ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

Ευφυή Συστήματα

Οντολογίες και εφαρμογές υγείας: Μια μελέτη περίπτωσης για τη διαχείριση ασθενών με απώλεια μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ιωάννη Ηγουμενάκη

Επιβλέπων : Γκουμόπουλος Χρήστος
Μέλη εξεταστικής επιτροπής:

Σάμος, Φεβρουάριος 2020

Πρόλογος και ευχαριστίες

Με την περάτωση της παρούσας διπλωματικής εργασίας αισθάνομαι ευτυχής που συνέβαλλα, έστω και λίγο με την υλοποίηση λογισμικού, στην αντιμετώπιση των ανοιών όπως το Αλτσχάιμερ.

Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή κύριο Χρήστο Γκουμόπουλο για την ευκαιρία που μου έδωσε να εργαστώ πάνω σε ενδιαφέρουσες τεχνολογίες για έναν ευγενή σκοπό. Επίσης θα ήθελα να τον ευχαριστήσω για την εμπιστοσύνη που μου έδειξε κατά την εκπόνηση της εργασίας, καθώς και για τις εύστοχες συμβουλές και υποδείξεις του.

Επίσης θα ήθελα να ευχαριστήσω τους γονείς μου, Χαρίδημο και Ανδριάνα για την συνεχή συμπαράσταση τους καθ' όλη τη διάρκεια των σπουδών και για την καθοδήγησή τους.

Τέλος θα ήθελα να ευχαριστήσω την κοπέλα μου, Inês Araújo. Η συνεχή υπομονή της και επιμονή της καθώς και η βοήθεια της τόσο πνευματική όσο και στις καθημερινές δουλείες, αποτέλεσαν σημαντικό αρωγό για την εκπόνηση της εργασίας. Επίσης θα ήθελα να την ευχαριστήσω για τις ιδέες και συμβουλές που μου έδωσε σχετικά με την σύνταξη του κειμένου.

Ιωάννη Ηγουμενάκη

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

Πίνακας περιεχομένων

Λίστα Σχημάτων

Λίστα Πινάκων

Περίληψη

Η άνοια είναι μία από τις ανίατες ασθένειες. Δρα μέσω του εκφυλισμού του εγκεφάλου με αποτέλεσμα την αποδυνάμωση των γνωστικών λειτουργιών του ασθενούς. Με το πέρασμα των χρόνων έχουν αναπτυχθεί διάφορα παίγνια, τα οποία επιβραδύνουν τα συμπτώματα της ασθένειας. Η ανάπτυξη ενός τέτοιου παιγνίου είναι χρονοβόρα καθώς θα πρέπει να εδραιωθούν οι κανόνες και στη συνέχεια να δημιουργηθεί το κατάλληλο οπτικοακουστικό υλικό. Η διαχείριση τέτοιου υλικού με παραδοσιακές μεθόδους (χαρτί και μολύβι) δεν είναι εύκολη και αρκετές φορές δημιουργείται η ανάγκη για την αναδημιουργία τους λόγω φθορών. Επίσης, η εξάσκηση του ασθενούς επιβάλει την επίσκεψη σε κάποια κλινική ώστε να αξιολογηθούν οι γνωστικές λειτουργίες του.

Η παρούσα εργασία εξετάζει ως μελέτη περίπτωσης μια εφαρμογή υγείας, που αυτόματα δημιουργεί στιγμιότυπα τέτοιων παιγνίων. Η εφαρμογή επιτρέπει την μοντελοποίηση των κανόνων που διέπουν τη λειτουργία των παιχνιδιών χρησιμοποιώντας μία οντολογία. Η οντολογία περιγράφει τους κανόνες κάθε παίγνιου και τα υλικά που χρειάζονται για το εκάστοτε παίγνιο. Επίσης έγινε ανάπτυξη λογισμικού το οποίο δημιουργεί αυτόματα τέτοια στιγμιότυπα χρησιμοποιώντας τους κανόνες της οντολογίας. Τέλος, η λειτουργικότητα αυτή είναι προσβάσιμη από πιθανές εφαρμογές πελάτη (clients) μέσω του πρωτοκόλλου HTTP.

Για την ανάπτυξη της οντολογίας χρησιμοποιήθηκαν δύο εφαρμογές Android οι οποίες παρέχουν τέτοια παίγνια. Οι εφαρμογές αυτές είχαν πεπερασμένο πλήθος επιπέδων και μικρό αριθμό στιγμιότυπων για κάθε είδος παιγνίου. Το πρώτο βήμα ήταν η μελέτη των διαθέσιμων παιγνίων των εφαρμογών αυτών, με σκοπό την ανίχνευση των κανόνων. Το δεύτερο βήμα ήταν η δημιουργία της οντολογίας ώστε να μοντελοποιηθούν αυτοί οι κανόνες. Σαν τρίτο και τελευταίο βήμα ήταν η ανάπτυξη ενός λογισμικού το οποίο με την χρήση της οντολογίας δημιουργεί νέα στιγμιότυπα που μπορούν να αξιοποιηθούν από εφαρμογές παιχνιδιών.

Λέξεις Κλειδιά: *Οντολογία, OWL, SPARQL, Knowledge Base, Protégé, Spring Framework, Rest API, Alzheimer*

Abstract

Dementia is one of the incurable diseases. It acts through the degeneration of the brain, resulting in the weakening of the patient's cognitive functions. Through time a lot of games have been developed, that slow down the symptoms of the disease. The development of such a game is time consuming, since first the rules must be established and then the media that are used to be created. The management of these media, when created with the traditional means (pen and paper) is not easy and often there is the need of their recreation because of natural decay. Moreover, the training of the patient makes mandatory a visit to a local clinic, where his cognitive functions could be evaluated.

This thesis is studying the case study of a health application, which automatically creates instances of such games. The application allows the modeling of the game rules by using an ontology. The ontology is describing the rules of each game and the media that are required by those games. Moreover, there has been implemented a software that reads the rules in the ontology and creates new instances of the games. Finally, this functionality is accessible to potential client – applications through HTTP.

For the development of the ontology, two Android application were used, which are providing games against dementia. These applications contain a finite number of difficulty levels and a small number of game instances. The first step was to study these game instances in order to identify and describe the rules. The second step was the creation of the ontology in accordance of these rules. The third and last step was the implementation of the application that with the help of the ontology creates new game instances that can be used by game applications, like these two Android apps.

Λέξεις Κλειδιά: *Ontology, OWL, SPARQL, Knowledge Base, Protégé, Spring Framework, Rest API, Alzheimer*

1

Εισαγωγή

1.1 Πλαίσιο Εργασίας

Άνοια είναι η επιδείνωση των γνωστικών και συναισθηματικών λειτουργειών ενός ατόμου. Το Alzheimer είναι ένα είδος άνοιας και εμφανίζεται κυρίως σε άτομα μεγάλης ηλικίας, με μεγαλύτερο ποσοστό από τις υπόλοιπες γεροντικές άνοιες. Το Alzheimer έχει ως αποτέλεσμα τον εκφυλισμό του εγκεφάλου, ο οποίος οδηγεί στην νέκρωση εγκεφαλικών κυττάρων (Papatheodoropoulos, 2015). Γενικά τα άτομα τα οποία πάσχουν από άνοια παρουσιάζουν επιδείνωση στη μνήμη, η οποία οδηγεί στην γνωστική ανεπάρκεια καθώς και σε δυσκολία συγκράτησης νέας πληροφορίας. Επίσης οι ασθενείς αυτοί παρουσιάζουν αυξημένη επιθετικότητα και βρίσκονται σε μία συναισθηματική πίεση καθώς τα συμπτώματα της ασθένειας οδηγούν σε αλλαγή προσωπικότητας (Gustafson, 1996).

Πειραματικά αποτελέσματα δείχνουν ότι η νοητική καθώς και η σωματική εξάσκηση βοηθούν στην ανάπτυξη νέων εγκεφαλικών νευρώνων, οι οποίοι μειώνουν τις επιπτώσεις των νόσων άνοιας (Papatheodoropoulos, 2015). Έρευνες έχουν δείξει ότι η νοητική εξάσκηση πρέπει να στοχεύει σε συγκεκριμένες εγκεφαλικές δραστηριότητες (Tong, Chan, & Chignell, 2017). Συγκεκριμένα οι δραστηριότητες αυτές είναι η βραχυπρόθεσμη (νέα πληροφορία) και η μακροπρόθεσμη

(διατήρηση πληροφορίας) μνήμη, η εναλλαγή μεταξύ διαφόρων εργασιών, η αναγνώριση λέξεων ή αντικειμένων και τέλος η χρονολογική και χωρική τοποθέτηση.

Τα τελευταία χρόνια με τη βοήθεια της τεχνολογίας έχουν μελετηθεί και δημιουργηθεί μια πληθώρα ηλεκτρονικών παιγνίων, τα οποία αποσκοπούν στην εξάσκηση των παραπάνω εγκεφαλικών δραστηριοτήτων [(Luciano, και συν., 2006), (Green & Bavelier, 2004), (Garcia Marin, Navarro, & Lawrence, 2011)]. Μία εκτενής λίστα αυτών μπορεί να βρεθεί στην αναφορά των (McCallum & Boletsis, 2013). Στη λίστα αυτή υπάρχουν παίγνια τα οποία χρησιμοποιούν μέσα όπως το Kinect, τα οποία αποσκοπούν επίσης στη σωματική εξάσκηση (Chartomatsidis & Goumopoulos, 2019).

Τα παραπάνω παίγνια καθώς αποτελούν ένα τελικό προϊόν, δεν μπορούν να προσαρμοστούν εύκολα ανάλογα με τις ανάγκες του χρήστη. Επιπροσθέτως, οι παραδοσιακές ασκήσεις, που προσφέρονται από το κλινικό προσωπικό στους ασθενείς, δεν έχουν μεγάλη ποικιλομορφία. Επίσης υπάρχει μια διαδικασία παραγωγής των αντικειμένων, που χρησιμοποιούνται από την άσκηση, όπως για παράδειγμα ένα παζλ. Για την αντιμετώπιση αυτής της δυσχέρειας οι (Quaglini, και συν., 2009) προτείνουν τη χρήση μιας βάσης δεδομένων και ενός γραφικού περιβάλλοντος που επιτρέπουν τη δημιουργία νέων παιγνίων και των αντικειμένων τους με έμφαση στην προσαρμογή τους ανάλογα με τον χρήστη. Μια άλλη έρευνα των (Leonardi, Panzarasa, & Quaglini, 2011) προτείνει την αυτόματη δημιουργία τέτοιων παιγνίων με την χρήση μίας οντολογίας, όπου περιγράφει τα παίγνια και τα αντικείμενα αυτών. Ο συνδυασμός των δύο αυτών ερευνών περιγράφεται από τους (Alloni, και συν., 2017), οι οποίοι δημιούργησαν ένα λογισμικό το οποίο δημιουργεί παίγνια και τα αντίστοιχα αντικείμενά τους σε μία οντολογία.

1.2 Στόχος Εργασίας

Όπως αναφέρθηκε η δημιουργία παιγνίων, τα οποία αποσκοπούν στην νοητική εξάσκηση καθίσταται χρονοβόρα διαδικασία. Επίσης η παραγωγή των αντικειμένων κάθε παιγνίου προσθέτει φόρτο εργασίας για τη σωστή διαχείριση των πόρων αυτών και ανανέωσής τους με πιθανές νεότερες εκδόσεις. Η δημιουργία μίας πλατφόρμας, που να διαχειρίζεται τα παραπάνω και να μπορεί να δημιουργεί αυτόματα στιγμιότυπα παιγνίων ανάλογα των αναγκών του τελικού χρήστη, αποτελεί την ιδανική λύση. Μέχρι στιγμής υπάρχουν κάποια προϊόντα τα οποία εξασφαλίζουν, αν όχι όλες, τις περισσότερες από τις προδιαγραφές αυτές. Ονομαστικά κάποια από αυτά είναι το CoRE (Alloni, et al., 2017) και το E-Prime (n.d.). Τα δύο αυτά προϊόντα ενώ καλύπτουν ένα μεγάλο φάσμα αναγκών, δεν καλύπτουν την περίπτωση της δημιουργίας παιγνίων on-demand από κάποιο πιθανό πρόγραμμα πελάτη.

Η παρούσα εργασία αποσκοπεί στην μελέτη, στον σχεδιασμό και στην υλοποίηση μιας λύσης για την αυτοματοποιημένη δημιουργία τέτοιων παιγνίων. Οι κανόνες των παιγνίων περιγράφονται σε μία οντολογία, η οποία επιτρέπει την μεταβολή των κανόνων ανάλογα με την πρόοδο του χρήστη ή με την παρεμβολή κάποιου ιατρικού προσωπικού. Επίσης, τα στιγμιότυπα των παιγνίων και των

αντικειμένων που τα απαρτίζουν εγγράφονται και αυτά σε μία οντολογία, ώστε οι μεταξύ τους συσχετίσεις να είναι καταγεγραμμένες. Τέλος, γίνεται δυνατή η χρήση της λειτουργικότητας αυτής μέσω του πρωτοκόλλου HTTP σε πιθανούς χρήστες οι οποίοι μπορούν να βρίσκονται στον προσωπικό τους χώρο και όχι σε κάποια κλινική.

1.3 Μεθοδολογική Προσέγγιση

Για την επίτευξη του στόχου πρώτα έπρεπε να γίνει αντιληπτή η ανάγκη για την αυτοματοποίηση και την πρόσβαση της δημιουργίας τέτοιων παιγνίων μέσω του διαδικτύου.

Ως πρώτο βήμα μελετήθηκε γιατί τα παίγνια αυτά μπορούν και επιβραδύνουν τις διάφορες μορφές της άνοιας με έμφαση στην ήπια γνωστική εξασθένιση. Στη συνέχεια μελετήθηκαν τα είδη των παιγνίων αυτών και ποιες εγκεφαλικές διεργασίες επηρεάζουν.

Το επόμενο βήμα ήταν η μελέτη υπαρχόντων περιπτώσεων τέτοιων παιγνίων, ή εφαρμογών που περιέχουν τέτοια παίγνια. Συγκεκριμένα, μελετήθηκαν εκτενέστερα δύο διαφορετικές εφαρμογές αναπτυγμένες για συσκευές Android, οι οποίες περιέχουν ένα σύνολο τέτοιων παιγνίων. Κάθε εφαρμογή έχει διαφορετικές κατηγορίες παιγνίων όπου κάθε κατηγορία στοχεύει σε συγκεκριμένη εγκεφαλική δραστηριότητα. Στη συνέχεια αναλύθηκαν και συγκρίθηκαν τα παίγνια τα οποία έχουν κοινά αντικείμενα για την υλοποίησή τους, για παράδειγμα όλα τα παίγνια που χρησιμοποιούν εικόνες. Οι δύο αυτές διαφορετικές αναλύσεις αποσκοπούν στην εξαγωγή χαρακτηριστικών των παιγνίων με βάση την εγκεφαλική δραστηριότητα και ταυτόχρονα τα χαρακτηριστικά των συνολικών αντικειμένων που χρησιμοποιούνται.

Ως τρίτο βήμα αποτέλεσε η υλοποίηση της οντολογίας που περιέχει τους κανόνες που βρέθηκαν από τη μελέτη των εφαρμογών. Εκτός των κανόνων των παιγνίων, παρατηρήθηκε ότι και τα αντικείμενα τα οποία συμμετάσχουν στο κάθε παίγνιο έχουν κάποιους κανόνες, για παράδειγμα οι συνώνυμες λέξεις ή ο αριθμός των γειτονικών κομματιών ενός τυχαίου κομματιού του παζλ. Στο βήμα αυτό μοντελοποιήθηκαν επίσης οι κανόνες των αντικειμένων.

Σαν τελευταίο βήμα έγινε η υλοποίηση μίας εφαρμογής, η οποία κάνει χρήση της οντολογία που αναπτύχθηκε στο τρίτο βήμα, με σκοπό την δημιουργία στιγμιότυπων των παιγνίων. Επίσης αναπτύχθηκε ένα REST API, το οποίο παρέχει τις δυνατότητες δημιουργίας και λύσης ενός στιγμιότυπου. Με τον τρόπο αυτό ένας client μπορεί να ζητήσει την λίστα των ήδη υπάρχοντών στιγμιότυπων, την δημιουργία ενός στιγμιότυπου ή να στείλει την λύση ενός στιγμιότυπου. Η επικοινωνία γίνεται μέσω HTTP πρωτοκόλλου χρησιμοποιώντας τα HTTP Verbs όπως έχουν περιγραφεί στο Παράρτημα.

1.4 Δομή Εργασίας

Το παρόν κείμενο περιέχει πέντε κύριες ενότητες, οι οποίες περιγράφουν τις τεχνολογίες που χρησιμοποιήθηκαν, την μεθοδολογία για την δημιουργία των κανόνων και τέλος την μελλοντική μελέτη που μπορεί να γίνει.

Το **Κεφάλαιο 2** αναλύει την έννοια της οντολογίας και τα τμήματα από τα οποία αυτή απαρτίζεται. Γίνεται μελέτη της Περιγραφικής Λογικής και πως αυτή συμβάλει στη σύλληψη μίας εκφραστικής γλώσσας, που ονομάζεται OWL, για τη δημιουργία της οντολογίας. Επίσης μελετώνται ο Σημασιολογικός Ιστός και οι τεχνολογίες που τον απαρτίζουν και ο τρόπος με τον οποίο οι τεχνολογίες αυτές επηρέασαν με την σειρά τους την OWL. Επιπλέον στο κεφάλαιο αυτό μελετάται ο τρόπος με τον οποίο ελέγχεται η ορθότητα μίας οντολογίας και τα SPARQL ερωτήματα, με τα οποία δημιουργούνται ερωτήσεις στην OWL. Επιπλέον, περιγράφει αναλυτικά τη χρήση microservices τα οποία θα χρησιμοποιηθούν για τη δημιουργία των παιγνίων.

Το **Κεφάλαιο 3** περιγράφει τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της λύσης αξιοποιώντας τις τεχνολογίες που μελετήθηκαν στο Κεφάλαιο 2.

Το **Κεφάλαιο 4** περιγράφει αναλυτικά την μεθοδολογία που ακολουθήθηκε για την δημιουργία των κανόνων για τα παίγνια και τα αντικείμενά τους που συνδυαστικά απαρτίζουν την οντολογία MCI.

Το **Κεφάλαιο 5** εξηγεί πως είναι εφικτή η κατανάλωση των κανόνων αυτών μέσα από την εφαρμογή χρησιμοποιώντας το framework Apache Jena. Τέλος, περιγράφει ως παράδειγμα ένα σενάριο δημιουργίας ενός παιγνίου που μπορεί να χρησιμοποιηθεί από την εφαρμογή.

Το **Κεφάλαιο 6** περιλαμβάνει τα συμπεράσματα και αναφέρει πιθανές μελλοντικές εργασίες που μπορούν να γίνουν ώστε να επεκταθεί περισσότερο η πλατφόρμα.

2

Θεωρητικό και Τεχνολογικό Υπόβαθρο

2.1 Εισαγωγή

Τα τελευταία χρόνια έχει παρατηρηθεί μεγάλη ανάπτυξη της χρήσης των υπολογιστών ή των έξυπνων συσκευών (Goumopoulos & Kameas, 2009), (Goumopoulos, Papa, & Stavrianos, 2017). Η δυνατότητα σύνδεσης των συσκευών αυτών στο Διαδίκτυο επέφερε την δημιουργία εφαρμογών οι οποίες μπορούν να αποκτούν πληροφορίες ανά πάσα ώρα και στιγμή (Gavalas, et al., 2017). Τέτοιες συσκευές αποτελούν το ιδανικό μέσο για την γνωστική ενδυνάμωση των ατόμων που πάσχουν από άνοια. Οι ασθενείς μπορούν άμεσα και γρήγορα να έχουν πρόσβαση στα αντίστοιχα παίγνια χωρίς να είναι αναγκαία η μετάβαση τους σε κάποια κλινική.

Κάθε παίγνιο έχει ορισμένους κανόνες, οι οποίοι πρέπει να ακολουθηθούν από τον παίκτη. Ακόμη ανάλογα το επίπεδο του παίκτη οι κανόνες μπορούν να μεταβάλλονται με σκοπό τα παίγνια να γίνονται πιο ανταγωνιστικά. Όπως έχει αναφερθεί στην ενότητα 1.2 Στόχος Εργασίας, η χρήση οντολογίας αποτελεί ιδανική λύση για την μοντελοποίηση των κανόνων ενός παιγνίου και για την αποθήκευση αυτών.

Μία εφαρμογή που επικοινωνεί με μία οντολογία, θα πρέπει να υλοποιεί ένα REST API, ώστε να παρέχει τις δυνατότητες της οντολογίας σε απομακρυσμένους πελάτες.

Σ' αυτό το κεφάλαιο θα αναλυθεί η ορολογία της οντολογίας, από ποια μέρη αποτελείται και πως μπορεί να απαντήσει σε δύο βασικά ερωτήματα: α) πώς μπορεί να δημιουργηθεί ένα στιγμιότυπο παιχνιδιού και β) αν η απάντηση που προτείνει ο χρήστης είναι σωστή. Επίσης, θα μελετηθούν τρόποι με τους οποίους η οντολογία αυτή μπορεί να γίνει προσιτή μέσω του πρωτοκόλλου HTTP.

2.2 Οντολογία

Ο όρος οντολογία έχει ρίζες στην αρχαία Ελλάδα όπου προσπαθεί να ορίσει, τι οντότητες υπάρχουν σε έναν χώρο, ποια είναι η κατηγορία τους και πώς αλληλοεπιδρούν μεταξύ τους. Σήμερα στην επιστήμη των υπολογιστών ο όρος αυτός έχει τον ίδιο στόχο και υπό το πρίσμα της τεχνολογίας γίνεται προσπάθεια της μοντελοποίησης των οντοτήτων ώστε να μπορούν να χρησιμοποιηθούν από κάποια εφαρμογή (Poli & Obrst, 2010).

Για να γίνει σωστή αυτή η μοντελοποίηση, όλες οι οντότητες, συσχετίσεις ή άλλες έννοιες αναμένεται ότι υπάρχουν στον χώρο για να ορισθούν. Αυτό υποδηλώνει ότι όλες οι βάσεις γνώσεις στηρίζονται πάνω σε αυτή την αντίληψη. Από τα παραπάνω ο (Gruber, 1993) ορίζει ως οντολογία τη ρητή προδιαγραφή μίας τέτοιας αντίληψης. Η προδιαγραφή αυτή πρέπει να μοντελοποιείται σε κάποια κατανοητή γλώσσα τόσο από τον άνθρωπο όσο και από την μηχανή (Baader, Horrocks, & Sattler, 2010).

Ο παραπάνω ορισμός είναι πολύ αφηρημένος, το οποίο επιτρέπει την δημιουργία μιας οντολογίας που να απαρτίζεται από πολλές οντότητες και πολλές συσχετίσεις. Η προσθήκη νέων οντοτήτων ή νέων συσχετίσεων σε μια οντολογία μπορεί να δημιουργήσει προβλήματα λογικής σε μία οντολογία καθιστώντας την μη χρήσιμη. Για παράδειγμα, έστω οι οντότητες Α, Β συσχετίζονται ως εξής, *A hasAncestor B.* Αν στη συνέχεια δημιουργηθεί μία νέα οντότητα C και συσχετίζεται ως εξής με τις υπόλοιπες οντότητες *B hasAncestor C* , *C hasAncestor A,* τότε δημιουργείται ασυνέχεια στην οντολογία καθώς η C δεν μπορεί να συσχετίζεται με την Α με την συσχέτιση *hasAncestor*. Αντιθέτως από το παραπάνω παράδειγμα χρησιμοποιώντας την ανθρώπινη λογική διαπιστώνεται εύκολα ότι η οντότητα Α *hasAncestor C*.

Από το παραπάνω παράδειγμα προκύπτει η ανάγκη της ύπαρξης ενός συντακτικού το οποίο θα επιτρέπει για τον έλεγχο της ορθότητας μίας οντολογίας. Επίσης το συντακτικό αυτό θα πρέπει να επιτρέπει την εύκολη αναπαράσταση της λογικής ώστε να είναι δυνατή η εύρεση οντοτήτων ή συσχετίσεων που δεν έχουν περιγραφεί στην οντολογία. Αυτό στο παραπάνω παράδειγμα είναι ότι οι οντότητες Α και C εννοούνται ότι συσχετίζονται μέσω της συσχέτισης *hasAncestor*. Η ύπαρξη τέτοιων συντακτικών είναι απόρροια μελέτης και έρευνας στον κλάδο της τεχνητής νοημοσύνης και πιο συγκεκριμένα στον χώρο της αναπαράστασης γνώσης, στον οποίο αναφέρονται ως **περιγραφική λογική** (description logic) (Nardi & Brachman, 2003).

2.3 Περιγραφική Λογική (Description Logic)

Οι γλώσσες **περιγραφικής λογικής** είναι μία οικογένεια γλωσσών αναπαράστασης της γνώσης, που χρησιμοποιούνται για να αναπαραστήσουν την γνώση ενός χώρου κάποιας εφαρμογής με δομημένο και εύληπτο τρόπο (Baader, Horrocks, & Sattler, 2010). Οι γλώσσες αυτές χρησιμοποιούν έννοιες για να χαρακτηρίσουν τις οντότητες και ρόλους για να χαρακτηρίσουν πως οι έννοιες αυτές αλληλοεπιδρούν. Για παράδειγμα, αν θέλουμε να περιγράψουμε την έννοια του παιγνίου ενός πάζλ χρησιμοποιώντας περιγραφική λογική, τότε μπορούμε να πούμε ότι «το παζλ είναι παιχνίδι που έχει τουλάχιστον 4 κομμάτια και τα κομμάτια ενώνονται μεταξύ τους»:

$$Game \sqcap (>= 4\ hasPiece) \sqcap \exists\ hasPiece.Piece \sqcap \forall (Piece \sqcap \exists connectsWith.Piece)$$

Στο παράδειγμα αυτό έχουμε τα *Game* και *Piece* ως έννοιες, ενώ ως ρόλους τα *hasPiece* και *connectsWith*.

Χρησιμοποιώντας την **περιγραφική λογική** μπορούμε να σχηματίσουμε «ακρωνύμια» για τις περιγραφές με σκοπό την απλούστευση του συνόλου των εννοιών. Ένα ακρωνύμιο για το παραπάνω παράδειγμα είναι:

$$Puzzle \equiv Game \sqcap (>= 4\ hasPiece) \sqcap \exists\ hasPiece.Piece \sqcap \forall (Piece \sqcap \exists connectsWith.Piece)$$

Το σύνολο των εννοιών και των ρόλων σε μία τέτοια γλώσσα ονομάζεται **TBox**, ενώ το σύνολο των εννοιών και των ρόλων, τα οποία προκύπτουν μέσα από την λογική επαγωγή ενός **TBox**, ονομάζεται **ABox** (Baader, 2002). Γενικά ισχύει ότι οι έννοιες και οι ρόλοι, που ανήκουν σε κάποιο **TBox**, μαζί με τις λογικά επαγόμενες έννοιες και τους λογικά επαγόμενους ρόλους του **ABox**, μας δίνουν τη συνολική γνώση που είναι διαθέσιμη στον εφαρμοσμένο χώρο. Απλούστερα ισχύει ότι ***TBox + ABox = Knowledge Base.***

Όπως αναφέρθηκε υπάρχει μεγάλη ποικιλία γλωσσών στην περιγραφική λογική. Αυτό δε σημαίνει ότι κάθε γλώσσα είναι τελείως διαφορετική από μια άλλη, αλλά ότι κάθε γλώσσα μπορεί να χρησιμοποιεί διάφορες τεχνικές δημιουργίας μίας έννοιας (Baader & Nutt, 2003). Κάθε τέτοια τεχνική έχει συγκεκριμένη ονομασία και συμβάλει στην ονομασία της γλώσσας, η οποία τη χρησιμοποιεί. Η Εικόνα 1 αναπαριστά τις τεχνικές αυτές με τις αντίστοιχες ονομασίες τους. Στην παρούσα εργασία θα μελετηθούν μόνο οι γλώσσες, οι οποίες συμβάλλουν στην εκπόνηση της.

Επιστρέφοντας στην οντολογία, επόμενος στόχος είναι να βρεθεί μια περιγραφική λογική η οποία να έχει μεγάλη εκφραστικότητα. Αυτό σημαίνει ότι από ένα σταθερό **TBox**, να μπορεί να διαπιστωθεί όσο το δυνατό μεγαλύτερο **ABox**. Οι (Schmidt-Schauβ & Smolka, 1991) στην έρευνα τους αποδεικνύουν ότι η γλώσσα $\mathcal{ALC}$ έχει μεγάλη εκφραστικότητα, αλλά οι υλοποιήσεις αυτής της γλώσσας αντιμετωπίζουν προβλήματα διαστάσεων *PSPACE*, δηλαδή προβλήματα πολυωνιμικού χώρου τα οποία μπορούν να λυθούν με χρήση κάποιας μηχανής Turing. Η γλώσσα αυτή, για την περιγραφή των εννοιών και των ρόλων ενός χώρου, χρησιμοποιεί αρνήσεις, τομές

και καθολικούς δείκτες. Οι ενώσεις και οι υπαρξιακοί δείκτες μπορούν να αναπαρασταθούν ως αρνήσεις της τομής και του καθολικού δείκτη αντίστοιχα.

Μια επέκταση της γλώσσας αυτής αποδεικνύεται να προσφέρει μεγαλύτερη εκφραστικότητα είναι η $\mathcal{SHIQ}$. Η γλώσσα αυτή χρησιμοποιεί τις ίδιες εκφράσεις με την γλώσσα $\mathcal{ALC}$ και επίσης εκφράζει τους ρόλους ως μεταβατικούς (Sattler, 1996). Προκειμένου να χαρακτηριστούν οι ιδιότητες αυτές με ένα σύμβολο – γράμμα χρησιμοποιείται το S. Επιπροσθέτως, ως επέκταση της $\mathcal{ALC}$, χρησιμοποιούνται εκφράσεις που εκφράζουν τους ανάστροφους ρόλους ($\mathcal{I}$), ιεραρχίες αυτών($\mathcal{H}$) και τον αριθμό των περιορισμών ($\mathcal{Q}$). Στην έρευνα των (Horrocks, Sattler, & Tobies) αποδεικνύεται ότι η περιγραφική λογική $\mathcal{SHIQ}$ είναι ικανή να περιγράψει ένα μεγάλο μέγεθος **ABox**, για ένα συγκεκριμένου μεγέθους **TBox**.

Notation	Description
S	ALC Description logic extended with transitive roles i.e. ALCR+
H	Role hierarchies
O/B	Nominals
Q	Qualified number restriction
N	Unqualified number restriction
(D)	Data types
A	Atomic concept
AL	Top, bottom, intersection and value restriction
U	Union
C	Negation
E	Existential quantifier
F	Agreement and disagreement with equality for feature chains
f	Agreement and disagreement without equality for feature chains
I	Role constructor inverse
R+	at subscript-restriction that some roles are
H	Role hierarchy with single inheritance
R	Role conjunctions

Εικόνα 1 Σύμβολα Περιγραφικής Λογικής (Abburu, 2012)

2.4 Σημασιολογικός Ιστός

Στην ενότητα 2.3 Περιγραφική Λογική (Description Logic) αναφέρθηκε μια γλώσσα η οποία μπορεί να χρησιμοποιηθεί για να περιγράψει μία οντολογία. Η **Περιγραφική Λογική** επιβάλλει την εγκατάσταση της επιλεγμένης γλώσσας σε κάθε συσκευή που επιθυμείται η χρήση της. Με

την ευρεία ανάπτυξη του Internet έχει αναπτυχθεί νέες γλώσσες περιγραφής οντοτήτων. Οι γλώσσες αυτές απαρτίζουν τον **Σημασιολογικό Ιστό** και επιχειρούν την περιγραφή των media στις ιστοσελίδες και τι εκπροσωπούν. Ο Σημασιολογικός Ιστός έχει περιγραφεί ως «μία επέκταση του σημερινού Διαδικτύου, όπου η πληροφορία έχει ουσιαστικό νόημα και επιτρέπει τη συνεργασία υπολογιστών - ανθρώπων» (Berners-Lee, Hendler, & Lassila, 2001). Από τον ορισμό συμπεραίνεται ότι η πληροφορία που περιέχεται σε μία ιστοσελίδα θα πρέπει να εκφραστεί με κάποιον τρόπο ώστε να μπορεί να γίνει αντιληπτή τόσο από τον άνθρωπο, όσο και από κάποιο υπολογιστικό σύστημα. Σύντομα καταλήγουμε στο συμπέρασμα ότι πρέπει να βρεθεί μία «γλώσσα» η οποία να μπορεί να κάνει μία τέτοια περιγραφή. Υπάρχουν δύο βασικές γλώσσες που μπορούν να χρησιμοποιηθούν ώστε να επιτευχθεί η επιθυμητή αναπαράσταση της πληροφορίας όπως παρουσιάζεται στη συνέχεια.

2.4.1 XML

Η πρώτη γλώσσα ονομάζεται **eXtensible Markup Language (XML)** (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2008). Η γλώσσα αυτή βασίζεται σε ορισμένους κανόνες για την δημιουργία των οντοτήτων. Οι κανόνες αυτοί ορίζονται από το XML Schema Definition Language (XSD) (Gao, Sperberg-McQueen, & Thompson, 2012). Παρόλο που η γλώσσα αυτή έχει κανόνες για τη δημιουργία εννοιών-οντοτήτων, αποτυγχάνει να αναπαραστήσει ρόλους – συσχετίσεις που μπορεί να υπάρχουν μεταξύ των οντοτήτων. Για το λόγο αυτό η γλώσσα αυτή, από μόνη της, δεν είναι ιδανική για να περιγράψει μία οντολογία.

2.4.2 RDFS

Η δεύτερη γλώσσα ονομάζεται **Resource Description Framework (RDF)** (Gandon & Schreiber, 2014). Η γλώσσα αυτή είναι επέκταση της XML. Ως επέκταση επιτρέπει την περιγραφή εννοιών – οντοτήτων και των ρόλων – συσχετίσεων μεταξύ αυτών. Η περιγραφή αυτή γίνεται με τη χρήση της έννοιας της τριπλέτας (triple), η οποία έχει μια δομή παρόμοια με την σύνταξη προτάσεων στην ανθρώπινη γλώσσα, ως υποκείμενο – ιδιότητα – αντικείμενο (Berners-Lee, Hendler, & Lassila, 2001). Για παράδειγμα *Puzzle(Subject) hasPiece(property) Piece(object)*, η τριπλέτα αυτή εκφράζει ότι μία οντότητα *Puzzle* συσχετίζεται με την οντότητα *Piece* μέσω της σχέσης *hasPiece*. Η οντότητα *Puzzle* ορίζεται ως το **domain,** ενώ η οντότητα *Piece* ως το **range** της ιδιότητας *hasPiece.* Το **domain** μίας ιδιότητας σημαίνει ότι τα στιγμιότυπα των οντοτήτων που έχουν ορισθεί ως **domain** έχουν ορίσει την ιδιότητα αυτή, ενώ ως **range** είναι το σύνολο τιμών το οποίο μπορεί να συσχετιστεί η ιδιότητα, είτε στιγμιότυπα άλλων οντοτήτων είτε primitive τιμές όπως ένας ακέραιος. Καθώς η **RDF** περιγράφει οντότητες στον Ιστό, είναι πιθανό ο ορισμός μίας οντότητας να βρίσκεται σε διαφορετική τοποθεσία από τις υπόλοιπες οντότητες του ίδιου χώρου – εγγράφου. Προκειμένου να επιτευχθεί η εύκολη χρήση τους, η **RDF** επιτρέπει την χρήση ενός *URI* ως επέκταση του ονόματος μίας οντότητας. Αυτό επιτρέπει σε κάποιο υπολογιστικό σύστημα να «κατεβάσει» τους ορισμούς της οντότητας και να μην περιορίζεται μόνο στις τοπικές οντότητες.

