτομα μέσω της απόστασης Levenshtein(2.4) και επιστρέφεται η καλύτερη από αυτές, δηλαδή αυτή που απέχει λιγότερο απο την λέξη που μας δίνεται. Η σύγκριση αυτή γίνεται μόνο με πεζούς χαρακτήρες, οπότε κάθε φορά μετατρέπονται σε πεζούς χαρακτήρες οι δυο λέξεις που συγκρίνονται(case insensitive). Για να πετύχει η σωστή αντιστοίχιση έχει προστέθει και ένα threshold στην σύγκριση, το οποίο για λέξεις πάνω από 4 γράμματα έχει τιμή 1, αλλιώς έχει τιμή 0. Δηλαδή:

$$threshold = \begin{cases} 1 & \text{if len(word)} > 4 \\ 0 & \text{otherwise} \end{cases}$$

Η σύγκριση αυτή δεν γίνεται πάντα με το όνομα της οντολογικής σχέσης/κλάσης/ατόμου, επειδή πολλές φορές τα ονόματα αυτά μπορεί να είναι περίπλοκα και δυσνόητα και να μην περιγράφουν ακριβώς το νόημα που θέλουμε. Ένα παράδειγμα τέτοιας οντολογίας είναι μια ανοιχτή οντολογία που περιγράφει ασθένειες (`http://www.disease-ontology.org/`). Παρακατω φαίνονται κάποια ονόματα κλάσεων αυτής της οντολογίας:

1. Bordetella parapertussis whooping cough
2. Mycobacterium avium complex disease
3. Parasitic helminthiasis infectious disease

όπως φαίνεται τα παραπάνω ονόματα μπορεί να είναι αρκετά περίπλοκα. Οπότε θεωρήθηκε καλύτερη η παρακάτω υλοποιήση που χρησιμοποιεί τα σχόλια των σχέσεων/κλάσεων/ατόμων. Για τα σχόλια όμως, επειδή μπορεί να περιέχουν πολλές φορές τον ορισμό της κλάσης/σχέσης έχει προστεθεί και ένα ακόμα threshold που κοιτάει αν το σχόλιο της λέξης περιέχει περισσότερες από έναν συγκεκριμένο αριθμό λέξεων. Το threshold αυτό είναι το παρακάτω:

$$threshold = 10$$

Για κάθε μια οντολογική σχέση/κλάση/άτομο κοιτάμε αν έχει σχόλια στην γλώσσα που ζητήθηκε από τον χρήστη και αν δεν έχει τότε γίνεται σύγκριση με το ίδιο το όνομα της. Αν έχει σχόλια τότε αρχικά αφού βρεθεί το κατάλληλο σχόλιο για την γλώσσα που ζητήθηκε, διαχωρίζεται στις αλλαγές γραμμής(υποστηρίζονται έτσι πολλά σχόλια σε μια γλώσσα) ώστε μετά για κάθε σχόλιο ανά γράμμή να γίνονται τα παρακάτω βήματα. Αρχικά γίνεται σύγκριση με όλο το σχόλιο και ύστερα διαχωρίζεται και αυτό στα κενά του και γίνεται αναζήτηση για την δοθείσα λέξη ακριβώς όπως μας δίνεται(δηλαδή με απόσταση Levenshtein ίση με 0). Ο αντίστοιχος ψευδοκώδικας φαίνεται παρακάτω:

```
// name, language: the name of the given word and the given language
// classname, comments: the name and the comments of the ontology object
minDistance ← 10000
hasComments ← False
for each comment in comments do
  if comment.lang = language then
    hasComments ← True
    comment_for_language ← comment
  end if
end for
if hasComments = False then
  classNameCompare ← Levenshtein(name,classname)
  if classNameCompare <= minDistance then
    minDistance ← classNameCompare
    result ← className
  end if
else
  multipleComments ← comment_for_language.split(”\n”)
  for each comment in multipleComments do
    commentCompare ← Levenshtein(name,comment)
    if compareComment <= minDistance then
      result ← className
      minDistance ← compareComment
    end if
    splitComment ← comment.split(” ”)
    for each part in splitComment do
      comparePart ← Levenshtein(name,part)
      if comparePart = 0 then
        result ← className
        minDistance ← 0
      end if
    end for
  end for
end if
```

Κώδικας 3.1: Ψευδοκώδικας αντιστοίχισης σχολίων

Αφού αυτή η διαδικασία τρέξει για τις κλάσεις, τις σχέσεις και τα άτομα κρατάμε την πιο κοντινή απόσταση από τα τρία και την αντιστοιχίζουμε με την λέξη που έχει δοθεί από τον χρήστη. Αυτή θεωρούμε ως την καλύτερη αντιστοίχιση για την λέξη. Αν υπάρχουν πολλές κοντινές διαφορετικές αντιστοιχίσεις(πολλές φορές είναι και τέλειες αντιστοιχίσεις, δηλαδή με απόσταση 0) τότε υπάρχει κάποια ασάφεια στα δεδομένα και το σύστημα κρατάει την τελευταία που βρήκε ως την καλύτερη αντιστοίχιση. Αυτό αποτελεί ένα θέμα που θα πρέπει μελλοντικά να λυθεί.

Εμπλουτισμός Αντιστοιχίσεων

Αν δεν βρεθεί αντιστοίχιση με την παραπάνω διαδικασία, τότε γίνεται μια προσπάθεια για τον εμπλουτισμό των αποτελεσμάτων και την απόδοση μιας αντιστοίχισης για την λέξη. Ακολουθούνται λοιπόν οι παρακάτω μέθοδοι με διαδοχικό τρόπο, τέτοιον ώστε αν σε οποιαδήποτε από αυτές βρεθεί μια αντιστοίχιση να μην προχωράμε στις επόμενες. Η σειρά των μεθόδων αυτών είναι:

1. Συνώνυμα
2. Υπερώνυμα
3. Ορθογραφικός έλεγχος

Για τα **συνώνυμα** χρησιμοποιείται η βιβλιοθήκη NLTK(A.2.4) και πιο συγκεκριμένα το πακέτο WordNet(A.2.5) που προσφέρεται στην συλλογή της(corpus). Για την δοθείσα λέξη γίνεται αναζήτηση για τα αντίστοιχα συνώνυμα της(synsets) και αφού συνδυαστούν μαζί με τα λήματα τους(lemmas), δημιουργείται μια λίστα, από την οποία προσπαθεί να γίνει η αντιστοίχιση της λέξης. Για κάθε στοιχείο της λίστας, γίνεται σύγκριση με τις οντολογικές κλάσεις/σχέσεις και αν βρεθεί τέλεια σύγκριση(μηδενική απόσταση Levenshtein), με κάποια από αυτές, τότε γυρνάμε την κλάση/σχέση αυτή ως την καλύτερη αντιστοίχιση.

Για τα **υπερώνυμα** χρησιμοποιείται η βιβλιοθήκη Pattern(A.2.6). Τα υπερώνυμα είναι γενικότερες έννοιες για μια λέξη και ένα παράδειγμα υπερώνυμου για την λέξη ασθενοφόρο είναι το αυτοκίνητο. Με την βιβλιοθήκη αυτή γίνεται να βρεθούν όλα τα υπερώνυμα για μια λέξη και για τα ίδια τα υπερώνυμα της τα υπερώνυμα τους αναδρομικά. Έτσι φτιάχτηκε μια λίστα με υπερώνυμα, τα οποία συγκρίνονται με τις οντολογικές κλάσεις/σχέσεις για να βρεθεί κάποια τέλεια σύγκριση.

Για τον **ορθογραφικό έλεγχο** χρησιμοποιείται πάλι η βιβλιοθήκη Pattern. Στην βιβλιοθήκη αυτή έχει υλοποιηθεί ο ορθογραφικός έλεγχος που έχει δημιουργήσει ο Peter Norvig[12] και προσφέρει ακρίβεια της τάξης των 70%. Κάθε στοιχείο της λίστας, που επιστρέφει ο παραπάνω αλγόριθμος, συγκρίνεται με όλες τις κλάσεις/σχέσεις της οντολογίας και γίνεται αναζήτηση για κάποια τέλεια σύγκριση.

Να αναφερθεί ότι οι παραπάνω τρόποι εμπλουτισμού λειτουργούν μόνο για την Αγγλική γλώσσα και δεν αφορούν οποιαδήποτε άλλη γλώσσα υποστηρίζεται. Επίσης, οι συγκρίσεις με την οντολογία, στις παραπάνω μεθόδους, γίνονται μόνο για τις κλάσεις και τις σχέσεις αυτής και δεν γίνονται για τα άτομα. Ο αντίστοιχος ψευδοκώδικας του εμπλουτισμού φαίνεται παρακάτω:

```
// word, language: the given word and the given language
// checkNames: the above function that checks for comments and names
// ontology: the given ontology
classes ← list(ontology.classes())
properties ← list(ontology.properties())
individuals ← list(ontology.individuals())

classResult, classDistance ← checkNames(classes, word, language)
propertyResult, propertyDistance ← checkNames(properties,word, language)
individualResult, individualDistance ← checkNames(individuals,word,language)

// Here we check if any mapping was successful to return it
if classResult and propertyResult then
  if classDistance < propertyDistance then
    if individualResult and individualDistance < classDistance then
      return individualResult, individualDistance
    end if
    return classResult, classDistance
  else
    if individualResult and individualDistance < propertyDistance then
      return individualResult, individualDistance
    end if
    return propertyResult, propertyDistance
  end if
else if classResult = null and propertyResult:
  if individualResult and individualDistance < propertyDistance then
    return individualResult, individualDistance
  end if
  return propertyResult, propertyDistance
else if propertyResult = null and classResult:
  if individualResult and individualDistance < classDistance then
    return individualResult, individualDistance
  end if
  return classResult, classDistance
else if individualResult:
  return individualResult, individualDistance

// Here we check if we can enrich our results with WordNet
// We search for synonyms for english language
if language = ”en”:
  synsets ← wordnet.synsets(word)
  synonmys ← []
```

```
43
44    for each synset in synsets do
45      if synset != word then
46        synonyms.add(synset.lemma_names())
47      end if
48    end for
49
50    // We check only for perfect matches
51    minSynonymDistance ← 0
52    minSynonym ← null
53    for each syn in synonyms do
54      synClassResult, synClassDistance ← checkNames(classes, syn, language)
55      synPropResult, synPropDistance ← checkNames(properties, syn, language)
56      if synClassResult and synClassDistance <= minSynonymDistance then
57        minSynonymDistance ← synClassDistance
58        minSynonym ← synClassResult
59      end if
60      if synPropResult and synPropDistance <= minSynonymDistance then
61        minSynonymDistance ← synPropDistance
62        minSynonym ← synPropResult
63      end if
64    end for
65
66    if minSynonym:
67      return minSynonym, minSynonymDistance
68    end if
69  end if
70
71  // We search for hypernyms for english language
72  if language = ”en”:
73    wordMap ← pattern.synsets(word)
74    if len(wordMap) > 0 then
75      hypernyms ← wordMap[0].hypernyms(recursive=True)
76
77      minHypernymDistance ← 0
78      minHypernym ← None
79      for each hypernym in hypernyms do
80        hypClassRes, hypClassDist ← checkNames(classes, hypernym, language)
81        hypPropRes, hypPropDist ← checkNames(classes, hypernym, language)
82
83        if hypClassRes and hypClassDist <= minHypernymDistance then
84          minHypernymDistance ← hypClassDist
```

```
85          minHypernym ← hypClassRes
86        end if
87        if hypPropRes and hypPropDist <= minHypernymDistance then
88          minHypernymDistance ← hypPropDist
89          minHypernym ← hypPropRes
90        end if
91      end for
92
93      if minHypernym:
94        return minHypernym, minHypernymDistance
95      end if
96    end if
97  end if
98
99  // We search for suggestions in spelling and we keep the best result
100 if language = ”en”:
101   spellingSuggestions ← pattern.suggest(word)
102   if len(spellingSuggestions) > 0:
103     minSpellingDistance ← 0
104     minSpelling ← None
105     for each spelling in spellingSuggestions do
106       spClassRes, spClassDistance ← checkNames(classes, spelling, language)
107       spPropRes, spPropDistance ← checkNames(properties, spelling, language)
108
109       if spClassRes and spClassDistance <= minSpellingDistance then
110         minSpellingDistance ← spClassDistance
111         minSpelling ← spClassRes
112       end if
113       if spPropRes and spPropDistance <= minSpellingDistance then
114         minSpellingDistance ← spPropDistance
115         minSpelling ← spPropRes
116       end if
117     end for
118
119     if minSpelling:
120       return minSpelling, minSpellingDistance
121     end if
122   end if
123 end if
124 return null, 10000
```

Κώδικας 3.2: Ψευδοκώδικας αντιστοίχισης και εμπλουτισμού λέξεων

Όπως φαίνεται από τον κώδικα παραπάνω πρώτα έλεγχουμε να γίνεται κάποια αντιστοίχιση στις κλάσεις/σχέσεις/άτομα της οντολογίας. Αν έχει γίνει επιστρέφουμε την καλύτερη από τις τρεις. Αν δεν έχει γίνει καμία από τις τρεις τότε αρχίζουμε να ψάχνουμε τρόπους να βελτιώσουμε τα αποτελέσματα μας για την Αγγλική γλώσσα ξεκινώντας με τα συνώνυμα της δοθείσας λέξης στο WordNet και ελέγχοντας αν υπάρχει κάποιο από αυτά τα συνώνυμα στην οντολογία μας(στις κλάσεις ή στις σχέσεις). Ύστερα αν δεν βρεθεί κάποιο συνώνυμο τότε ψάχνουμε τα υπερώνυμα πάλι στο WordNet και κάνουμε τον ίδιο έλεγχο. Τελικά αν δεν βρεθούν είτε υπερώνυμα τότε κάνουμε ορθογραφικό έλεγχο μέσω του αλγόριθμου που αναφέραμε και πραγματοποιούμε ξανά τον έλεγχο αν κάποιες από τις προτάσεις του υπάρχουν στην οντολογία. Να αναφέρουμε ότι στις περιπτώσεις του εμπλουτισμού κάνουμε αναζήτηση μόνο για τέλεια αποτελέσματα(απόσταση Levenshtein 0) στην οντολογία και δεν έχουμε κάποιο ελάχιστο threshold.

Συνδυασμός Λέξεων

Αν γίνει επιτυχής αντιστοίχιση μιας λέξης με μια οντολογική σχέση/κλάση, γίνεται επιπλέον ένας έλεγχος αν μπορεί να συνδυαστεί η τωρινή λέξη με την προηγούμενη, την επόμενη ή και με τις δυο μαζί για να δημιουργήθει μια πιο ισχυρή σχέση. Αν η απόσταση που θα βρεθεί εκτελώντας τους τρεις αυτούς ελέγχους είναι καλύτερη απο την αρχική απόσταση ή και ίση(σε περίπτωση τέλειας σύγκρισης με μηδενική απόσταση Levenshtein), θεωρούμε ότι ο συνδυασμός είναι καλύτερος από τον αρχικό και γίνεται αυτή η αντιστοίχιση των λέξεων.

Πιο συγκεκριμένα αν γίνει αντιστοίχιση με την προηγούμενη λέξη και είχε και αυτή αντιστοιχιθεί, τότε η προηγούμενη αφαιρείται. Αν γίνει αντιστοίχιση με την επόμενη λέξη τότε την προσπερνάμε και αν γίνει αντιστοίχιση και με τις δυο πραγματοποιούνται και τα δυο προηγούμενα. Τέλος, θεωρούμε ότι αν έχει γίνει αντιστοίχιση είτε με την προηγούμενη είτε με την επόμενη και γίνεται και με τις δυο θεωρούμε ως σημαντικότερη την διπλή αντιστοίχιση. Ο ψευδοκώδικας του συνδυασμού φαίνεται παρακάτω:

```
1  // nodes, language: the nodes of the tree and the given language
2  // best_match: the above function that matches a word with ontology
3  // best_match_combination: a similar function without enrich results
4  // ontology: the given ontology
5  // sort: function that sorts the array with the given key ascending
6  result ← []
7  skipNext ← False
8  pos ← 0
9  for each word in nodes do
10   // with skipNext flag we skip the current word
11   if skipNext:
12     skipNext ← False
13     continue
14
15   matchForOneWord, mainDist ← best_match(word, onto, language)
```

```
16
17  if matchForOneWord:
18    combResults = []
19
20    // We combine with previous word
21    if pos > 0 then
22      combWithPrev ← node[pos−1] + ” ” + word
23      combResult,combDistance ← best_match_combination(word,combWithPrev,
24                                                        onto, language)
25      if combResult then
26          combResults.add([combDistance, combResult, ”prev”])
27      end if
28    end if
29
30    // We combine with next word
31    if pos <= len(node)−2 then
32      combWithNext ← word + ” ” + node[pos+1]
33      combResult,combDistance ← best_match_combination(word,combWithNext,
34                                                        onto, language)
35      if combResult then
36          combResults.add([combDistance, combResult, ”next”])
37      end if
38    end if
39
40    // We combine with both next and previous
41    if pos > 0 and pos <= len(node)−2 then
42      combWithBoth ← node[pos−1] + ” ” + word + ” ” + node[pos+1]
43      combResult,combDistance ← best_match_combination(word,combWithBoth,
44                                                        onto, language)
45      if combResult then
46          combResults.add({[combDistance, combResult, ”both”])
47      end if
48    end if
49
50    sort(combResults, key = 0)
51
52    // If any of the results is better than the main distance
53    // we append that one as stronger relationship
54    // If both has also the lowest distance we return that
55    // instead of next or prev
56    if len(combResults) > 0 then
57      minDistance ← combResults[0][0]
```

```
58        combResult ← combResults[0]
59        for each res in combResults do
60          if minDistance = res[0] and res[2] = ”both” then
61            combResult ← res
62          end if
63        end for
64
65        if combResult[0] <= mainDist then
66          // We skip the next word
67          if combResult[2] = ”next” or combResult[2] = ”both” then
68            skipNext ← True
69          end if
70
71          if combResult[2] = ”prev” or combResult[2] = ”both” then
72            // We remove the previous word
73            if result[−1] = node[pos−1] or result[−1] = combResult:
74              result.remove(result[−1])
75            end if
76          end if
77          result.add(combResult[1])
78          continue
79        end if
80      end if
81
82      // If there isn’t a combination we add the word
83      result.add(matchForOneWord)
84    end if
85    pos ← pos + 1
86  end for
87  return result
```