Η χρήση των **RDF** τριπλέτων επιτρέπει την περιγραφή των οντοτήτων και των συσχετίσεων τους σε ένα χώρο. Επίσης, είδαμε ότι η **RDF** επιτρέπει την χρήση των URI με αποτέλεσμα να μπορούμε να χρησιμοποιούμε οντότητες που έχουν ορισθεί από άλλους φορείς. Στο σημείο αυτό δημιουργείται ένα πρόβλημα που αποτρέπει την **RDF** να χρησιμοποιηθεί ως γλώσσα μίας οντολογίας. Σε μία οντολογία, οι οντότητες και οι συσχετίσεις τους πρέπει να ακολουθούν μια συγκεκριμένη και προσυμφωνημένη δομή, ώστε ένα υπολογιστικό σύστημα να μπορεί να διακρίνει τις οντότητες, την ιεραρχία αυτών και τις συσχετίσεις τους. Αυτή η ανάγκη δημιούργησε το **RDF Schema (RDFS)** το οποίο αποτελεί μια επέκταση της **RDF**. Με το **RDFS** μπορούμε να χρησιμοποιήσουμε κάποιες ιδιότητες, οι οποίες προσφέρουν την αναγκαία πληροφορία ώστε να γίνει η παραπάνω διάκριση (Pan, 2010). Οι οντότητες – έννοιες χαρακτηρίζονται μέσω της ιδιότητας **rdf:type**, ενώ η δήλωση της ιεραρχίας μεταξύ οντοτήτων ή ιδιοτήτων γίνεται μέσω των **rdfs:subClassOf** και **rdfs:subPropertyOf** αντίστοιχα. Τέλος το **RDFS** επιτρέπει την δήλωση των οντοτήτων που μπορούν να συσχετιστούν με μία συγκεκριμένη ιδιότητα. Από το παραπάνω παράδειγμα, *Puzzle hasPiece Piece,* μπορούμε να πούμε ότι η ιδιότητα *hasPiece* έχει ως **domain** οντότητες της κλάσης *Puzzle* και ως **range** οντότητες της κλάσης *Piece*. Χρησιμοποιώντας το **RDFS** μπορούμε να περιγράψουμε το παράδειγμα αυτό ως εξής:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix mci: <http://example.org/Mci#>
mci:Game rdf:type rdfs:Class .
mci:Piece rdf:type rdfs:Class .
mci:Puzzle rdf:type rdfs:Class ; rdfs:subClassOf mci:Game .
mci:hasPiece rdf:type rdf:Property ; rdfs:domain mci:Puzzle ; rdfs:range mci:Piece .
```

Παρόλο που το **RDFS** επιτρέπει σε κάποιο υπολογιστικό σύστημα να κατανοήσει τις διαφορές μεταξύ οντοτήτων, πώς αυτές υπόκεινται σε μία ιεραρχία και πώς συσχετίζονται, δεν έχει την ίδια εκφραστικότητα με μία περιγραφική λογική όπως η $\mathcal{SHIQ}$.

2.5 OWL

Το επόμενο βήμα στην εύρεση μίας εκφραστικής γλώσσας που να μπορεί να περιγράφει πλήρως μία οντολογία είναι η δημιουργία μίας περιγραφικής λογικής η οποία να επωφελείται από το συντακτικό του RDFS και να έχει την ίδια δυνατή εκφραστικότητα της $\mathcal{SHIQ}$ (Horrocks, Patel-Schneider, & van Harmelen). Ο συνδυασμός αυτός οδήγησε στη δημιουργία της περιγραφικής λογικής DAML+OIL (Horrocks, 2002), η οποία στη συνέχεια υιοθετήθηκε από τον παγκόσμιο οργανισμό W3C ο οποίος δημιούργησε τη γλώσσα OWL (McGuiness & van Harmelen, 2004), μια βιομηχανοποιημένη έκδοση της λογικής αυτής, και της OWL 2 (W3C OWL Working Group, 2012), την 2η έκδοση της γλώσσας αυτής. Στην (Horrocks, DAML+OIL: a Description Logic,

2002)Εικόνα 2 εμφανίζονται οι βασικές κλάσεις της DAML+OIL, ενώ στην Εικόνα 3 τα βασικά αξιώματα.

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
toClass	$\forall P.C$	∀hasChild.Doctor
hasClass	$\exists P.C$	∃hasChild.Lawyer
hasValue	$\exists P.\{x\}$	∃citizenOf.{USA}
minCardinalityQ	$\geq n P.C$	≥2hasChild.Lawyer
maxCardinalityQ	$\leq n P.C$	≤1hasChild.Male
cardinalityQ	$=n\ P.C$	=1 hasParent.Female

Εικόνα 2 DAML+OIL κλάσεις (Horrocks, DAML+OIL: a Description Logic, 2002)

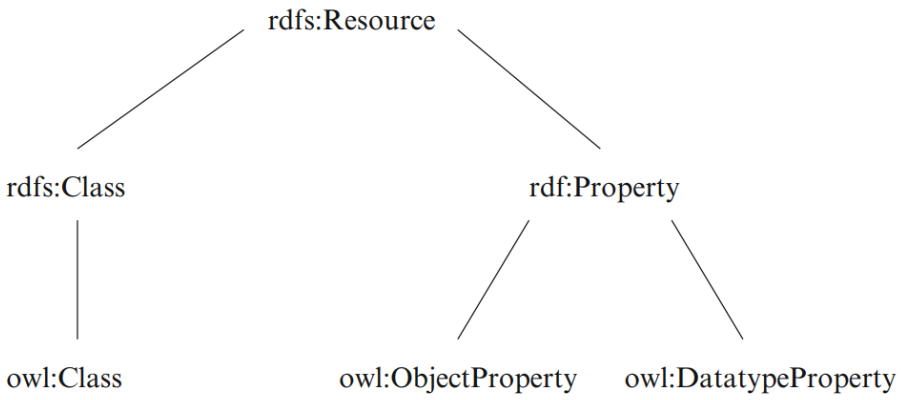
Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human ⊑ Animal ⊓ Biped
sameClassAs	$C_1 \equiv C_2$	Man ≡ Human ⊓ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter ⊑ hasChild
samePropertyAs	$P_1 \equiv P_2$	cost ≡ price
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male ⊑ ¬Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} ≡ {G_W_Bush}
differentIndividualFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} ⊑ ¬{peter}
inverseOf	$P_1 \equiv P_2^-$	hasChild ≡ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ ⊑ ancestor
uniqueProperty	$\top \sqsubseteq \leq 1 P$	⊤ ⊑ ≤1 hasMother
unambiguousProperty	$\top \sqsubseteq \leq 1 P^-$	⊤ ⊑ ≤1 isMotherOf$^-$

Εικόνα 3 DAML+OIL αξιώματα (Horrocks, DAML+OIL: A Description Logic, 2002)

Η OWL επιτρέπει τη δημιουργία βάσης γνώσης, η οποία διακατέχεται από τις εξής ιδιότητες (Antoniou & van Harmelen) όπως αυτές αναπαρίστανται στην Εικόνα 3:

- **Ιεραρχία κλάσεων**, αν μία κλάση Α έχει ως υποκλάση την Β, τότε ένα αντικείμενο Β εννοείται ότι είναι και αντικείμενο της Α.
- **Ισότητα κλάσεων**, αν μία κλάση Α ισούται με την Β και η Β με την Γ, τότε η Α ισούται με την Γ

- **Συνεκτικότητα**, αν η κλάση Α είναι υποκλάση της Β και οι δύο αυτές κλάσεις έχουν δηλωθεί ως ανεξάρτητες μεταξύ τους, τότε να είναι εφικτή η εμφάνιση της μη – συνεκτικότητας
- **Κατηγοριοποίηση**, αν ένα αντικείμενο έχει τις βασικές συσχετίσεις μίας κλάσης Α, τότε μπορούμε να συμπεράνουμε ότι το αντικείμενο αυτό ανήκει στην κλάση αυτή.



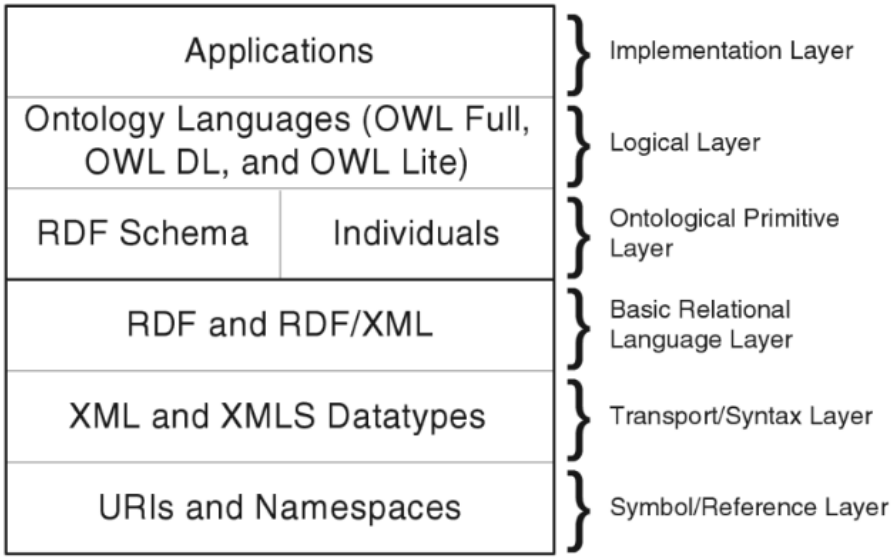
Εικόνα 4 Συσχέτιση RDF-RDFS-OWL (Antoniou & van Harmelen)

Εκτός αυτών, η OWL επιτρέπει οι συσχετίσεις να έχουν ως **range** περιορισμούς κλάσεων και όχι μία καλώς ορισμένη οντότητα. Για παράδειγμα αν έχουμε τις παρακάτω **RDFS** οντότητες, οι οποίες περιγράφουν μια οντότητα Synonims ως υποκλάση της οντότητας Game και μία οντότητα AntonymWord.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix mci: <http://example.org/Mci#>
mci:Game rdf:type rdfs:Class .
mci:AntonymWord rdf:type rdfs:Class .
mci:Synonims rdf:type rdfs:Class ; rdfs:subClassOf mci:Game .
```

τότε μπορούμε να πούμε με τη μορφή ψευδό-RDFS γλώσσας ότι η ιδιότητα *hasSynonym* έχει ως **domain** την οντότητα *Synonims* και σαν **range** όλες τις οντότητες που δεν είναι *AntonymWord* ή δεν έχουν την οντότητα αυτή ως υπερκλάση.

```
mci:hasSynonym rdf:type rdf:Property ; rdfs:domain mci: Synonims; rdfs:range not mci: AntonymWord.
```



Εικόνα 5 Επίπεδα του Σημασιολογικού Ιστού (Lacy, 2005)

Επίσης, επιτρέπει δύο κλάσεις να δηλωθούν ως ανεξάρτητες μεταξύ τους, δηλαδή η τομή τους να είναι το κενό σύνολο. Εκτός αυτού, με την **OWL** κάποιος μπορεί να ορίσει μία κλάση ως την ένωση ή την τομή άλλων κλάσεων, ή του συμπληρώματος μία άλλης και να δημιουργήσει περιορισμούς που πρέπει να ικανοποιούνται για να μπορεί ένα αντικείμενο να ανήκει στην κλάση αυτή. Κάποιες φορές είναι επιθυμητό να υπάρχει ποσοτικοποίηση ως προς τους περιορισμούς αυτούς, για παράδειγμα ένα παζλ έχει τουλάχιστον 4 κομμάτια. Η **OWL** επιτρέπει κάτι τέτοιο παρέχοντας τη δυνατότητα ορισμού cardinality στους περιορισμούς αυτούς όπως min, max, exactly, only και exists. Επιπλέον, η **OWL** κατηγοροποιεί τις συσχετίσεις σε **DataTypeProperty**, οι οποίες έχουν ως **range** κάποιο datatype όπως ένα xsd:string, και σε **ObjectProperty**, οι οποίες ως **range** έχουν κάποια οντότητα η οποία έχει χαρακτηρισθεί ως **rdf:type** *owl:Class*.

Τέλος, με την OWL είναι εφικτή η σήμανση πληροφορίας κάθε συσχέτισης, όπως για παράδειγμα αν κάθε οντότητα μπορεί να έχει το πολύ μία συσχέτιση με κάποιο άλλη οντότητα μέσω αυτής της συσχέτισης. Η **OWL** υποστηρίζει τις εξής «ιδιότητες» των συσχετίσεων:

- ***Functional***: Κάθε οντότητα, που ανήκει στο **domain** της συσχέτισης αυτής, μπορεί να ενώνεται το πολύ με μία άλλη οντότητα, που ανήκει στο **range**, μέσω της συσχέτισης αυτής. Για παράδειγμα εάν μία εικόνα έχει έναν τίτλο τότε αυτό μπορεί να μοντελοποιηθεί ως *Image hasTitle Word*. Όταν δημιουργούνται individuals της οντότητας *Image* τότε αυτά πρέπει να ενώνονται το πολύ μία φορά μέσω της συσχέτισης *hasTitle* με μία οντότητα τύπου *Word*.
- ***Inverse functional***: Είναι το αντίστροφο της ιδιότητας **Functional**. Κάθε οντότητα που ανήκει στο range της συσχέτισης αυτής μπορεί να έχει πολλαπλές εισερχόμενες συσχετίσεις μέσω της συσχέτισης αυτής. Για παράδειγμα *Greece hasCurrency Euro*, *Italy hasCurrency Euro*, το individual *Euro* έχει πολλαπλές εισερχόμενες συσχετίσεις μέσω της

συσχέτισης *hasCurrency* και τα individuals *Greece* και *Italy* μπορούν να έχουν το πολύ μία εξερχόμενη συσχέτιση μέσω του *hasCurrency*.

- ***Transitive***: Αν μία οντότητα X ενώνεται με μια οντότητα Y μέσω αυτής της συσχέτισης και αν η Y ενώνεται με μία οντότητα Z με την ίδια συσχέτιση, τότε υποδηλώνεται η X μπορεί να ενωθεί με την Z μέσω της ίδιας συσχέτισης. Για παράδειγμα, αν η συσχέτιση *hasChild* είναι transitive και αν *Bob hasChild Alice* και *Alice hasChild Nick*, τότε *Bob hasChild Nick*.
- ***Symmetric****(OWL2)*: Αν μία οντότητα X ενώνεται με μια οντότητα Y μέσω αυτής της συσχέτισης τότε η Y ενώνεται με την X μέσω της ίδιας συσχέτισης. Για παράδειγμα αν η συσχέτιση *hasFriend* είναι **Symmetric** και αν *Bob hasFriend Alice*, τότε *Alice hasFriend Bob*
- ***Asymmetric****(OWL2)*: Αν μία οντότητα X ενώνεται με μια οντότητα Y μέσω αυτής της συσχέτισης τότε η Y δεν μπορεί να ενωθεί με την X μέσω της ίδιας συσχέτισης. Για παράδειγμα αν η συσχέτιση *hasChild* είναι **Asymmetric** και αν *Bob hasChild Alice*, τότε δεν μπορεί να ισχύει *Alice hasChild Bob*.
- ***Reflexive****(OWL2)*: Υποδηλώνει ότι μία οντότητα μπορεί να συσχετιστεί με τον εαυτό της μέσω της συσχέτισης αυτής. Για παράδειγμα αν *Bob knows Alice*, τότε ισχύει *Bob knows Bob*
- ***Irreflexive****(OWL2)*: Υποδηλώνει ότι μία οντότητα δεν μπορεί να συσχετιστεί με τον εαυτό της μέσω της συσχέτισης αυτής. Για παράδειγμα αν *Bob hasChild Alice*, τότε δεν ισχύει *Bob hasChild Bob*.

2.5.1 OWL Profiles

Το σύνολο των απαιτήσεων μιας οντολογίας φαίνεται αδύνατο να επιτευχθεί από μια μόνο γλώσσα γιατί απαιτεί τον συνδυασμό της αποτελεσματικής υποστήριξης της συλλογιστικής και την ευκολία έκφρασης τόσο ισχυρή όσο ένας συνδυασμός **RDF Schema** με πλήρη λογική. Θεωρούμε γενικά ότι όσο μεγαλύτερη είναι η εκφραστική δύναμη μιας γλώσσας, τόσο καλύτερη είναι και η διεξαγωγή συμπερασμών αυξανόμενης πολυπλοκότητας. Οι απαιτήσεις αυτές οδήγησαν στην δημιουργία τριών προφίλ της **OWL, OWL Full, OWL DL** και **OWL Lite**. Η δημιουργία της OWL 2 έχει προσθέσει τρία νέα προφίλ τα **OWL 2 EL, OWL 2 QL και OWL 2 RL**.

Μεταξύ των OWL Full, OWL DL και OWL Lite ισχύουν τα εξής:

- Κάθε έγκυρη οντολογία OWL Lite είναι μια έγκυρη οντολογία OWL DL.
- Κάθε έγκυρη οντολογία OWL DL είναι μια έγκυρη οντολογία OWL Full
- Κάθε έγκυρο συμπέρασμα OWL Lite είναι ένα έγκυρο συμπέρασμα OWL DL
- Κάθε έγκυρο συμπέρασμα OWL DL είναι ένα έγκυρο συμπέρασμα OWL Full

Ενώ, και η OWL DL και η OWL Full χρησιμοποιούν το ίδιο λεξιλόγιο, η OWL DL θέτει κάποιους περιορισμούς στη χρήση του. Επίσης να σημειώσουμε ότι η OWL εξακολουθεί να χρησιμοποιεί την **RDF** και το RDF Schema σε μεγάλο βαθμό. Όλα τα είδη της OWL χρησιμοποιούν την **RDF**

για τη σύνταξή τους. Τα στιγμιότυπα ορίζονται όπως στην **RDF**, χρησιμοποιώντας τις **RDF** περιγραφές και εισάγοντας την πληροφορία, ενώ δομές (Constructors) της OWL όπως owl:Class, owl:DataTypeProperty και owl:ObjectProperty αποτελούν εξειδικεύσεις των αντίστοιχων της **RDF**. Οι υπεύθυνοι για την ανάπτυξη οντολογιών που υιοθετούν την **OWL** πρέπει να εξετάσουν ποιο προφίλ ανταποκρίνεται καλύτερα στις ανάγκες τους. Παρακάτω αναφέρονται οι ουσιαστικές διαφορές μεταξύ των γλωσσών. Οι τεχνικές προδιαγραφές των προφίλ αναγράφονται αναλυτικά τόσο για την **OWL** στο (Bechhofer, et al., 2004), όσο και για την OWL 2 στο (W3C OWL Working Group, 2012).

OWL Full

Η OWL Full προορίζεται για χρήστες που επιθυμούν μέγιστη εκφραστικότητα και την πλήρη εκφραστική ελευθερία του **RDF** χωρίς όμως εγγυήσεις επιλυσιμότητας. Χρησιμοποιεί όλες τις θεμελιώδεις αρχές των υπογλωσσών **OWL** και επιτρέπει το συνδυασμό των αρχών αυτών με την **RDF** και το **RDF Schema**. Η επιλογή μεταξύ της OWL Lite και της OWL DL εξαρτάται από το βαθμό στον οποίο οι χρήστες απαιτούν περισσότερο εκφραστικές οντολογίες. Η OWL Full περιέχει στην ουσία ολόκληρη τη γλώσσα **OWL** και χρησιμοποιεί όλα τα δομικά συστατικά της. Επιτρέπει επιπλέον το συνδυασμό των συστατικών αυτών με οποιοδήποτε αυθαίρετο τρόπο με τα στοιχεία της **RDF** και της **RDF Schema**. Στα θετικά στοιχεία της OWL Full περιλαμβάνεται το ότι είναι πλήρως αναδρομικά συμβατή με τις RDF, RDFS, OWL Lite και OWL DL. Οποιοδήποτε έγκυρο έγγραφο στις γλώσσες αυτές είναι επίσης και έγκυρο στην OWL Full και κάθε πόρισμα που προκύπτει από αυτά είναι επίσης έγκυρο και στην OWL Full. Το βασικό μειονέκτημα της OWL Full είναι ότι σαν γλώσσα είναι τόσο ισχυρή έτσι ώστε να μην είναι δυνατή η εξασφάλιση υποστήριξης από υπηρεσίες συλλογιστικής (reasoning). Η ελευθερία στην περιγραφή της OWL Full δεν εξασφαλίζει ότι οι διαδικασίες αυτές μπορεί να λήγουν σε πεπερασμένο χρόνο (Antoniou & van Harmelen)

OWL Lite

Η OWL Lite έχει σχεδιαστεί με σκοπό την έκφραση ιεραρχιών ταξινόμησης και απλών περιορισμών ιδιοτήτων. Είναι πιο εύκολο να σχεδιαστούν εργαλεία και να αντιστοιχιστούν όροι και ταξινομήσεις στην OWL Lite από ότι στα άλλα εκφραστικότερα επίπεδα. Ακόμη αυτή είναι πιο εύκολη στην εκμάθηση από τους χρήστες. Το σημαντικότερο μειονέκτημά της είναι ασφαλώς η περιορισμένη εκφραστικότητα. Σαν γλώσσα, περιέχει μόνο ένα μικρό υποσύνολο του λεξικού της **OWL**. Συγκεκριμένα, έχει αφαιρεθεί η δήλωση για αριθμημένες (enumerated) κλάσεις, η δήλωση για ανεξάρτητες (disjoint) κλάσεις και ο αυθαίρετος περιορισμός στον αριθμό των στοιχείων των κλάσεων (πληθάριθμος - cardinality) (Antoniou & van Harmelen).

OWL DL

Η υπογλώσσα OWL DL σχεδιάστηκε για τους χρήστες που επιθυμούν τη μέγιστη δυνατή εκφραστικότητα, διατηρώντας παράλληλα την υπολογιστική πληρότητα (όλα τα συμπεράσματα να είναι εγγυημένα υπολογίσιμα) και την διεξαγωγή συμπερασμών σε πεπερασμένο χρόνο. Η OWL DL ονομάζεται έτσι λόγω της αντιστοιχίας της με τις **Περιγραφικές Λογικές** (DL). Η OWL DL έχει σχεδιαστεί έτσι ώστε να είναι περιγραφική σαν γλώσσα ώστε να μπορεί να δέχεται

«εξυπηρετητές υπηρεσιών συλλογιστικής ή συστήματα συλλογιστικής» (reasoners) των οποίων οι αλγόριθμοι να τερματίζουν σε πολυωνυμικό χρόνο. Τα χαρακτηριστικά της OWL DL τέτοια ώστε να βελτιστοποιείται η υπολογισιμότητα των εννοιών της διατηρώντας παράλληλα και τη βέλτιστη απόδοση σε θέματα πολυπλοκότητας. Όσον αφορά το λεξικό που χρησιμοποιείται στην OWL DL, είναι προφανώς περιορισμένο σε σχέση με την πλήρη εκφραστικότητα της OWL Full. Στα αρνητικά σημεία της, ωστόσο, περιλαμβάνεται το ότι έχει χαθεί η συμβατότητα με το **RDF**. Ένα έγγραφο **RDF** θα πρέπει να τροποποιηθεί και να επεκταθεί σε αρκετά σημεία ώστε να καταστεί συμβατό με την OWL DL. Αντίθετα, όμως, κάθε έγκυρο OWL DL έγγραφο είναι επίσης και έγκυρο **RDF** έγγραφο (Antoniou & van Harmelen).

OWL 2 EL

Το προφίλ αυτό είναι παρόμοιο με το OWL DL. Η ονομασία προήλθε από την οικογένεια περιγραφικής λογικής EL++, όπου οι γλώσσες αυτές χρησιμοποιούν κυρίως υπαρξιακούς ποσοδείκτες. Η OWL 2 EL έχει σχεδιαστεί λαμβάνοντας υπόψη τις ανάγκες που υπάρχουν στις βιολογικές και επιστημονικές οντολογίες, καθώς επίσης στις οντολογίες που περιγράφουν τις ρυθμίσεις ενός συστήματος ή την καταγραφή εμπορευμάτων. Το κοινό στις οντολογίες αυτές είναι η πολύπλοκη δομή των κλάσεων που τα απαρτίζουν. Οι οντολογίες αυτές έχουν πολύ μεγάλο αριθμό κλάσεων και οι κλάσεις αυτές συνήθως έχουν μεγάλη και πολύπλοκη αλυσίδα ιεραρχίας. Ενώ η OWL 2 EL μπορεί να ανταπεξέλθει με πολύ μεγάλη εκφραστικότητα σε τέτοιου είδους οντολογίες, ταυτόχρονα έχει κάποιους περιορισμούς ως προς τις εκφράσεις που μπορούν να χρησιμοποιηθούν. Η OWL 2 EL δεν επιτρέπει την έκφραση της άρνησης και της ανεξαρτησίας μίας κλάσης. Επιπλέον αποκλείονται οι καθολικοί δείκτες και οι αντίστροφες συσχετίσεις (W3C OWL Working Group, 2012).

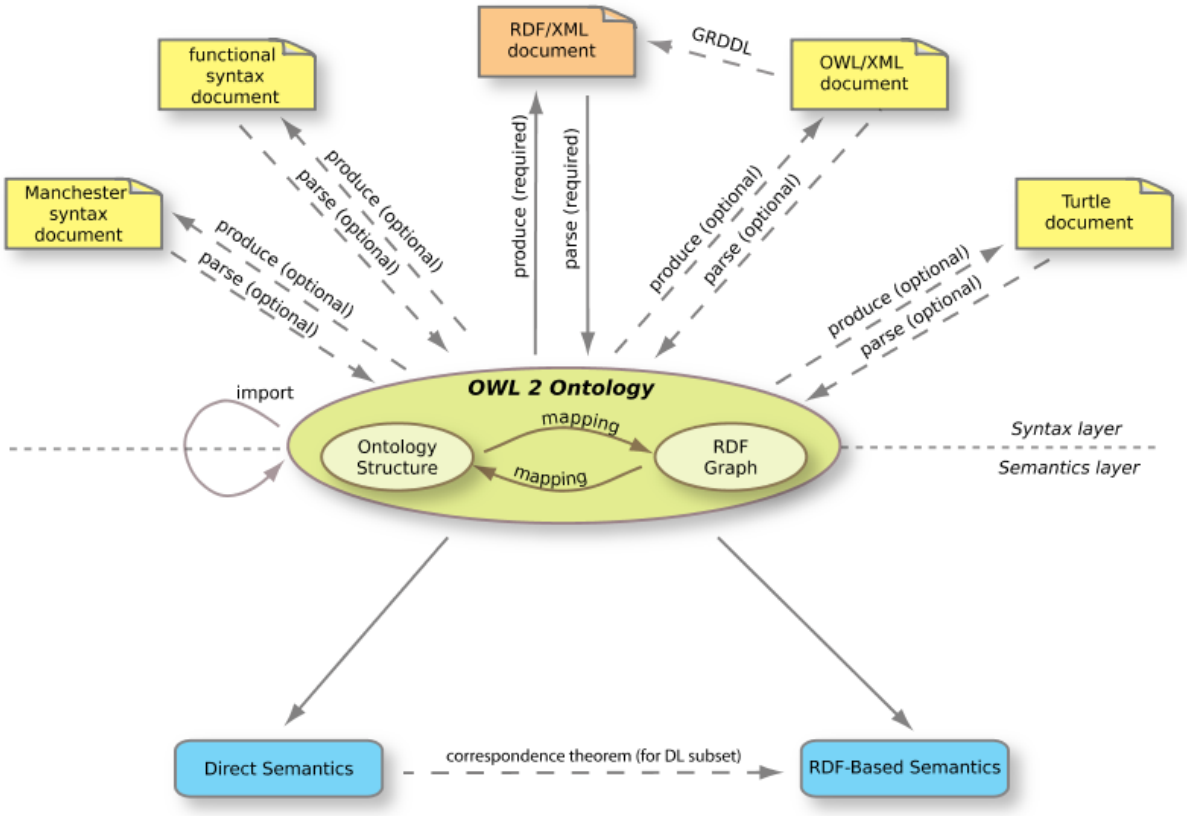
OWL 2 QL

Το προφίλ αυτό καλύπτει τις ανάγκες του χρήστη που επιθυμεί την χρήση κλασσικών οντοτήτων - συσχετίσεων Βάσεων Δεδομένων. Ουσιαστικά η γλώσσα αυτή μπορεί να θεωρηθεί μια «επέκταση» της SQL και να επωφεληθεί από τα χαρακτηριστικά της όπως η ταυτόχρονη χρήση της από πολλούς χρήστες. Επίσης χρησιμοποιείται ως ένα ενδιάμεσο επίπεδο το οποίο μεταφράζει συσχετίσεις από ένα UML διάγραμμα σε SQL γλώσσα. Η γλώσσα αυτή υποστηρίζει κάποια βασικά χαρακτηριστικά της **RDFS** όπως αντίστροφες ιδιότητες και τις ιεραρχίες τους. Παρόλα αυτά, αποτρέπει την χρήση των υπαρξιακών ποσοδεικτών, την δημιουργία ιδιοτήτων – αλυσίδες και της ισότητας αυτών (W3C OWL Working Group, 2012).

OWL 2 RL

Το προφίλ αυτό απευθύνεται σε εφαρμογές οι οποίες χρησιμοποιούν την OWL 2 και επιθυμούν να θυσιάσουν μερική από την εκφραστικότητα της γλώσσας με στόχο την καλύτερη υπολογιστική απόδοση. Επίσης το ίδιο προφίλ μπορεί να χρησιμοποιηθεί από εφαρμογές που χρησιμοποιούν μόνο **RDFS** και επιθυμούν κάποια από την εκφραστικότητα της OWL. Αυτό επιτυγχάνεται ορίζοντας ένα υποσύνολο του συντακτικού της OWL 2. Το υποσύνολο αυτό είναι κατανοητό από τεχνολογίες που βασίζονται σε κανόνες (rule based) με αποτέλεσμα την δημιουργία first-order ισχυρισμών οι οποίοι μπορούν να χρησιμοποιηθούν από τις εφαρμογές που χρησιμοποιούν το

προφίλ αυτό. Αυτό το υβριδικό προφίλ έχει κάποιους περιορισμούς όπως ότι η ύπαρξη ενός αντικειμένου δεν μπορεί να προσδιορίσει την ύπαρξη ενός άλλου. Επίσης κάποια αξιώματα τα οποία μπορούν να χρησιμοποιηθούν για να εκφράσουν υποκλάσεις, δεν μπορούν να εκφράσουν υπερκλάσεις (W3C OWL Working Group, 2012).



Εικόνα 6 OWL 2 (W3C OWL Working Group, 2012)

2.6 Reasoners

Μέχρι στιγμής τόσο στην ενότητα που αναλύεται η **Περιγραφική Λογική**, όσο και στην ενότητα που αναλύεται η γλώσσα **OWL** γίνεται συχνά αναφορά στην εκφραστικότητα της γλώσσας. Όπως αναφέρθηκε παραπάνω υψηλή εκφραστικότητα σημαίνει ότι από ένα συγκεκριμένο **TBox** μπορεί να διαπιστωθεί ένα πολύ μεγάλο **ABox**. Στην ενότητα αυτή θα αναλυθεί ο τρόπος με τον οποίο γίνεται αυτός ο συμπερασμός του **ABox**. Πιο συγκεκριμένα θα μελετηθούν κάποιοι αλγόριθμοι, που ονομάζονται reasoners, οι οποίοι είναι υπεύθυνοι για την συνοχή μίας οντολογίας και για τον συμπερασμό ενός **ABox**.

Σύμφωνα με την έρευνα των (Matentzoglu, Leo, Hudhra, Parsia, & Sattler, 2015) ένας reasoner έχει κάποια βασικά χαρακτηριστικά. Υπάρχουν reasoners που προσφέρουν ικανοποιητική κατηγοριοποίηση ενός **TBox**, ενώ άλλοι είναι δυνατοί στην απάντηση ερωτημάτων. Από την

πλευρά της χρήσης, οι συγγραφείς υποστηρίζουν ότι τα χαρακτηριστικά ενός reasoner είναι οι υποστηριζόμενες υπηρεσίες συμπερασμού, το επίπεδο της εκφραστικότητας και η πληρότητα του αλγορίθμου. Βεβαίως η υλοποίηση ενός τέτοιου αλγορίθμου μπορεί να βασίζεται σε διαφορετικούς λογισμούς όπως ο λογισμός Tableau (Hähnle, 2001) ή ο Consequence based λογισμός (Simancík, Kazakov, & Horrocks, 2011), κάτι το οποίο να ενδιαφέρει τους ερευνητές οι οποίοι επιθυμούν να βρουν αποδοτικότερους τρόπους συμπερασμού.

		Pellet	RACER	FACT++	Snorocket	SWRL-IQ	HermiT	CEL	TrOWL	ELK
Methodology		Tableau based	Tableaux based	tableau based	Completion rules	SWRL rules	Hypertableau based	Completion rules	Completion rules	Consequence based
Soundness		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Completeness		Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Expressivity		SROIQ(D	SHIQ	SROIQ(D	EL+	-	SROIQ(D)	EL+	SROIQ	EL
Native Profile		DL, EL	DL	DL	EL	-	DL	EL	DL, EL	EL
Incremental Classification	**Additio**	Yes	No	No	Yes	Y/N	No	Yes	No	Yes
	Remova	Yes	No	No	No	Y/N	No	No	No	Yes
Rule Support		Yes (SWRL)	Yes (SWRL)	No	No	Yes (SWRL)	Yes (SWRL)	No	No	Yes (Own rule format)
Platforms		all	all	all	all	all	all	Linux	all	all
Justifications		Yes	Yes	No	No	Yes	No	Yes	No	No
ABOX Reasoning		Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No
OWL API		Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
OWL Link API		Yes	Yes	Yes	No	No	Yes	Yes	No	Y/N
Protégé Support		Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
NeOn Support		Yes	No	No	No	No	Yes	No	No	No
License		DULI: AGPL	own	GLGPL	own	Y/N	GLGPL	Apache License	DULI: AGPL	Apache License 2.0
Jena Support		Yes	No	No	No	No	No	No	Yes	Y/N
Impl. Language		Java	LISP	C++	Java	Prolog	Java	LISP	Java	Java
Availability		Open source	Commercial	Open Source	Commercial	Y/N	Open source	Open source	Commercial	Open source

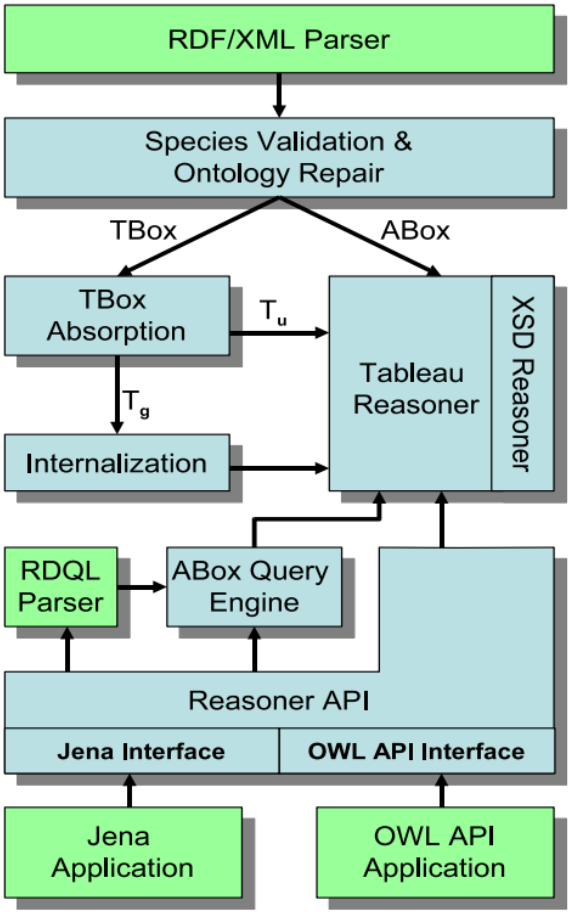
Εικόνα 7 OWL Reasoners (Abburu, 2012)

Η **OWL** έχει πολλούς reasoners. Από τους πιο διαδομένους είναι οι Pellet, FaCT++ και HermiT. Στη συνέχεια θα αναφερθούν κάποιες βασικές ιδιότητες των αλγορίθμων αυτών.

2.6.1 Pellet

Ο αλγόριθμος Pellet δημιουργήθηκε από τους (Parsia & Sirin, 2004) και στοχεύει κυρίως το OWL DL προφίλ της **OWL**. Ο αλγόριθμος αυτός προσφέρει έλεγχο για την ορθότητα και τη συνεκτικότητα μιας οντολογίας. Παρόλο που ο αλγόριθμος έχει σχεδιαστεί για το OWL DL,

παρέχει τη δυνατότητα χρήσης του ελέγχου ορθότητας μίας οντολογίας γραμμένη σε OWL Full. Επίσης μπορεί να ελέγξει αν πιθανοί περιορισμοί σε κάποιες τιμές εκπληρώνονται, για παράδειγμα μία ιδιότητα να δέχεται αριθμούς σε συγκεκριμένο πεδίο τιμών, ή μια άλλη να έχει συγκεκριμένες τιμές αλφαριθμητικών. Αξιοσημείωτο είναι ότι εκτός από το σύνηθες σύνολο υπηρεσιών reasoning που προσφέρει ο Pellet επιπλέον παρέχει και υπηρεσίες για αποσφαλμάτωση και επεξήγηση οντολογιών. Η διαφορά του Pellet όμως από τους υπόλοιπους reasoners είναι ότι από την αρχή σχεδιάστηκε ώστε να συνεργάζεται με την **OWL**. Επιπλέον ο αλγόριθμος αυτός προσφέρει συμπερασμό στην οντολογία και τη δυνατότητα να απαντηθούν ερωτήματα σχετικά με το **ABox**. Τέλος, ο αλγόριθμος αυτός είναι βασισμένος στον λογισμό Tableau και έχει υλοποιηθεί στην γλώσσα προγραμματισμού Java (Parsia & Sirin, 2004).