Κώδικας 3.3: Ψευδοκώδικας συνδυασμού λέξεων

3.2.4 Δημιουργία της τελικής οντολογικής σχέσης

Το τελευταίο και πιο κρίσιμο κομμάτι είναι ουσιαστικά το κομμάτι που δημιουργείται η οντολογική σχέση σύμφωνα με τις οντολογικές αντιστοιχίσεις που έγιναν στο προηγούμενο στάδιο. Στο συγκεκριμένο στάδιο γίνονται διαδοχικές προσπελάσεις της δοθείσας πρότασης για να διευκρινιστεί σε ποιο οντολογικό άτομο αναφέρονται οι κλάσεις και οι σχέσεις που έχουν αντιστοιχιστεί.

Η διαδικασία αυτή ξεκινάει απο το προηγούμενο στάδιο, στο οποίο εκτός από την κλάση/άτομο-/σχέση που έχει αντιστοιχηθεί, επιστρέφονται και κάποιες πληροφορίες για το είδος της αντιστοίχισης καθώς επίσης και για τον τύπο που θα έχει αργότερα αυτή κατά την δημιουργία της τελικής οντολογικής σχέσης. Η διαφορά που έχουν οι σχέσεις με τις κλάσεις είναι ότι οι κλάσεις αναφέρονται σε ένα

άτομο στην οντολογία ενώ οι σχέσεις αναφέρονται σε δυο. Επομένως, θα πρέπει να διευκρινίζεται, στην τελική οντολογική σχέση, για τις σχέσεις σε ποια δυο άτομα αναφέρονται και για τις κλάσεις σε ποιο άτομο αναφέρονται.

Παρακάτω παρουσιάζονται βασικά κομμάτια της υλοποίησης για την δημιουργία της τελικής οντολογικής σχέσης.

Αλλαγή μεταβλητής

Το κύριο πρόβλημα στην δημιουργία της τελικής σχέσης είναι να διαπιστωθεί που ακριβώς γίνεται η αλλαγή της μεταβλητής, δηλαδή του ατόμου, στο οποίο αναφέρεται η παρούσα λέξη, είτε αυτή είναι σχέση είτε κλάση. Διαπιστώθηκε, ότι στις παρακάτω περιπτώσεις πρέπει να γίνει αλλαγή της μεταβλητής:

- Αν η λέξη που αντιστοιχήθηκε είναι ρήμα και έχει αντιστοιχηθεί με σχέση στην οντολογία
- Αν η λέξη είναι ρήμα και είναι κτητικό, είτε έχει αντιστοιχηθεί είτε όχι, δηλαδή στην αγγλική γλώσσα οι λέξεις has και have
- Αν η λέξη είναι συνδέσμος είτε προθέση και αναπαριστάται από το Spacy με το **IN** μέρος του λόγου
- Αν έχει γίνει αντιστοίχιση συνδυασμού λέξεων με σχέση

Ειδικά για το τελευταίο μπορεί ο μηχανικός που θα φτιάξει την οντολογία να προσθέσει στα σχόλια των σχέσεων, που θέλει να γίνει η αλλαγή, τον συνδυασμό αυτό, ώστε να πετύχει τα καλύτερα αποτέλεσματα.

Η διαδικασία της δημιουργίας της σχέσης αρχικοποιείται με μια ακέραια μεταβλητή που αναπαριστά το εκάστωτε άτομο, στο οποίο αναφέρονται οι σχέσεις/κλάσεις, και είναι αρχικοποιημένη στο 0. Αυτή η μεταβλητή αυξάνεται κάθε φορά που πρέπει να γίνει μια αλλαγή ώστε οι επόμενες σχέσεις/κλάσεις να αναφέρονται στο καινούργιο άτομο.

Η χρήση της σύζευξης

Κατά την σύζευξη, δεν είναι γνωστό σε ποιο άτομο αναφέρονται οι επόμενες οντολογικές αντιστοιχίσεις, αφού είναι εφικτό η επόμενη σχέση που θα παρουσιαστεί να αναφέρεται είτε στο δεύτερο άτομο μιας σχέσης είτε στο πρώτο. Δυο παραδείγματα, στα οποία η σύζευξη έχει διαφορετικό ρόλο είναι τα παρακάτω:

1. A person that is married and in love
2. A person that is married to a beautiful and healthy person

Στα παραδείγματα αυτά φαίνεται ότι στην πρώτη περίπτωση το in love θα αναφέρεται στο πρώτο άτομο, ενώ στην δεύτερη περίπτωση φαίνεται ότι το and χρησιμοποιείται για να χαρακτηρίσει παραπάνω το δεύτερο άτομο, δηλαδή το άτομο που αναφέρεται δεύτερο στην σχέση married.

Για να αποφευχθεί η λανθασμένη αναφορά σε διαφορετικό άτομο κατά την εμφάνιση μιας συζευκτικής λέξης, είτε ενός ρήματος(το ίδιο μπορεί να παρουσιαστεί και με την χρήση τους), γίνεται αναζήτηση μέσω των εξαρτήσεων της πρότασης, για να βρεθεί η λέξη στην οποία αναφέρεται η σύζευξη.

Μέσω αυτής αλλάζει η μεταβλητή του ατόμου, ώστε οι επόμενες κλάσεις/σχέσεις να αναφέρονται στο σωστό.

Η χρήση των κτητικών ρημάτων

Όπως προαναφέρθηκε κατά την χρήση κτητικών ρημάτων γίνεται αλλαγή της μεταβλητής, ώστε να αναπαριστά το καινούργιο άτομο που θα αναφερθεί παρακάτω. Αυτό γίνεται για τις λέξεις has και have στην αγγλική γλώσσα. Η αλλαγή στα κτητικά ρήματα γίνεται για να βελτιωθούν αποτελέσματα όπως το παρακάτω:

A person that has a female friend

όπου παράγεται το παρακάτω αποτέλεσμα:

Person(X0), Female(X1), Friendof(X0,X1)

Σχέσεις

Κατά την δημιουργία των σχέσεων θα πρέπει να ξέρουμε ή να δημιουργήσουμε το δεύτερο άτομο στο οποίο αναφέρεται η σχέση. Υπάρχουν δυο κύριες περιπτώσεις εδώ για να βρούμε την μεταβλητή αυτή, αν έχει γίνει προηγούμενη αλλαγή με κάποιο κτητικό ρήμα ή αν δεν έχει γίνει.

Αν έχει γίνει αλλαγή σε ένα κτητικό ρήμα, τότε για την εκάστοτε σχέση χρησιμοποιείται σαν δεύτερο άτομο στην σχέση είτε η προτελευταία μεταβλητή, δηλαδή η μεταβλητή πριν την αλλαγή, είτε η μεταβλητή, στην οποία αναφέρεται η λέξη από την οποία εξαρτάται η σχέση. Η δεύτερη είναι πιο ισχυρή σαν σύνδεση και προτιμάται από την πρώτη.

Στο σημείο αυτό έχει υλοποιηθεί και η αλλαγή που μπορεί να γίνει στην σειρά των ατόμων σε μια σχέση. Αν σε οποιοδήποτε comment της σχέσης υπάρχει η λέξη has/have, τότε γίνεται αντιστροφή των ατόμων στην σχέση για να προσδιορισθεί σωστά η σχέση, ανάλογα με την έννοια που της έχει προσδώσει ο μηχανικός της οντολογίας. Οπότε η σχέση:

$$FatherOf(X, Y)$$

που κανονικά υποδυκνύει ότι ο Χ είναι πατέρας του Υ με την συγκεκριμένη διαδικασία μπορεί να γίνει

$$FatherOf(Y, X)$$

που υποδηλώνει ότι ο Χ έχει έναν πατέρα που είναι ο Υ. Είναι στο χέρι του μηχανικού της οντολογίας να βρίσκει τις σχέσεις που θέλει να πετύχει αυτό το αποτέλεσμα και μπορεί να τις τροποποιεί αντίστοιχα για να το πετύχει.

Αν δεν έχει γίνει αλλαγή με ένα κτητικό ρήμα, τότε προκύπτουν οι παρακάτω δυο περιπτώσεις.

1. Αν έχει χρησιμοποιηθεί συνδετική λέξη πρίν απο την σχέση που εξετάζεται(μέχρι να βρεθεί κάποια άλλη σχέση) τότε απλώς χρησιμοποιείται σαν δεύτερο άτομο για την σχέση, είτε ο αμέσως επόμενος αριθμός μεταβλητής(δηλαδή αύξηση της τωρινής μεταβλητής κατά 1), είτε αν η τελευταία μεταβλητή είναι όνομα, δημιουργείται ένας ακέραιος που αναπαριστά το άτομο.

2. Αν δεν έχει χρησιμοποιηθεί συνδετική λέξη, τότε αν πρέπει να γίνει αλλαγή μεταβλητής στο εκάστοτε σημείο, γίνεται ένας ακόμα έλεγχος αν η σχέση μεταξύ των δυο ατόμων που θα δημιουργούταν έχει χρησιμοποιηθεί ήδη. Αν έχει χρησιμοποιηθεί δημιουργείται μια καινούργια μεταβλητή που είναι είτε 2 φόρές μεγαλύτερη απο την προηγούμενη(αν η τωρινή μεταβλητή είναι ακέραιος) ή έχει την τιμή 1 αν η τωρινή μεταβλητή είναι όνομα. Αν οποιοσδήποτε απο τους παραπάνω ελέγχους δεν ισχύει τότε ακολουθείται η πρώτη περίπτωση.

Ο αντίστοιχος ψευδοκώδικας του συγκεκριμένου σημείου είναι ο παρακάτω:

```
// usedProperties: list with the used properties until now
// changeVariable: the flag is true if we need to change the variable
// currentVariable: the currentVariable
// CheckPreviousWords: function to check for an and in previous words
// CreateProperty: function to create a new property
usedAnd ← CheckPreviousWords()
if usedAnd = False and changeVariable then
  newVariable ← currentVariable + 1
  if (currentVariable, newVariable) in usedProperties then
    newVariable ← currentVariable + 2
    CreateProperty(currentVariable, newVariable)
    continue
  end if
end if

newVariable ← currentVariable + 1
CreateProperty(currentVariable, newVariable)
```

Κώδικας 3.4: Ψευδοκώδικας ελέγχου σχέσεων

Η παραπάνω διαδικασία χρησιμοποιείται για να γίνει σωστή αντιστοίχιση συγκεκριμένων προτάσεων στις οποίες γίνεται χρήση δυο σχέσεων η μια μετά την άλλη. Ένα παράδειγμα τέτοιας πρότασης, στην οποία οι λέξεις parent και love είναι σχέσεις, είναι η παρακάτω:

A parent that is in love.

όπου δημιουργείται το παρακάτω αποτέλεσμα:

Parent(X0,X1), Love(X0,X2)

το ρήμα love στην συγκεκριμένη περίπτωση αντιστοιχίζεται με σχέση, οπότε γίνεται σε αυτό αλλαγή μεταβλητής.

Σχέσεις που εννοούνται

Πολλές φορές στον προφορικό και στο γραπτό λόγο λέξεις μπορεί να εννοούνται και να μην περιλλαμβάνονται σε μια πρόταση. Ένα παράδειγμα τέτοιας πρότασης είναι η παρακάτω:

The parent of a male and of a female

Στην συγκεκριμένη περίπτωση θα πρέπει η σχέση parent να επαναληφθεί δυο φορές, ώστε να είναι σωστή η οντολογική ερώτηση που θα δημιουργηθεί. Οπότε για να καλυφθεί και αυτή η περίπτωση, σε περιπτώσεις που πρέπει να γίνει αλλαγή μεταβλητής και δεν υπάρχει σχέση στην οποία αναφέρεται η καινούργια μεταβλητή, τότε γίνεται δημιουργία της τελευταίας σχέσης που έχει χρησιμοποιηθεί με δεύτερο άτομο την καινούργια μεταβλητή και το ίδιο πρώτο άτομο. Στο παραπάνω παράδειγμα με την συγκεκριμένη διαδικασία θα έχουμε το παρακάτω αποτέλεσμα:

Parentof(X0,X1), Male(X1), Parentof(X0,X2), Female(X2)

Άτομα

Όταν μια λέξη έχει αντιστοιχιστεί με ένα άτομο, έτσι όπως γίνεται η προσπέλαση των αντιστοιχισμένων λέξεων, πρέπει να πάει να αντικαταστήσει την τελευταία μεταβλητή, η οποία μπορεί ήδη να έχει χρησιμοποιηθεί σε σχέσεις και κλάσεις. Οπότε για να δημιουργηθεί σωστή οντολογική ερώτηση πρέπει να γίνουν αντικαταστάσεις στις παρακάτω κατηγορίες:

- Στις ήδη υπάρχουσες σχέσεις, αν η μεταβλητή είναι είτε το πρώτο είτε το δεύτερο άτομο αυτών
- Στις ήδη υπάρχουσες κλάσεις
- Στις σχέσεις που θα εισαχθούν επειδή εννούνται, όπως αναφέρθηκε και στην παραπάνω παράγραφο
- Στην λίστα με τις χρησιμοποιούμενες σχέσεις για να γίνεται αργότερα έλεγχος αν έχει χρησιμοποιηθεί σχέση με το άτομο αυτό

Αφαιρέσεις

Αφού έχουν γίνει όλοι οι παραπάνω έλεγχοι και έχουν βρεθεί τα άτομα στα οποία αναφέρονται οι σχέσεις και οι λέξεις της πρότασης, όλες οι λέξεις στις οποίες δεν έχει γίνει κάποια αντιστοίχιση διαγράφονται απο την τελική ερώτηση.

Επίσης γίνεται και ένας έλεγχος απο τις αντιστοιχισμένες σχέσεις και κλάσεις, ώστε να αφαιρεθούν τα συνεχόμενα διπλότυπα αν έχουν δημιουργηθεί. Τέτοιες περιπτώσεις μπορούν να τύχουν αν ο μηχανικός της οντολογίας έχει προσθέσει στην ίδια κλάση ή σχέση παραπάνω από μια διαφορετική ερμηνεία στα σχόλια της. Αν τύχει να υπάρξουν λοιπόν, δυο συνεχόμενες αντιστοιχίσεις της ίδιας κλάσης ή σχέσης με την ίδια μεταβλητή, η μια απο αυτές διαγράφεται.

Άρνηση

Τέλος έχει υλοποιηθεί και άρνηση για τις οντολογικές σχέσεις. Κατά το τελευταίο στάδιο που γίνεται μόνο η συλλογή των αντιστοιχισμένων λέξεων γίνεται και ένας έλεγχος για το αν έχει γίνει χρήση κάποιας άρνησης. Αυτός ο έλεγχος γίνεται με την εξάρτηση **neg** που παρέχει το Spacy για τις αρνήσεις. Το αποτέλεσμα είναι η προσθήκη του συμβόλου **\neg** στις οντολογικές σχέσεις. Ένα παράδειγμα είναι:

A parent that is not married.

η οποία παράγει το παρακάτω αποτέλεσμα

Parentof(X0,X1), \not Marriedto(X0,X2)

Για την αγγλική γλώσσα συγκεκριμένα έχει υλοποιηθεί και η χρήση του cannot(δεν μπορώ) η οποία είναι διαφορετική από την απλή άρνηση. Το πρόβλημα στο συγκεκριμένο κομμάτι ήταν ότι το Spacy αναλύει το cannot σαν δυο λέξεις ένα can και ένα not. Επομένως γίνεται έλεγχος αν πρίν απο μια λέξη not εμφανίζεται η λέξη can ώστε να προστεθεί το σύμβολο **\cneg** αντί του not. Ένα παράδειγμα για το cannot είναι:

A female that cannot get married.