Εικόνα 8 Αρχιτεκτονική Pellet (Parsia & Sirin, 2004)

2.6.2 HermiT

Ένας άλλος διαδεδομένος reasoner είναι ο HermiT (Shearer, Motik, & Horrocks). Γενικά οι Tableau reasoners προκειμένου να ελέγξουν την συνεκτικότητα μιας οντολογίας, δημιουργούν διαφορετικά μοντέλα που αναπαριστούν την βάση γνώσης. Ο αλγόριθμος αυτός βασίζεται στον

«hypertableau» λογισμό, ο οποίος ελαχιστοποιεί τον αριθμό των διαφορετικών μοντέλων που μπορούν να δημιουργηθούν. Το βασικό χαρακτηριστικό του αλγορίθμου αυτού είναι ότι μπορεί να χρησιμοποιηθεί σε οντολογίες που χρησιμοποιούν περιγραφές-γράφους, οι οποίοι δεν μπορούν να αναπαρασταθούν αξιόπιστα μέσω της **OWL**. Ο reasoner αυτός είναι υλοποιημένος σε Java και μπορεί να χρησιμοποιηθεί μέσω του Java API και της γραμμής εντολών.

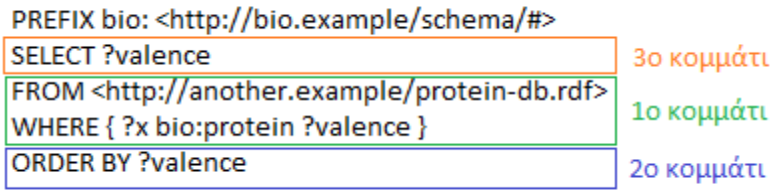
2.6.3 FaCT++

Αυτός ο reasoner δημιουργήθηκε από τους (Tsarkov & Horrocks, 2006) και είναι ένας από τους «ορθούς και πλήρεις» reasoners, παρόμοια με τον Pellet. Ο reasoner αυτός υλοποιεί Tableau λογισμό για την περιγραφική λογική $\mathcal{SHOIQ}$ με υποστήριξη για δεδομένα τύπου ακεραίων ή αλφαριθμητικών. Ο αλγόριθμος αυτός έχει τρία βασικά στάδια με τα οποία επιτυγχάνεται ο έλεγχος της ορθότητας μίας οντολογίας. Σαν πρώτο βήμα ο αλγόριθμος «φορτώνει» την βάση γνώσης στη μνήμη ενώ ταυτόχρονα εκτελεί κάποιες βελτιστοποιήσεις σε αυτή. Στο δεύτερο βήμα γίνεται η κατηγοριοποίηση και ξανά βελτιστοποιήσεις στο αποτέλεσμα της κατηγοριοποίησης και σαν τελευταίο βήμα, ο αλγόριθμος ελέγχει την ορθότητα της κατηγοριοποίησης. Εν αντιθέσει με τους προαναφερθείς reasoners, ο FaCT++ είναι υλοποιημένος σε C++.

2.7 SPARQL

Αφού εφαρμόσουμε reasoning σε μία οντολογία, το αποτέλεσμα θα περιέχει το σύνολο της βάσης γνώσης, δηλαδή το **TBox** και το **ABox**. Ένα από τα δυνατά σημεία της **OWL** είναι ότι κάποιος μπορεί να κάνει ερωτήσεις στην οντολογία, σχετικά με το **TBox** ή με το **ABox**, και να πάρει την αντίστοιχη απάντηση. Η **SPARQL** είναι μία γλώσσα, που επιτρέπει την σύνταξη και την εκτέλεση τέτοιων ερωτήσεων σε μία οντολογία βασισμένη σε **RDF** (W3C SPARQL Working Group, 2013). Εφόσον η OWL αποτελεί επέκταση της **RDF** όπως έχει αναφερθεί προηγουμένως, μπορούμε να χρησιμοποιήσουμε την **SPARQL** ώστε να κάνουμε ερωτήσεις σε μια **OWL** οντολογία.

Η **SPARQL** χρησιμοποιεί **RDF** γράφους για την δομή των ερωτήσεων και για την εμφάνιση των απαντήσεων. Όπως αναφέρθηκε παραπάνω, η **RDF** χρησιμοποιεί τριπλέτες για την περιγραφή μιας συσχέτισης μεταξύ δύο οντοτήτων. Αν κάποιος σχεδιάσει τις τριπλέτες αυτές, τότε θα καταλήξει να έχει έναν γράφο, όπου οι κορυφές είναι οι οντότητες και οι ακμές είναι οι συσχετίσεις που τις ενώνουν. Το κάθε **SPARQL** ερώτημα αποτελείται από τρία βασικά κομμάτια. Το πρώτο περιέχει τον γράφο που ερωτάται, ο οποίος μπορεί να περιέχει προαιρετικά τμήματα ή φίλτρα που εφαρμόζονται σε τιμές, να αποτελεί ένωση πολλαπλών ή εμφωλεύσιμων γράφων. Το δεύτερο κομμάτι επιτρέπει την επεξεργασία της απάντησης – λύσης του ερωτήματος, για παράδειγμα distinct, order by, limit. Το τελευταίο κομμάτι είναι η ίδια η απάντηση, η οποία μπορεί να είναι ένας υπογράφος της γνώσης βάσης, ή η περιγραφή των οντοτήτων ή απλά ένα ναι/όχι. (Pérez, Arenas, & Gutierrez, 2006)



Εικόνα 9 Κομμάτια ενός SPARQL ερωτήματος

2.7.1 Συντακτικό SPARQL

Η **SPARQL** υποστηρίζει ερωτήματα τύπου SELECT, INSERT, DELETE παρόμοια με την SQL. Επίσης ως querying γλώσσα επιτρέπει στον χρήστη την δήλωση του γράφου που επιθυμεί να ρωτήσει, καθώς και να ορίσει τις τιμές συγκεκριμένων μεταβλητών μέσω του WHERE clause. Καθώς αυτές οι έννοιες είναι παρόμοιες και στην SQL, ο αναγνώστης μπορεί να βρει περισσότερες λεπτομέρειες και παραδείγματα στο specification της **SPARQL** (Prud'hommeaux & Seaborne, 2008). Παρακάτω θα αναφερθούν δύο νέοι τύποι ερωτημάτων που δεν έχουν αντίστοιχη ερμηνεία στην SQL.

CONSTRUCT

Η SPARQL υποστηρίζει ερωτήματα τύπου SELECT, INSERT, DELETE παρόμοια με την SQL. Εκτός αυτών η SPARQL υποστηρίζει τον τύπο CONSTRUCT, με τον οποίο κάποιος μπορεί να δημιουργήσει νέες τριπλέτες χρησιμοποιώντας τις τιμές των μεταβλητών που ρωτήθηκαν. Για παράδειγμα έχουμε μία οντολογία που περιγράφει ένα άτομο ως το όνομα (name) τους και την διεύθυνση email (mbox) τους :

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a    foaf:name   "Alice" .
_:a    foaf:mbox   <mailto:alice@example.org> .
```

Μπορούμε να ορίσουμε το παρακάτω CONSTRUCT ερώτημα το οποίο θα δημιουργήσει μία νέα τριπλέτα στην βάση γνώσης που θα προσθέσει την ιδιότητα *vcard:FN* για τα στιγμιότυπα τα οποία έχουν την ιδιότητα *foaf:name*. Το ερώτημα αυτό μπορεί να μεταφραστεί σε ανθρώπινη γλώσσα ως εξής, *"Δημιούργησε την συσχέτιση, στο Person στιγμιότυπο Alice, vcard:FN με τιμή το όνομα, ώπου το όνομα ορίζεται ως η τιμή της ιδιότητας foaf:name"*

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

θα απαντήσει την τριπλέτα που δημιούργησε στη βάση γνώσης. Το στιγμιότυπο Alice της οντότητας Person έχει την ιδιότητα *vcard:FN* με τιμή *"Alice"*

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://example.org/person#Alice> vcard:FN "Alice" .
```

ASK

Αυτός ο τύπος ερώτησης χρησιμοποιείται για να ελέγξουμε αν ένας γράφος είναι αληθής ή όχι. Για παράδειγμα, αν έχουμε την οντολογία

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .

_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .

_:b  foaf:mbox       <mailto:bob@work.example> .
```

Τότε αν ρωτηθεί αν υπάρχει ένα στιγμιότυπο με *foaf:name* να έχει την τιμή *Alice*

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

ASK  { ?x foaf:name  "Alice" }
```

Θα απαντηθεί απλά `yes`

2.7.2 SPARQL Protocol Operations

Ένα από τα πλεονεκτήματα της **SPARQL** είναι ότι πρόσφατα έχει υλοποιηθεί ένα HTTP πρωτόκολλο το οποίο επιτρέπει την εκτέλεση ενός **SPARQL** ερωτήματος μέσω του Internet. Με τον τρόπο αυτό η οντολογία μπορεί να βρίσκεται σε διαφορετικό εξυπηρετητή από την εφαρμογή. Εφόσον ο εξυπηρετητής που φιλοξενεί την οντολογία, υλοποιεί το **SPARQL** πρωτόκολλο τότε οποιαδήποτε εφαρμογή μπορεί να εκτελέσει ερωτήσεις απομακρυσμένα.

Το πρωτόκολλο αυτό υποστηρίζει μόνο δύο λειτουργίες, query και update. Η λειτουργία query επιτρέπει ερωτήματα τα οποία «διαβάζουν» μία οντολογία, όπως τα SELECT και ASK, ενώ η λειτουργία update επιτρέπει ερωτήματα τα οποία μεταβάλλουν μια οντολογία είτε προσθέτοντας, είτε αφαιρώντας, είτε αλλάζοντας τις τιμές ενός γράφου, όπως τα INSERT ή DELETE.

	HTTP Method	Query String Parameters	Request Content Type	Request Message Body
query via GET	GET	`query` (exactly 1) `default-graph-uri` (0 or more) `named-graph-uri` (0 or more)	None	None
query via URL-encoded POST	POST	None	`application/x-www-form-urlencoded`	URL-encoded, ampersand-separated query parameters. `query` (exactly 1) `default-graph-uri` (0 or more) `named-graph-uri` (0 or more)
query via POST directly	POST	`default-graph-uri` (0 or more) `named-graph-uri` (0 or more)	`application/sparql-query`	Unencoded SPARQL query string

Εικόνα 10 Τρόποι χρήσης SPARQL Protocol για την query λειτουργία (Feigenbaum, Williams, Clark, & Torres, 2013)

	HTTP Method	Query String Parameters	Request Content Type	Request Message Body
update via URL-encoded POST	POST	None	`application/x-www-form-urlencoded`	URL-encoded, ampersand-separated query parameters. `update` (exactly 1) `using-graph-uri` (0 or more) `using-named-graph-uri` (0 or more)
update via POST directly	POST	`using-graph-uri` (0 or more) `using-named-graph-uri` (0 or more)	`application/sparql-update`	Unencoded SPARQL update request string

Εικόνα 11Τρόποι χρήσης SPARQL Protocol για την update λειτουργία (Feigenbaum, Williams, Clark, & Torres, 2013)

Ένα από τα δυνατά σημεία του SPARQL Protocol είναι ότι τα αποτελέσματα των ερωτήσεων μπορούν να ακολουθούν όλες τις διαδεδομένες μορφοποιήσεις όπως XML (Beckett & Broekstra, 2013) και JSON (Clark, Feigenbaum, & Torres, 2013) .

2.8 REST

Η αρχιτεκτονική Representational State Transfer (REST), είναι μια αρχιτεκτονική, προσαρμοσμένη στην ζήτηση υπηρεσιών, για κατανεμημένα συστήματα. Η αρχιτεκτονική REST ορίζει συγκεκριμένες αρχές για τον σχεδιασμό υπηρεσιών Ιστού, με βάση τους πόρους (resources) και τις αναπαραστάσεις τους (representations), και έχει σαν στόχο την απροβλημάτιστη διαλειτουργικότητα ετερογενών συστημάτων χαλαρής ζεύξης (Fielding, 2000).

Με τον όρο resource μπορεί να χαρακτηριστεί οποιαδήποτε πληροφορία ή έννοια. Έτσι, σε ένα συστήματα δημιουργίας παιγνίων, ως resource μπορούμε να χαρακτηρίσουμε ένα παίγνιο, μία λίστα παιγνίων ενός χρήστη, μία λύση κ.ά.

Ωστόσο, ενώ resource μπορεί να είναι οποιαδήποτε πληροφορία ή έννοια, η αναπαράσταση αυτής (representation) είναι το έγγραφο ή το αρχείο συγκεκριμένης μορφοποίησης που περιγράφει την τρέχουσα κατάσταση του resource (Pautasso, 2013). Για κάθε resource είναι δυνατό να υπάρχουν πολλές διαφορετικές αναπαραστάσεις, όπου οι πιο συχνά χρησιμοποιούμενες είναι οι απλές HTML σελίδες και οι σελίδες που ακολουθούν τις μορφοποιήσεις XML και JSON. Αυτό επιτρέπει σε κάθε πελάτη να μπορεί κατά την αποστολή ενός αιτήματος για την τρέχουσα κατάσταση ενός πόρου, να ζητήσει και συγκεκριμένη μορφοποίηση για την επιστρεφόμενη αναπαράσταση. Βεβαίως υπάρχει περίπτωση ο εξυπηρετητής να μην μπορεί να επιστρέψει την αναπαράσταση στην αιτούμενη μορφοποίηση, ενημερώνοντας τον πελάτη με κατάλληλο μήνυμα του HTTP πρωτοκόλλου.

Όπως ήδη αναφέρθηκε, η αιτούμενη μορφοποίηση της αναπαράστασης βρίσκεται μέσα στο απεσταλμένο μήνυμα. Αυτό συμβαίνει διότι η αρχιτεκτονική REST επιβάλει τα μηνύματα να είναι αυτό-περιγραφόμενα, δηλαδή να περιέχουν όλη την πληροφορία που χρειάζονται ώστε να επεξεργαστούν, για παράδειγμα δίχως να γνωρίζει ο εξυπηρετητής κάτι για την κατάσταση του πελάτη (Pautasso, Wilde, & Alarcon, 2014). Έτσι λοιπόν όλα τα μηνύματα θα πρέπει να περιέχουν πληροφορία για τη μορφοποίηση των αναπαραστάσεων, πληροφορίες για την προσωρινή αποθήκευση κ.α.

Στην αρχιτεκτονική REST είναι δυνατή η πρόσβαση στις λειτουργίες των υπηρεσιών Ιστού μέσω των Universal Resource Identifier (URI). Μέσω της κλήσης μιας υπηρεσίας Ιστού και κάνοντας χρήση του HTTP πρωτοκόλλου και των HTTP μεθόδων (GET/PUT/POST/DELETE) είναι δυνατή η πρόσβαση και ο χειρισμός των πόρων. Σε μια αρχιτεκτονική τύπου REST το μόνο προαπαιτούμενο προκειμένου να είναι δυνατή η πρόσβαση στην υπηρεσία, είναι η γνώση του URI της.

Για να μπορεί μια υπηρεσία Ιστού να θεωρηθεί RESTful, θα πρέπει αυτή να ικανοποιεί τους περιορισμούς της REST αρχιτεκτονικής. Οι περιορισμοί αυτοί είναι:

- *Client-server*, κάθε σύστημα που βασίζεται στην αρχιτεκτονική REST θα πρέπει να χαρακτηρίζεται από διαχωρισμό των ευθυνών (separation of concerns). Πιο συγκεκριμένα, ζητήματα όπως η αποθήκευση των δεδομένων θα πρέπει να αποτελούν έγνοια μόνο του εξυπηρετητή, ενώ ο εξυπηρετητής δε θα πρέπει να κρατάει πληροφορία σχετική με την κατάσταση στην οποία βρίσκεται ο κάθε πελάτης.
- *Uniform interface*, η διεπαφή (interface) κάθε συστατικού μέλους μιας REST αρχιτεκτονικής θα πρέπει να ακολουθεί συγκεκριμένους γενικούς κανόνες ομοιογένειας, κάτι που διευκολύνει την πρόσβαση στις υπηρεσίες τους, ενώ παράλληλα συμβάλει στη διευκόλυνση της κλιμακωσιμότητας (scalability).
- *Stateless*, κάθε αίτημα ενός client περιέχει όλες τις απαραίτητες πληροφορίες για την επεξεργασία του μηνύματος καθώς και για την κατάσταση που βρίσκεται τη συγκεκριμένη στιγμή ο client. Δεν είναι λοιπόν απαραίτητο ο server να κρατάει πληροφορίες σχετικά με την κατάσταση του κάθε client, κάτι που έχει μεγάλο αντίκτυπο στην κλιμακωσιμότητα και την αξιοπιστία του συστήματος.
- *Cacheable*, οι εξυπηρετητές επιτρέπουν κάποιες απαντήσεις τους να αποθηκευτούν στην προσωρινή μνήμη στην πλευρά του πελάτη. Αυτό συμβαίνει ιδιαίτερα σε πληροφορίες που δεν αλλάζουν με ταχείς ρυθμούς. Αντίθετα, είναι συχνό φαινόμενο το να μην επιτρέπουν την προσωρινή αποθήκευση απαντήσεων, ιδιαίτερα όταν αφορούν πληροφορίες οι οποίες έχουν μικρή διάρκεια ζωής. Με τον τρόπο αυτό, οι clients μπορούν σε ορισμένες περιπτώσεις να αποφύγουν την άσκοπη επικοινωνία με τον server, κάτι που μπορεί να βελτιώσει σε μεγάλο βαθμό την απόδοση ενός συστήματος ιδιαίτερα σε συστήματα μεγάλης κλίμακας, ενώ ταυτόχρονα προστατεύονται από τη χρήση μη επικαιροποιημένων πληροφοριών, κάτι που θα μπορούσε να οδηγήσει σε αλλοίωση δεδομένων.
- *Layered system*, στις αρχιτεκτονικές τύπου REST οι clients μπορούν να συνδεθούν είτε απευθείας στον server που παρέχει μια υπηρεσία, είτε σε ενδιάμεσους κόμβους-εξυπηρετητές δίχως να το γνωρίζουν. Αυτό συμβαίνει διότι η αρχιτεκτονική REST χρησιμοποιεί ιεραρχικά επίπεδα για την κατανομή των κόμβων που συμμετέχουν σε ένα σύστημα, όπου κάθε κόμβος μπορεί να επικοινωνεί μόνο με τους κόμβους στους οποίους βρίσκεται πλησιέστερα.
- *Code on demand* (προαιρετικό), ο κώδικας κατά απαίτηση (code on demand) είναι ο μοναδικός περιορισμός που είναι προαιρετικός για την αρχιτεκτονική REST. Πέρα από τις

ζητούμενες πληροφορίες και τις αναπαραστάσεις πόρων, οι server σε κάποιες περιπτώσεις μπορούν να προσφέρουν και εκτελέσιμο κώδικα στον client, με τις πιο συνήθης υλοποιήσεις να αφορούν κώδικα σε JavaScript.

2.8.1 Εντολές HTTP

Η αρχιτεκτονική REST χρησιμοποιεί τέσσερα βασικά ρήματα (εντολές) με τα οποία ένας client μπορεί να ζητήσει μία υπηρεσία.

GET

Η μέθοδος GET χρησιμοποιείται για την ανάκτηση resources από το συγκεκριμένο server χρησιμοποιώντας ένα συγκεκριμένο URI. Οι αιτήσεις τύπου GET θα πρέπει μόνο να ανακτούν τα δεδομένα και να μην έχουν καμία επίδραση πάνω τους. Μία αίτηση GET μπορεί να γίνει υπό όρους ώστε να αποφευχθεί η άσκοπη μεταφορά δεδομένων προς τον πελάτη.

POST

Κάθε μέθοδος POST επιτρέπεται την δημιουργία ενός στιγμιότυπου ενός resource. Υπάρχει όμως η πιθανότητα να μην σταλεί η πληροφορία με το αίτημα, όπως επίσης να μην μπορούν να ληφθούν πληροφορίες από την απάντηση. Τα προς αποστολή δεδομένα δεν προσκολλώνται στο URL παρά τοποθετούνται στο HTTP Request Body

PUT

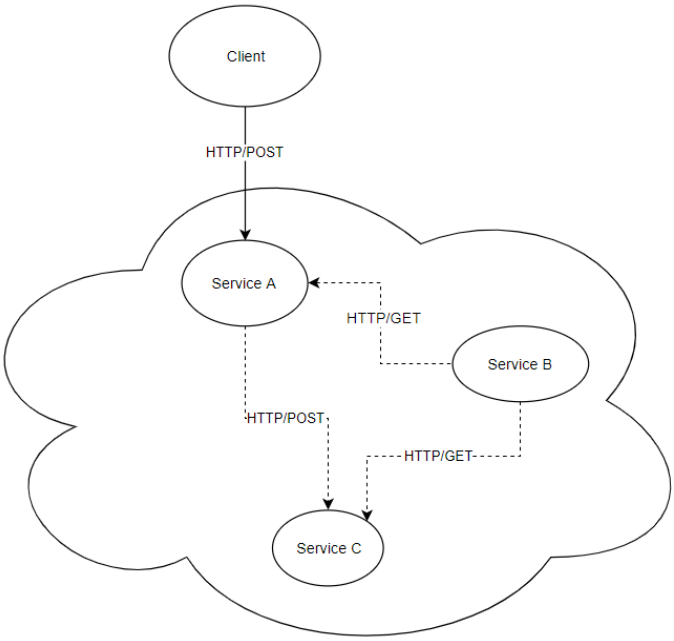
Η μέθοδος PUT ζητά από τον server να αντικαταστήσει ένα ήδη υπάρχον resource με αυτό του μηνύματος. Παρόμοια με το ρήμα POST, τα δεδομένα υπάρχουν στο HTTP Request Body. Όταν γίνεται μια αίτηση PUT, ο πελάτης γνωρίζει την ταυτότητα του πόρου προς δημιουργία ή προς ενημέρωση. Η αποστολή του ίδιου μηνύματος περισσότερες από μία φορά δεν θα έχει καμία επίδραση επί της υποκείμενης υπηρεσίας.

2.9 Μικροϋπηρεσίες (microservices)

Έχοντας περιγράψει τους τρόπους με τους οποίους μία οντολογία μπορεί να είναι προσβάσιμη μέσω του SPARQL πρωτοκόλλου και πως η αρχιτεκτονική REST επιτρέπει την επικοινωνία ενός client με κάποιον server, στη συνέχεια θα περιγραφεί η αρχιτεκτονική των μικροϋπηρεσιών (microservices), η οποία θα επιτρέψει την δυναμική και ανεξάρτητη πρόσβαση της οντολογίας από πιθανούς clients.

Τα microservices είναι μία αρχιτεκτονική στην πλευρά του server, που επιτρέπει τον διαχωρισμό μίας εφαρμογής σε ένα σύνολο μικρών και ανεξάρτητων υπηρεσιών. Κάθε μία από αυτές εκτελείται σε δική της αυτόνομη διεργασία. Αυτές, μπορούν να επικοινωνούν μέσω κάποιου ελαφριού μηχανισμού (συνήθως HTTP). Επίσης, αυτές οι υπηρεσίες μπορούν να υλοποιηθούν αυτόνομα, καθώς και η διαχείριση αυτών είναι μία ξεχωριστή υπηρεσία. Έχουν, ακόμη τη

δυνατότητα, να υλοποιηθούν σε διαφορετικές γλώσσες και να χρησιμοποιήσουν διαφορετικά μοντέλα δεδομένων (Newman, 2016).



Εικόνα 12 Microservices Topology

2.9.1 Μονολιθική αρχιτεκτονική vs Microservices

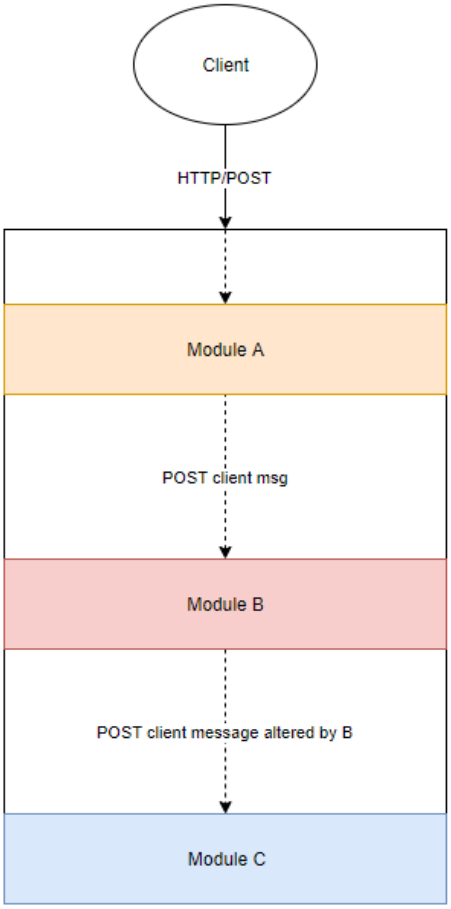
Τα πλεονεκτήματα που μας προσφέρει η αρχιτεκτονική των microservices γίνονται περισσότερο κατανοητά, μετά από σύγκριση της με την κλασική μονολιθική αρχιτεκτονική (Dragoni, et al., 2017).

Μονόλιθος

Προτού εμφανιστούν τα microservices, η αρχιτεκτονική που επικρατούσε στα κατανεμημένα συστήματα ήταν ο *μονόλιθος* (monolith). Οι εφαρμογές που ακολουθούν την αρχιτεκτονική αυτή έχουν όλα τα επιμέρους κομμάτια τους ενωμένα άρρηκτα. Αυτό επιφέρει αρκετά προβλήματα όπως,

- Δυσκολία στην επεκτασιμότητα, καθώς τα επιμέρους κομμάτια τους είναι ενωμένα αυτό οδηγεί στην δυσκολία ενσωμάτωσης νέας λειτουργίας.
- Δυσκολία στην διατηρησιμότητα της εφαρμογής, καθώς μονολιθικές εφαρμογές τείνουν να έχουν αυξημένη πολυπλοκότητα.
- Δυσκολία στην κλιμακωσιμότητα, καθώς η κατανομή των πόρων μπορεί να είναι άνιση μεταξύ των κομματιών. Αυτό σημαίνει ότι τα κομμάτια τα οποία χρειάζονται περισσότερους πόρους μπορούν να οδηγήσουν στην στέρηση των πόρων από τα υπόλοιπα.

- Υπόκεινται στο επονομαζόμενο “dependency hell”, όπου η διαχείριση των εκδόσεων των βιβλιοθηκών μπορεί να προκαλέσει προβλήματα σε ένα ή περισσότερα κομμάτια, την κατάρρευση της εφαρμογής.
- Κάθε νέα αλλαγή σε ένα κομμάτι της εφαρμογής σημαίνει την επανεκκίνηση της σε όσους server έχει αναπτυχθεί.
- Τέλος επιβάλλουν τις τεχνολογίες που θα χρησιμοποιηθούν για την εφαρμογή εφόρου ζωής.



Εικόνα 13 Μονόλιθος

Πλεονεκτήματα microservices

Τα προβλήματα αυτά λύνονται με την αρχιτεκτονική των microservices καθώς κάθε μικρο-υπηρεσία είναι ανεξάρτητη από τις υπόλοιπες. Εκτός αυτών η χρήση των microservices επιφέρουν πλεονεκτήματα όπως,

- Ευκολία στην μετάβαση σε νεότερες εκδόσεις των υπηρεσιών. Καθώς κάθε υπηρεσία είναι αυτόνομη, οι προγραμματιστές μπορούν για ένα συγκεκριμένο χρονικό διάστημα να έχουν

δύο εκδόσεις μίας υπηρεσίας προσβάσιμες από άλλες υπηρεσίες ή clients. Με τον τρόπο αυτό επιτυγχάνεται η ευκολότερη διατηρησιμότητα της εφαρμογής ενώ ταυτόχρονα επιτρέπει την συνέχιση της λειτουργίας των υπολοίπων υπηρεσιών.

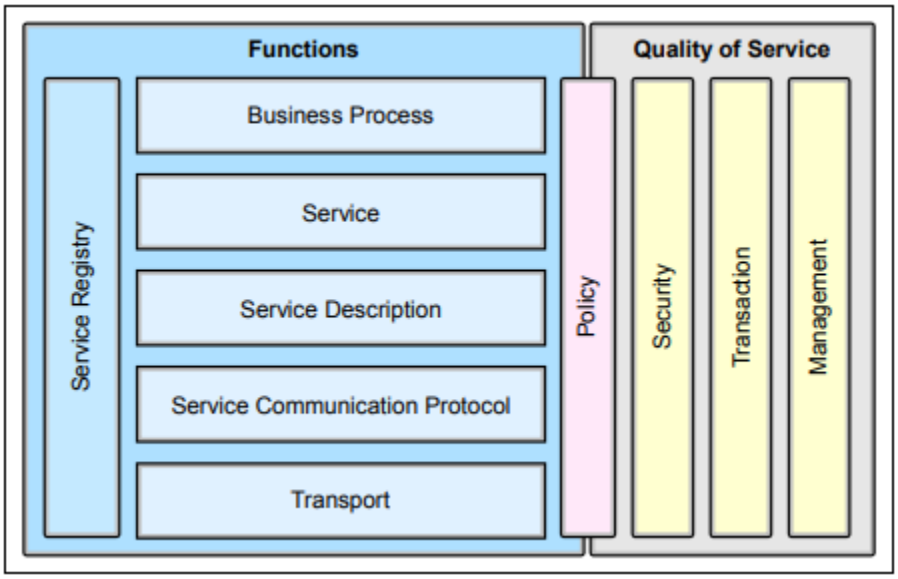
- Η αλλαγή μίας υπηρεσίας δεν επιφέρει επανεκκίνηση όλων των υπολοίπων αλλά μόνο αυτής. Καθώς μόνο μία υπηρεσία υπόκειται σε επανεκκίνηση, ο χρόνος που δεν είναι προσβάσιμη είναι ελάχιστος συγκριτικά με αυτόν του μονόλιθου.
- Τα microservices υποστηρίζουν την ανεξάρτητη ανάπτυξή τους σε κάποιον server. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δημιουργήσουν διαφορετικούς τρόπους, ή να επαναχρησιμοποιήσουν άλλους, με τους οποίους κάποιο microservice μπορεί να γίνει deployed σε κάποιον server.
- Κάθε microservice μπορεί να κλιμακωθεί αυτόνομα και ανεξάρτητα από τα υπόλοιπα. Με τον τρόπο αυτό οι προγραμματιστές μπορούν να καθορίσουν τον αριθμό των πόρων ανά microservice.
- Καθώς κάθε microservice μπορεί να γίνει deployed ανεξάρτητα από τα υπόλοιπα, συνεπάγεται η δυνατότητα της χρήσης διαφορετικών τεχνολογιών για την ανάπτυξή τους. Ο μόνος περιορισμός που προκύπτει είναι ότι οι τεχνολογίες που θα χρησιμοποιηθούν να υποστηρίζουν την χρήση του HTTP πρωτοκόλλου.

2.9.2 SOA vs Microservices

Ο τρόπος με τον οποίο τα microservices επικοινωνούν μεταξύ τους, περιεγράφηκε για πρώτη φόρα στην αρχιτεκτονική Service-Oriented Architecture (SOA) (Endrei, et al., 2004), ως αποτέλεσμα τα microservices να αποτελούν μία επέκταση της SOA.

SOA

Η αρχιτεκτονική SOA διαχωρίζεται σε δύο βασικά μέρη. Το πρώτο μέρος αποτελείται από τις λειτουργικές δυνατότητες της εφαρμογής ενώ το δεύτερο από την ποιότητα αυτών των δυνατοτήτων.



Εικόνα 14 SOA stack (Endrei, et al., 2004)

Η αρχιτεκτονική αυτή υποστηρίζει αυτόνομες υπηρεσίες, με σκοπό την επαναχρησιμοποίησή τους. Αυτό οδηγεί στην μείωση του κόστους και χρόνου όταν μία εφαρμογή υλοποιείται εκ νέου. Οι υπηρεσίες αυτές μπορούν να επικοινωνήσουν μεταξύ τους μέσω διαφόρων πρωτοκόλλων, ασύγχρονα με στόχο την παράλληλη επεξεργασία από ετερογενή συστήματα. Καθώς μπορούν να χρησιμοποιηθούν διαφορετικά πρωτόκολλα επικοινωνίας από κάθε υπηρεσία, αυτό οδηγεί σε δυσκολία ανακάλυψης των διαθέσιμων υπηρεσιών. Επίσης μπορεί να προκαλέσει προβλήματα κατά την υλοποίηση μίας υπηρεσίας καθώς αυτόματα αυξάνεται η πολυπλοκότητα της. Ένα ακόμα θετικό στοιχείο της αρχιτεκτονικής αυτής είναι η εύκολη ευθυγράμμιση της επιχειρησιακής στρατηγικής με την πληροφορική. Η αρχιτεκτονική αυτή επιτρέπει εύκολη και γρήγορη προσαρμογή ενός πληροφοριακού συστήματος με βάση την στρατηγική της εκάστοτε επιχείρησης. Με τον τρόπο αυτό δίνεται η δυνατότητα στον επιχειρηματικό κλάδο να αλλάξει τη στρατηγική που ακολουθείται χωρίς η αλλαγή αυτή να επιφέρει κόστη για την αντίστοιχη αλλαγή των πληροφοριακών συστημάτων.

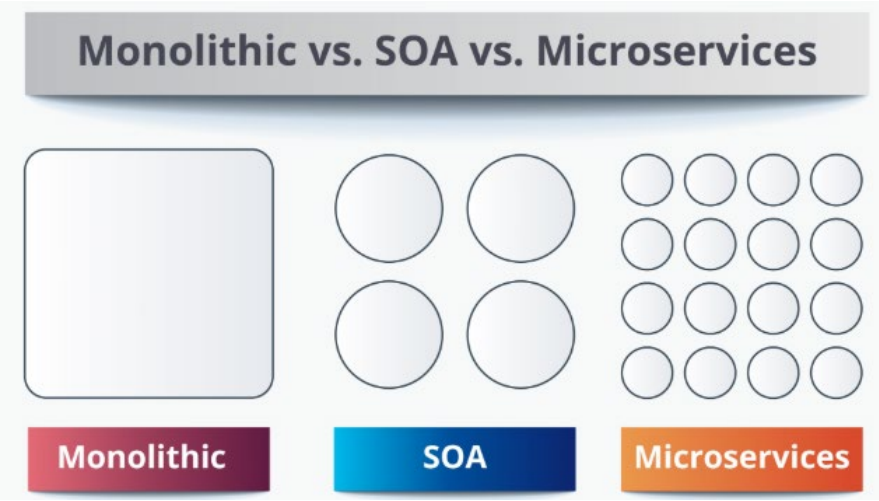
Πλεονεκτήματα microservices

Πολλά από τα προτερήματα των SOA εμφανίζονται και στην αρχιτεκτονική των microservices, το οποίο είναι λογικό καθώς η δεύτερη αποτελεί επέκταση της πρώτης. Η διαφορά μεταξύ των δύο αυτών αρχιτεκτονικών είναι πολύ μικρή και κυρίως περιστρέφεται γύρω από το μέγεθος ευθύνης μίας υπηρεσίας, εξού και η ονομασία micro-service (Dragoni, και συν., 2017). Συγκεκριμένα,

- Το μέγεθος μίας υπηρεσίας είναι αρκετά μικρό συγκριτικά με αυτό μίας υπηρεσίας που υπόκειται στην αρχιτεκτονική SOA. Στην αρχιτεκτονική των microservices κάθε υπηρεσία στοχεύει στην υλοποίηση μίας επιχειρηματικής δυνατότητας, ενώ στην αρχιτεκτονική

SOA μία υπηρεσία είναι μια υλοποίηση μιας ολοκληρωμένης επιχειρηματικής λειτουργίας.

- Στην αρχιτεκτονική των microservices παρόμοιες λειτουργικότητες ομαδοποιούνται σε μια επιχειρηματική δυνατότητα, η οποία στη συνέχεια υλοποιείται ως μία υπηρεσία
- Τα microservices εξυψώνουν την ευελιξία των πληροφοριακών συστημάτων στις νέες επιχειρηματικές στρατηγικές καθώς αποτελούνται από μικρές επιχειρηματικές δυνατότητες.
- Καθώς κάθε επιχειρηματική δυνατότητα έχει μικρό πεδίο δράσης η διατηρησιμότητα μίας υπηρεσίας είναι εύκολη επιτρέποντας έτσι την εύκολη και γρήγορη ανέλιξή της.



Εικόνα 15 Monolithic vs SOA vs Microservices

3

Τεχνολογίες

Μέχρι στιγμής μελετήθηκε τι είναι μία Οντολογία και πως μέσα από ένα πλήθος οντοτήτων και των αναμεταξύ τους συσχετίσεων είναι εφικτός ο συμπερασμός νέας γνώσης. Με την χρήση ενός αλγορίθμου, που βασίζεται σε κάποια περιγραφική λογική, είναι εφικτοί τέτοιοι συμπερασμοί από ένα υπαρκτό πλήθος οντοτήτων. Επίσης είδαμε ότι με τη χρήση της **SPARQL** μπορούν να δημιουργηθούν ερωτήσεις που στοχεύουν την βάση γνώσης μίας οντολογίας. Με το **SPARQL** πρωτόκολλο οι ερωτήσεις αυτές μπορούν να γίνουν απομακρυσμένα χρησιμοποιώντας το πρωτόκολλο HTTP ως μέσω επικοινωνίας. Τέλος μελετήθηκε η Microservices αρχιτεκτονική η οποία βασίζεται στην ανταλλαγή μηνυμάτων HTTP, η οποία παρέχει ανεξάρτητες υπηρεσίες.