η οποία παράγει το παρακάτω αποτέλεσμα

Female(X0), \cnot Marriedto(X0,X1)

Δημιουργία της οντολογικής ερώτησης

Αφού έχουν γίνει όλα τα παραπάνω δημιουργείται τελικά η τελευταία οντολογική σχέση με την μορφή που έχει αναφερθεί σε όλα τα παραπάνω. Η μορφή αυτή ενώνει τις κλάσεις και τις σχέσεις με κόμματα, τα οποία αναπαριστούν την σύζευξη. Ο ψευδοκώδικας για την δημιουργία της τελικής ερώτησης είναι ο παρακάτω:

```
// strip: removes characters from the end of a string
resultString ← ""
for each result in Results do
  if result = "neg" then
    if result.cannot = True then
      resultString ← resultString + "\\cnot "
    else
      resultString ← resultString + "\\not "
    end if
    continue
  end if
  if result is class then
    resultString ← resultString + "(" + result.variable + "), "
  else
    resultString ← resultString + "(" + result.propertyVar1 + ","
                                      + result.propertyVar2 + "), "
  end if
end for
resultString ← resultString.strip(", ")
```

Κώδικας 3.5: Ψευδοκώδικας της δημιουργίας της ερώτησης

3.3 Web API και οδηγίες χρήσης

Όπως αναφέρθηκε και παραπάνω για την εύκολη χρήση της παραπάνω εφαρμογής, αναπτύχθηκε παράλληλα και ένα Web API(A.1.3) με το οποίο ο χρήστης μπορεί να χρησιμοποιήσει την οντολογία και την γλώσσα που θέλει για να υπολογίσει μια οντολογική ερώτηση. Για την δημιουργία του API χρησιμοποιήθηκε το framework της Python που λέγεται Flask(A.2.7) με το οποίο δημιουργήθηκε μια μέθοδος που μπορεί ο χρήστης να κάλεσει για να πάρει πίσω την απάντηση από την εφαρμογή.

Ο χρήστης θα μπορεί να κάνει μια κλήση POST(A.1.4) με περιεχόμενο σε μορφή JSON(A.1.5) όπως αυτή:

```
{
  "Ontology": "https://test.com/text.owl",
  "Text": "A test sentence",
  "Language": "en",
}
```

Κώδικας 3.6: Παράδειγμα δομής κλήσης του API

το πεδίο **Ontology** είναι η διεύθυνση στην οποία υπάρχει η οντολογία που θέλει να χρησιμοποιήσει ο χρήστης, το πεδίο **Text** είναι η πρόταση την οποία θέλει να ρωτήσει ο χρήστης και το πεδίο **Language** είναι η γλώσσα με την οποία θέλει να χρησιμοποιήσει την οντολογία. Το πεδίο Ontology αν δεν έχει δοθεί χρησιμοποιείται η εσωτερική οντολογία με την οποία έχουν γίνει και τα τέστ της εφαρμογής. Το πεδίο γλώσσας υποστηρίζει τις παρακάτω τιμές:

1. **En**: English
2. **De**: German
3. **Es**: Spanish
4. **Fr**: French
5. **Pt**: Portugal

αν δεν δοθεί καμία απο τις παραπάνω επιλογές ή αν δοθεί κάποια λανθασμένη, η γλώσσα που χρησιμοποιείται είναι η Αγγλική.

Αν δεν έχει δοθεί πεδίο Text ή αν υπάρχει πρόβλημα με την φόρτωση της οντολογίας τότε επιστρέφεται απάντηση με status 400 τύπου Bad Request(A.1.4) και με την παρακάτω δομή σε JSON:

```
{
  "Error": "No text given"
}
```

Κώδικας 3.7: Παράδειγμα σφάλματος κλήσης του API

τα δυο μηνύματα που μπορεί να επιστραφούν είναι:

- **No text given**: όταν δεν έχει δοθεί πεδίο text

- **Problem loading ontology**: όταν υπάρχει πρόβλημα στην φόρτωση της οντολογίας

Αν όλα έχουν πάει καλά επιστρέφεται μια απάντηση με status 200(OK) και την παρακάτω δομή σε JSON:

```
{
  "Ontology": "https://test.com/text.owl",
  "Result": "Test(X0)",
  "Language": "en",
}
```

Κώδικας 3.8: Παράδειγμα απάντησης του API

Τα πεδία Ontology και Language είναι τα ίδια με την δομή της κλήσης, όπως παραπάνω, και το πεδίο **Result** είναι η τελική ερώτηση που δίνει η εφαρμογή για την πρόταση που έδωσε ο χρήστης στο πεδίο Text. Αν υπάρξει κάποιο πρόβλημα θα επιστραφεί απάντηση με status 500(Internal Server Error).

3.4 Επιπλέον πράγματα που υλοποιήθηκαν

Έχουν επίσης υλοποιηθεί και δυο μηχανισμοί που μπορούν να εμπλουτίσουν την οντολογία που χρησιμοποιείται κάθε φορά για την δημιουργία της οντολογίκης ερώτησης. Οι δυο αυτοί μηχανισμοί είναι:

- Κατά την φόρτωση της οντολογίας γίνεται μια διαδικασία αναζήτησης του ορισμού των κλάσεων και σχέσεων μέσω της βιβλιοθήκης NLTK. Με αυτό τον ορισμό εμπλουτίζουμε τα σχόλια της αντίστοιχης κλάσης και σχέσης για να χρησιμοποιηθούν αργότερα στις παραπάνω διαδικασίες. Η συγκεκριμένη διαδικασία χρησιμοποιείται μόνο στην Αγγλική γλώσσα και οι ορισμοί που εισάγονται στα σχόλια των κλάσεων/σχέσεων θα πρέπει να είναι μια λέξη και όχι ολόκληρη πρόταση.
- Μια ακόμα διαδικασία που υλοποιήθηκε είναι ο εμπλουτισμός της οντολογίας μέσω της δημιουργίας καινούργιων κλάσεων. Αφού γίνει η δημιουργία της οντολογικής ερώτησης, οι αρχικές λέξεις της δοθείσας πρότασης που είναι επίθετα(**ADJ** μέρος του λόγου στο Spacy) και δεν έχουν αντιστοιχιστεί με κάποια οντολογική σχέση/κλάση/άτομο, μπορούν με την συγκατάθεση του χρήστη να δημιουργηθούν σαν κλάσεις στην οντολογία που χρησιμοποιείται. Αυτή η διαδικασία μπορεί να χρησιμοποιηθεί από τον μηχανικό της οντολογίας για να εμπλουτίσει γρήγορα την οντολογία, αν παρατηρήσει κάποια ερώτηση που έχει δημιουργηθεί και δεν είχε τα απολύτως επιθυμητά αποτελέσματα.

Οι παραπάνω δυο διαδικασίες δεν λειτουργούν στην API έκδοση της εφαρμογής, αλλά μόνο στην τοπική εφαρμογή που τρέχει σε παράθυρο εντολών(terminal).

Κεφάλαιο 4

Αποτελέσματα

Για την αξιολόγηση της εφαρμογής δημιουργήθηκε μια γεννήτρια τυχαίων οντολογικών σχέσεων και μια διαδικτυακή εφαρμογή που την αξιοποιεί. Με την εφαρμογή αυτή κλήθηκαν να γράψουν οι χρήστες την τυχαία κάθε φορά οντολογική σχέση σε φυσική γλώσσα. Τα αποτελέσματα που πάρθηκαν απο την εφαρμογή χρησιμοποιήθηκαν σαν δείγμα για την αξιολόγηση της εφαρμογής. Παρακάτω πρώτα περιγράφεται η γεννήτρια τυχαίων οντολογικών σχέσεων(Query Generator) και ύστερα παρουσιάζονται τα αποτελέσματα.

4.1 Δημιουργία Query Generator

Όπως έχει αναφερθεί και στο θεωρητικό μέρος(2.2.1) οι οντολογικές σχέσεις έχουν μια δενδρική δομή και η γεννήτρια τυχαίων οντολογικών σχέσεων που αναπτύχθηκε δημιουργεί τέτοιες δομές για τις σχέσεις που παράγει. Πιο συγκεκριμένα δημιουργούνται δενδρικές δομές με αναδρομικό τρόπο μέχρι βάθος 3 και κάθε κόμβος έχει τα παρακάτω χαρακτηριστικά:

- Το πολύ 2 φύλλα με τα οποία συνδέεται με διαφορικές σχέσεις
- Από 1 μέχρι 2 κλάσεις που το περιγράφουν
- Ακέραια μεταβλητή για όνομα είτε κάποιο μη χρησιμοποιημένο άτομο της οντολογίας

Ο ψευδοκώδικας για την δημιουργία της τυχαίας οντολογικής σχέσης είναι ο παρακάτω:

```
1  // Find a random object from a group(individuals,classes,properties)
2  // randomInt: function that return a random int from a range
3  // individuals, classes, relations: the ontology entities
4  function findARandomFromClass(group, used):
5    number ← randomInt(0,len(group)−1)
6    obj ← group[number]
7    // We search for a not used object
8    while obj in used do
9      number ← random.randint(0,len(group)−1)
10     obj ← group[number]
11   end while
12   used.add(obj)
13   return obj
14 end function
```

```
15
16  // Create a nodes name
17  function createName(usedVariables):
18    hasName ← randInt(0,1)
19    allIndividualsUsed ← True
20    for each name in individuals do
21      if name not in usedVariables then
22        allIndividualsUsed ← False
23      end if
24    end for
25
26    if hasName and not allIndividualsUsed then
27      name ← findARandomFromClass(individuals,usedVariables)
28    else
29      variables = []
30      for each name in usedVariables do
31        if type(name) is int then
32          variables.add(name)
33        end if
34      end for
35
36      if len(variables) > 0 then
37        usedVariables.add(variables[−1] + 1)
38      else
39        usedVariables.add(0)
40      end if
41      name ← usedVariables[−1]
42    end if
43    return name
44  end function
45
46  // Create a node
47  function createNode(usedVariables, existedName=null):
48    subString ← ""
49    if existedName != null then
50      name ← createName(usedVariables)
51    else
52      name ← existedName
53    end if
54    numOfClasses ← randInt(1,2)
55    usedClasses ← []
56    for i from 0 to numOfClasses do
```

```
57      className ← findARandomFromClass(classes,usedClasses)
58      if type(name) is int then
59        classString ← className + ”(X” + str(name) + ”)”
60        subString ← subString + className + ”(X” + str(name) + ”), ”
61        name ← ”X” + str(name)
62      else
63        classString ← className + ”(” + name + ”)”
64        subString ← subString + className + ”(” + name + ”), ”
65      end if
66    end for
67    return subString, name
68  end function
69
70  // depth: the depth of the tree
71  // usedVariables: already used variables either int or individuals
72  // nodeName: the name of the node
73  function recursiveCreateNodes(depth, usedVariables, nodeName):
74    question ← ””
75    if depth > 0 then
76      subSt, nodeName = createNode(usedVariables,nodeName)
77      question ← question + subSt
78      numOfChildren ← random(0,2)
79      for child from 0 to numOfChildren do
80        childName ← createName(usedVariables)
81        prop = findARandomFromClass(properties,[])
82        question ← question + createProperty(nodeName,childName,prop)
83        subSt ← recursiveCreateNodes(depth−1,usedVariables, childName)
84        question ← question + subSt
85      end for
86    end if
87    return question
88  end function
```

Κώδικας 4.1: Ψευδοκώδικας δημιουργίας τυχαίας οντολογικής σχέσης

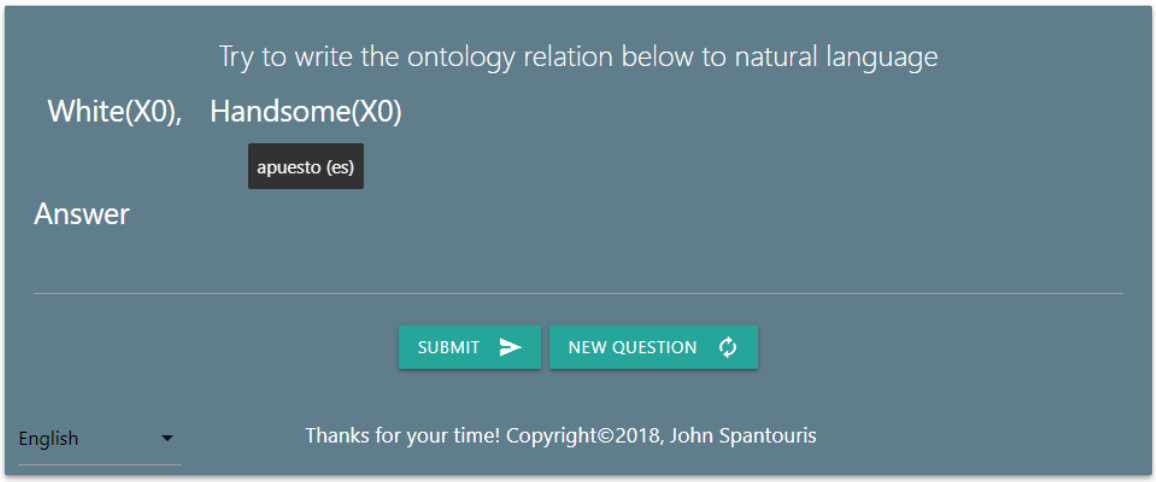
Στον παραπάνω ψευδοκώδικα φαίνονται όλες οι συναρτήσεις που χρησιμοποιούνται για να δημιουργηθεί μια τυχαία οντολογική σχέση. Οι συναρτήσεις που χρησιμοποιούνται κάνουν τα παρακάτω:

- **findARandomFromClass**: αναζητάει έναν τυχαίο αντικείμενο από την λίστα που της δίνεται
- **createName**: δημιουργεί ένα όνομα για τον κόμβο και τυχαία είτε θα είναι νούμερο με αρχικό το X ή θα διαλέξει απο την λίστα με τα άτομα της οντολογίας αν δεν έχουν ήδη χρησιμοποιηθεί όλα
- **createNode**: δημιουργεί έναν κόμβο. Αρχικά κοιτάει αν έχει όνομα αλλιώς του δημιουργεί μέσω

της παραπάνω συνάρτησης και μετά κοιτάει πόσες κλάσεις θα έχει αυτός ο κόμβος

- **recursiveCreateNodes**: δημιουργεί αναδρομικά ένα δέντρο με κόμβους και τους συνδέει με τυχαίες οντολογικές σχέσεις μεταξύ τους

Με την γεννήτρια τυχαίων σχέσεων δημιουργήθηκε παράλληλα μια ιστοσελίδα με την οποία μπορούν οι χρήστες να υποβάλλουν την αντίστοιχη φυσική γλώσσα για κάθε οντολογική σχέση που παράγεται. Η ιστοσελίδα που παράχθηκε έχει την μορφή που φαίνεται στην εικόνα παρακάτω.



Η ιστοσελίδα δημιουργήθηκε και αυτή με την βιβλιοθήκη Flask(A.2.7) που παρέχει εργαλεία για την επικοινωνία του front end μιας ιστοσελίδας με τον server(back end). Στην παραπάνω ιστοσελίδα δίνεται η δυνατότητα αλλαγής γλώσσας, όπως φαίνεται κάτω αριστερά καθώς επίσης και η δυνατότητα για δημιουργία νέας ερώτησης αν αυτή που έχει παραχθεί δεν μπορεί να αναπαρασταθεί σε φυσική γλώσσα. Ένα ακόμα χαρακτηριστικό είναι η εμφάνιση των αντίστοιχων σχολίων(comments) που έχει κάθε σχέση ή κλάση στην αντίστοιχη οντολογία σε κάθε γλώσσα όταν ο χρήστης περνάει το δείκτη πάνω στην αντίστοιχη λέξη, όπως φαίνεται για την λέξη Handsome.

Τα αποτελέσματα που δημιουργούν οι χρήστες αποθηκεύονται σε έναν πίνακα που φιλοξενείται στο cloud μέσω του Microsoft Azure(A.1.6). Στην ίδια υπηρεσία φιλοξενείται και η ιστοσελίδα. Για τα αποτελέσματα βρίσκουμε και το hash(sha256)(A.1.7) της ερώτησης και το αποθηκεύουμε και αυτό, ώστε μετά να ξέρουμε αν η ίδια ερώτηση έχει παραχθεί ξανά χωρίς εκτενές ψάξιμο.

4.2 Παρουσίαση αποτελεσμάτων

Για να δούμε τα αποτελέσματα σε επίπεδο πολυγλωσσίας και επειδή δεν μας δόθηκαν αποτελέσματα σε άλλες γλώσσες μεταφράσαμε αυτόματα τις προτάσεις που έδωσαν οι χρήστες στις αντίστοιχες γλώσσες που θέλουμε να δοκιμάσουμε. Η οντολογία μεταφράστηκε και αυτή στα Γαλλικά, Ισπανικά και Γερμανικά ώστε να μπορούμε να δούμε την ακρίβεια των αποτελεσμάτων του συστήματος (ευχαριστούμε θερμά τα άτομα που βοήθησαν στην μετάφραση). Επίσης να αναφέρουμε ότι η οντολογία που χρησιμοποιήθηκε περιέχει 18 κλάσεις, 13 σχέσεις και 4 άτομα.

Στο σύνολο παράχθηκαν 106 προτάσεις σε φυσική γλώσσα στα Αγγλικά από άτομα στα οποία απλώς δόθηκαν μερικά παραδείγματα μετατροπής και η οδηγία ότι πρέπει οι απαντήσεις που θα παρά-

γουν να είναι μόνο μια πρόταση. Για πιο σωστή αναπαράσταση των αποτελεσμάτων κρατήθηκαν από τις 106 προτάσεις οι πρώτες 100.

Τα παρακάτω αποτελέσματα αξιολόγησαν πόσο κοντά είναι η απάντηση που έδωσε η εφαρμογή στην πρόταση που απάντησε ο τελικός χρήστης στην τυχαία παραχθήσα οντολογική σχέση. Στα αποτελέσματα που παρήχθησαν εφαρμόσαμε ένα threshold της τάξης τους **80%**.