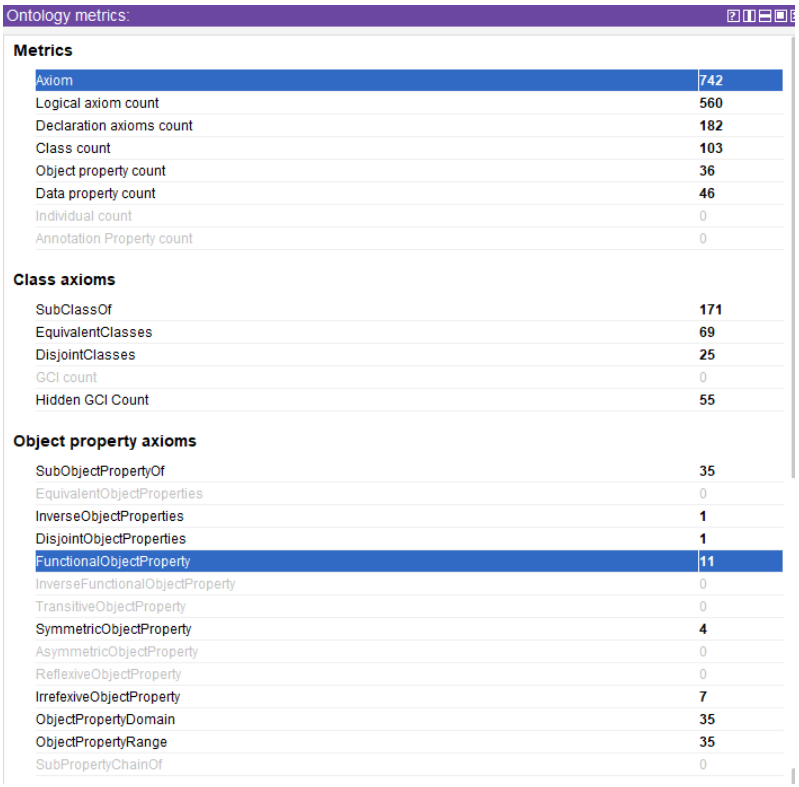
Στο κεφάλαιο αυτό θα μελετηθούν τα εργαλεία τα οποία χρησιμοποιήθηκαν για την επίτευξη της δημιουργίας μία οντολογίας, για την ενσωμάτωση του **SPARQL** πρωτοκόλλου, καθώς και κάποια frameworks που θα χρησιμοποιηθούν για την ανάπτυξη ενός server που θα επιτρέπει την επικοινωνία μεταξύ ενός client και μίας οντολογίας.

3.1 Turtle-Syntax

Στο κεφάλαιο 2 μελετήθηκε η **OWL** ως κύρια γλώσσα μίας οντολογίας. Συνοπτικά ο λόγος που την καθιστά ως ιδανική γλώσσα είναι η μεγάλη εκφραστικότητα που την διακρίνει και μπορεί να περιγράψει πλήρως μία οντολογία. Καθώς μία οντολογία περιγράφεται μέσα από ένα σύνολο τριπλέτων, το τελικό αποτέλεσμα αποτελεί ένα αρχείο το οποίο περιέχει το σύνολο των τριπλέτων αυτών. Ένα τέτοιο αρχείο μπορεί να γίνει χαοτικό είναι σημαντικό να επιλεχθεί το σωστό format το οποίο επιτρέπει τη μείωση του όγκου. Το πιο διαδεδομένο format που επιτυγχάνει κάτι τέτοιο είναι το Turtle-Syntax (Beckett, Berners-Lee, Prud'hommeaux, Carothers, & Machina, 2014).

3.2 Protégé

Για την δημιουργία της οντολογίας χρησιμοποιείται το Protégé (5.5.0). Το Protégé είναι ένα λογισμικό το οποίο επιτρέπει τη δημιουργία μιας **OWL** οντολογίας και υποστηρίζει τις δυνατότητες της **OWL** γλώσσας. Καθώς η **OWL** διαθέτει 5 διαφορετικά προφίλ, το Protégé καλείται να καλύψει όλα τα προφίλ. Αντ' αυτού καθώς η OWL Full είναι το προφίλ που διαθέτει όλα τα χαρακτηριστικά της **OWL**, το Protégé δημιουργεί οντολογίες με βάση το προφίλ αυτό. Το ποιο προφίλ θα χρησιμοποιηθεί για το reasoning αφήνεται στην κρίση του χρήστη. Επίσης διαθέτει κάποιους reasoners οι οποίοι ελέγχουν το consistency της οντολογίας καθώς και το reasoning. Τέλος παρέχει χρήσιμες πληροφορίες για την οντολογία όπως το σύνολο των αξιωμάτων, δηλαδή των αυτο-αποδεικνυούμενων τιμών, που υπάρχουν στην οντολογία.


Ontology metrics:

Metrics	
Axiom	742
Logical axiom count	560
Declaration axioms count	182
Class count	103
Object property count	36
Data property count	46
Individual count	0
Annotation Property count	0
Class axioms	
SubClassOf	171
EquivalentClasses	69
DisjointClasses	25
GCI count	0
Hidden GCI Count	55
Object property axioms	
SubObjectPropertyOf	35
EquivalentObjectProperties	0
InverseObjectProperties	1
DisjointObjectProperties	1
FunctionalObjectProperty	11
InverseFunctionalObjectProperty	0
TransitiveObjectProperty	0
SymmetricObjectProperty	4
AsymmetricObjectProperty	0
ReflexiveObjectProperty	0
IrrefexiveObjectProperty	7
ObjectPropertyDomain	35
ObjectPropertyRange	35
SubPropertyChainOf	0

Εικόνα 16 Πληροφορίες οντολογίας για τα παίγνια

3.2.1 Δημιουργία Οντοτήτων

Τα βήματα για την δημιουργία μίας οντότητας είναι απλά και το γραφικό περιβάλλον του Protégé καθοδηγεί τον χρήστη σε κάθε βήμα. Όπως είδαμε η OWL υποστηρίζει την ιεραρχία των οντοτήτων-κλάσεων. Το Protégé επιτρέπει την ιεραρχία αυτή καθώς και την κληρονομικότητα που μπορεί να υπάρχει μεταξύ δύο οντοτήτων. Επίσης παρέχει την δυνατότητα δημιουργίας συσχετίσεων - περιορισμών. Για παράδειγμα έχουμε ορίσει την οντότητα *EasyAntonyms* ως υποκλάση της οντότητας *Antonyms* με περιορισμό ότι έχει ακριβώς 3 αντώνυμες λέξεις. Ο περιορισμός αυτός διαφοροποιεί την οντότητα αυτή από την *MediumAntonyms* η οποία έχει 4 αντώνυμες λέξεις. Στο γραφικό περιβάλλον του Protégé μπορούμε να δούμε τους περιορισμούς αυτούς για κάθε επιλεγμένη οντότητα είτε να δημιουργήσουμε νέους.

Η σύνταξη των περιορισμών αυτών καθώς και των συσχετίσεων που κληρονομούνται είναι απλή. Για μία επιλεγμένη οντότητα το Protégé απαριθμεί τις συσχετίσεις ως εξής,

<όνομα συσχέτισης> <καρδιότητα> <τιμή καρδιότητας (προαιρετικό)> <range συσχέτισης>

Για παράδειγμα όταν επιλεχθεί η οντότητα EasyAntonyms έχουμε τον εξής περιορισμό:

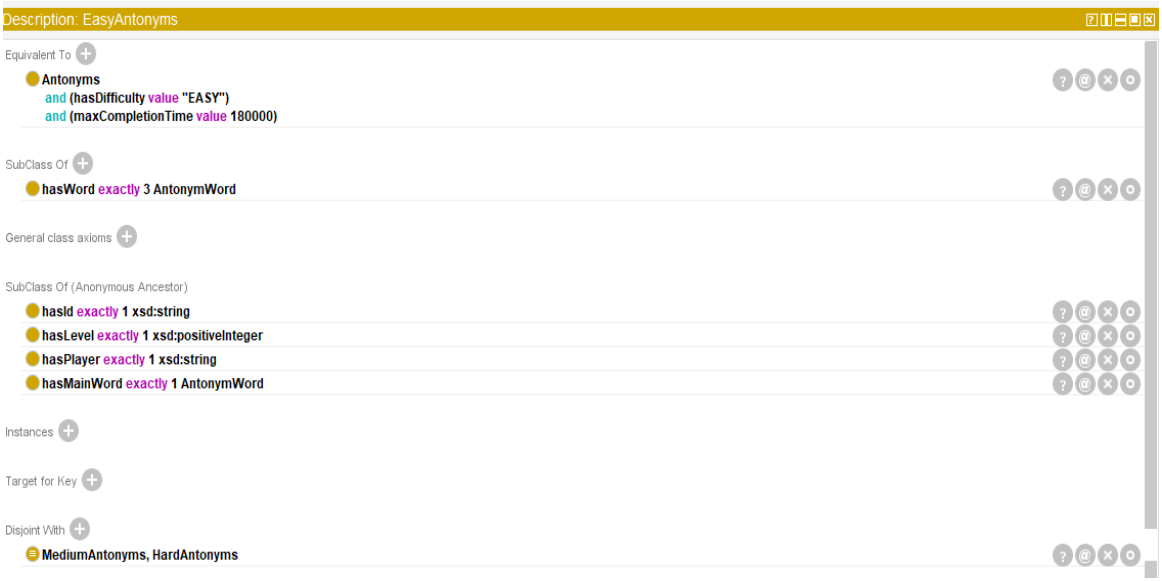
- hasWord exactly 3 AntonymWord

ενώ κληρονομεί τους παρακάτω:

- hasId exactly 1 xsd: string
- hasLevel exactly 1 xsd: positiveInteger
- hasPlayer exactly 1 xsd: string
- hasMainWord exactly 1 AntonymWord

Ένα άλλο χαρακτηριστικό που έχει η **OWL** είναι η δήλωση μίας οντότητας ως Equivalent ή Complement άλλων οντοτήτων. Το Protégé παρέχει τη δυνατότητας έκφρασης μίας τέτοιας συσχέτισης κατά την δημιουργία μίας οντότητας.

Τέλος δίνεται η δυνατότητα της σημείωσης της οντότητα ως disjoint με άλλες οντότητες. Αυτό σημαίνει ότι τα individuals της οντότητας αυτής ανήκουν μόνο στην οντότητα αυτή, ή αλλιώς η τομή της οντότητας αυτής με άλλες είναι το κενό σύνολο. Ένα παράδειγμα δημιουργίας μίας οντότητας με το Protégé φαίνεται στην Εικόνα 17



Εικόνα 17 Δημιουργία Οντότητας μέσω του Protégé

3.2.2 Δημιουργία Συσχετίσεων

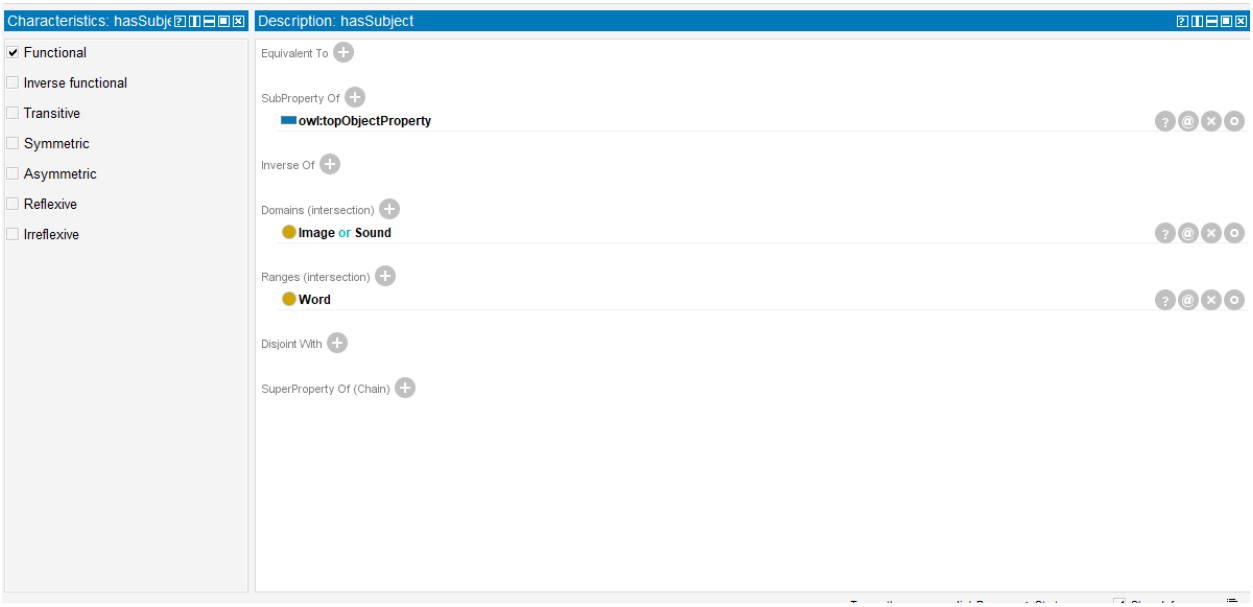
Παρόμοια με τα βήματα για τη δημιουργία μίας οντότητας, το Protégé καθοδηγεί τον χρήστη για την δημιουργία μίας συσχέτισης. Το Protégé υποστηρίζει τους δύο διαφορετικούς τύπους συσχετίσεων της **OWL**, τα **ObjectProperties**, τα οποία έχουν ως range κάποια άλλη οντότητα, και τα **DatatypeProperties** τα οποία έχουν ως range κάποιους βασικούς τύπους δεδομένων όπως έναν αριθμό.

Τέλος με την **OWL** είναι εφικτή η σήμανση της πληροφορίας κάθε συσχέτισης, όπως για παράδειγμα αν κάθε οντότητα μπορεί να έχει το πολύ μία συσχέτιση με κάποιο άλλη οντότητα μέσω αυτής της συσχέτισης. Στην Ενότητα 2.5 OWL έχουν περιγραφεί οι ιδιότητες των συσχετίσεων. Στο Protégé δίνεται εύκολα η δυνατότητα στον χρήστη να υποσημειώσει τις

ιδιότητες που επιθυμεί να έχει η συσχέτιση που δημιουργεί. Οι ιδιότητες αυτές είναι οι ίδιες και στους δύο τύπους των συσχετίσεων. Επιπλέον, κάποιες ιδιότητες δεν είναι συμβατές μεταξύ τους, όπως για παράδειγμα Functional και Inverse Functional. Το Protégé επιτρέπει τον λογικό έλεγχο τέτοιων περιπτώσεων και ειδοποιεί αντίστοιχα των χρήστη σε περιπτώσεις παραβίασης.

ObjectProperties

Με τη δημιουργία μίας νέας συσχέτισης δίνεται η δυνατότητα συμπλήρωσης των ιδιοτήτων της όπως αναφέρθηκαν στην Ενότητα 2.5 OWL καθώς και όλα τα **domains** και **ranges** της ιδιότητας αυτής όπως έχουν ορισθεί στην Ενότητα 2.4.2 RDFS. Για παράδειγμα, η Εικόνα 18περιγράφει την συσχέτιση *hasSubject* να έχει ως **domain** κάποιο στιγιμιότυπο το οποίο είναι είτε *Image* είτε *Sound* και ως **range** ένα στιγιότυπο *Word*.



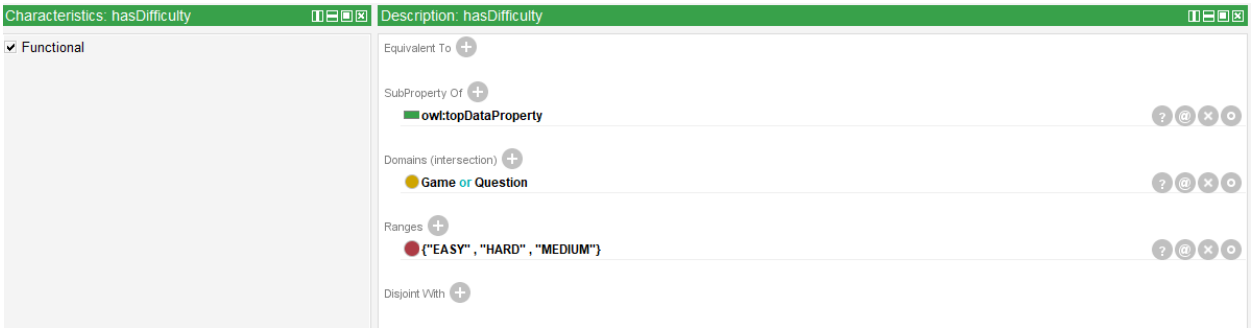
Εικόνα 18 Δημιουργία ObjectProperty μέσω του Protégé

Κατά τη δημιουργία της οντολογίας παρατηρήθηκε ότι δεν είναι ευνοϊκό να υπάρχει συσχέτιση της οποίας το **domain** και το **range** να απαρτίζονται από πολλαπλές οντότητες, δηλαδή να υπάρχει μία συσχέτιση m προς n μεταξύ του **domain** και του **range** , καθώς καθίσταται δύσκολο να γίνουν αντιλληπτές οι ακριβείς συσχετίσεις μεταξύ των οντοτήτων. Αυτό σημαίνει ότι αν η συσχέτιση τηςΕικόνα 18, *hasSubject* έχει ως **range** *Word* OR *Image* τότε μπορεί να διαπιστωθεί ότι μία *Image* οντότητα *hasSubject* μια άλλη *Image* οντότητα το οποίο στα πλαίσια της εργασίας δεν είναι κάποια επιθυμητή κατάσταση.

DatatypeProperties

Για τα **DatatypeProperties** κάποιες φορές οι τιμές που μπορούν να δοθούν ως **range** πρέπει να ανήκουν σε ένα συγκεκριμένο πεδίο τιμών. Επίσης δίνει την δυνατότητα χρήσης του

Manchester Syntax (Horridge & Patel-Schneider, 2008) όταν είναι αναγκαίο να περιοριστούν οι διαθέσιμες τιμές. Το **Mancester Syntax** είναι ένας λακωνικός τρόπος περιγραφής των τιμών που μπορούν να δηλώθούν ως range. Στην Εικόνα 19, για το **dataPorperty** *hasDifficulty* μπορούμε να περιορίσουμε το πλήθος των τιμών στις τιμές *"EASY, MEDIUM, HARD"*. Χωρίς το **Mancester Syntax** θα έπρεπε να περιγράψουμε τις τιμές αυτές ως εξείς, *hasDifficulty exactly 1 xsd:String AND hasDifficulty value ("EASY" OR "MEDIUM" OR "HARD")*



Εικόνα 19 Δημιουργία DatatypeProperty με enumerated Range μέσω του Protégé

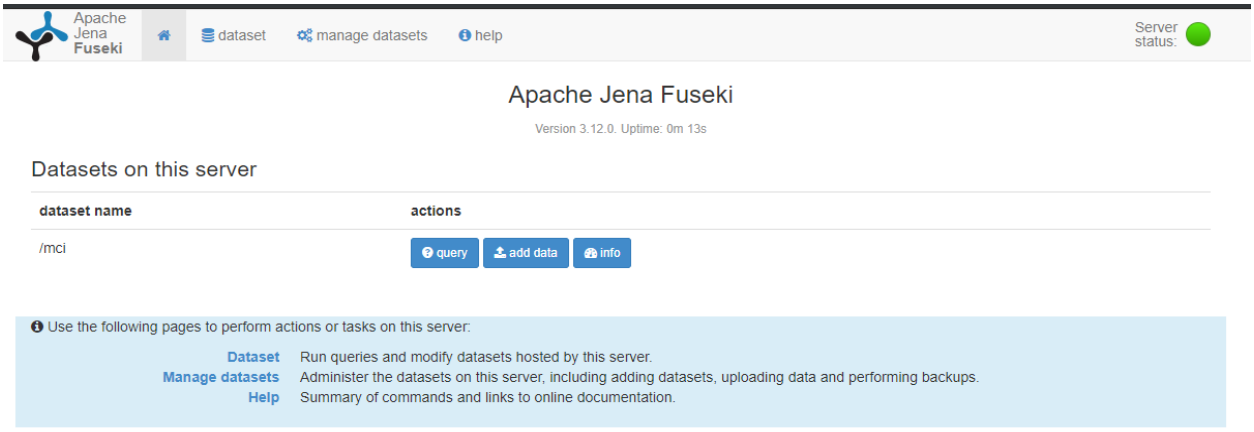
3.3 Pellet Reasoner

Μία οντολογία μπορεί να περιέχει ένα μεγάλο πλήθος οντοτήτων και συσχετίσεων τα οποία μπορούν να αλληλοεπιδρούν μεταξύ τους. Κάποιες φορές λανθασμένα ή μη λογικά. Για τον λόγο αυτό χρειάζεται ένας μηχανισμός - αλγόριθμος ο οποίος να εξασφαλίζει μια οντολογία να είναι «συνεχής» και λογική. Ο μηχανισμός – αλγόριθμος αυτός ονομάζεται *Reasoner*. Το Protégé διαθέτει ένα μεγάλο πλήθος reasoners. Κατά τη προσπάθεια να βρεθεί ο κατάλληλος reasoner παρατηρήθηκε ότι ο Pellet (Sirin, Parsia, Grau, Kalyanpur, & Katz, 2007) μπορεί να χρησιμοποιηθεί τόσο εντός του Protégé με την μορφή plugin, όσο και εκτός, στην υλοποίηση του server, σε μορφή maven dependency. Αυτός είναι ο μόνος λόγος που επιλέχθηκε ο Pellet reasoner, καθώς διαφορετικά δεν έχει κάποιο πλεονέκτημα έναντι των υπολοίπων (Bock, Ji, Haase, & Volz, 2008) .

3.4 Apache Fuseki

Όπως αναφέρθηκε προηγουμένως μία οντολογία μπορεί να ερωτηθεί απομακρυσμένα χρησιμοποιώντας το πρωτόκολλο SPARQL. Ο Apache Fuseki είναι ένας server, βασισμένος στον Tomcat server, ο οποίος υποστηρίζει το **SPARQL** Protocol όπως αναπτύχθηκε παραπάνω. Ο Apache Fuseki εκτός από SPARQL Protocol provider μπορεί να λειτουργήσει και ως μία βάση οντολογίας, δηλαδή ως μία βάση η οποία περιέχει όλα τα individuals και τις αναμεταξύ τους συσχετίσεις. Επίσης δίνει τη δυνατότητα εκτέλεσης ερωτήσεων **SPARQL** απευθείας στη βάση, επιτρέποντας έτσι την γρήγορη και άμεση αποσφαλμάτωση της βάσης. Η βάση που

χρησιμοποιείται δεν είναι μία παραδοσιακή SQL βάση δεδομένων, αλλά όπως ήδη έχει αναφερθεί, ένα σύνολο τριπλετών της μορφής: οντότητα – συσχέτιση – οντότητα/τιμή. Ένα είδος βάσης που μπορεί να αποθηκεύει τέτοιες τριπλέτες ή και τους γράφους αυτών είναι η TDB (Apache, 2019).



Εικόνα 20 Apache Fuseki κεντρική σελίδα

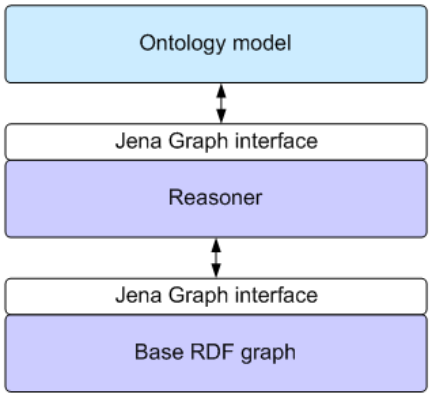
Εκτός από standalone server, ο Apache Fuseki μπορεί να χρησιμοποιηθεί ως ένα service ή και ως ένα web application.

3.5 Apache Jena

Ο Apache Fuseki κάνει χρήση του framework Apache Jena για να διεκπεραιώσει τις αντίστοιχες λειτουργίες όπως να κάνει INSERT μία τριπλέτα στην βάση γνώσης. Το Apache Jena είναι ένα πλούσιο framework το οποίο αποτελείται από τρία διαφορετικά APIs. Το πρώτο είναι το RDF API το οποίο υποστηρίζει τη δημιουργία και την ανάγνωση RDF εγγράφων. Το δεύτερο API αφορά τις λειτουργίες που χρησιμοποιούνται από τον Apache Fuseki, καθώς επίσης και την προγραμματιστική δημιουργία μίας TDB βάση γνώσης. Τέλος, το τρίτο API, και το οποίο θα αναπτυχθεί για την συγκεκριμένη εργασία, είναι το OWL API. Το API αυτό προσφέρει εργαλεία και μεθόδους για την ανάγνωση και την εγγραφή κανόνων σε μία ήδη υπάρχουσα οντολογία.

Το OWL API, εκτός της ανάγνωσης και της εγγραφής οντοτήτων και των συσχετίσεων τους, επιτρέπει και την εκτέλεση των επιθυμητών reasoners. Αυτό επιφέρει μεγαλύτερη ευρωστία κατά τη διάρκεια της υλοποίησης μίας εφαρμογής, καθώς επιτρέπει στον προγραμματιστή να χρησιμοποιήσει inferred πληροφορία on demand. Ο τρόπος με τον οποίο αυτό επιτυγχάνεται, γίνεται κατά τη διάρκεια που το μοντέλο μίας οντολογίας «φορτώνεται» στην μνήμη. Εάν έχει επιλεχθεί κάποιος reasoner τότε το αποτέλεσμα θα περιέχει τις τριπλέτες που υπάρχουν στο αρχικό μοντέλο, συν τις τριπλέτες οι οποίες δημιουργήθηκαν κατά την εκτέλεση του reasoner. Με τον

τρόπο αυτό, η χρήση ή μη ενός reasoner δεν αλλάζει τον τρόπο με τον οποίο μία εφαρμογή μπορεί να καταναλώσει το αποτέλεσμα.



Εικόνα 21 Παραγόμενο μοντέλο του OWL API

Εφόσον μπορεί να εμφανιστεί κάποιος reasoner κατά τη διάρκεια της δημιουργίας ενός μοντέλου, κάποιος μπορεί να αναρωτηθεί εάν είναι εφικτή η επιλογή του προφίλ που επιθυμεί να χρησιμοποιήσει. Το OWL API παρέχει τη δυνατότητα επιλογής μέσα από τρία διαφορετικά OWL προφίλ (OWL Full, OWL DL, OWL Lite) κατά τη διάρκεια της «φόρτωσής» του στη μνήμη. Επίσης περιέχει κάποιους reasoners out-of-the-box οι οποίοι μπορούν να χρησιμοποιηθούν χωρίς ο χρήστης να βρει κάποιον διαφορετικό. Η Εικόνα 22 περιέχει τον πίνακα με όλες τις out-of-the-box ρυθμίσεις και σε κάποιες περιπτώσεις και ο default reasoner που χρησιμοποιείται.

OntModelSpec	Language profile	Storage model	Reasoner
OWL_MEM	OWL full	in-memory	none
OWL_MEM_TRANS_INF	OWL full	in-memory	transitive class-hierarchy inference
OWL_MEM_RULE_INF	OWL full	in-memory	rule-based reasoner with OWL rules
OWL_MEM_MICRO_RULE_INF	OWL full	in-memory	optimised rule-based reasoner with OWL rules
OWL_MEM_MINI_RULE_INF	OWL full	in-memory	rule-based reasoner with subset of OWL rules
OWL_DL_MEM	OWL DL	in-memory	none
OWL_DL_MEM_RDFS_INF	OWL DL	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_DL_MEM_TRANS_INF	OWL DL	in-memory	transitive class-hierarchy inference
OWL_DL_MEM_RULE_INF	OWL DL	in-memory	rule-based reasoner with OWL rules
OWL_LITE_MEM	OWL Lite	in-memory	none
OWL_LITE_MEM_TRANS_INF	OWL Lite	in-memory	transitive class-hierarchy inference
OWL_LITE_MEM_RDFS_INF	OWL Lite	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_LITE_MEM_RULES_INF	OWL Lite	in-memory	rule-based reasoner with OWL rules

Εικόνα 22 Υποστηριζόμενα OWL Profiles

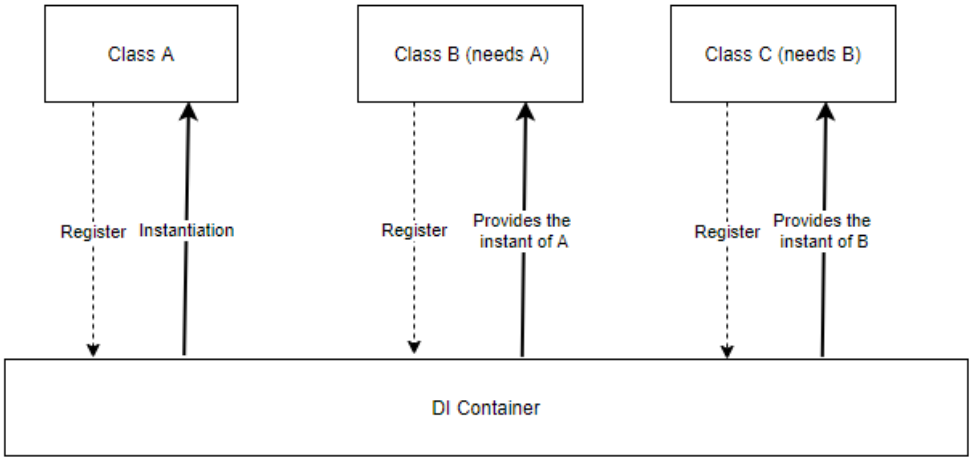
Αξίζει να παρατηρηθεί ότι όλα τα προφίλ χρησιμοποιούν την memory του υπολογιστικού συστήματος. Αυτό συμβαίνει διότι όπως ειπώθηκε στο προηγούμενο κεφάλαιο, ένα τέτοιο μοντέλο έχει υψηλή πολυπλοκότητα με αποτέλεσμα ο χρόνος εκτέλεσης ενός reasoner μπορεί να

είναι μεγάλος. Εάν κατά την λειτουργία μίας εφαρμογής, το μοντέλο διαβάζεται/εγγράφεται από κάποιο filesystem, τότε θα πρέπει να εκτελεστούν λειτουργίες synchronization για την προσπέλαση του μοντέλου. Γεγονός που μπορεί να οδηγήσει σε concurrency προβλήματα τα οποία προσθέτουν στην ήδη υπάρχουσα χρονοβόρα πολυπλοκότητα. Από την άλλη μεριά, η επιλογή να αποθηκεύεται το μοντέλο στην μνήμη οδηγεί σε χρονοβόρες επανεκκινήσεις της εφαρμογής, καθώς σε κάθε εκκίνηση το μοντέλο πρέπει να δημιουργηθεί εκ νέου (Apache, 2019).

3.6 Spring MVC

Όσον αφορά την ανάπτυξη της εφαρμογής χρησιμοποιείται το Spring MVC Framework. Το Spring είναι ένα framework το οποίο χρησιμοποιεί Dependency Injection για την δημιουργία singleton στιγμιότυπων μίας κλάσης. Επίσης επιτρέπει την δημιουργία ενός Web API επιπέδου επάνω σε αυτά τα στιγμιότυπα, επιτρέποντας έτσι την προσπέλασή τους μέσω του πρωτοκόλλου HTTP. Τέλος υποστηρίζει όλα τα HTTP ρήματα, πράγμα που επιτρέπει την ανάπτυξη με βάση την αρχιτεκτονική REST.

Από τεχνικής άποψης, το DI είναι μια κλάση – container η οποία δημιουργεί και αποθηκεύει τα στιγμιότυπα των registered κλάσεων. Το injection γίνεται κατά τη δημιουργία των στιγμιότυπων και εφόσον η dependency - κλάση έχει ήδη δημιουργηθεί. Υπάρχουν δύο βασικά στάδια, το στάδιο του registration και το στάδιο του instantiation. Η Εικόνα 23 περιγράφει τα στάδια αυτά:



Εικόνα 23 Dependency Injection Container

Το DI Container μπορεί να υλοποιηθεί ως ένα class το οποίο δημιουργείται με το ξεκίνημα της εφαρμογής. Στη συνέχεια γίνονται register στο instance όλες οι κλάσεις που θέλουμε να χρησιμοποιήσουμε. Τέλος δημιουργούμε μία μέθοδο *create* η οποία δημιουργεί όλα τα instances

των κλάσεων σύμφωνα με το διάγραμμα της Εικόνας 23, ώστε να καλύπτονται όλα τα πιθανά dependencies μεταξύ των κλάσεων αυτών.

Εφόσον το container έχει αρχικοποιηθεί τότε μπορούμε να χρησιμοποιήσουμε τα instances των κλάσεων από το container αυτό χωρίς να χρειάζεται κάθε φορά να δημιουργούμε νέο instance της κλάσης αυτής και όλων των dependent κλάσεων. Με την τακτική αυτή αποφεύγουμε να έχουμε προβλήματα με την μνήμη όπως πιθανά memory leaks ή μεγάλο όγκο garbage κλάσεων οδηγώντας έτσι σε κατάχρηση της μνήμης.

Ανάλογα την γλώσσα προγραμματισμού υπάρχουν διαφορετικές υλοποιήσεις DI container με διαφορετικές δυνατότητες. Όπως αναφέρθηκε στα πλαίσια της εργασίας επιλέχθηκε ως γλώσσα προγραμματισμού η Java. Το *Spring* είναι ένα DI Framework το οποίο υλοποιεί το container όπως είδαμε παραπάνω καθώς επίσης παρέχει μια πληθώρα δυνατοτήτων, για παράδειγμα τα instances των registered κλάσεων μπορούν να έχουν συγκεκριμένο lifecycle (per request, singletons).

Τέλος το Dependency Injection επιτρέπει την ευκολότερη δημιουργία Unit Tests τα οποία εξασφαλίζουν την ακέραια λειτουργία της κλάσης. Αυτό γίνεται διότι η κλάση έχει σχεδιαστεί έτσι ώστε όλα τα dependencies να είναι απαραίτητα για την αρχικοποίηση της. Με τον τρόπο αυτό τα dependencies μπορούν να γίνουν mock από την Unit Test κλάση καθώς η λειτουργία τους είναι ανεξάρτητη από την λειτουργία της κλάσης που δοκιμάζεται.

4

Οντολογία MCI

Στο Κεφάλαιο 3 – Τεχνολογίες αναπτύχθηκαν τα εργαλεία που χρησιμοποιούνται για την ανάπτυξη της εφαρμογής και πως αυτά είναι συμβατά με τις τεχνολογίες που μελετήθηκαν στο Κεφάλαιο 2 – Θεωρητικό και Τεχνολογικό Υπόβαθρο . Το παρόν κεφάλαιο περιγράφει τον τρόπο με τον οποίο χρησιμοποιήθηκαν τα εργαλεία αυτά ώστε να επιτευχθεί ο στόχος της εργασίας αυτής.

Το πρώτο βήμα που χρειάστηκε να διεκπεραιωθεί είναι η δημιουργία της οντολογίας και να αποσαφηνιστεί ποιος είναι ο σκοπός της.

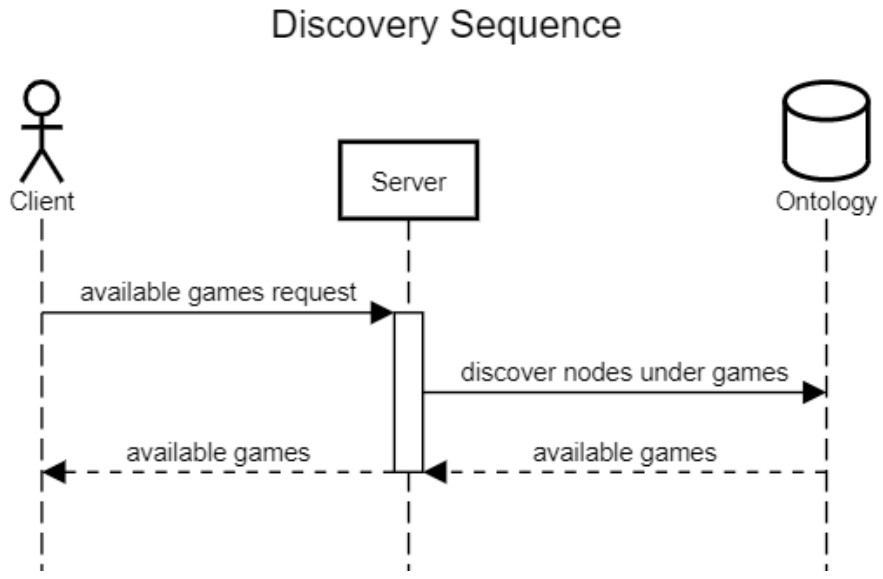
4.1 Σκοπός

Η οντολογία που δημιουργήθηκε περιλαμβάνει παίγνια τα οποία περιγράφονται από κανόνες όπως μέγιστος επιτρεπόμενος χρόνος, επίπεδο δυσκολίας κ.λπ. Στην παρούσα εργασία η οντολογία

πρέπει να περιγράφει συνολικά 15 παίγνια, αλλά θα πρέπει να σχεδιαστεί με τέτοιο τρόπο ώστε να είναι επεκτάσιμη.

Αναλύοντας τα παίγνια διαπιστώθηκε ότι το κάθε ένα χρησιμοποιεί αντικείμενα για να περιγράφουν οι κανόνες που το απαρτίζουν. Κάθε αντικείμενο μπορεί να χρησιμοποιηθεί από περισσότερους κανόνες ή και παίγνια. Για τον λόγο αυτό τα αντικείμενα έχουν σχεδιαστεί σαν μια διαφορετική κατηγορία από τα παίγνια. Η κατηγορία των παιγνίων περιέχει όλα τα παίγνια σε μία επίπεδη ιεραρχία και όχι ομαδοποιημένα ανάλογα της γνωστικής λειτουργίας. Η μοντελοποίηση αυτού του είδους δεν επιδιώχθηκε καθώς δεν υπάρχει κάποια αλληλεπίδραση μεταξύ των παιγνίων της ίδιας γνωστικής λειτουργίας στα πλαίσια της εργασίας αυτής. Ένα παράδειγμα που θα μπορούσε να αξιοποιηθεί αυτή η ιεραρχία είναι η συνολική απόδοση του ασθενή σε κάθε γνωστική λειτουργία και χρήση της συνολικής εικόνας ώστε η δημιουργία των παιγνίων να είναι προσωποποιημένη.

Επίσης η οντολογία θα πρέπει να χρησιμοποιηθεί από το λογισμικό, το οποίο θα διαβάσει τους κανόνες αυτούς και στη συνέχεια θα δημιουργήσει αυτόματα τα στιγμιότυπα των παιγνίων. Μία από τις δυσκολίες που εμφανίστηκε είναι η «ανακάλυψη» των διαθέσιμων παιγνίων από το λογισμικό καθώς έχει διαφορετικό κύκλο ζωής από την οντολογία. Η ανακάλυψη αυτή γίνεται εύκολα καθώς όλα τα διαθέσιμα παίγνια βρίσκονται κάτω από έναν συγκεκριμένο κόμβο – πατέρα. Με αυτόν τον τρόπο καθίσταται εφικτή η επικοινωνία με τον χρήστη από την συσκευή του καθώς μπορεί να δει κάθε στιγμή ποια παίγνια είναι διαθέσιμα, χωρίς την μεσολάβηση κάποιου τρίτου, όπως ιατρικό προσωπικό. Ένα use case είναι η ανακάλυψη των διαθέσιμων παιγνίων. Η περίπτωση αυτή εμφανίζεται γραφικά στην Εικόνα 24:



Εικόνα 24 Use-Case: ανακάλυψη διαθέσιμων παιγνίων

Η οντολογία θα πρέπει να περιέχει το πλήθος των αντικειμένων που χρησιμοποιούνται από τους κανόνες των παιγνίων. Τα αντικείμενα αυτά μπορούν να είναι λέξεις, εικόνες, τετράγωνα ή ήχοι. Επίσης δίνεται η δυνατότητα της συσχέτισης μεταξύ τους με αποτέλεσμα την μεγαλύτερη ποικιλία αντικειμένων και ευκολότερη συσχέτιση με τους κανόνες των παιγνίων. Η δυνατότητα της συσχέτισης των αντικειμένων μεταξύ τους μπορεί να οδηγήσει στην δημιουργία νέων παιγνίων, όπως για παράδειγμα στην εύρεση του ήχου που μπορεί να χαρακτηρίζει μία λέξη. Η θετική πλευρά της ποικιλομορφίας των αντικειμένων στο παραπάνω παράδειγμα είναι ότι οι κανόνες των παιγνίων παραμένουν απλοί, αρκεί να χρησιμοποιηθούν, για το συγκεκριμένο παράδειγμα, ένας ήχος και ένας αριθμός εικόνων με συσχέτιση με κάποιον ήχο, αντί ενός ήχου *και* n αριθμός εικόνων *και* μόνο εικόνες με ήχο. Η αρνητική πλευρά είναι ότι τα αντικείμενα τείνουν να γίνονται πολύπλοκα καθώς κάθε συσχέτιση μπορεί να δημιουργήσει μια υποκατηγορία του αντικειμένου. Μεταξύ της προσπέλασης πολύπλοκων κανόνων και πολύπλοκων αντικειμένων διαπιστώθηκε ότι η αυτόματη δημιουργία των αντικειμένων είναι πιο προσιτή καθώς όπως έχει αναφερθεί ένα αντικείμενο μπορεί να επαναχρησιμοποιηθεί από άλλα παίγνια.

Ο σκοπός της οντολογίας είναι η αναπαράσταση ως γνώση των κανόνων των παιγνίων και των αντικειμένων τους ώστε να είναι εφικτή μετέπειτα η αυτόματη δημιουργία τους από κάποιο λογισμικό. Επίσης, η οντολογία θα πρέπει να είναι εύκολα επεκτάσιμη και να μπορεί να περιγράφει νέα παίγνια ακολουθώντας ένα συγκεκριμένο μοτίβο το οποίο θα περιγραφεί παρακάτω.

4.2 Μηχανική Οντολογίας

Αφού μελετήθηκαν δύο εφαρμογές οι οποίες υλοποιούν παίγνια για την ενίσχυση της γνωστικής λειτουργίας για περιπτώσεις ήπιας γνωστικής εξασθένισης διαπιστώθηκαν τα εξής μοτίβα τα οποία είναι κοινά για όλα τα παίγνια:

1. Υπάρχει επίπεδο δυσκολίας
2. Υπάρχει επιτρεπόμενο χρονικό όριο
3. Υπάρχει επιτρεπτός αριθμός επαναλήψεων.
4. Μετά από *n* πλήθος σωστών απαντήσεων το επίπεδο δυσκολίας αυξάνεται.
5. Κάθε παίγνιο χρησιμοποιεί εικόνες, αλφαριθμητικά ή και τα δύο για την υλοποίηση του.

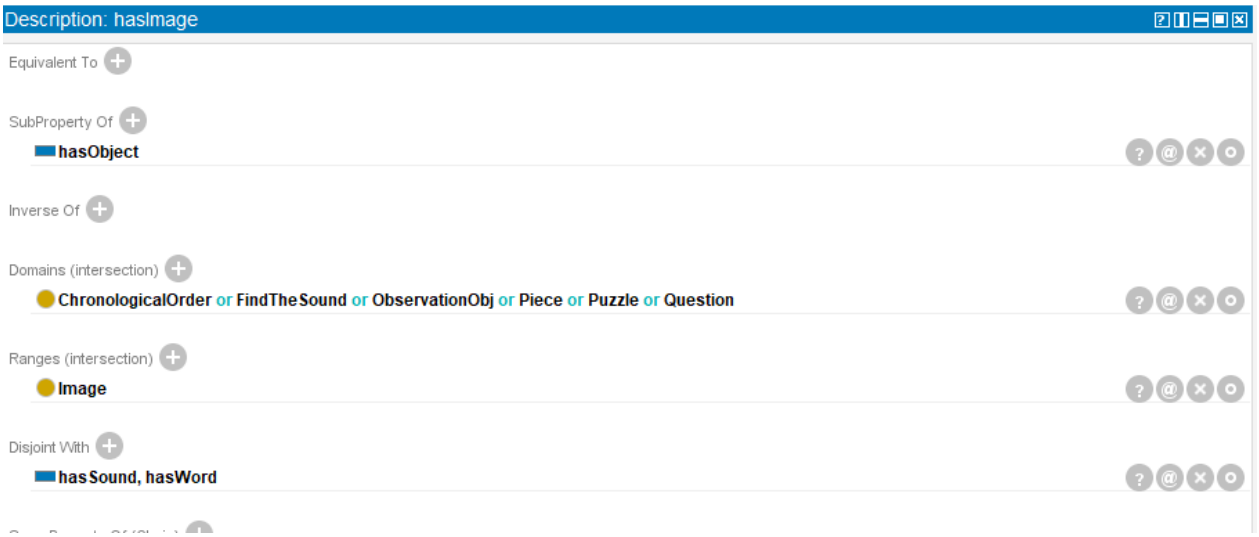
Επίσης για κάθε παίγνιο παρατηρήθηκε ότι ανάλογα του επιπέδου δυσκολίας τα αντικείμενα που χρησιμοποιούνται γίνονται πιο δύσκολα ή εύκολα. Θέλοντας να αποφευχθεί η μετάβαση της δυσκολίας ως περιορισμός στα αντικείμενα, δημιουργήθηκαν *n* διαφορετικές υποκλάσεις για κάθε παίγνιο, όπου κάθε μία αναφέρεται σε κάθε επίπεδο δυσκολίας που υπάρχει διαθέσιμο για το παίγνιο αυτό. Κάθε τέτοια υπο-οντότητα περιέχει τα αντίστοιχα αντικείμενα ή το αντίστοιχο πλήθος των αντικειμένων, που πρέπει να χρησιμοποιηθούν από το επιλεγμένο επίπεδο δυσκολίας.

Εκτός του επιπέδου δυσκολίας, παρατηρήθηκε ότι όλα τα παίγνια έχουν κάποιους κοινούς περιορισμούς, όπως για παράδειγμα, το όνομα του παίχτη, τον μέγιστο επιτρεπόμενο χρόνο, τον χρόνο λύσης, το μοναδικό ID κάθε παίγνιου. Προκειμένου να αποφευχθεί η επανάληψη των συσχετίσεων αυτών σε κάθε παίγνιο, αυτές αποτελούν περιορισμούς για την οντότητα *Game* που αποτελεί πατέρα όλων των οντοτήτων – παιγνίων. Αυτό έχει ως αποτέλεσμα όλες οι οντότητες των παίγνιων να κληρονομούν αυτούς τους περιορισμούς.

Όπως αναφέρθηκε κάθε παίγνιο χρησιμοποιεί κάποια αντικείμενα για την υλοποίησή του. Από τα αντικείμενα αυτά δημιουργήθηκαν οντότητες μόνο για τις εικόνες, τους ήχους, τα αλφαριθμητικά (έχουν το ρόλο της λέξης στα παίγνια) και των τετραγώνων. Η δημιουργία οντότητας που να χαρακτηρίζει έναν αριθμό αποφεύχθηκε καθώς εν τέλει θα πρέπει να υλοποιηθούν εκ νέου όλες οι λογικές και αριθμητικές πράξεις που ένας αριθμός μπορεί να κάνει. Απεναντίας για τα παίγνια τα οποία χρησιμοποιούν αριθμούς χρησιμοποιείται το ήδη υπάρχον *datatype xsd:positiveInteger*.

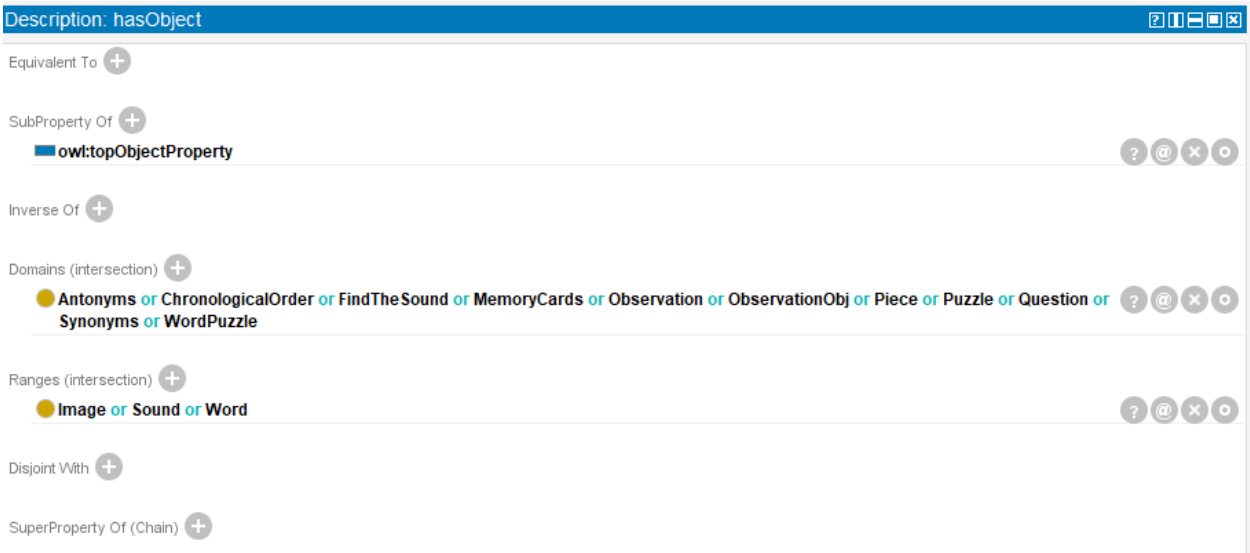
Επίσης για τα **ObjectProperties** επικρατεί κυρίως ένα naming convention. Το όνομα κάθε τέτοιας συσχέτισης ξεκινάει με την λέξη has και στη συνέχεια ακολουθεί το όνομα του αντικειμένου. Είναι κατανοητό ότι με τον τρόπο αυτό ο αριθμός των συσχετίσεων είναι ίδιος με τον αριθμό αντικειμένων και αυτό μπορεί να περιορίζει το γνωστικό επίπεδο της οντολογίας, καθώς μεγαλύτερο πλήθος συσχετίσεων σημαίνει πλουσιότερα αξιώματα και φυσικά περισσότερες συσχετίσεις μεταξύ των οντοτήτων. Όταν η ονομασία της συσχέτισης δεν είναι αρκετή τότε επιδιώκεται κάποια ιεραρχική μορφή με πατέρα κάποια συσχέτιση της μορφής *hasWord* και παιδί μια συσχέτιση που προσδίδει κάποια παραπάνω πληροφορία για παράδειγμα *hasMainWord*. Όταν και αυτό δεν είναι εφικτό τότε απλά δημιουργείται μια νέα συσχέτιση στο ίδιο επίπεδο με τις υπόλοιπες. Ο λόγος που ακολουθείται αυτή η σειρά είναι ώστε όλες οι συσχετίσεις να είναι οργανωμένες και εύκολα κατανοητές.

Ένα παίγνιο είναι ανεξάρτητο από κάποιο άλλο και δεν υπάρχει κάποια συσχέτιση μεταξύ τους. Αυτό σημαίνει ότι δεν υπάρχει κάποιο **ObjectProperty** με **domain** και **range** ένα παίγνιο. Το **domain** ενός **ObjectProperty** μπορεί να είναι ένα ή περισσότερα παίγνια ή αντικείμενα, αλλά το **range** θα είναι πάντα ένα αντικείμενο. Με τον τρόπο αυτό είναι εύκολο να διαπιστωθεί με ποιο αντικείμενο η δοθείσα οντότητα συσχετίζεται και μέσω ποιας συσχέτισης. Για παράδειγμα η συσχέτιση της Εικόνα 25 *hasImage* έχει ως **domain** κάποιο παίγνιο ή κάποιο αντικείμενο, αλλά το **range** είναι πάντα ένα *Image*.



Εικόνα 25 Περίπτωση κληρονομικότητας: hasImage

Στην Εικόνα 25 παρατηρείται ότι η συσχέτιση αυτή είναι sub-property της *hasObject*. Αυτό σημαίνει ότι οι οντότητες που ενώνονται με την συσχέτιση *hasImage* μπορούν να συσχετιστούν και με την *hasObject*. Επίσης η συσχέτιση *hasObject,* της Εικόνας 26, αποτελεί υπερσύνολο της *hasImage*, με αποτέλεσμα το **domain** και το **range** της *hasObject* να είναι υπερσύνολο αυτών της *hasImage*.



Εικόνα 26 Περίπτωση κληρονομικότητας: hasObject

Παρόμοια λογική χρησιμοποιήθηκε και για την δημιουργία των **DataProperties**.

Για κάθε επίπεδο δυσκολίας οι τιμές που παίρνουν κάποια **DataProperties** είναι συγκεκριμένες, όπως για παράδειγμα το επίπεδο δυσκολίας. Τα **DataProperties** αυτά λειτουργούν ως διαχωριστικά μιας οντότητας. Για παράδειγμα το παίγνιο *EasyPuzzle* είναι ίδιο με μια οντότητα Puzzle όταν έχει την τιμή "*EASY*" για την συσχέτιση *hasDifficulty*.

Οι κανόνες κάθε παιγνίου καθώς και το πλήθος των αντικειμένων που χρησιμοποιούν είναι τα πραγματικά κριτήρια τα οποία διαχωρίζουν ένα εύκολο παίγνιο από ένα δύσκολο. Οι περιορισμοί των παιγνίων έχουν υπαρξιακές συσχετίσεις με κάποια αντικείμενα. Αυτό σημαίνει ότι ένας περιορισμός μπορεί να είναι είτε σε ένα **ObjectProperty** είτε σε ένα **DataProperty** αρκεί να έχει ένα από τα παρακάτω cardinality:

1. **min**, οι ελάχιστες εμφανίσεις του property
2. **max**, οι μέγιστες εμφανίσεις του property
3. **exactly**, ο ακριβές αριθμός εμφανίσεων του property
4. **only**, φράζει τον τύπο του range του property

Με τον παραπάνω τρόπο δημιουργούνται οι επιθυμητές οντότητες που περιγράφουν είτε ένα παίγνιο είτε ένα αντικείμενο.

4.3 Οντότητες

Οι δύο βασικές οντότητες της οντολογίας είναι οι:

1. **Game**, αποτελεί την βασική οντότητα η οποία συσχετίζεται με τις βασικές λειτουργίες ενός παιγνίου όπως το επίπεδο δυσκολίας. Κάθε παίγνιο το οποίο μοντελοποιείται πρέπει να έχει ως πατέρα την συγκεκριμένη οντολογία.
2. **Object**, αποτελεί τον πατέρα για κάθε αντικείμενο το οποίο μοντελοποιείται. Το μόνο κοινό χαρακτηριστικό μεταξύ των διαφορετικών αντικειμένων είναι ένα μοναδικό id.

Οι οντότητες αυτές είναι ανεξάρτητες μεταξύ τους, δηλαδή μια οντότητα *Game* δεν έχει κάποια τομή με μία οντότητα *Object*. Αυτό σημαίνει ότι ένα individual *Game* δεν μπορεί να είναι και *Object*.

Στη συνέχεια περιγράφονται αναλυτικότερα οι υπό-οντότητες των δύο βασικών οντοτήτων καθώς και οι συσχετίσεις τους με άλλες οντότητες. Για την εύκολη ανάγνωση των συσχετίσεων με *OP* έχουν σημειωθεί όλες οι **ObjectProperty** συσχετίσεις, με *DP* όλες οι **DataProperty** και με obj όλες οι οντότητες οι οποίες είναι αντικείμενα. Οι οντότητες των αντικειμένων μπορούν να βρεθούν στην αντίστοιχη κατηγορία *Objects* παρακάτω. Επίσης, κάθε συσχέτιση θα ακολουθείται από ένα cardinality, το πλήθος, και τέλος την οντότητα με την οποία συσχετίζεται (**range**). Για παράδειγμα ο περιορισμός:

hasMathOperation(OP) max 5 MathOperator(obj)

ερμηνεύεται ως η συσχέτιση *hasMathOperation* είναι **ObjectProperty** με max cardinality το πλήθος 5 της οντότητας - αντικειμένου *MathOperator*. Στη συνέχεια ο αναγνώστης μπορεί να ανατρέξει στην οντότητα *MathOperator* η οποία έχει περιγραφεί παρακάτω στο κείμενο.

4.3.1 Game

Η οντότητα *Game* περιέχει όλες τις οντότητες των παιγνίων. Οι οντότητες αυτές μοιράζονται τις παρακάτω κοινές συσχετίσεις.

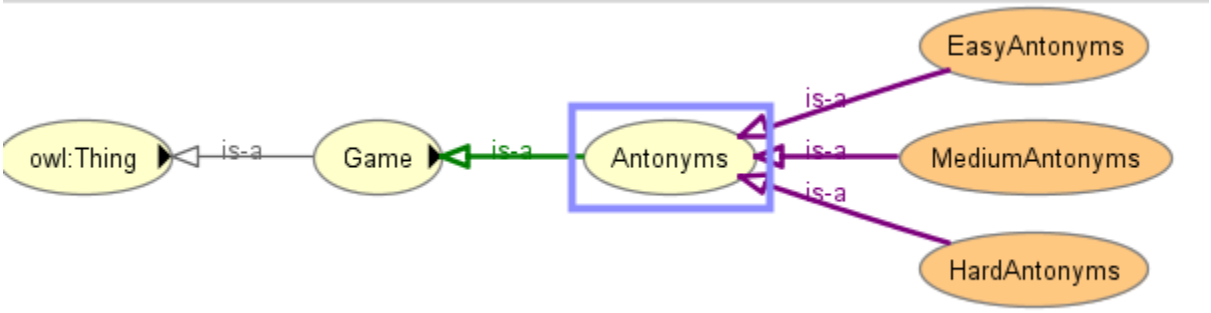
1. *maxCompletionTime(DP) value n,* ο επιτρεπόμενος χρόνος εκτέλεσης. Το n μπορεί να μεταβάλλεται ανάλογα με την δυσκολία του παιχνιδιού. Η τιμή είναι ένας ακέραιος αριθμός ο οποίος αναλογεί στα ms για την διεκπεραίωση του παίγνιου.
2. *completedDate (DP) value xsd: string:* Περιγράφει την ημερομηνία τερματισμού του τρέχοντος επιπέδου.
3. *hasDifficulty (DP) value {"EASY", "MEDIUM", "HARD"}:* η δυσκολία του παίγνιου. Μπορεί να πάρει μία από τις δοθείσες τιμές.
4. *hasGameId (DP) value xsd: string:* ένα μοναδικό αναγνωριστικό του individual παίγνιου. Το αναγνωριστικό αυτό είναι της μορφής *<Όνομα του παίγνιου>_<επίπεδο δυσκολίας>_<όνομα παίχτη>_<αριθμός επιπέδου>*.
5. *hasLevel (DP) value xsd: string:* το τρέχον επίπεδο του individual.
6. *hasPlayer (DP) value xsd: string:* το ψευδώνυμο του παίχτη.
7. *isCompletedIn (DP) value xsd: positiveInteger:* ο χρόνος σε ms τερματισμού του επιπέδου.

Στη συνέχεια περιγράφονται οι οντότητες των παιγνίων αναλυτικότερα.

Antonyms

Η οντότητα αυτή, όπως εμφανίζεται στην Εικόνα 27,περιγράφει το παίγνιο που προτρέπει τον χρήστη να βρει ένα αντώνυμο της δοθείσας λέξης επιλέγοντας από ένα πλήθος λέξεων. Το πλήθος των λέξεων αυξάνεται ανάλογα με το επίπεδο δυσκολίας.

1. *hasWord(OP) exactly n AntonymWord(obj)*, ο αριθμός των λέξεων-επιλογών του χρήστη
2. *hasMainWord(OP) exactly 1 AntonymWord(obj),* η κυρίως λέξη-απάντηση

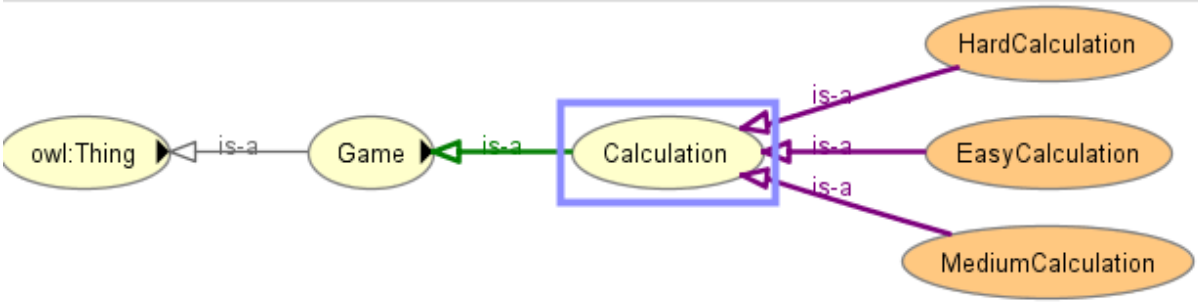


Εικόνα 27 Οντότητα Antonyms

Calculation

Η οντότητα της Εικόνας 28 περιγράφει το παίγνιο εύρεσης των αριθμητικών πράξεων και αριθμών. Ο τύπος των αριθμητικών πράξεων καθώς και το διάστημα των αριθμών εξαρτάται ανάλογα της δυσκολίας του παίγνιου. Οι περιορισμοί είναι οι παρακάτω:

1. *hasMathOperation (OP) max 5 MathOperator(obj)*, το πλήθος των αριθμητικών πράξεων
2. *hasMathOperation (OP) only n(obj),* οι επιτρεπόμενες αριθμητικές πράξεις. Κάθε πράξη συσχετίζεται με αριθμούς και για αυτό δεν υπάρχει κάποιος περιορισμός όσον αφορά το πλήθος των αριθμών.
3. *hasNumber (OP) only UpToTen(obj)*, το επιτρεπόμενο διάστημα των αριθμών. Ο περιορισμός αυτός δηλώνει ότι οι αριθμοί που θα χρησιμοποιηθούν στις πράξεις πρέπει να είναι individuals της οντότητας *UpToTen*

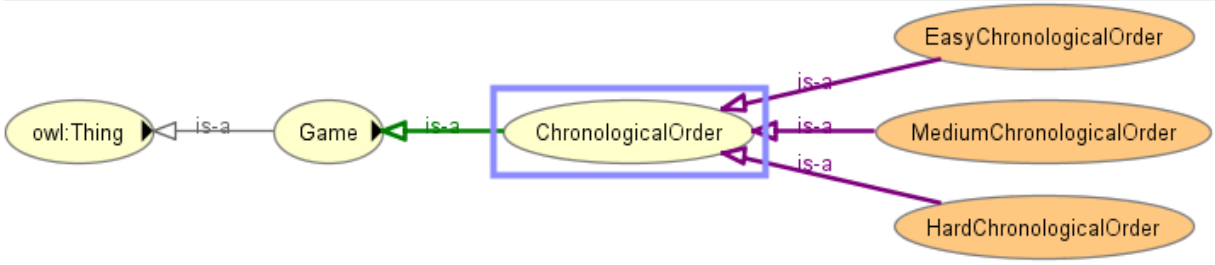


Εικόνα 28 Οντότητα Calculation

ChronologicalOrder

Η οντότητα της Εικόνας 29 περιγράφει το παίγνιο της χρονολογικής τοποθέτησης των εικόνων. Το πλήθος των εικόνων αυξάνεται ανάλογα της δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasOrderedImage (OP) exactly n OrderedImage (obj),* το πλήθος των εικόνων που θα χρησιμοποιηθούν.

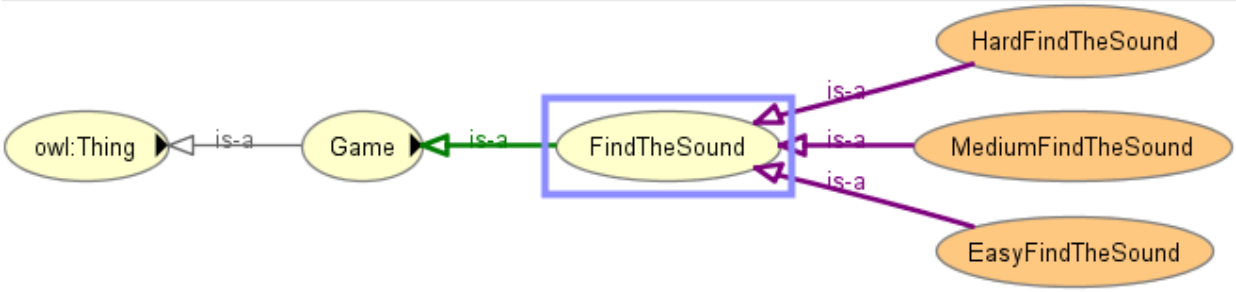


Εικόνα 29 Οντότητα ChronologicalOrder

FindTheSound

Η οντότητα της Εικόνας 30 περιγράφει το παίγνιο συσχέτισης των εικόνων με τον ήχο. Το πλήθος των εμφανιζόμενων εικόνων αυξάνεται ανάλογα της δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

2. *hasImage (OP) exactly n ImageSound(obj),* το πλήθος των εικόνων που θα χρησιμοποιηθούν ως επιλογές με τους σχετιζόμενους ήχους.
3. *hasSound (OP) exactly 1 Sound(obj),* ο ήχος που ο χρήστης ακούει με την σχετιζόμενη εικόνα

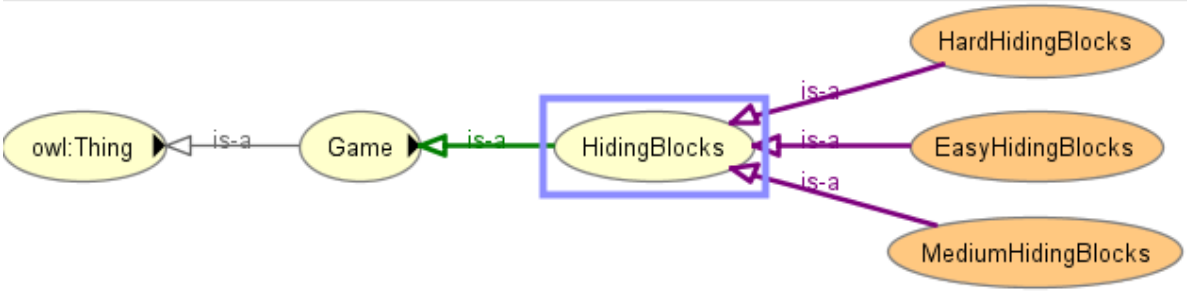


Εικόνα 30 Οντότητα FindTheSound

HidingBlocks

Η οντότητα της Εικόνας 31 περιγράφει το παίγνιο εύρεσης των σημειωμένων τετραγώνων από ένα πλήθος των 9 τετραγώνων. Το πλήθος των σημειωμένων τετραγώνων αυξάνεται ανάλογα του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasBlock (OP) exactly 9 Block (obj),* ο αριθμός των τετραγώνων που είναι διαθέσιμα.
2. *hasHidingBlock (OP) exactly n StationaryBlock(obj)*, ο αριθμός των τετραγώνων που είναι σημειωμένα

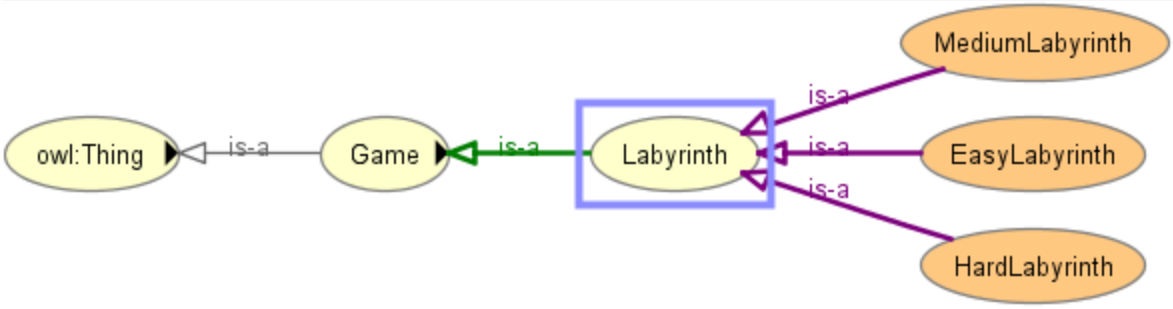


Εικόνα 31 Οντότητα HidingBlocks

Labyrinth

Η οντότητα της Εικόνας 32 περιγράφει το παίγνιο του λαβυρίνθου. Ο αριθμός των τετραγώνων που χρησιμοποιούνται αυξάνεται ανάλογα του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasStartingBlock (OP) exactly 1 LabyrinthBlock(obj),* το αρχικό τετράγωνο.
2. *hasBlock (OP) exactly n LabyrinthBlock(obj),* το πλήθος των τετραγώνων που θα δημιουργηθούν.
3. *hasEndingBlock (OP) exactly 1 LabyrinthBlock(obj),* το τελικό τετράγωνο.

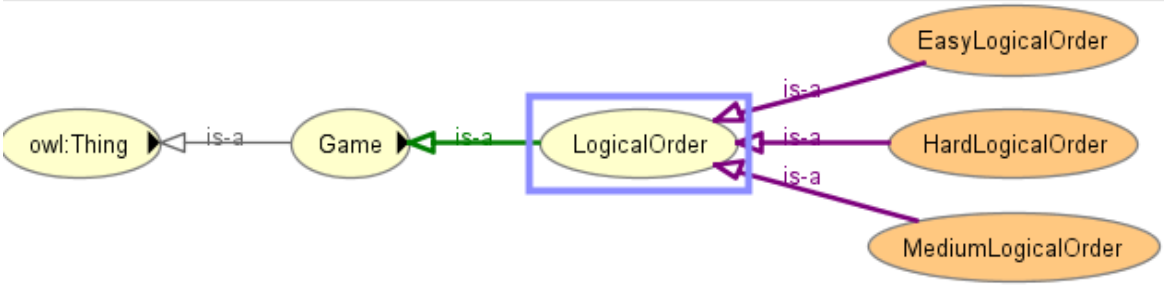


Εικόνα 32 Οντότητα Labyrinth

LogicalOrder

Η οντότητα της Εικόνας 33 περιγράφει το παίγνιο εύρεσης της λογικής συνέχειας μίας αλληλουχίας κινήσεων ενός ή περισσότερων τετραγώνων από ένα πλήθος των 9 τετραγώνων. Το πλήθος των τετραγώνων είναι πάντοτε 9. Αυτό έχει μοντελοποιηθεί χρησιμοποιώντας τα DP *hasSquareColumns*, *hasSquareRows* με τιμή 3. Το παίγνιο αυτό έχει παρατηρηθεί ότι σε κάθε επίπεδο δυσκολίας δίνονται δύο πιθανές επιλογές στον χρήστη. Έχει δημιουργηθεί το DP *choices*, που αναφέρεται στο πλήθος των δυνατών επιλογών, το οποίο έχει πάντα την τιμή 2. Επίσης έχει παρατηρηθεί ότι κάθε τετράγωνο μετατοπίζεται κατά ένα τετράγωνο. Αυτή η ιδιότητα έχει μοντελοποιηθεί με το DP *hasStep* το οποίο έχει τιμή 1. Τέλος, το πλήθος των τετραγώνων που μετατοπίζονται αλλάζει με το επίπεδο δυσκολίας, αλλά γνωρίζουμε ότι οι τιμές είναι 1, 2 ή 3 ανάλογα του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasBlockSet (OP) exactly 4 BlockSet (obj),* οι διαθέσιμες επιλογές του χρήση. Κάθε blockset περιγράφει έναν πίνακα τετραγώνων ο οποίος περιέχει τα κινούμενα τετράγωνα. διαθέσιμων .
2. *hasCorrectBlockSet (OP) exactly 1 BlockSet (obj),* περιγράφει την σωστή απάντηση του παίγνιου. Εφόσον η συσχέτιση hasCorrectBlockSet είναι sub property της συσχέτισης hasBlockSet αυτό σημαίνει ότι η απάντηση θα συσχετιστεί και την συσχέτιση hasBlockSet.
3. *hasMovement (DP) exactly 1 {"D", "L", "LD", "LU", "R", "RD", "RU", "U"}*, η επιτρεπόμενη κίνηση. Από το παίγνιο παρατηρήθηκε ότι ένα τετράγωνο μπορεί να κάνει μόνο μια κίνηση. Οι τιμές αυτές ορίζονται ως εξής:
    a. D, κίνηση προς τα κάτω (Down)
    b. L, κίνηση προς τα αριστερά (Left)
    c. LD, κίνηση διαγώνια προς τα αριστερά και κάτω(Left Down)
    d. LU, κίνηση διαγώνια προς τα αριστερά και πάνω (Left Up)
    e. R, κίνηση προς τα δεξιά (Right)
    f. RD, κίνηση διαγώνια προς τα δεξιά και κάτω (Right Down)
    g. RU, κίνηση διαγώνια προς τα δεξιά και πάνω(Right Up)
    h. U, κίνηση προς τα πάνω (Up)

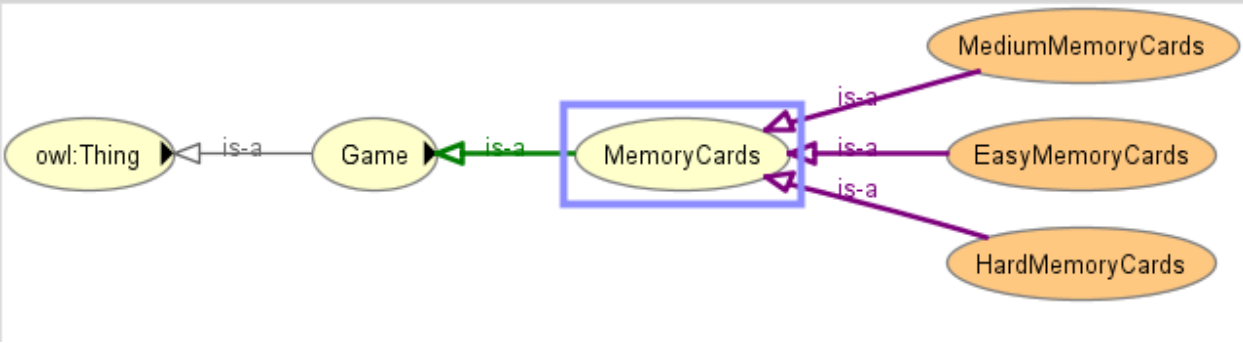


Εικόνα 33 Οντότητα LogicalOrder

MemoryCards

Η οντότητα της Εικόνα 34 περιγράφει το παίγνιο εύρεσης όλων των ζευγαριών εικόνων. Το πλήθος των ζευγαριών αυξάνεται ανάλογα του επιπέδου δυσκολίας. Από το παίγνιο παρατηρήθηκε ότι κάθε εικόνα εμφανίζεται 2 φορές. Παρόλα αυτά το πλήθος των εμφανίσεων ενός αντικειμένου έχει μοντελοποιηθεί με την συσχέτιση *objectsPerCard* και έχει ως default τιμή 2. Οι περιορισμοί είναι οι παρακάτω:

1. *hasObject (OP) exactly n Object (obj),* το πλήθος των διαθέσιμων μοναδικών αντικειμένων. Το συνολικό πλήθος των αντικειμένων είναι *objectsPerCard* * n, όπου n είναι το πλήθος τον αντικειμένων που συσχετίζονται μέσω της *hasObject.* Ο αριθμός αυτός μεταβάλλεται ανάλογα της δυσκολίας του παιγνίου. Επιλέχθηκε η συσχέτιση hasObject και όχι η hasImage, καθώς δίνεται η δυνατότητα δημιουργίας νέου περιορισμού που να αναφέρει ποια αντικείμενα να χρησιμοποιηθούν. Με τον τρόπο αυτό μπορεί να δημιουργηθεί ένα παίγνιο με την εύρεση παρόμοιων ήχων ή λέξεων ή εικόνων.