Τα αποτελέσματα που παρήχθησαν ανά γλώσσα και κάποια στατιστικά στοιχεία φαίνονται στον πίνακα που ακολουθεί:

Γλώσσα	Αγγλικά	Γαλλικά	Ισπανικά	Γερμανικά
Προτάσεις που βρέθηκαν	80	37	48	42
Μέσο μήκος πρότασης	8.7	8.18	8.46	8.36
Μέσο μήκος πρότασης που βρέθηκε	7.14	4.14	5.36	5.54
Μέσο μήκος πρότασης που δεν βρέθηκε	14.95	10.56	11.15	10.53
Ακρίβεια	**80%**	**37%**	**48%**	**42%**

Όπως φαίνεται και παραπάνω, τα αποτελέσματα είναι αρκετά ικανοποιητικά για την Αγγλική γλώσσα. Για τις υπόλοιπες γλώσσες τα αποτελέσματα δεν είναι τόσο καλά και αυτό οφείλεται σε δυο λόγους. Αρχικά οι προτάσεις που χρησιμοποιήθηκαν ήταν auto translated μέσω του Google Translate με αποτέλεσμα να μην ήταν αρκετά ακριβής η μετάφραση των προτάσεων αυτών. Επίσης η αντιστοίχιση των λέξεων με αυτές της οντολογίας δεν ήταν ικανοποιητική και αυτό οφείλεται στο ότι η οντολογία είχε αναπτυχθεί στα Αγγλικά και είχε δοθεί στην γλώσσα αυτή μεγαλύτερη βαρύτητα από τις υπόλοιπες.

Οι αντιστοιχίσεις παίζουν μεγάλο ρόλο στην διαδικασία δημιουργίας του αποτελέσματος και η μη σωστή αντιστοίχιση με κλάσεις της οντολογίας οδηγεί σε διαφορετικά αποτελέσματα απο ότι θα έπρεπε. Στα αποτελέσματα παρατηρήθηκε ότι πολλές λέξεις δεν είχαν αντιστοιχηθεί με τις σωστές κλάσεις με αποτέλεσμα την παραγωγή λάθος αποτελεσμάτων.

Κεφάλαιο 5

Επίλογος

5.1 Σύνοψη

Στην παρούσα διπλωματική εργασία προσπαθήσαμε να δημιουργήσουμε οντολογικές σχέσεις από φυσική γλώσσα μέσω της ανάλυσης της δοθείσας πρότασης από το χρήστη με ένα εργαλείο επεξεργασίας φυσικής γλώσσας και ύστερα με την αντιστοίχιση των λέξεων της πρότασης αυτής να δημιουργήσουμε από μια οντολογία την αντίστοιχη οντολογική σχέση. Το σύστημα το οποίο προσπαθήσαμε να αναπτύξουμε ήταν πολυγλωσσικό και ανεξάρτητο της δοθείσας οντολογίας. Παράλληλα δημιουργήσαμε και ένα εργαλείο παραγωγής τυχαίων οντολογικών σχέσεων για την αξιολόγηση του πρώτου συστήματος. Τις τυχαίες αυτές οντολογικές σχέσεις που προσπάθησαν να αναπαραστήσουν οι χρήστες που ερωτήθηκαν χρησιμοποιήσαμε για την αξιολόγηση της ακρίβειας του συστήματος. Τέλος το σύστημα που προσπαθήσαμε να αναπτύξουμε θέλαμε να δίνει την δυνατότητα στον μηχανικό να μπορεί να βελτιώσει τα αποτελέσματα του και μέσω της χρήσης των σχολίων ο μηχανικός της οντολογίας μπορεί να βλέπει τις προβληματικές περιπτώσεις του συστήματος και να αλλάζει αναλόγως την οντολογία.

Τελικώς το σύστημα το οποίο αναπτύχθηκε δεν είχε την επιθυμητή ακρίβεια που περιμέναμε μέσω των πειραμάτων που πραγματοποιήθηκαν. Όπως φαίνεται η ακρίβεια στην Αγγλική γλώσσα ήταν αρκετά καλή αλλά στις υπόλοιπες γλώσσες αντιμετώπιζε προβλήματα. Τα μέτρια αποτελέσματα οφείλονταν κυρίως στην αυτοματοποιημένη μετάφραση των προτάσεων σε άλλες γλώσσες καθώς επίσης και στην χρήση των σχολίων στην οντολογία που αναπτύχθηκε για τις γλώσσες αυτές. Η χρήση των σχολίων είναι αρκετά σημαντική για την δημιουργία σωστών αποτελεσμάτων και η έλλειψη αυτών και των διάφορων περιπτώσεων που μπορεί να παρουσιαστούν προκάλεσε χαμηλά αποτελέσματα στις άλλες γλώσσες.

5.2 Επεκτάσεις

Για την ενίσχυση του συστήματος θα μπορούσε μελλοντικά να βρεθεί κάποιος καλύτερος τρόπος με τον οποίο θα μπορούσε να γίνει η αντιστοίχιση των λέξεων της δοθείσας από τον χρήστη πρότασης στις αντίστοιχες οντολογικές σχέσεις. Μια λύση θα μπορούσε να ήταν η δημιουργία κάποιου προσαρμοσμένου parser για τις προτάσεις των χρηστών ώστε να γίνεται καλύτερη και πιο ακριβής αντιστοίχιση στο μετέπειτα στάδιο της ανάλυσης.

Θα πρέπει ακόμη να δοθεί έμφαση στον τρόπο με τον οποίο θα μπορούσε το σύστημα να διαχειριστεί καλύτερα την πολυγλωσσία. Εδώ θα πρέπει να αναπτυχθεί και κάποιος τρόπος με τον οποίο θα μπορούσαν να εμπουτιστούν τα αποτελέσματα άλλων γλωσσών μέσω κάποιας μετάφρασης ή κάποιας

βιβλιοθήκης για συνώνυμα όπως χρησιμοποιήθηκε στην παρούσα διπλωματική για τα Αγγλικά.

Έμφαση θα πρέπει να δοθεί και σε έναν τρόπο για να λυθούν οι ασάφειες που πολλές φορές δημιουργούνται στα αποτελέσματα και ποιός είναι ο καλύτερος τρόπος για να επιλυθούν π.χ. με ερωτήσεις στους χρήστες, όπως κάνουν πολλές παρόμοιες υλοποιήσεις.

Τέλος θα πρέπει να διερευνηθεί πως θα μπορούσε να γίνει ανάλυση παραπάνω από μια προτάσεων των οποίων τα νοήματα αλληλοεξαρτώνται και επίσης προτάσεων που να έχουν πολλά υποκείμενα με πολλές διαφορετικές ιδιότητες και σχέσεις μεταξύ τους. Οι σχέσεις αυτές θα μπορούσαν να ήταν και μεγαλύτερης τάξης από δυο πράγμα που σε αυτή την διπλωματική δεν αναπτύχθηκε.

Παράρτημα Α

Τεχνολογίες και βιβλιοθήκες ανάπτυξης

Στο παράρτημα αυτό παρουσιάζονται οι τεχνολογίες τα οποία χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής καθώς επίσης και οι βιβλιοθήκες που χρησιμοποιήθηκαν για την γλώσσα Python.

Α.1 Τεχνολογίες

Α.1.1 Python

Η γλώσσα προγραμματισμού Python δημιουργήθηκε το 1991 και αποτελεί μια γλώσσα υψηλού επιπέδου και γενικού σκοπού. Χρησιμοποιείται σε διάφορους τομείς προγραμματισμού, αφού προσφέρει αντικειμενοστράφεια, συναρτησιακό και προστατικό προγραμματισμό, με κυριότερους την τεχνητή μάθηση, τον δικτυακό προγραμματισμό και μαθηματικές εφαρμογές.

Α.1.2 HTML: Hypertext Markup Language

Αποτελεί την βασική γλώσσα σήμανσης που χρησιμοποιείτε στους φυλλομετρητές ιστού για την αναπαράσταση ιστοσελίδων. Μπορεί να χρησιμοποιηθεί παράλληλα με τεχνολογίες όπως τα Cascading Style Sheets(CSS) και με scripting γλώσσες όπως η Javascript.

Α.1.3 Web API

Ένα Web API είναι μια δικτυακή εφαρμογή που προσφέρει διευθύνσεις, στις οποίες γίνεται επικοινωνία μέσω μηνυμάτων τύπου αίτησης - απάντησης με τρίτες εφαρμογές. Μέσω αυτών των αιτημάτων μπορεί η εφαρμογή να πραγματοποιεί εργασίες στο σέρβερ στον οποίο φιλοξενείται, όπως σύνδεσεις με βάσεις δεδομένων ή απλώς τροποποίηση των δεδομένων του αιτήματος και επιστροφή κατάλληλης απάντησης. Οι απαντήσεις ενός Web API είναι συνήθως σε μορφή JSON(Α.1.5)

Α.1.4 HTTP

HTTP είναι ένα πρωτόκολλο επικοινωνίας στο διαδίκτυο με το οποίο γίνεται μεταφορά δεδομένων μεταξύ μηνυμάτων τύπου αίτησης - απάντησης. Αποτελεί την βάση της επικοινωνίας στον παγκόσμιο ιστό και δημιουργήθηκε πρώτη φορά το 1989.