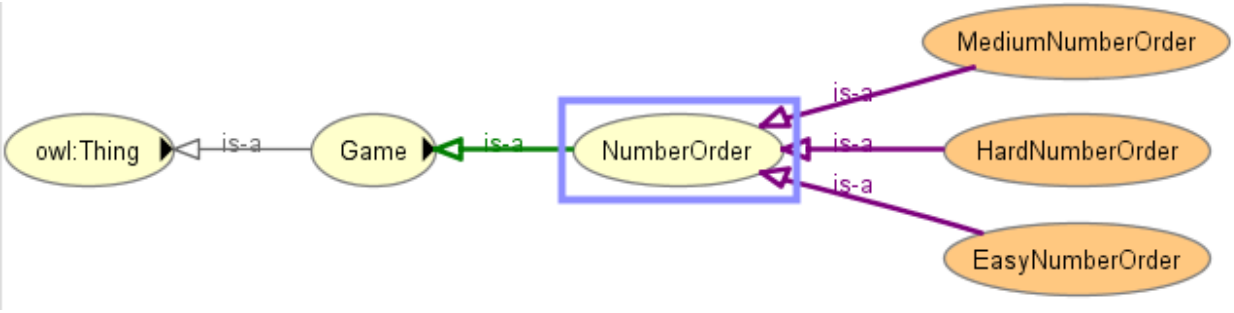


Εικόνα 34 Οντότητα MemoryCards

NumberOrder

Η οντότητα της Εικόνας 35 περιγράφει το παίγνιο τοποθέτησης των αριθμών στη σωστή σειρά είτε από το μικρότερο στο μεγαλύτερο είτε αντίστροφα. Το πλήθος των διαθέσιμων αριθμών αυξάνεται με βάση το επίπεδο δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasNumberValue (DP) exactly n xsd: positiveInteger*, οι αριθμοί που θα χρησιμοποιηθούν για το παίγνιο. Η σωστή σειρά είναι μία λογική πράξη μεταξύ ακεραίων οπότε δεν υπάρχει η ανάγκη δημιουργίας νέας συσχέτισης
2. *hasNumberValue (DP) only xsd: positiveInteger [<= "1000"^^xsd: positiveInteger]*, οι πιθανές τιμές των παραγόμενων αριθμών μπορούν να είναι θετικοί αριθμοί μέχρι το 1000

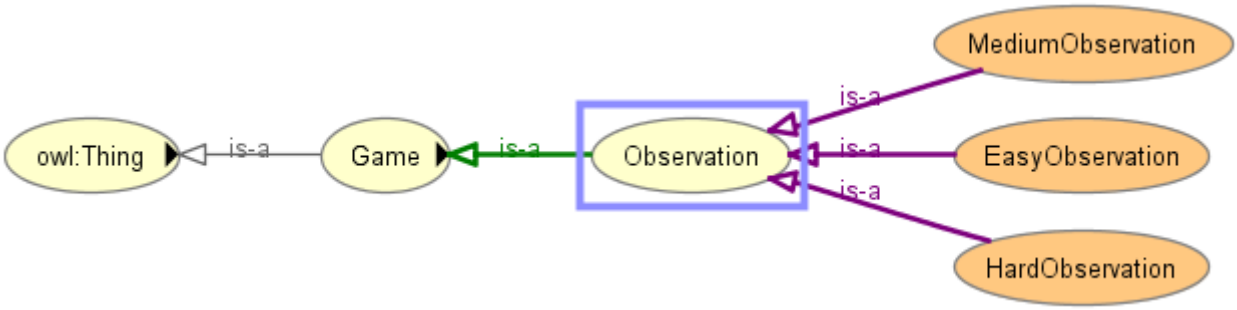


Εικόνα 35 Οντότητα NumberOrder

Observation

Η οντότητα της Εικόνας 36 περιγράφει το παίγνιο εύρεσης του συνολικού πλήθους των ζητούμενων εικόνων. Το πλήθος των διαθέσιμων εικόνων αυξάνεται ανάλογα με το επίπεδο δυσκολίας και έχει μοντελοποιηθεί με την συσχέτιση hasTotalImages(DP). Οι περιορισμοί είναι οι παρακάτω:

1. *hasObservation (OP) exactly 3 ObservationObj (obj)*, το πλήθος των παρατηρήσεων. Μια παρατήρηση περιγράφεται από το αντικείμενο ObservationObj το οποίο περιέχει την εικόνα, το πλήθος των εμφανίσεων αυτής καθώς και την λέξη με την οποία συσχετίζεται.



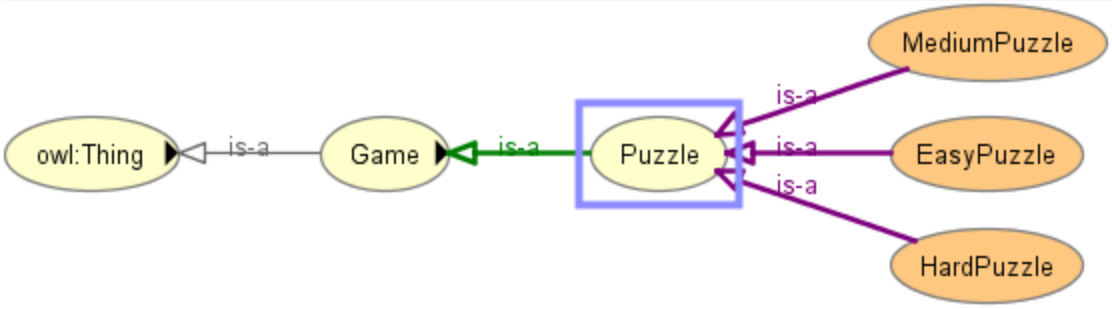
Εικόνα 36 Οντότητα Observation

Puzzle

Η οντότητα της Εικόνας 37 περιγράφει ένα παζλ. Το πλήθος των κομματιών αυξάνεται με βάση του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasImage (OP) exactly 1 Image(obj)*, η σχετιζόμενη εικόνα
2. *hasCornerPiece (OP) exactly 4 Piece(obj),* το πλήθος των γωνιακών κομματιών

3. *hasBorderLinePiece (OP) some Piece (obj)*, υποδηλώνει ότι κάθε πάζλ συσχετίζεται μέσω της συσχέτισης *hasBorderLinePiece* με κάποια κομμάτια. Το πλήθος αυτών είναι υπολογίσιμο κατά τη δημιουργία τους.
4. *hasInteriorPiece (OP) some Piece (obj)*, υποδηλώνει ότι κάθε πάζλ συσχετίζεται μέσω της συσχέτισης *hasInteriorPiece* με κάποια κομμάτια. Το πλήθος αυτών είναι υπολογίσιμο κατά τη δημιουργία τους.
5. *hasColumns (DP) exactly 1 xsd: positiveInteger [>= "4"^^xsd: positiveInteger, <= "5"^^xsd: positiveInteger]*, περιγράφει το πλήθος των στηλών του παζλ.
6. *hasRows (DP) exactly 1 xsd: positiveInteger [>= "4"^^xsd: positiveInteger, <= "5"^^xsd: positiveInteger]*, περιγράφει το πλήθος των γραμμών του παζλ.

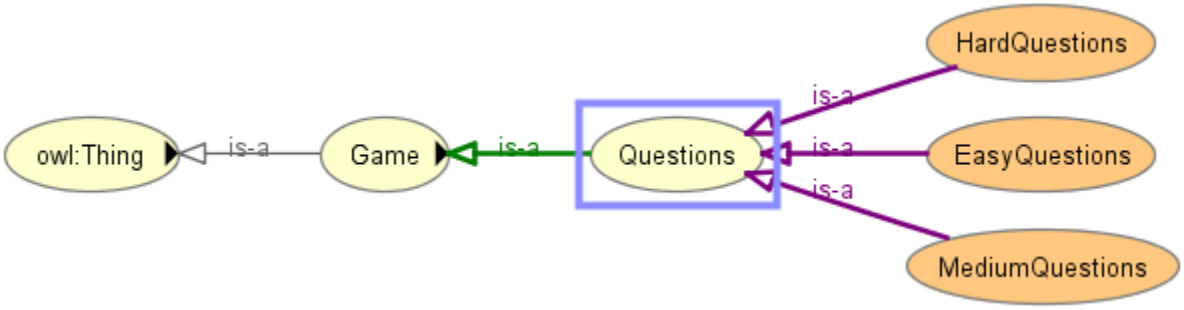


Εικόνα 37 Οντότητα Puzzle

Questions

Η οντότητα της Εικόνας 38 περιγράφει το παίγνιο των ερωτήσεων με 1 απάντηση και 4 επιλογές. Στο παρών παίγνιο δεν έχει παρατηρηθεί κάποιος κανόνας που να ορίζει το επίπεδο δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasQuestion(OP) exactly 1 Question(obj)*, η ερώτηση η οποία συσχετίζεται με το παίγνιο.
2. *hasChoice (OP) exactly 4 Word(obj)*, το πλήθος των διαθέσιμων επιλογών
3. *hasAnswer (OP) exactly 1 Word(obj)*, η σωστή απάντηση. Καθώς η συσχέτιση είναι sub property της hasChoice αυτό σημαίνει ότι η απάντηση συσχετίζεται και μέσω αυτής.
4. *hasCategory (OP) some Word(obj)*, η κατηγορία της ερώτησης.

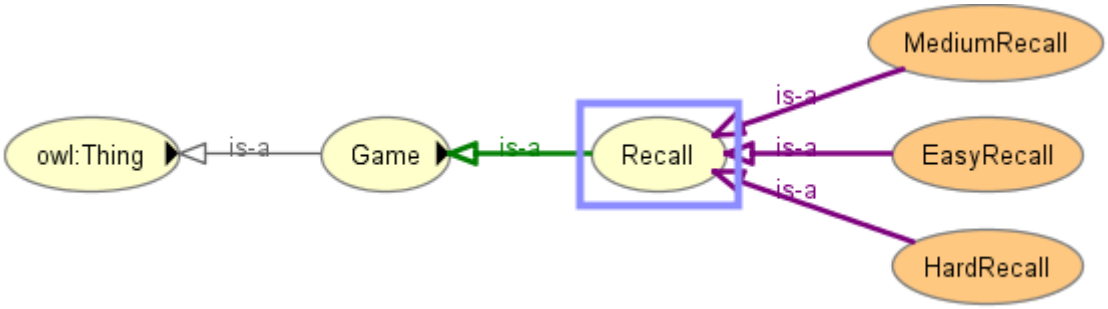


Εικόνα 38 Οντότητα Questions

Recall

Η οντότητα της Εικόνας 39 περιγράφει την εύρεση του εμφανιζόμενου αριθμού από ένα σετ αριθμών οι οποίοι είναι παρόμοιοι με τον ζητούμενο, για παράδειγμα αν ο ζητούμενος είναι ο 1149 τότε ένας παρόμοιος αριθμός είναι ο 1194. Το διάστημα από το οποίο ένας αριθμός μπορεί να επιλεχθεί, καθώς και το πλήθος των διαθέσιμων επιλογών αυξάνονται ανάλογα του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω:

1. *hasNumberValue (DP) exactly 5 xsd: positiveInteger*, οι αριθμοί προς επιλογή
2. *hasNumberValue (DP) only xsd: positiveInteger [>= "1000"^^xsd: positiveInteger , <= "9999"^^xsd: positiveInteger]*, το μέγεθος των αριθμών που θα χρησιμοποιηθούν
3. *hasRecallNumber (DP) exactly 1 xsd: positiveInteger*, ο ζητούμενος αριθμός.

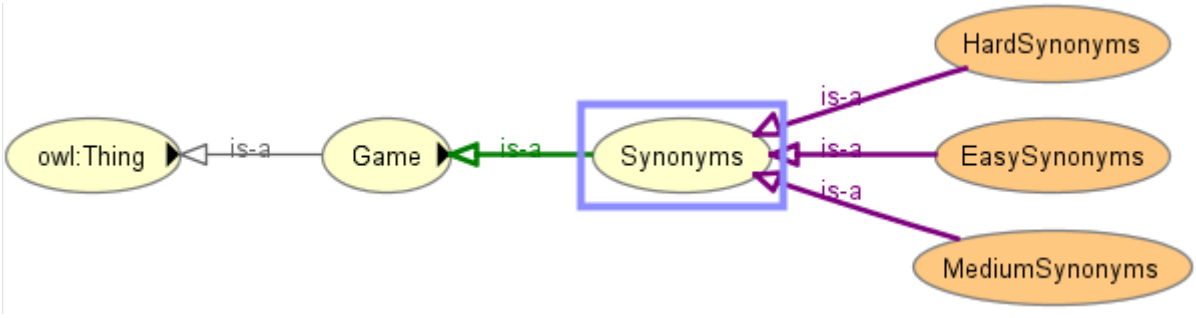


Εικόνα 39 Οντότητα Recall

Synonyms

Η οντότητα της Εικόνας 40 περιγράφει το παίγνιο της εύρεσης ενός συνώνυμου από μία δοθείσα λέξη. Το πλήθος των επιλογών αυξάνεται με βάση το επίπεδο δυσκολίας. Οι περιορισμοί είναι οι παρακάτω

1. *hasWord(OP) n SynonimWord(obj)*, ο αριθμός των λέξεων-επιλογών του χρήστη
2. *hasMainWord(OP) 1 Word(obj),* η κυρίως λέξη-απάντηση

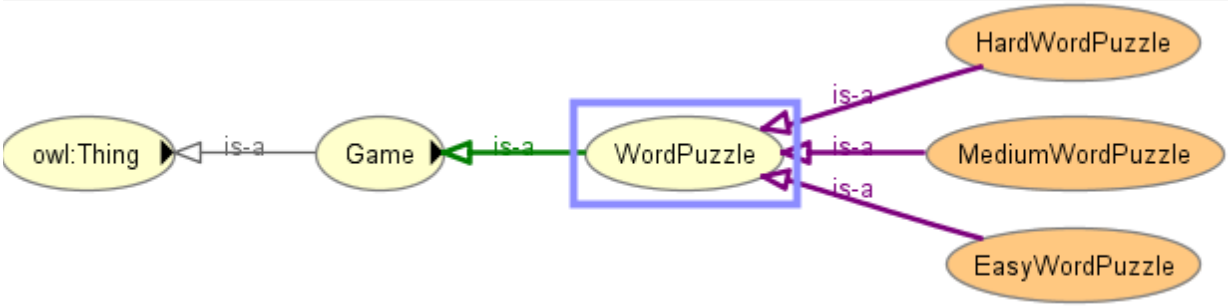


Εικόνα 40 Οντότητα Synonyms

WordPuzzle

Η οντότητα της Εικόνας 41 περιγράφει το παίγνιο τοποθέτησης των γραμμάτων στη σωστή σειρά ώστε να δημιουργηθεί η σωστή λέξη. Το μέγεθος της λέξης αυξάνεται ανάλογα του επιπέδου δυσκολίας. Οι περιορισμοί είναι οι παρακάτω

1. *hasWord exactly 1 Word(obj)*, η λέξη που ζητείται να βρεθεί.



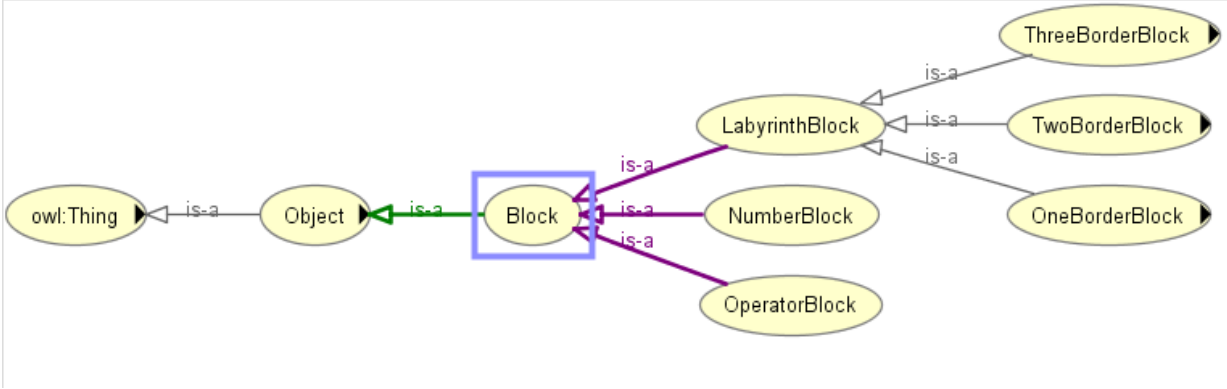
Εικόνα 41 Οντότητα WordPuzzle

4.3.2 GameObjects

Παρακάτω περιγράφονται οι οντότητες για τα αντικείμενα που χρησιμοποιούνται από τις οντότητες των παιγνίων.

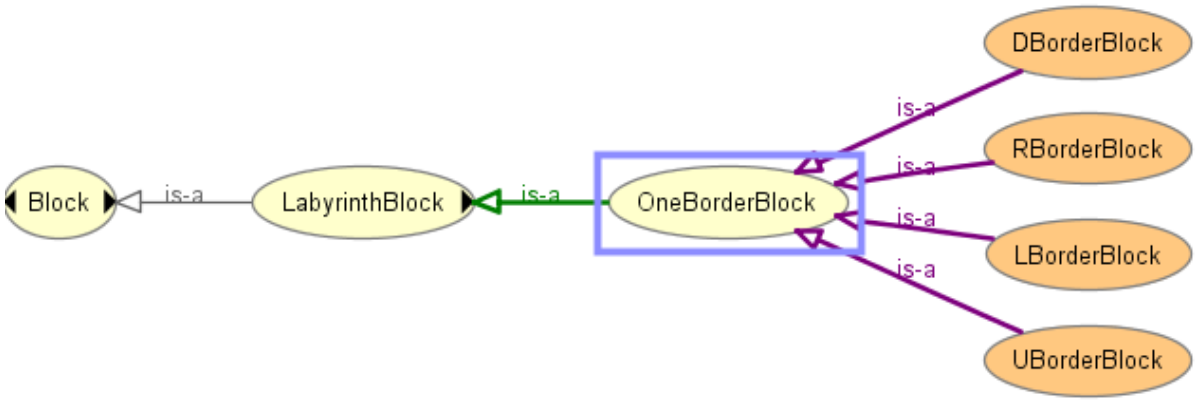
Block

Η οντότητα της Εικόνας 42 περιγράφει το αντικείμενο το οποίο χρησιμοποιείται από όλα τα παιχνίδια που κάνουν χρήση τετραγώνων. Καθώς το block αποτελεί μέρος ενός πίνακα και επειδή η θέση στον πίνακα είναι κρίσιμη, υπάρχουν δύο data properties τα οποία καθορίζουν τη θέση της οντότητας σε έναν πίνακα. Οι συσχετίσεις αυτές είναι οι *hasColumnNumber*, *hasRowNumber* και οι τιμές είναι ένας θετικός ακέραιος.



Εικόνα 42 Οντότητα Block

1. **LabyrinthBlock**, περιγράφει το κάθε τετράγωνο για το παιχνίδι του λαβυρίνθου. Κάθε τέτοιο τετράγωνο μπορεί να ενώνεται με ένα πλήθος γειτονικών τετραγώνων. Το πλήθος αυτό εξαρτάται από το είδος του τετραγώνου. Σχεδιάστηκαν 3 κατηγορίες τετραγώνων οι οποίες είναι οι εξής:
    a. **OneBorderBlock**, η οντότητα αυτή εμφανίζεται στην Εικόνα 43 και περιγράφει ένα τετράγωνο το οποίο έχει μία ακμή, πιο συγκεκριμένα έχουμε τις παρακάτω οντότητες με τους αντίστοιχους περιορισμούς:
        i. **DBorderBlock** (= *OneBorderBlock(obj) and (hasBorder (DP) value "D")*, περιγράφει το τετράγωνο το οποίο έχει ακμή την κάτω πλευρά του, με τους παρακάτω περιορισμούς :
            1. *hasConnectingBlock (OP) max 1 RLBorderBlock(obj)*
            2. *hasConnectingBlock (OP) max 2 UDBorderBlock(obj)*
        ii. **LBorderBlock** (= *OneBorderBlock(obj) and (hasBorder(DP) value "L")*, περιγράφει το τετράγωνο το οποίο έχει ακμή την αριστερή πλευρά του, με τους παρακάτω περιορισμούς:
            1. *hasConnectingBlock (OP) max 2 RLBorderBlock(obj)*
            2. *hasConnectingBlock (OP) max 1 UDBorderBlock(obj)*
        iii. **RBorderBlock** (= *OneBorderBlock(obj) and (hasBorder(DP) value "R")*, περιγράφει το τετράγωνο το οποίο έχει ακμή την δεξιά πλευρά του με περιορισμούς ίδιους με το αντικείμενο LBorderBlock
        iv. **UBorderBlock** (= *OneBorderBlock(obj) and (hasBorder(DP) value "U")*, περιγράφει το τετράγωνο το οποίο έχει ακμή την πάνω πλευρά του με περιορισμούς ίδιους με το αντικείμενο DBorderBlock



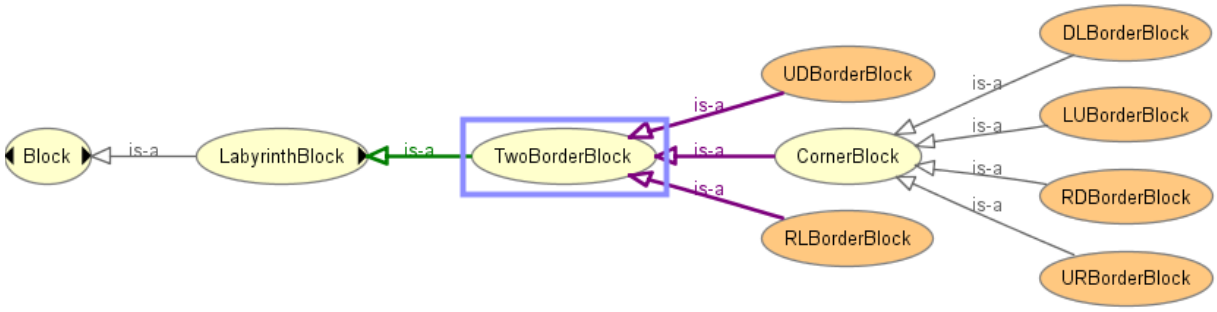
Εικόνα 43 Οντότητα OneBorderBlock

    b. **TwoBorderBlock**, η οντότητα αυτή εμφανίζεται στην Εικόνα 44 και περιγράφει ένα τετράγωνο το οποίο έχει ακριβώς δύο σύνορα σε δύο οποιεσδήποτε ακμές, πιο συγκεκριμένα έχουμε τις παρακάτω οντότητες με τους αντίστοιχους περιορισμούς:

i. **CornerBlock**, περιγράφει τα τετράγωνα που οι ακμές τους σχηματίζουν γωνία. Οι περιορισμού για τα τετράγωνα αυτά είναι οι εξής:
   1. *hasConnectingBlock (OP) max 1 RLBorderBlock(obj)*
   2. *hasConnectingBlock (OP) max 1 RLBorderBlock(obj)*

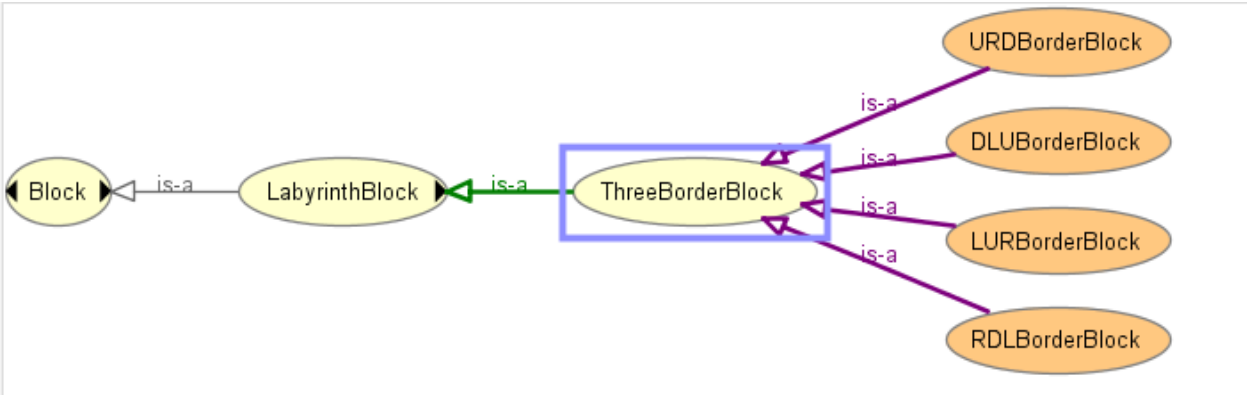
   Τα τετράγωνα – γωνίες έχουν ονομαστεί στις παρακάτω οντότητες:

   a. **DLBorderBlock**, η γωνία που σχηματίζεται από την κάτω και την αριστερή πλευρά του τετραγώνου: *CornerBlock(obj) and (hasBorder(DP) value "D") and (hasBorder(DP) value "L")*
   b. **LUBorderBlock**, η γωνία που σχηματίζεται από την αριστερή και την πάνω πλευρά του τετραγώνου: *CornerBlock(obj) and (hasBorder(DP) value "U") and (hasBorder(DP) value "L")*
   c. **RDBorderBlock**, η γωνία που σχηματίζεται από την δεξιά και την κάτω πλευρά του τετραγώνου: *CornerBlock(obj) and (hasBorder(DP) value "R") and (hasBorder(DP) value "D")*
   d. **URBorderBlock**, η γωνία που σχηματίζεται από την δεξιά και την πάνω πλευρά του τετραγώνου: *CornerBlock(obj) and (hasBorder(DP) value "R") and (hasBorder(DP) value "U")*
   3. **RLBorderBlock** (= *TwoBorderBlock and (hasBorder value "L") and hasBorder value "R"*), περιγράφει το τετράγωνο που έχει ακμές την αριστερή και την δεξιά πλευρά, με τον μοναδικό περιορισμό:
      a. *hasConnectingBlock (OP) max 2 (OneBorderBlock (obj) or CornerBlock (obj) or RLBorderBlock (obj) or LURBorderBlock (obj) or RDLBorderBlock (obj))*
   4. **UDBorderBlock** (= *TwoBorderBlock and (hasBorder value "U") and hasBorder value "D"*), περιγράφει το τετράγωνο που έχει ακμές την πάνω και την κάτω πλευρά, με τον μοναδικό περιορισμό:
      a. *hasConnectingBlock (OP) max 2 (OneBorderBlock (obj) or CornerBlock (obj) or UDBorderBlock (obj) or URDBorderBlock (obj) or DLUBorderBlock (obj))*



Εικόνα 44 Οντότητα TwoBorderBlock

c. **ThreeBorderBlock**, η οντότητα εμφανίζεται στην Εικόνα 45 και περιγράφει ένα τετράγωνο το οποίο έχει ακριβώς δύο σύνορα σε τρεις διαδοχικές ακμές, πιο συγκεκριμένα έχουμε τις παρακάτω οντότητες με τους αντίστοιχους περιορισμούς:
   i. **DLUBorderBlock** (= *ThreeBorderBlock and (hasBorder value "D") and hasBorder value "L") and (hasBorder value "U")*), περιγράφει το τετράγωνο με ακμές τις κάτω, αριστερά και πάνω πλευρές του. Έχει ως μοναδικό περιορισμό:
      1. *hasConnectingBlock (OP) exactly 1 RLBorderBlock (obj)*
   ii. **URDBorderBlock** (= *ThreeBorderBlock and (hasBorder value "U") and hasBorder value "R") and (hasBorder value "D")*), περιγράφει το τετράγωνο με ακμές τις πάνω, δεξιά και κάτω πλευρές του. Ο μοναδικός περιορισμός του είναι ίδιος με την οντότητα **DLUBorderBlock.**
   iii. **LURBorderBlock** (= *ThreeBorderBlock and (hasBorder value "L") and hasBorder value "U") and (hasBorder value "R")*), περιγράφει το τετράγωνο με ακμές τις αριστερά, πάνω και δεξιά πλευρές. Έχει ως μοναδικό περιορισμό:
      1. *hasConnectingBlock (OP) exactly 1 UDBorderBlock (obj)*
   iv. **RDLBorderBlock** (= *ThreeBorderBlock and (hasBorder value "R") and hasBorder value "D") and (hasBorder value "L")*), περιγράφει το τετράγωνο με ακμές τις δεξιά, κάτω και αριστερή πλευρές του. Ο μοναδικός περιορισμός του είναι ίδιος με την οντότητα **LURBorderBlock**



Εικόνα 45 Οντότητα ThreeBorderBlock

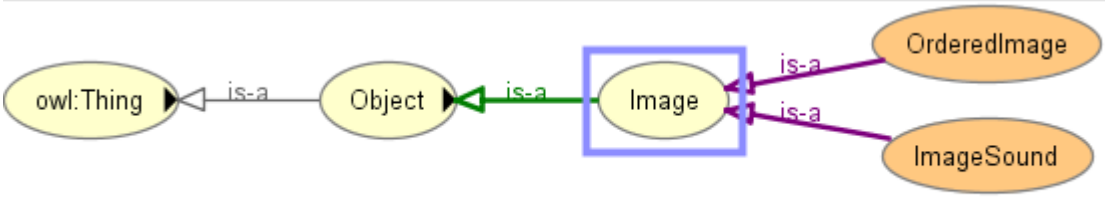
2. **NumberBlock**, η οντότητα αυτή όπως εμφανίζεται στην Εικόνα 42, περιγράφει ένα τετράγωνο το οποίο περιέχει έναν αριθμό
3. **OperatorBlock**, η οντότητα αυτή όπως εμφανίζεται στην Εικόνα 42, περιγράφει ένα τετράγωνο το οποίο περιέχει έναν μαθηματικό συντελεστή. Επίσης ενώνεται με γειτονικά **NumberBlock** μέσω του OP *hasConnectingBlock*.

Image

Η οντότητα αυτή εμφανίζεται στην Εικόνα 46 και περιγράφει μία εικόνα. Κάθε εικόνα συσχετίζεται μέσω των OP hasTitle και hasSubject με δύο οντότητες Word. Με τον τρόπο αυτό

μπορεί μια εικόνα να κατηγοριοποιηθεί με βάση την συσχέτιση hasSubject και να δημιουργηθούν αντίστοιχοι κανόνες που να ζητούν εικόνες από μία κατηγορία. Επίσης με τον τρόπο κάποια παιχνίδια που έχουν ως κύριο αντικείμενο μια οντότητα Word, μπορούν να περιέχουν κανόνες με τις αντίστοιχες εικόνες, όπως για παράδειγμα να βρεθούν τα αντίθετα συναισθήματα.

1. **OrderedImage**, περιγράφει μια εικόνα και συσχετίζεται με μία άλλη μέσω του OP *hasPreviousImage*. Παρατηρήθηκε ότι κάθε εικόνα μπορεί να έχει το πολύ μια προηγούμενη, πράγμα που καθιστά την συσχέτιση ως Functional
2. **ImageSound**, περιγράφει την συσχέτιση μίας εικόνας με έναν ήχο μέσω του OP *hasAssociatedSound*

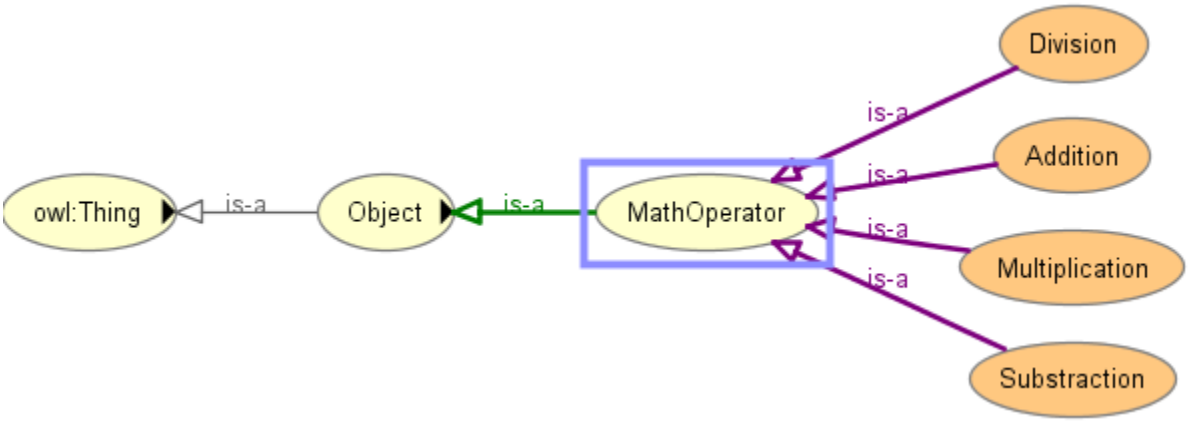


Εικόνα 46 Οντότητα Image

MathOperator

Η οντότητα αυτή εμφανίζεται στην Εικόνα 47 και περιγράφει μία αριθμητική πράξη

1. **Addition**, περιγράφει την πρόσθεση και έχει ως περιορισμούς την ύπαρξη δύο προσθετέων και ενός αθροίσματος
2. **Division**, περιγράφει την διαίρεση και έχει ως περιορισμούς την ύπαρξη ενός διαιρετέου, ενός διαιρέτη και ενός πηλίκου
3. **Multiplication**, περιγράφει τον πολλαπλασιασμό και έχει ως περιορισμούς την ύπαρξη ενός πολλαπλασιαστέου, ενός πολλαπλασιαστή και ενός γινομένου
4. **Substraction**, περιγράφει την αφαίρεση και έχει ως περιορισμούς την ύπαρξη ενός μειωτέου, ενός αφαιρετέου και μίας διαφοράς



Εικόνα 47 Οντότητα MathOperator

Piece

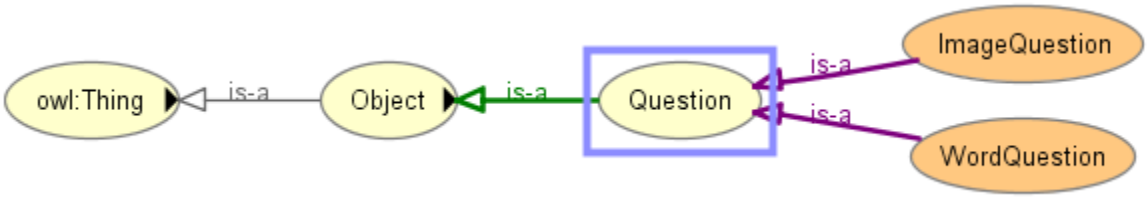
Η οντότητα αυτή εμφανίζεται στην Εικόνα 48 και περιγράφει ένα κομμάτι παζλ. Για την δημιουργία της οντότητας αυτής είναι αναγκαίο να υπάρχουν συσχετίσεις με άλλα αντικείμενα μέσω του OP *hasConnectingPiece***,** παρατηρήθηκε ότι κάθε κομμάτι ενώνεται με 2 έως 4 άλλα κομμάτια. Καθώς κάθε κομμάτι δεν μπορεί να ενωθεί με τον εαυτό του η συσχέτιση αυτή έχει χαρακτηρισθεί ως Irreflexive. Επίσης η συσχέτιση είναι αμφίδρομη οπότε έχει χαρακτηρισθεί ως Symmetric.

Εικόνα 48 Οντότητα Piece

Question

Η οντότητα αυτή εμφανίζεται στην Εικόνα 49 και περιγράφει μία ερώτηση. Παρατηρήθηκε ότι κάθε ερώτηση ανήκει σε τουλάχιστον μία κατηγορία. Η συσχέτιση αυτή χαρακτηρίζεται μέσω του OP *hasCategory*

1. **ImageQuestion**, περιγράφει μια ερώτηση με τους περιορισμούς αυτής καθώς επίσης ότι έχει εκφώνηση μια εικόνα που περιγράφει τι ζητείται να βρεθεί ή να επιλεχθεί. Η συσχέτιση αυτή γίνεται μέσω του OP *hasImage* και είναι υποχρεωτικό να υπάρχει συσχέτιση με μια εικόνα
2. **WordQuestion,** περιγράφει μια ερώτηση με τους περιορισμούς αυτής καθώς επίσης ότι έχει μια εκφώνηση-κείμενο



Εικόνα 49 Οντότητα Question

Sound

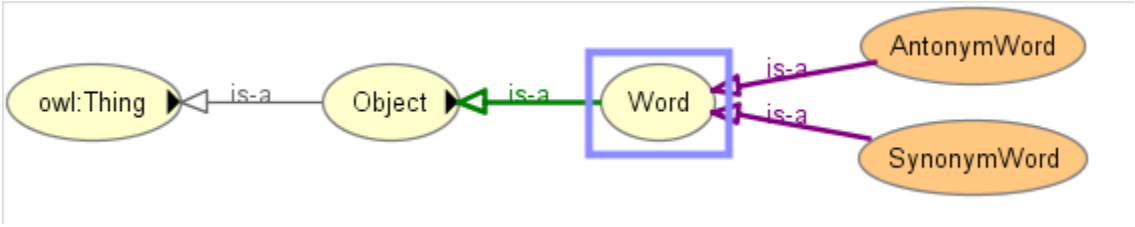
Η οντότητα αυτή εμφανίζεται στην Εικόνα 50 και περιγράφει έναν ήχο. Κάθε ήχος συσχετίζεται με μια οντότητα Word μέσω του OP *hasSubject* και περιγράφει το αντικείμενο που συμβολίζει ο ήχος. Καθώς και μία οντότητα Image συσχετίζεται με μία οντότητα Word μέσω της ίδιας συσχέτισης, είναι εύκολο να συνδυαστεί μία εικόνα με έναν ήχο.

Εικόνα 50 Οντότητα Sound

Word

Η οντότητα αυτή εμφανίζεται στην Εικόνα 51 και περιγράφει μια λέξη

1. **AntonymWord**, περιγράφει μια λέξη και συσχετίζεται με τουλάχιστον μία άλλη η οποία αποτελεί ένα αντώνυμο της πρώτης. Η συσχέτιση αυτή είναι υποχρεωτική και γίνεται μέσω του OP hasAntonym. Επειδή μία λέξη δεν αποτελεί αντώνυμο του εαυτού της, η συσχέτιση αυτή έχει χαρακτηρισθεί ως Irreflexive. Επίσης η ίδια λέξη αποτελεί αντώνυμο της άλλης, για το λόγο αυτό η συσχέτιση αυτή είναι Symmetric.
2. **SynonymWord**, περιγράφει μια λέξη και συσχετίζεται με τουλάχιστον μία άλλη η οποία αποτελεί ένα συνώνυμο της πρώτης. Παρόμοια με την οντότητα AntonymWord, υπάρχει η Irreflexive και Symmetric συσχέτιση hasSynonym, η οποία ενώνει δύο λέξεις.



Εικόνα 51 Οντότητα Word

4.4 Συσχετίσεις

Όπως αναφέρθηκε παραπάνω, η οντολογία περιέχει κάποιες συσχετίσεις οι οποίες ενώνουν τα παίγνια με τα αντικείμενα και κάποια αντικείμενα με κάποια άλλα. Παρακάτω εξηγούνται καλύτερα οι συσχετίσεις αυτές και ποιος είναι ο ρόλος τους. Για την καλύτερη κατανόηση, οι συσχετίσεις θα αναγραφούν με την μορφή property(D: Domains, R: Range) και με «*» σημαίνει ότι αποτελούν περιορισμό είτε για παίγνιο, είτε για την δημιουργία κάποιου αντικειμένου.

4.4.1 Object Properties

1. ***hasAntonym* (D: Word, R: Word)***: Η συσχέτιση αυτή ενώνει δύο οντότητες Word και υποδηλώνει ότι η μία λειτουργεί ως αντώνυμο της άλλης. Καθώς μία λέξη μπορεί να έχει παραπάνω από ένα αντώνυμο, αλλά όχι τον εαυτό της, η συσχέτιση αυτή έχει δηλωθεί ως Irreflexive. Τέλος η συσχέτιση αυτή είναι αμφίδρομη, δηλαδή όταν μία λέξη έχει ως αντώνυμο μια άλλη λέξη, τότε και η άλλη έχει την πρώτη ως αντώνυμο . Για τον λόγο αυτό έχει σημειωθεί η ιδιότητα Symmetric.
2. ***hasSynonym* (D*: Word, R: Word)**: παρόμοια με την συσχέτιση hasAntonym, υπάρχει και η αντίστοιχη συσχέτιση για τα συνώνυμα. Οι ιδιότητες και στις δύο περιπτώσεις είναι ίδιες καθώς οι ίδιοι κανόνες ισχύουν και για τις δύο.
3. ***hasAssociatedSound* (D: Image, R: Sound)***: Με την συσχέτιση αυτή υποδηλώνεται ένωση μεταξύ μίας οντότητας **Image** με μία τύπου **Sound**.
4. ***hasAssociatedImage (D: Sound, R: Image)***: Αντίστροφη συσχέτιση της **hasAssociatedSound**. Επειδή όταν έχει δηλωθεί ως αντίστροφη σημαίνει ότι όταν μία οντότητα Image ενωθεί με μία οντότητα Sound, τότε η οντότητα Sound θα ενωθεί με την ίδια οντότητα Image μέσω της **hasAssociatedImage**. Ο σκοπός της συσχέτισης αυτής είναι ο εμπλουτισμός της πληροφορίας της οντολογίας και όχι κάποιος περιορισμός για την δημιουργία κάποιου παίγνιου ή αντικειμένου.
5. ***hasBlock* (D: BlockSet or Calculation or HidingBlocks or Labyrinth or LogicalOrder, R: Block):** Η συσχέτιση αυτή είναι γενική καθώς συσχετίζει παίγνια ή αντικείμενα με μια οντότητα **Block** χωρίς να παρέχεται κάποια παραπάνω γνωστική πληροφορία. Η συσχέτιση αυτή χρησιμοποιείται όταν θέλει να ορισθεί ο συνολικός αριθμός των **Block** που μπορεί να έχει μία οντότητα από το Domain. Καθώς η συσχέτιση αυτή έχει sub properties, αυτό σημαίνει ότι ο αριθμός των συσχετίσεων των sub properties συμπεριλαμβάνεται στο συνολικό αριθμό της συσχέτισης hasBlock. Για παράδειγμα για το παίγνιο **HidingBlocks** έχουν ορισθεί οι παρακάτω περιορισμοί: *hasBlock exactly 9 Block **AND** hasHidingBlock exactly 2 Block*. Αυτό σημαίνει ότι στο σύνολο των 9 Block τα 2 θα συσχετίζονται μέσω της συσχέτισης hasHidingBlock. Η συσχέτιση **hasBlock** έχει τα εξής sub properties:
    a. ***hasHidingBlock* (D: HidingBlocks, R: Block)***: Η συσχέτιση αυτή υποδηλώνει ότι η οντότητα Block έχει χαρακτηριστεί ως hiding για το παίγνιο HidingBlocks

b. ***hasMovingBlock* (D: BlockSet, R: Block)***: Η συσχέτιση αυτή υποδηλώνει ότι η επιλεγμένη οντότητα Block μετακινείται στα πλαίσια της οντότητας **Blockset,** η οποία χρησιμοποιείται από το παίγνιο LogicalOrder,
c. ***hasNumberBlock* (D: Calculation, R: Block)***: Η συσχέτιση αυτή υποδηλώνει ότι το συγκεκριμένο **Block** για το παίγνιο **Calculation** περιέχει αριθμό.
d. ***hasOperatorBlock* (D: Calculation, R: OperatorBlock)***: Η συσχέτιση αυτή υποδηλώνει ότι το συγκεκριμένο **OperatorBlock** για το παίγνιο **Calculation** περιέχει μαθηματικό σύμβολο.

6. ***hasBlockSet* (D: LogicalOrder, R: BlockSet)***: Η συσχέτιση αυτή δηλώνει ποιες οντότητες τύπου **BlockSet** ανήκουν στο παίγνιο **LogicalOrder.** Το μοναδικό sub property της συσχέτισης αυτής είναι το :
    a. ***hasCorrectBlockSet*(D: LogicalOrder, R: BlockSet)***, το οποίο δηλώνει τη σωστή επιλογή του παίγνιου
7. ***hasCategory*(D: Question or Questions, R: Word)***: η συσχέτιση προσδίδει μία κατηγορία σε μία ερώτηση
8. ***hasChoice*(D: Questions, R: Word)***: η συσχέτιση αυτή υποδηλώνει τις πιθανές απαντήσεις μίας ερώτησης. Το μοναδικό sub property που έχει είναι το εξής:
    a. ***hasAnswer*(D: Questions, R: Word)***, που υποδηλώνει τη σωστή απάντηση.
9. ***hasConnectingBlock*(D: Block, R: Block)***: υποδηλώνει την ένωση δύο οντοτήτων **Block**. Έχει σημανθεί ως *Irreflexive,* καθώς ένα **Block** δεν μπορεί να ενωθεί με τον εαυτό του και ως *Symmetric*, καθώς η σχέση αυτή είναι αμφίδρομη.
10. ***hasConnectingPiece*(D: Piece, R: Piece)***: υποδηλώνει την ένωση δύο οντοτήτων **Piece**. Έχει σημανθεί ως *Irreflexive,* καθώς ένα **Piece** δεν μπορεί να ενωθεί με τον εαυτό του και ως *Symmetric*, καθώς η σχέση αυτή είναι αμφίδρομη.
11. ***hasMathOperator*(D: Calculation, R: OperatorBlock)***: δηλώνει ποια τετράγωνα περιέχουν μαθηματικές πράξεις για το παίγνιο **Calculation**
12. ***hasObject*(D: Antonyms or ChronologicalOrder or FindTheSound or MemoryCards or Observation or ObservationObj or Piece or Puzzle or Question or Synonyms or WordPuzzle, R: Image or Sound or Word)***: η συσχέτιση αυτή είναι γενική και μπορεί να συσχετίσει συγκεκριμένα παίγνια και αντικείμενα με οντότητες τύπου **Image, Sound** ή **Word.** Ο λόγος δημιουργίας μίας τέτοιας συσχέτισης είναι η προσπάθεια της δημιουργίας κλειστού – γενικού τύπου περιορισμού ώστε στο μέλλον να καθίσταται εφικτή η διαφοροποίηση των ήδη υπάρχων παιγνίων ώστε να μπορούν να χρησιμοποιηθούν άλλα μέσα. Για παράδειγμα ένα παίγνιο τύπου **Questions** στο οποίο η ερώτηση να είναι ένας ήχος και να ως απαντήσεις κάποιες λέξεις. Η γενική αυτή συσχέτιση περιέχει τα εξής sub properties τα οποία μικραίνουν το φάσμα ευχρηστίας συνειδητά.
    a. ***hasImage*(D: ChronologicalOrder or FindTheSound or ObservationObj or Piece or Puzzle or Question, R: Image)***, συσχετίζει μία εικόνα με μία από τις οντότητες του Domain. Κατά εξακολούθηση το μοναδικό sub property αυτής της συσχέτισης είναι το εξής:

   i. ***hasOrderedImage*(D: ChronologicalOrder, R: OrderedImage)***, που συσχετίζει εικόνες με συγκεκριμένη σειρά για το παίγνιο **ChronologicalOrder**
- b. ***hasSound*(D: FindTheSound, R: Sound),*** που συσχετίζει τον ζητούμενο ήχο για το παίγνιο FindTheSound
- c. ***hasWord*(D: Antonyms or Observation or Synonyms or WordPuzzle, R: Word)***, η οποία συσχετίζει μία λέξη. Κατά εξακολούθηση η συσχέτιση αυτή έχει ως μοναδικό sub property
   i. ***hasMainWord*(D: Antonyms or Synonyms, R: Word)***, η οποία δηλώνει τη λέξη για την οποία πρέπει να βρεθεί το αντώνυμο ή το συνώνυμο για τα παίγνια Antonyms και Synonyms. Καθώς τα παίγνια αυτά έχουν μόνο μία τέτοια λέξη, η συσχέτιση αυτή έχει την ιδιότητα *Functional***.**

13. ***hasObservation*(D: Observation, R: ObservationObj)***: η συσχέτιση αυτή δηλώνει τις παρατηρήσεις για το παίγνιο **Observation**
14. ***hasPiece*(D: Puzzle, R: Piece)***: υποδηλώνει τις οντότητες τύπου **Piece** που ανήκουν στο παίγνιο **Puzzle.** Η συσχέτιση αυτή έχει τα παρακάτω sub properties τα οποία το καθένα υποδηλώνει τον τύπο του **Piece**, για παράδειγμα αν είναι κάποιο γωνιακό κομμάτι:
   a. ***hasBorderLinePiece* (D: Puzzle, R: Piece)***
   b. ***hasCornerPiece*(D: Puzzle, R: Piece)***
   c. ***hasInteriorPiece*(D: Puzzle, R: Piece)***
15. ***hasPreviousBlockSet*(D: BlockSet, R: BlockSet)***, σκοπός της συσχέτισης αυτής είναι να δώσει κάποια σειρά για τα αντικείμενα **BlockSet**, για το παίγνιο **LogicalOrder,** ώστε να επαληθευθεί η απάντηση του χρήστη. Καθώς το προηγούμενο σε σειρά **BlockSet** μπορεί να είναι μόνο ένα, η συσχέτιση έχει δηλωθεί ως *Functional* και ως *Irreflexive* επειδή το προηγούμενο **BlockSet** δεν μπορεί να είναι το ίδιο.
16. ***hasPreviousImage*(D: OrderedImage, R: OrderedImage)*****,** παρόμοια με τη συσχέτιση ***hasPreviousBlockSet***
17. ***hasQuestion*(D: Questions, R: Question)***: η ερώτηση για το παίγνιο **Questions**. Καθώς κάθε τέτοιο παίγνιο έχει μία ερώτηση, η συσχέτιση αυτή έχει δηλωθεί ως *Functional*
18. ***hasQuestion*(D: Questions, R: Question)***: η ερώτηση για το παίγνιο **Questions**. Καθώς κάθε τέτοιο παίγνιο έχει μία ερώτηση, η συσχέτιση αυτή έχει δηλωθεί ως *Functional*
19. ***hasSubject*(D: Image or Sound, R: Word)*** η συσχέτιση αυτή υποδηλώνει το αντικείμενο που δείχνει μία εικόνα ή που κάνει τον συγκεκριμένο ήχο
20. ***hasTitle*(D: Image or Sound, R: Word)*** η συσχέτιση αυτή υποδηλώνει τον τίτλο μίας εικόνας ή ενός ήχου.

4.5 Επεκτασιμότητα Οντολογίας

Η Οντολογία έχει δημιουργηθεί με τέτοιο τρόπο ώστε να είναι ανοιχτή σε αλλαγές των ήδη υπάρχοντών κανόνων, καθώς και στην δημιουργία νέων. Οι αλλαγές αυτές μπορεί να επιφέρουν αλλαγές και στην εφαρμογή. Οι αλλαγές αυτές αναφέρονται στην Ενότητα 5.5 Επεκτασιμότητα εφαρμογής.

4.5.1 Δημιουργία νέου παιγνίου

Για την δημιουργία ενός νέου παιγνίου, η οντότητα θα πρέπει να δημιουργηθεί ως υποκλάση της οντότητας *Game*, με όνομα, το όνομα του παιγνίου. Με τον τρόπο αυτό επιτυγχάνουμε την κληρονομικότητα κάποιων ιδιοτήτων, οι οποίες είναι κοινές για όλα τα παίγνια, όπως αναφέρονται στην υπό-ενότητα 4.3.1 Game.

Περιορισμοί

Στο σημείο αυτό μπορούμε να δημιουργήσουμε διάφορους περιορισμούς για το παίγνιο αυτό. Η υπό-ενότητα 4.3.1 Game αναφέρει τέτοια παραδείγματα όπως για το παίγνιο *Synonims* έχουμε τον περιορισμό *hasMainWord(OP) 1 Word(obj).* Στο παράδειγμα αυτό μπορούμε να διαπιστώσουμε ότι ο περιορισμός αυτός είναι κοινός για όλα τα επίπεδα δυσκολίας. Κοινώς, ενθαρρύνεται η τακτική αυτή καθώς έτσι ο περιορισμός αυτός θα κληρονομηθεί σε πιθανές υποκλάσεις.

Ιεραρχία

Ο τρόπος με τον οποίο θα ιεραρχηθεί το νέο παίγνιο είναι ανάλογος με τους κανόνες του παιγνίου και πως αυτό συναναστρέφεται με τα αντικείμενα. Για παράδειγμα έστω το νέο παίγνιο *NewGame* μπορεί να χρησιμοποιεί είτε εικόνες, είτε ήχους. Μία καλή τακτική που μπορεί να χρησιμοποιηθεί είναι η δημιουργία των υποκλάσεων *ImageNewGame* με περιορισμό την ιδιότητα *hasImage* και *SoundNewGame* με περιορισμό την ιδιότητα *hasSound.* Με τον τρόπο αυτό επιτυγχάνεται η διαφοροποίηση της υλοποίησης ενός στιγμιότυπου στο να χρησιμοποιεί εικόνες ή ήχους.

Disjoint

Επίσης είναι σημαντικό για κάθε νέα οντότητα που δημιουργείται να δηλωθεί ως **disjoint** από της γειτονικές οντότητες (siblings). Ο ορισμός **disjoint** σημαίνει ότι τα στιγμιότυπα της οντότητας δεν μπορούν να ανήκουν και σε μία άλλη οντότητα. Το **disjointness** των οντοτήτων βοηθάει ώστε να υπάρχει μία συνεπής οντολογία και με τον τρόπο αυτό αποφεύγονται πιθανά λογικά λάθη. Στο παράδειγμα του παιγνίου *NewGame,* εάν δεν δηλωθούν οι υποκλάσεις *ImageNewGame* και *SoundNewGame* ως **disjoint** υπάρχει η πιθανότητα ένα στιγμιότυπο *ImageNewGame* να ανήκει και στην οντότητα *SoundNewGame.*

Επίπεδο Δυσκολίας

Έχει παρατηρηθεί ότι ανάλογα του επιπέδου δυσκολίας, αλλάζουν ο αριθμός των αντικειμένων που είναι διαθέσιμα στον παίκτη. Ένα πάζλ θα έχει περισσότερα κομμάτια καθώς η δυσκολία αυξάνεται, ή σε ένα παίγνιο καρτών μνήμης θα υπάρχουν περισσότερα ζεύγη. Για την διευθέτηση αυτού του προβλήματος μπορούν να δημιουργηθούν νέες οντότητες που αποτελούν υποκλάσεις της οντότητας του παιγνίου, οι οποίες χειρίζονται τέτοιου είδους κανόνες, όπως τον συνολικό

αριθμό των κομματιών που έχει ένα παζλ. Τα διαθέσιμα επίπεδα δυσκολίας είναι τα εξής *EASY, MEDIUM και HARD.*

Οι υποκλάσεις που δημιουργούνται για κάθε επίπεδο δυσκολίας ακολουθούν μία συγκεκριμένη συνθήκη ονομασίας. Αυτό γίνεται ώστε να μπορούν να ανιχνευθούν εύκολα από την εφαρμογή οι περιορισμού για κάθε επίπεδο δυσκολίας. Η συνθήκη ονομασίας είναι η εξής, *<Δυσκολία><Όνομα_παιγνίου>*. Έτσι για το νέο παίγνιο *ImageNewGame* θα έχουμε τις εξής υποκλάσεις *EasyImageNewGame, MediumImageNewGame, HardImageNewGame*.

Μία οντότητα δυσκολίας παιγνίου είναι ίση με την οντότητα παιγνίου όταν αυτή έχει κάποιες συγκεκριμένες τιμές για κάποια **DataProperties**. Δύο **DataProperties** που κληρονομούνται από την οντότητα *Game* είναι τα *maxCompletionTime* και *hasDifficulty*. Στο παράδειγμα του παιγνίου *ImageNewGame*, η οντότητα *EasyImageNewGame* μπορεί να δημιουργηθεί ως εξής, *ImageNewGame AND hasDifficulty value ”EASY" AND maxCompletionTime value 180000.* Αντίστοιχα για την οντότητα *MediumImageNewGame* θα έχουμε, *ImageNewGame AND hasDifficulty value ”MEDIUM" AND maxCompletionTime value 180000*. Ο χρόνος ολοκλήρωσης του επιπέδου μπορεί να διαφέρει με το επίπεδο δυσκολίας και μπορεί να τεθεί η επιθυμειτή τιμή για κάθε επίπεδο δυσκολίας μέσω του **dataProperty** *maxCompletionTime*. Επίσης ανάλογα με τους κανόνες του παιγνίου, μπορούν να δημιουργηθούν άλλα παρόμοια dataProperties των οποίων οι τιμές τους είναι γνωστές για κάθε επίπεδο δυσκολίας. Για παράδειγμα ο αριθμός των στηλών και των γραμμών για το παίγνιο *LogicalOrder* όπως περιγράφεται στην υπό-ενότητα 4.3.1 Game

4.5.2 Δημιουργία νέου αντικειμένου

Επίσης είναι εφικτή η δημιουργία ενός νέου αντικειμένου. Κάθε νέο αντικείμενο που δημιουργείται θα πρέπει να είναι υποκλάση της οντότητας *GameObject.* Με τον τρόπο αυτό ομαδοποιούνται όλες οι οντότητες που περιγράφουν αντικείμενα. Πριν από την εισαγωγή νέου αντικειμένου είναι σημαντικό να αναγνωριστεί η ανάγκη για την δημιουργία του και αν μπορεί να καλυφθεί από τα ήδη υπάρχοντα αντικείμενα, καθώς ήδη υπάρχει ήδη μία ευρεία επιλογή από αντικείμενα.

4.5.3 Δημιουργία νέων συσχετίσεων-ιδιοτήτων

Η οντότητα υποστηρίζει την δημιουργία νέων συσχετίσεων-ιδιοτήτων ώστε να υποστηρίζει τυχών νέες ανάγκες για νέα παίγνια. Η αλλαγή των ήδη υπαρχόντων μπορεί να επιφέρει αλλαγές στην οντολογία αλλά και στην εφαρμογή όπως περιγράφονται στην Ενότητα 5.5 Επεκτασιμότητα εφαρμογής

5

Εφαρμογή MCI

5.1 Αρχιτεκτονική Server

Ένας άλλος σημαντικός στόχος της εργασίας ήταν η υλοποίηση ενός server ο οποίος θα δημιουργεί αυτόματα ένα παίγνιο. Η αρχιτεκτονική που έχει επιλεχθεί για τον σκοπό αυτό μιμείται την αρχιτεκτονική των microservices (Bakshi, 2017), όπως έχει ήδη αναφερθεί. Συνήθως τα microservices επικοινωνούν μεταξύ τους με HTTP, όμως στα πλαίσια τις εργασίας χρησιμοποιήθηκε το design pattern *dependency injection* (*DI*). Το dependency injection είναι ένα design pattern το οποίο αυξάνει το decoupling μεταξύ των κλάσεων ενός προγράμματος επιτυγχάνοντας έτσι μεγαλύτερη απομόνωση μεταξύ των κλάσεων, ευκολότερη δοκιμή των λειτουργιών καθώς και την υποστήριξη της αρχιτεκτονικής των microservices (Prasanna, 2009). Ο λόγος που ακολουθήθηκε αυτή η τεχνική είναι οι περιορισμοί στους διαθέσιμους υπολογιστικούς πόρους.

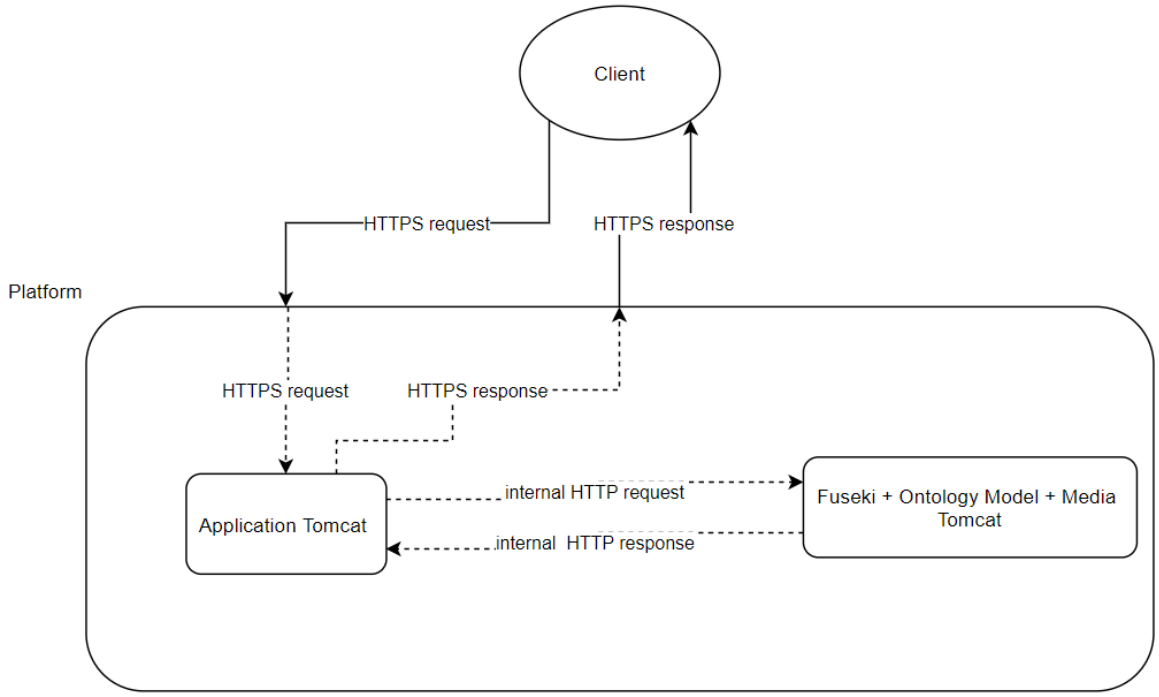
Εφόσον η αρχιτεκτονική για τον server μιμείται τα microservices αναζητήθηκε μια τεχνική για την υλοποίησή της. Μια τεχνική που ακολουθείται πρόσφατα για το deployment των microservices είναι η χρήση Docker Images.

5.1.1 Docker

Το Docker είναι μία τεχνολογία η οποία βασίζεται στην έννοια του containerization, η οποία υποστηρίζεται από τα microservices. Αυτό σημαίνει ότι κάθε εφαρμογή, στη συγκεκριμένη περίπτωση είναι microservice, μπορεί να πακεταριστεί σε ένα container και να γίνει deployed ανεξάρτητα από τα υπόλοιπα containers. Ένα container μπορεί να θεωρηθεί ως ένα virtual machine το οποίο μπορεί να ρυθμιστεί ανάλογα με την υπηρεσία που περιέχει. Αυτό έρχεται σε ευθυγράμμιση με τα προτερήματα των microservices, που τα θέλουν ανεξάρτητα μεταξύ τους και οι πόροι μπορούν να κατανεμηθούν αντιστοίχως της ζήτησης κάθε υπηρεσίας. Στην περίπτωση αυτής της εργασίας κάθε service αναπαριστά ένα παίγνιο. Αυτό σημαίνει ότι θα πρέπει να δημιουργηθούν ο αντίστοιχος αριθμός docker containers ώστε να ακολουθηθεί πλήρως η έννοια των microservices. Κάτι τέτοιο καθίσταται δύσκολο καθώς για την εκπόνηση της εργασίας αυτής διατίθεται ένας τυπικός υπολογιστής με πεπερασμένους πόρους.

5.1.2 Tomcat

Εξαιτίας των περιορισμένων πόρων αποφασίστηκε η χρήση ενός Tomcat server για την εκτέλεση των services – παιγνίων και ενός δεύτερου Tomcat server ο οποίος περιέχει το μοντέλο της οντολογίας, που πρέπει να «φορτωθεί» στη μνήμη της εφαρμογής κατά την εκκίνησή της, πιθανά αρχεία τα οποία χρειάζονται από κάποια αντικείμενα της οντολογίας, όπως εικόνες, και φυσικά τη βάση γνώσης με όλα τα στιγμιότυπα των παιγνίων. Στο σημείο αυτό χρησιμοποιείται ο Apache Fuseki για την υλοποίηση της βάσης αυτής και για τη χρήση του SPARQL πρωτοκόλλου. Η Εικόνα 52 απεικονίζει τη διάταξη στοιχείων του server.



Εικόνα 52 Διάταξη στοιχείων του server

5.2 Αρχιτεκτονική Εφαρμογής

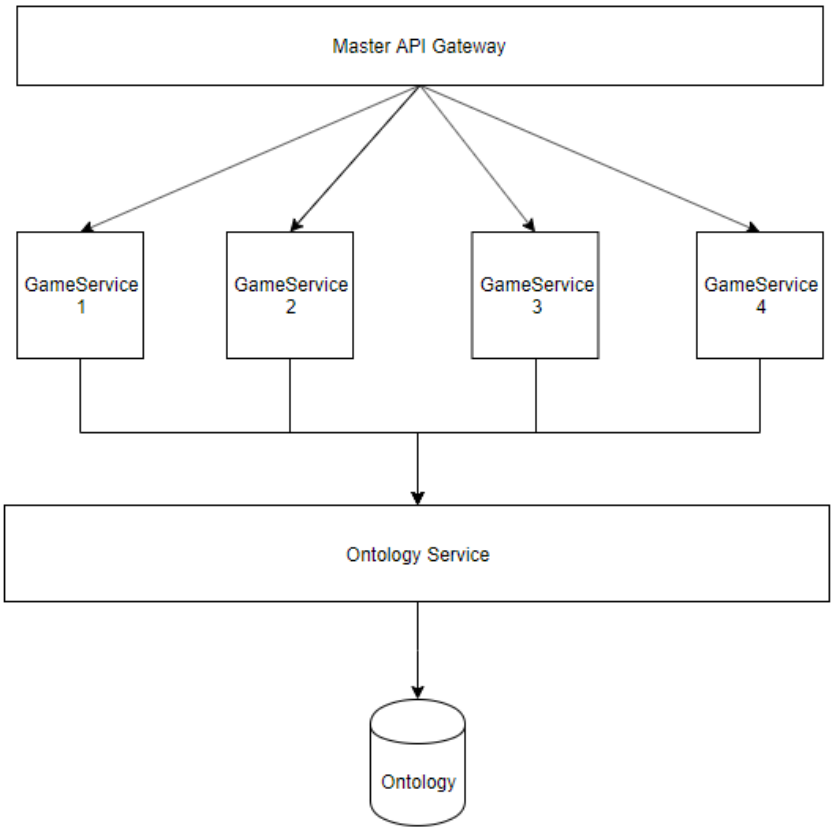
Επειδή η αρχιτεκτονική του server μιμείται τα microservices, αυτό συνεπάγεται ότι κάθε service είναι ανεξάρτητο από τα υπόλοιπα. Στην περίπτωση αυτής της εργασίας κάθε service αναπαριστά ένα παίγνιο. Κάθε service περιέχει την λογική η οποία περιγράφει ποια αντικείμενα θα χρησιμοποιηθούν για το παίγνιο καθώς επίσης αν κάποιο αντικείμενο χρειαστεί να δημιουργηθεί, ποια θα είναι η τιμή του σύμφωνα με τους περιορισμούς που έχουν αναπαρασταθεί στην οντολογία. Επίσης κάθε service θα πρέπει να είναι προσπελάσιμο από το διαδίκτυο οπότε θα πρέπει να υλοποιηθεί για κάθε service ένα API gateway. Με τον τρόπο αυτό επιτυγχάνεται η απομόνωση των services μεταξύ τους και στην περίπτωση που απαιτηθεί ένα service να επικοινωνήσει με κάποιο άλλο τότε μπορεί να χρησιμοποιηθεί το πρωτόκολλο HTTPS.

Εκτός των services – παιγνίων υπάρχει το service που είναι υπεύθυνο για την ανάγνωση των κανόνων κάθε οντότητας και επικοινωνεί με τη βάση δεδομένων για τη δημιουργία των ζητούμενων οντοτήτων. Το service αυτό «κρύβει» από τα υπόλοιπα services πληροφορίες σχετικά με τη βάση της οντολογίας καθώς επίσης και την ίδια οντολογία. Με τον τρόπο αυτό είναι δυνατή η αλλαγή της τεχνολογίας της βάσης δεδομένων, καθώς επίσης και της οντολογίας.

Η εφαρμογή πελάτη (π.χ. μια εφαρμογή υγείας, ένας agent, ένας ειδικός του χώρου υγείας μέσω ενός front-end) μπορεί να έχει πρόσβαση μόνο στα services των παιγνίων και όχι στο service της οντολογίας. Επειδή τα services έχουν δικά τους API gateways για την μεταξύ τους επικοινωνία,

είναι αναγκαία η δημιουργία ενός master API gateway το οποίο εξαγάγει μόνο τα gateways των services – παιγνίων.

Η αρχιτεκτονική της εφαρμογής αναπαρίσταται στην Εικόνα 53.



Εικόνα 53 Αρχιτεκτονική Εφαρμογής

Όλα τα gateways της εφαρμογής μοιράζονται κάποια κοινά χαρακτηριστικά. Τα gateways δέχονται μόνο requests με την μορφή *application/json* και απαντούν στην ίδια μορφοποίηση. Ο λόγος που επιλέχτηκε η μορφοποίηση αυτή είναι η ευκολία χρήσης τόσο από τον client όσο και από τον server. Επίσης επιτυγχάνεται γρήγορα και εύκολα τα Serialization/Deserialization από JSON σε Java Bean και αντίστροφα. Επίσης όλα τα Requests πρέπει να θέτουν υποχρεωτικά το request header *X-INFO-PLAYER* με τιμή το όνομα του χρήστη. Καθώς δεν έχει υλοποιηθεί κάποιος μηχανισμός login, το header αυτό θα επιτρέψει στο μέλλον τη δημιουργία του χωρίς να γίνουν σημαντικές αλλαγές στην υλοποίηση όλων των gateway των services. Στον Πίνακας 1 αναγράφονται τα URL με τη μορφή resources τα οποία υλοποιούνται από όλα τα Services καθώς και τα ρήματα HTTP που χρησιμοποιούνται και την περιγραφή τους.

HTTP Verb	Resource URL	Περιγραφή
GET	/mci/{resource}	Επιστρέφει όλα τα instances τύπου {resource}

POST	/mci/{resource}	Δημιουργία ενός instance τύπου {resource}
GET	/mci/{resource}/{id}	Επιστρέφει το instance τύπου {resource} με αναγνωριστικό {id}
PUT	/mci/{resource}/{id}	Ανανέωση του instance τύπου {resource} με αναγνωριστικό {id}. Το ρήμα αυτό χρησιμοποιείται για την λύση ενός παιγνίου

Πίνακας 1 Μορφή HTTP URL της εφαρμογής

Από τον παραπάνω πίνακα τα εξής resources έχουν αναπτυχθεί στην εφαρμογή

Όνομα	Περιγραφή
antonyms	Αναφέρεται στο παίγνιο αντωνυμιών
observations	Αναφέρεται στο παίγνιο παρατηρήσεων
wordPuzzle	Αναφέρεται στο παίγνιο παζλ λέξεων
synonyms	Αναφέρεται στο παίγνιο συνωνυμιών
recall	Αναφέρεται στο παίγνιο ανάμνησης αριθμού
numberOrder	Αναφέρεται στο παίγνιο σειράς αριθμών
memoryCards	Αναφέρεται στο παίγνιο κάρτες μνήμης
chronologicalOrders	Αναφέρεται στο παίγνιο χρονολογικής σειράς
findTheSounds	Αναφέρεται στο παίγνιο συσχέτισης εικόνας με ήχο
questions	Αναφέρεται στο παίγνιο ερωτήσεων
hidingBlocks	Αναφέρεται στο παίγνιο εύρεσης των κρυμμένων τετραγώνων
logicalOrder	Αναφέρεται στο παίγνιο λογικής σειράς
puzzles	Αναφέρεται στο παίγνιο λύσης ενός παζλ

Πίνακας 2 Διαθέσιμα resources

Τέλος για το id κάθε στιγμιότυπου ακολουθήθηκε το εξής μοτίβο:

Game id = resourceName_DIFFICULTY_user_Level

Με τον τρόπο αυτό επιτυγχάνεται η εύκολη και μοναδική δημιουργία του, καθώς επίσης και η εύκολη ανάγνωση από τον προγραμματιστή για πιθανή αποσφαλμάτωση. Για παράδειγμα, το id Antonyms_MEDIUM_Postman_3 υποδηλώνει ένα παίγνιο τύπου Antonyms, δυσκολίας MEDIUM, χρήστη Postman για το επίπεδο 3.