Http Verbs/Methods

Το πρωτόκολλο HTTP ορίζει διάφορες μεθόδους(methods) που πολλές φορές ονομάζονται και ρήματα(verbs) με τα οποία δηλώνεται το είδος επικοινωνίας που επιθυμεί να κάνει η εφαρμογή απο-

στολέα στην εφαρμογή δέκτη. Το ρήμα που έχει χρησιμοποιηθεί στην παρούσα διπλωματική είναι το παρακάτω:

- **POST**: Το ρήμα κυρίως αναφέρεται στην δημιουργία κάποιου πόρου στην εφαρμογή. Στην δική μας εφαρμογή χρησιμοποιείται για να μεταφέρονται οι απαραίτητες πληροφορίες για την δημιουργία της οντολογικής σχέσης.

Http Statuses

Το πρωτόκολλο HTTP στην απάντηση του επιστρέφει και έναν κωδικό κατάστασης(status code), ο οποίος χαρακτηρίζει το είδος της απάντησης, και τον οποίο χρησιμοποιεί η εφαρμογή του χρήστη για να διαχειριστή διαφορετικά την απάντηση. Υπαρχουν διάφορες ομάδες κωδικών αλλά εμάς μας ενδιαφέρουν η κατηγορία 2XX και η κατηγορία 4XX. Πιο συγκεκριμένα έχουν αναφερθεί οι παρακάτω κωδικοί κατάστασης στην διπλωματική:

- **200(OK)**: Με τον κωδικό αυτό δηλώνουμε ότι η αίτηση διεκπεραιώθηκε με επιτυχία.
- **400(Bad Request)**: Υπάρχει πρόβλημα με τα δεδομένα που έχει δώσει η εφαρμογή του χρήστη.
- **500(Internal Server Error)**: Υπήρξε εσωτερικό πρόβλημα στην εφαρμογή.

A.1.5 JSON

JSON(Javascript Object Notation) είναι μια ανοιχτή μορφή αρχείου που χρησιμοποιεί ευανάγνωστο κείμενο για την μεταφορά δεδομένων με την μορφή ιδιότητα - τιμή. Χρησιμοποιείται κατά κύριο λόγο στην μεταφορά δεδομένων μεταξύ διαδικτυακών εφαρμογών. Ένα παράδειγμα JSON είναι το παρακάτω, στο οποίο παρουσιάζονται όλα τα χαρακτηριστικά τιμών που μπορεί να υποστηριχθούν:

```
{
  "First Name": "George",
  "Last Name": "Smith",
  "Alive": true,
  "Age": 27,
  "Address": {
    "Address": "462 Illinois Avenue",
    "City": "Palm Coast",
    "State": "FL",
    "PostalCode": "32137"
  },
  "Phone Numbers": [
    {
      "Type": "Home",
      "Number": "321 888−4567"
    },
    {
```

```
      "Type": "Work",
      "Number": "646 999−2344"
    }
  ],
  "Children": [],
  "Spouse": null
}
```

Κώδικας Α.1: Παράδειγμα JSON

Α.1.6 Microsoft Azure

Αποτελεί την cloud λύση της Microsoft με πάνω από 600 διαφορετικές υπηρεσίες για την κατασκευή, τον έλεγχο, την ανάπτυξη και την διαχείριση εφαρμογών και υπηρεσιών μέσα από τα κέντρα δεδομένων της.

Α.1.7 Hashing

Οι μέθοδοι hashing ή στα ελληνικά μέθοδοι κατακερματισμού είναι μέθοδοι με τις οποίες δεδομένα οποιδήποτε μεγέθους μπορούν να αντιστοιχηθούν με δεδομένα συγκεκριμένου μεγέθους. Υπάρχουν πολλές μέθοδοι κατακερματισμού και στην συγκεκριμένη διπλωματική χρησιμοποιήσαμε τον SHA-256 που είναι ένας κρυπτογραφικός αλγόριθμος κατακερματισμού.

Α.2 Βιβλιοθήκες που χρησιμοποιήθηκαν

Α.2.1 Owlready2

Βιβλιοθήκη για την ανάγνωση οντολογιών. Υποστηρίζει OWL 2.0 και οντολογίες γραμμένες σε RDF και μπορεί να τις διαβάσει, τροποποιήσει και αποθηκεύσει. Δίνει πρόσβαση στις κλάσεις, σχέσεις και άτομα της οντολογίας που φορτώνει καθώς επίσης και στα σχόλια σε όσες γλώσσες είναι αυτά γραμμένα.

Α.2.2 Spacy

Είναι μια βιβλιοθήκη για Επεξεργασία Φυσικής Γλώσσας(Natural Language Processing) για την Python που προσφέρει dependency and constituency parse με πολύ καλή ακρίβεια σε πολλές γλώσσες με ήδη έτοιμα εκπαιδευμένα μοντέλα.

Α.2.3 StanfordNLP

Είναι και αυτή μια βιβλιοθήκη για Επεξεργασία Φυσικής Γλώσσας που έχει αναπτυχθεί από το Stanford και είναι γραμμένη σε Java.[10]

A.2.4 Nltk

Βιβλιοθήκη για επεξεργασία δεδομένων φυσικής γλώσσας που προσφέρει πρόσβαση στο WordNet και άλλες χρήσιμες βιβλιοθήκες επεξεργασία κειμένου[2].

A.2.5 WordNet

Το WordNet[5] είναι μια λεξιλογική βάση δεδομένων στα Αγγλικά, η οποία περιέχει ομαδοποιημένα συνώνυμα(synsets) από ουσιαστικά, επίθετα, ρήματα και επιρρήματα που συνδέονται μεταξύ τους με εννοιολογικές σημασίες και λεξιλογικές σχέσεις.

A.2.6 Pattern

Μια ακόμα βιβλιοθήκη που προσφέρει πρόσβαση στο WordNet, αλλά έχει πιο εύκολη πρόσβαση σε βασικές σχέσεις λέξεων όπως τα υπερώνυμα και προσφέρει επίσης και έλεγχο ορθογραφίας μέσω του αλγορίθμου του Peter Norvig[12], όπως έχει ήδη αναφερθεί.

A.2.7 Flask

Είναι ένα microframework για την Python που χρησιμοποιείται για την δημιουργία διαδικτυακών εφαρμογών με εύκολο και γρήγορο τρόπο. Είναι ιδανική βιβλιοθήκη για μικρές εφαρμογές που δεν απαιτούν περίπλοκες διασυνδέσεις με βάσεις δεδομένων και είναι αρκετά επεκτάσιμο.

Βιβλιογραφία

[1] Swartz A. Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.

[2] Edward Loper Bird, Steven and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[3] Cunningham Hamish Damljanovic Danica, Agatonovic Milan. Freya: An interactive way of querying linked data using natural language. *The Semantic Web: ESWC 2011 Workshops*, 7117:125–138, 2011.

[4] Chris Nowak Dominique Estival and Andrew Zschorn. Towards ontology-based natural language processing. Technical report, Defence Science and Technology Organisation, PO Box 1500, Edinburgh SA 5111 Australia, 2004.

[5] Miller GA. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[6] Sundheim B. Grishman R. Message understanding conference - 6: A brief history. *Proceedings of the 16th conference on Computational linguistics*, 1:466–471, 1996.

[7] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[8] Miloslav Konopík Ivan Habernal. Swsnl: Semantic web search using natural language. *Expert Systems with Applications*, 40:3649–3664, 2013.

[9] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[10] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[11] Miguel A. Rodriguez-Garcia Mario A. Peredes-Valverde, Rafael Valencia-Garcia, Ricardo Colomo-Palacios, and Giner Alor-Hernandez. A semantic-based approach for querying linked data using natural language. *Journal of Information Science*, 42(6):851–862, 2016.

[12] P. Norvig. How to write a spelling corrector. `http://norvig.com/spell-correct.html`. [Online, accessed Feb 2007].

[13] Jorg Heizmann Philipp Cimianoa, Peter Haasea, Matthias Mantel, and Rudi Studer. Towards portable natural language interfaces to knowledge bases – the case of the orakel system. *Data and Knowledge Engineering*, 65:325–354, 2008.

[14] Seaborne A et al. Prud'Hommeaux E. Sparql query language for rdf. *W3C Recommendation*, 15, 2008.

[15] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: an Introduction*. MIT Press., 1998.

[16] Dieter Fensel Stefan Decker, Michael Erdmann and Rudi Studer. *Ontobroker: ontology based access to distributed and semi-structured information*, page 351–369. 1999.

[17] Cynthia A. Thompson, Raymond J. Mooney, and Lappoon R. Tang, editors. *Learning to Parse Natural Language Database Queries into Logical Form*, Austin, TX 78712 - 1188, 7 1997. University of Texas, ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Aquicision.