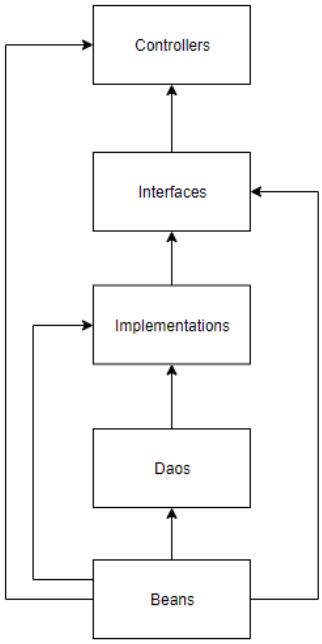
5.2.1 Δομή Service

Κάθε service έχει συγκεκριμένο τρόπο δόμησης ώστε να επιτευχθεί ομοιογένεια μεταξύ των διαφορετικών services και απομόνωση των πιθανών διαφορετικών λειτουργιών εντός του service. Κάθε service αποτελείται από τα παρακάτω μέρη:

1. **Controllers**, το κομμάτι αυτό είναι υπεύθυνο για την δημιουργία του API Gateway του service. Επίσης λειτουργεί ως ενορχηστρωτής των πιθανών διαφορετικών implementations. Έχει ως dependencies τουλάχιστον ένα Interface και τουλάχιστον ένα Bean
2. **Interfaces**, το κομμάτι αυτό αποτελεί ένα abstraction layer ώστε να κρύβεται το κάθε implementation από τους controllers ή άλλα implementations

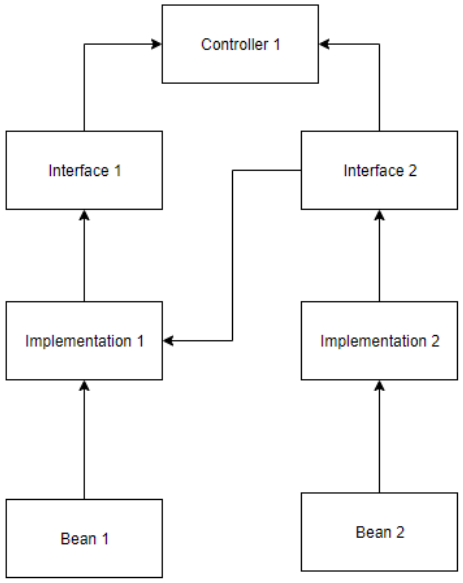
3. **Implementations**, το κομμάτι αυτό περιέχει την λογική της λειτουργικότητας κάθε service. Είναι πιθανό να υπάρχουν περισσότερα από ένα implementations. Κάθε implementation πρέπει να υλοποιεί ένα interface από το κομμάτι Interfaces. Τα dependencies του κομματιού αυτού είναι τα κομμάτια Interfaces, Daos και Beans
4. **Daos**, το κομμάτι αυτό είναι υπεύθυνο για την εκτέλεση διάφορων queries στην βάση της οντολογίας και της αντιστοίχισης των αποτελεσμάτων σε beans τα οποία έχουν κάποια αξία για το service. Τα dependencies του κομματιού αυτού είναι με κάποιο άλλο Dao ή με κάποια Beans. Επίσης το κομμάτι αυτό συνήθως χειρίζεται κάποιο db connection.
5. **Beans**, τέλος το κομμάτι αυτό περιέχει όλες τις κλάσεις τα οποία περιγράφουν κάποια αντικείμενα ώστε η επικοινωνία μεταξύ των Implementations ή μεταξύ των services να είναι strongly typed. Το μοναδικό dependency που μπορεί να έχει κάποιο bean είναι ένα άλλο bean του ίδιου service.

Τα παραπάνω μέρη αναπαρίστανται στο επόμενο σχήμα. Οι ακμές που ενώνουν δύο διαφορετικά κομμάτια υποδηλώνουν τα dependencies μεταξύ των κομματιών:



Εικόνα 54 Reference strategy μέσα στο service

Όταν έχουμε πολλαπλά μέρη τότε το παραπάνω σχήμα μπορεί να μετατραπεί ως εξής:



Εικόνα 55 Reference strategy μεταξύ services

5.3 Use – Case: Λύση παιγνίου Antonyms

Στη συνέχεια θα δούμε ένα αληθινό use-case για την λύση ενός παιγνίου Antonyms, στο οποίο ζητείται από τον χρήστη να βρεί το αντώνυμο μία δοθείσας λέξης.

5.3.1 Δημιουργία στιγμιότυπου

Για αρχή ο client πρέπει να ζητήσει την δημιουργία ενός στιγμιότυπου του παίγνιου δυσκολίας medium. Αυτό επιτυγχάνεται αν εκτελέσει το HTTP request του Πίνακα 3:

```
POST /mci/antonyms HTTP/1.1
Host: localhost:8443
Content-Type: application/json
X-INFO-PLAYER: Postman
cache-control: no-cache
{
      "difficulty": "medium"
}
```

Πίνακας 3 Παράδειγμα POST Antonyms

Το request αυτό θα δημιουργήσει ένα νέο στιγμιότυπο του παίγνιου *Antonyms* για τον χρήστη Postman. Χρησιμοποιώντας το γραφικό περιβάλλον του Apache Fuseki μπορούμε να επαληθεύσουμε την δημιουργία του παιγνίου καθώς και όλων των αντικειμένων που χρειάζονται.

5.3.2 Επαλήθευση στιγμιότυπου

Εκτελώντας το **SPARQL** ερώτημα του Πίνακα 4, επιστρέφει τις τριπλέτες που δημιουργήθηκαν για το παίγνιο με id Antonyms_MEDIUM_Postman_1, όπως περιγράφονται στον Πίνακα 5

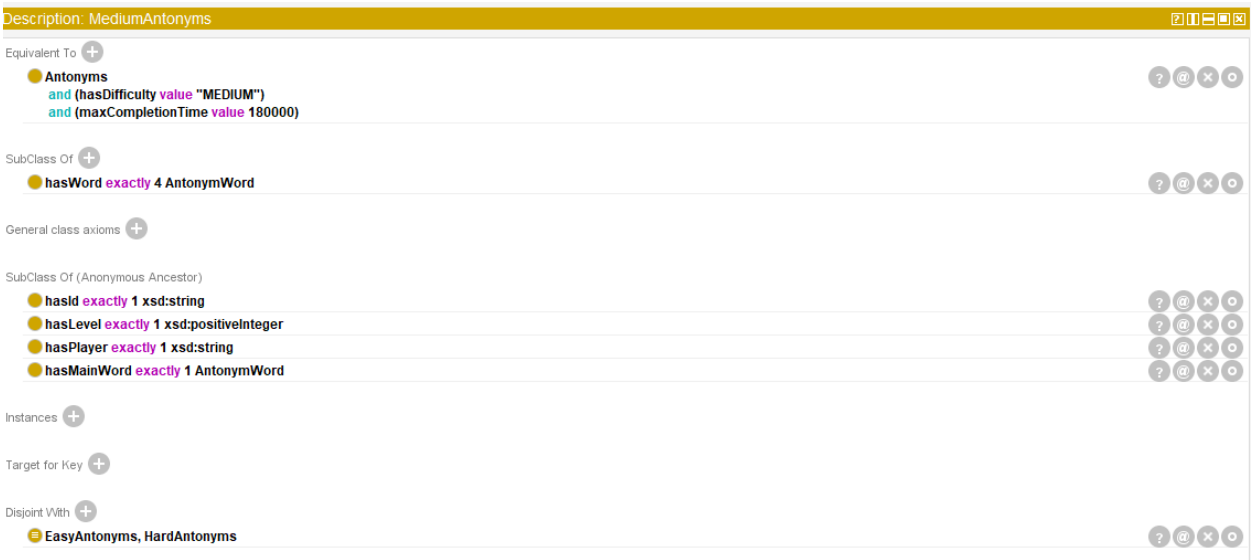
```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "Antonyms_MEDIUM_Postman_1";
           ?predicate ?object
}
```

Πίνακας 4 SPARQL ερώτημα για την επιστροφή παιγνίου με id Antonyms_MEDIUM_Postman_1

subject	predicate	object
<mci:Antonyms_MEDIUM_Postman_1>	<rdf:type>	<mci:MediumAntonyms>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasId>	"Antonyms_MEDIUM_Postman_1"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasLevel>	"1"^^xsd:integer
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasDifficulty>	"MEDIUM"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:maxCompletionTime>	"180000"^^xsd:integer
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasPlayer>	"Postman"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasMainWord>	<mci:Word_small>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_bad>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_unfortunate>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_tall>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_short>

Πίνακας 5 Απάντηση στο ερώτημα του Πίνακα 4

Από το μοντέλο της οντότητας *MediumAntonyms*, που εμφανίζεται στην Εικόνα 56, επαληθεύεται ότι όλοι οι απαραίτητοι περιορισμού για την δημιουργία ενός στιγμιότυπου έχουν ακολουθηθεί.



Εικόνα 56 Περιορισμοί MediumAntonyms

Στη συνέχεια πρέπει να επαληθευθούν ότι και τα απαραίτητα αντικείμενα έχουν δημιουργηθεί και συσχετιστεί σωστά σύμφωνα με τους περιορισμούς του μοντέλου.

Ο Πίνακας 6 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Word_small*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακα 7 και απαρτίζουν όλες τις συσχετίσεις της λέξης *small.*

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "Word_small";
           ?predicate ?object
}
```

Πίνακας 6 SPARQL ερώτημα για την λέξη με id Word_small

subject	predicate	object
<mci:Word_small>	<rdf:type>	<mci:Word>
<mci:Word_small>	<mci:hasStringValue>	"small"
<mci:Word_small>	<mci:hasWordLength>	"5"^^xsd:integer
<mci:Word_small>	<mci:hasId>	"Word_small"
<mci:Word_small>	<mci:isAntonym>	"true"^^xsd:boolean
<mci:Word_small>	<mci:hasAntonym>	<mci:Word_tall>

Πίνακας 7 Απάντηση στο ερώτημα του Πίνακα 6

Ο Πίνακας 8 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Word_bad*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακα 9 και απαρτίζουν όλες τις συσχετίσεις της λέξης *bad*

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "Word_bad";
           ?predicate ?object
}
```

Πίνακας 8 SPARQL ερώτημα για την λέξη με id Word_bad

subject	predicate	object
<mci:Word_bad>	<rdf:type>	<mci:Word>
<mci:Word_bad>	<mci:hasStringValue>	"bad"
<mci:Word_bad>	<mci:hasWordLength>	"3"^^xsd:integer
<mci:Word_bad>	<mci:hasId>	"Word_bad"
<mci:Word_bad>	<mci:isAntonym>	"true"^^xsd:boolean
<mci:Word_bad>	<mci:hasAntonym>	<mci:Word_good>

Πίνακας 9 Απάντηση στο ερώτημα του Πίνακα 7

Ο Πίνακας 10 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Word_unfortunate*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακα 11 και απαρτίζουν όλες τις συσχετίσεις της λέξης *unfortunate*

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "Word_unfortunate";
           ?predicate ?object
}
```

Πίνακας 10 SPARQL ερώτημα για την λέξη με id Word_unfortunate

subject	predicate	object
<mci:Word_unfortunate>	<rdf:type>	<mci:Word>
<mci:Word_unfortunate>	<mci:hasStringValue>	"unfortunate"
<mci:Word_unfortunate>	<mci:hasWordLength>	"11"^^xsd:integer
<mci:Word_unfortunate>	<mci:hasId>	"Word_unfortunate"
<mci:Word_unfortunate>	<mci:isAntonym>	"true"^^xsd:boolean
<mci:Word_unfortunate>	<mci:hasAntonym>	<mci:Word_happy>

Πίνακας 11 Απάντηση στο ερώτημα του Πίνακα 10

Ο Πίνακας 12 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Word_tall*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακα 13 και απαρτίζουν όλες τις συσχετίσεις της λέξης *tall*

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> " Word_tall ";
           ?predicate ?object
}
```

Πίνακας 12 SPARQL ερώτημα για την λέξη με id Word_tall

subject	predicate	object
<mci:Word_tall>	<rdf:type>	<mci:Word>
<mci:Word_tall>	<mci:hasStringValue>	"tall"
<mci:Word_tall>	<mci:hasWordLength>	"4"^^xsd:integer
<mci:Word_tall>	<mci:hasId>	"Word_tall"
<mci:Word_tall>	<mci:isAntonym>	"true"^^xsd:boolean
<mci:Word_tall>	<mci:hasAntonym>	<mci:Word_short>
<mci:Word_tall>	<mci:hasAntonym>	<mci:Word_small>
<mci:Word_tall>	<mci:isSynonym>	"true"^^xsd:boolean
<mci:Word_tall>	<mci:hasSynonym>	<mci:Word_big>
<mci:Word_tall>	<mci:hasSynonym>	<mci:Word_high>

Πίνακας 13 Απάντηση στο ερώτημα του Πίνακα 12

Ο Πίνακας 14 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Word_short*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακα 15 και απαρτίζουν όλες τις συσχετίσεις της λέξης *short*

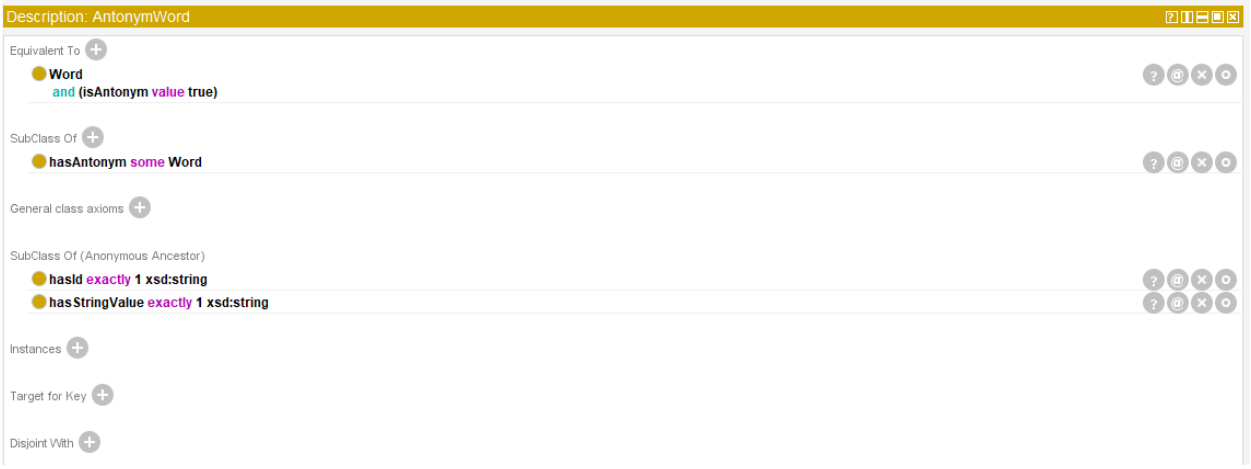
```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> " Word_short ";
            ?predicate ?object
}
```

Πίνακας 14 SPARQL ερώτημα για την λέξη με id Word_short

subject	predicate	object
<mci:Word_short>	<rdf:type>	<mci:Word>
<mci:Word_short>	<mci:hasStringValue>	"short"
<mci:Word_short>	<mci:hasWordLength>	"5"^^xsd:integer
<mci:Word_short>	<mci:hasId>	"Word_short"
<mci:Word_short>	<mci:isAntonym>	"true"^^xsd:boolean
<mci:Word_short>	<mci:hasAntonym>	<mci:Word_tall>

Πίνακας 15 Απάντηση στο ερώτημα του Πίνακα 14

Κάποιος μπορεί να παρατηρήσει ότι οι τριπλέτες του Πίνακας 13είναι περισσότερες συγκριτικά με τις τριπλέτες των υπολοίπων λέξεων. Αυτό συμβαίνει γιατί τα τρέχοντα δεδομένα που χρησιμοποιούνται για λέξεις επιτρέπουν κάτι τέτοιο. Παρόλα αυτά βλέπουμε ότι όλες οι τριπλέτες έχουν δημιουργηθεί με βάση τους κανόνες του μοντέλου, όπως αυτοί εμφανίζονται στην Εικόνα 57



Εικόνα 57 Περιορισμοί αντικειμένου AntonymWord

Επίσης βλέπουμε στον Πίνακα 5, ότι η λέξη της οποίας ζητείται το αντώνυμο είναι η λέξη *small*. Ο Πίνακας 13 περιγράφει την λέξη με id *Word_tall*. Χρησιμοποιώντας την ανθρώπινη λογική

μπορούμε να συμπεράνουμε ότι όντως η λέξη *tall* είναι αντώνυμο της λέξης *small*. Το ίδιο συμβαίνει και στην οντολογία καθώς η συσχέτιση *hasAntonym* υποδηλώνει ότι μία λέξη έχει ως αντώνυμο μία άλλη. Στον ίδιο πίνακα έχουμε την εξής τριπλέτα του Πίνακα 16, που υποδηλώνει ότι η λέξη *tall* έχει ως αντώνυμο την λέξη *small*.

<mci:Word_tall>	<mci:hasAntonym>	<mci:Word_small>

Πίνακας 16 Συσχέτιση αντώνυμων λέξεων

Παρατηρώντας τις ιδιότητες της συσχέτισης *hasAntonym* βλέπουμε ότι έχει υποσημειωθεί ως **Symmetric** και **Irreflexive**. Το αποτέλεσμα που περιμένουμε είναι ότι καμία λέξη δεν συσχετίζεται με τον εαυτό της μέσω αυτής της συσχέτισης (**Irreflexive**) και ότι η οντότητα στο **range** της συσχέτισης, συσχετίζεται με την ίδια συσχέτιση με την οντότητα του **domain**. Από τον Πίνακα 7 βλέπουμε ότι έχει δημιουργηθεί και εκεί η ίδια τριπλέτα.

<mci:Word_small>	<mci:hasAntonym>	<mci:Word_tall>

Εφόσον έχουν δημιουργηθεί σωστά οι τριπλέτες για το παίγνιο αυτό, το service θα στείλει στον client την απάντηση του Πίνακα 17, η οποία μπορεί να χρησιμοποιηθεί από κάποιο γραφικό περιβάλλον.

```
{
    "payload": {
        "word": "small",
        "choices": [
            "bad",
            "unfortunate",
            "tall",
            "short"
        ],
        "game": {
            "id": "Antonyms_MEDIUM_Postman_1",
            "difficulty": "MEDIUM",
            "playerName": "Postman",
            "level": 1,
            "maxCompletionTime": 180000
        },
        "solved": false
    }
}
```

Πίνακας 17 HTTP response του service Antonyms

Η απάντηση αυτή περιέχει όλες τις ζώσες πληροφορίες ώστε να χρησιμοποιηθούν από κάποιον client. Το πεδίο "word" περιέχει την λέξη – δεδομένο, το πεδίο "choices" περιέχει όλες τις πιθανές απαντήσεις, το πεδίο "game" περιέχει πληροφορίες σχετικά με το παίγνιο, όπως ο μέγιστος επιτρεπόμενος χρόνος λύσης και τέλος το πεδίο "solved" ενημερώνει αν το παίγνιο έχει επιλυθεί.

5.3.3 Αναζήτηση πληροφοριών παιγνίου

Καθώς είναι πιθανό το σενάριο ο τελικός χρήστης να σταματήσει προσωρινά την λειτουργία της τελικής εφαρμογής, έχουν αναπτυχθεί δύο διαφορετικά endpoints τα οποία μπορούν να ενημερώσουν τον client ποια είναι η κατάσταση των παιγνίων του χρήστη.

Το πρώτο endpoint επιστρέφει όλα τα παίγνια του χρήστη. Για την χρήση του, ο client αρκεί να κάνει το request του Πίνακας 18και θα λάβει ως απάντηση το response του Πίνακας 19.

```
GET /mci/antonyms? HTTP/1.1
Host: localhost:8443
X-INFO-PLAYER: Postman
cache-control: no-cache
```

Πίνακας 18 GET request για όλα τα παίγνια Antonyms για τον χρήστη Postman

```
{
    "payload": [
        {
            "game": {
                "id": "Antonyms_MEDIUM_Postman_1",
                "difficulty": "MEDIUM",
                "playerName": "Postman",
                "level": 1,
                "maxCompletionTime": 180000
            },
            "solved": false
        }
    ]
}
```

Πίνακας 19 Απάντηση στο request του Πίνακα 18

Το παραπάνω endpoint έχει στόχο μόνο την αναζήτηση όλων των παιγνίων τύπου Antonyms. Εάν ο client επιθυμεί περισσότερες πληροφορίες σχετικά με ένα συγκεκριμένο στιγμιότυπο τότε αρκεί να κάνει το request του Πίνακας 20. Η απάντηση που θα λάβει έχει ήδη αναλυθεί στον Πίνακα 17.

```
GET /mci/antonyms/Antonyms_MEDIUM_Postman_1 HTTP/1.1
Host: localhost:8443
X-INFO-PLAYER: Postman
cache-control: no-cache
```

Πίνακας 20 GET request για το παίγνιο με id Antonyms_MEDIUM_Postman_1

5.3.4 Λύση παιγνίου

Το τελικό βήμα είναι η λύση του παιγνίου. Στο βήμα αυτό, ο client πρέπει να κάνει ένα request σύμφωνα με αυτό του Πίνακα 21. Για όλα τα requests λύσης της εφαρμογής η δομή είναι η ίδια όπως περιγράφεται στον Πίνακα 17.

```
PUT /mci/antonyms/Antonyms_MEDIUM_Postman_1 HTTP/1.1
```

Host: localhost:8443
X-INFO-PLAYER: Postman
Content-Type: application/json
cache-control: no-cache
{
"completionTime": 18000,
"solution" :"bad"
}

Πίνακας 21 Request για την λύση του παιγνίου με id Antonyms_MEDIUM_Postman_1

Στην περίπτωση που κάποιος ζητήσει την λύση ενός παιγνίου το οποίο δεν υπάρχει για τον χρήστη, το service θα επιστρέψει μία απάντηση παρόμοια του Πίνακα 22.

```
{
    "error": {
        "code": "Antonyms.4",
        "message": "Antonyms.4: Game with id Antonyms_MEDIUM_Postman_2 and player Postman doesn't exist"
    }
}
```

Πίνακας 22 Απάντηση σε εσφαλμένο request για την λύση ενός παιγνίου

Στην περίπτωση που κάποιος δώσει λάθος απάντηση, τότε το service θα επιστρέψει μία απάντηση όπως έχει περιγραφεί στον Πίνακα 17.

Τέλος όταν κάποιος δώσει τη σωστή απάντηση, τότε το service θα επιστρέψει μία απάντηση όπως έχει περιγραφεί στον Πίνακα 23, με τις διαφορές ότι στο πεδίο "*game*" θα υπάρχει η πληροφορία του *completedDate* και *completionTime*, ενώ το πεδίο "*solved*" θα έχει την τιμή true.

```
{
    "payload": {
        "word": "small",
        "choices": [
            "bad",
            "unfortunate",
            "tall",
            "short"
        ],
        "game": {
            "id": "Antonyms_MEDIUM_Postman_1",
            "difficulty": "MEDIUM",
            "playerName": "Postman",
            "level": 1,
            "maxCompletionTime": 180000,
            "completionTime": 18000,
            "completedDate": "1569191585462"
        },
```

```
        "solved": true
    }
}
```

Πίνακας 23 Απάντηση για την σωστή λύση του παιγνίου με id Antonyms_MEDIUM_Postman_1

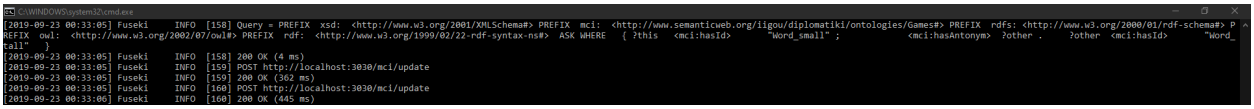
5.3.5 Επαλήθευση λύσης

Επιστρέφοντας στο γραφικό περιβάλλον του Apache Fuseki και αν εκτελέσουμε το **SPARQL** ερώτημα του Πίνακα 4, θα δούμε ότι έχουν δημιουργηθεί νέες τριπλέτες που υποδηλώνουν τη σωστή λύση. Οι τριπλέτες αυτές αναγράφονται στον Πίνακα 24

subject	predicate	object
<mci:Antonyms_MEDIUM_Postman_1>	<rdf:type>	<mci:MediumAntonyms>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasId>	"Antonyms_MEDIUM_Postman_1"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasLevel>	"1"^^xsd:integer
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasDifficulty>	"MEDIUM"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:maxCompletionTime>	"180000"^^xsd:integer
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasPlayer>	"Postman"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:completedDate>	"1569191585462"
<mci:Antonyms_MEDIUM_Postman_1>	<mci:isCompletedIn>	"18000"^^xsd:integer
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasMainWord>	<mci:Word_small>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_bad>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_unfortunate>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_tall>
<mci:Antonyms_MEDIUM_Postman_1>	<mci:hasWord>	<mci:Word_short>

Πίνακας 24 Απάντηση στο ερώτημα του Πίνακα 4 με την απάντηση

Για το παίγνιο αυτό η λογική με την οποία συμπεραίνεται ότι η δοθείσα λύση είναι η σωστή, γίνεται με την ερώτηση ενός ASK **SPARQL** ερωτήματος στην βάση. Από τα logs του Apache Fuseki, όπως απεικονίζονται στην Εικόνα 58, μπορούμε να δούμε την εκτέλεση του ερωτήματος.



Εικόνα 58 Εκτέλεση ASK SPARQL ερωτήματος

5.4 Use – Case: Λύση παιγνίου Find the Sound

Στη συνέχεια αναπτύσεται η περίπτωση για το παίγνιο Find the Sound. Στο παίγνιο αυτό ο χρήστης έχει στην διάθεση του ένα πλήθος εικόνων και έναν ήχο. Ο παίκτης καλείται να βρεί την σωστή εικόνα η οποία αντιστοιχεί στον ήχο αυτό.

5.4.1 Δημιουργία στιγμιότυπου

Για αρχή ο client πρέπει να ζητήσει την δημιουργία ενός στιγμιότυπου του παίγνιου δυσκολίας easy. Αυτό επιτυγχάνεται αν εκτελέσει το HTTP request του Πίνακα 25:

```
POST /mci/findTheSounds HTTP/1.1
```

```
Host: localhost:8443
Content-Type: application/json
X-INFO-PLAYER: Postman
cache-control: no-cache
{
"difficulty": "easy"
}
```

Πίνακας 25 Παράδειγμα POST FindTheSound

Το request αυτό θα δημιουργήσει ένα νέο στιγμιότυπο του παίγνιου *FindTheSound* για τον χρήστη Postman. Η απάντηση στο request αυτό περιγράφεται στον


```
{
    "payload": {
        "images": [
            {
                "id": "Image_9c90b4a254c142edb29c9bfca7287a1c",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\cat.jpg"
            },
            {
                "id": "Image_16ba6b21974f411f9b3931b0cafa360c",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\dog.jpg"
            },
            {
                "id": "Image_6442d418cd584e969a644c87671fdea2",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\donkey.jpg"
            }
        ],
        "soundId": "Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83",
        "soundPath":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\sounds\\donkey.mp3",
        "game": {
            "id": "FindTheSound_EASY_Postman_1",
            "difficulty": "EASY",
            "playerName": "Postman",
            "level": 1,
            "maxCompletionTime": 180000
        },
        "solved": false
    }
```

```
}
```

Πίνακας 26 Απάντηση στο request του Πίνακας 25

Χρησιμοποιώντας το γραφικό περιβάλλον του Apache Fuseki μπορούμε να επαληθεύσουμε την δημιουργία του παιγνίου καθώς και όλων των αντικειμένων που χρειάζονται.

5.4.2 Επαλήθευση στιγμιότυπου

Εκτελώντας το **SPARQL** ερώτημα του Πίνακας 27, επιστρέφει τις τριπλέτες που δημιουργήθηκαν για το παίγνιο με id Antonyms_MEDIUM_Postman_1, όπως περιγράφονται στον Πίνακας 28

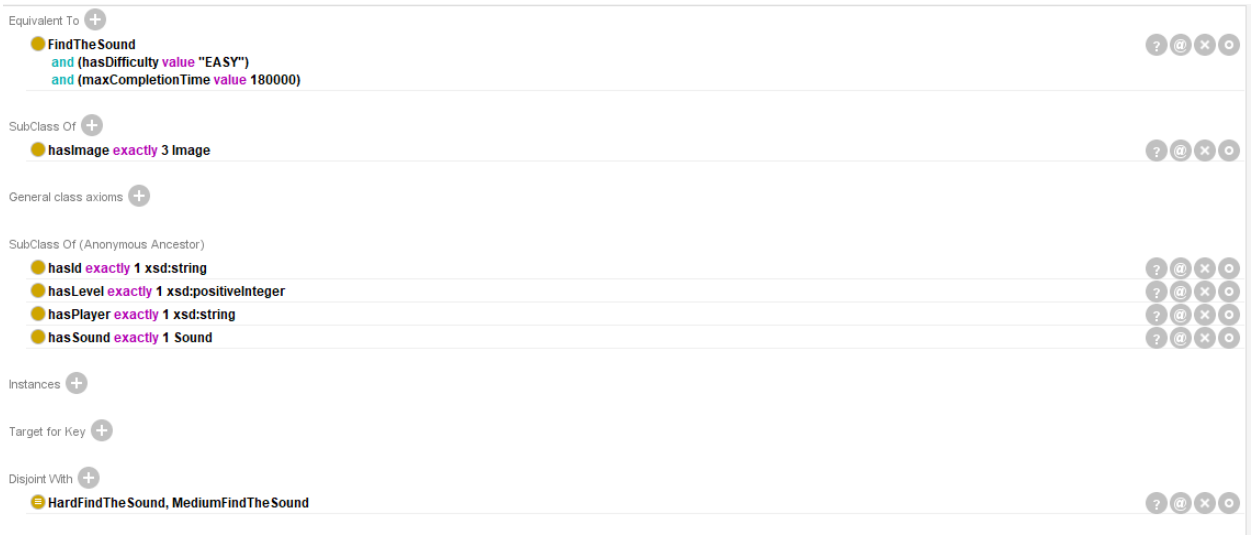
```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "FindTheSound_EASY_Postman_1";
           ?predicate ?object
}
```

Πίνακας 27 SPARQL ερώτημα για την επιστροφή παιγνίου με id FindTheSound_EASY_Postman_1

subject	predicate	object
<mci:FindTheSound_EASY_Postman_1>	<rdf:type>	<mci:EasyFindTheSound>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasId>	"FindTheSound_EASY_Postman_1"
<mci:FindTheSound_EASY_Postman_1>	<mci:hasLevel>	"1"^^xsd:integer
<mci:FindTheSound_EASY_Postman_1>	<mci:hasDifficulty>	"EASY"
<mci:FindTheSound_EASY_Postman_1>	<mci:maxCompletionTime>	"180000"^^xsd:integer
<mci:FindTheSound_EASY_Postman_1>	<mci:hasPlayer>	"Postman"
<mci:FindTheSound_EASY_Postman_1>	<mci:hasSound>	<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_9c90b4a254c142edb29c9bfca7287a1c>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_16ba6b21974f411f9b3931b0cafa360c>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_6442d418cd584e969a644c87671fdea2>

Πίνακας 28 Απάντηση στο ερώτημα του Πίνακας 27

Από το μοντέλο της οντότητας *EasyFindTheSound*, που εμφανίζεται στην Εικόνα 56, επαληθεύεται ότι όλοι οι απαραίτητοι περιορισμού για την δημιουργία ενός στιγμιότυπου έχουν ακολουθηθεί.



Εικόνα 59 Περιορισμοί EasyFindTheSound

Στη συνέχεια πρέπει να επαληθευθούν ότι και τα απαραίτητα αντικείμενα έχουν δημιουργηθεί και συσχετιστεί σωστά σύμφωνα με τους περιορισμούς του μοντέλου.

Ο Πίνακας 29 περιγράφει το **SPARQL** ερώτημα για την εύρεση των τριπλετών που συσχετίζονται με την συσχέτιση *hasId* με τιμή *Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83*. Οι τριπλέτες αυτές περιγράφονται στον Πίνακας 30 και απαρτίζουν όλες τις συσχετίσεις της λέξης του ήχου που επιλέχθηκε για το στιγμιότυπο αυτό.

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?subject ?predicate ?object
WHERE {
?subject <mci:hasId> "Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83";
            ?predicate ?object
}
```

Πίνακας 29 SPARQL ερώτημα για την ήχο με id Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83

subject	predicate	object
<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<rdf:type>	<mci:Sound>
<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<mci:hasId>	"Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83"
<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<mci:hasAssetPath>	"D:\WorkBench\Diplomatiki\mci\objects\sounds\donkey.mp3"
<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<mci:hasSubject>	<mci:Word_donkey>
<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<mci:isImageAssociated>	"true"^^xsd:boolean

<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>	<mci:hasAssociatedImage>	<mci:Image_6442d418cd584e969a644c87671fdea2>

Πίνακας 30 Απάντηση στο ερώτημα του Πίνακας 29

Από τον Πίνακας 30 παρατηρούμε ότι ο ήχος που συσχετίστηκε με το στιγμιότυπο είναι ο ήχος ενός γαϊδουριού. Στη συνέχεια πρέπει να επαληθευτεί ότι μία από τις τρείς συσχετισμένες εικόνες του Πίνακας 28 παρουσιάζει ένα γαϊδούρι και συσχετίζεται με την ίδια λέξη-αντικείμενο *<mci:Word_donkey>* του ήχου. Εκτελώντας το **SPARQL** ερώτημα του Πίνακας 31 επιστρέφονται τα στιγμιότυπα των εικόνων, που έχουν συσχετισθεί με το στιγμιότυπο παιγνίου με id *" FindTheSound_EASY_Postman_1",* καθώς και το στιγμιότυπο λέξης-αντικείμενου που εμφανίζεται στις εικόνες αυτές.

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>

SELECT ?image ?imageSubject
WHERE {
  ?subject <mci:hasId> "FindTheSound_EASY_Postman_1";
                   <mci:hasImage> ?image.
  ?image <mci:hasSubject> ?imageSubject
}
```

Πίνακας 31 SPARQL ερώτημα για την εύρεση των εικόνων του στιγμιότυπου "FindTheSound_EASY_Postman_1"

image	imageSubject
<mci:Image_9c90b4a254c142edb29c9bfca7287a1c>	<mci:Word_bird>
<mci:Image_16ba6b21974f411f9b3931b0cafa360c>	<mci:Word_car>
<mci:Image_6442d418cd584e969a644c87671fdea2>	<mci:Word_donkey>

Πίνακας 32 Απάντηση στο ερώτημα του Πίνακας 31

Από τον Πίνακας 32 διαπιστώνεται ότι μία από τις τρείς εικόνες εμφανίζει ένα γαϊδούρι. Στη συνέχεια πρέπει να διαπιστωθεί ότι η εικόνα με id *" Image_6442d418cd584e969a644c87671fdea2"* έχει συσχετισθεί σωστά με τον ήχο με id *"Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83"*. Ο Πίνακας 33περιγράφει το **SPARQL** ερώτημα για την εύρεση της τριπλέτας συσχετίζει την εικόνα με τον ήχο μέσω της συσχέτισης *hasAssociatedSound* και το αποτέλεσμα της σωστής αυτής συσχέτισης φαίνεται στον Πίνακας 34

```
PREFIX mci: <http://localhost:3030/mci/ontology/mci#>
SELECT ?subject ?predicate ?object
WHERE {
  ?subject <mci:hasId> "Image_6442d418cd584e969a644c87671fdea2";
         <mci:hasAssociatedSound> ?object.
}

```

Πίνακας 33 SPARQL ερώτημα για την εύρεση της συσχέτισης μεταξύ εικόνας και ήχου

subject	object
<mci:Image_6442d418cd584e969a644c87671fdea2>	<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>

Πίνακας 34 Απάντηση στο ερώτημα του Πίνακας 33

5.4.3 Αναζήτηση πληροφοριών παιγνίου

Καθώς είναι πιθανό το σενάριο ο τελικός χρήστης να σταματήσει προσωρινά την λειτουργία της τελικής εφαρμογής, έχουν αναπτυχθεί δύο διαφορετικά endpoints τα οποία μπορούν να ενημερώσουν τον client ποια είναι η κατάσταση των παιγνίων του χρήστη.

Το πρώτο endpoint επιστρέφει όλα τα παίγνια του χρήστη. Για την χρήση του, ο client αρκεί να κάνει το request του Πίνακας 35 και θα λάβει ως απάντηση το response του Πίνακας 36

```
GET /mci/ findTheSounds? HTTP/1.1
Host: localhost:8443
X-INFO-PLAYER: Postman
cache-control: no-cache
```

Πίνακας 35 GET request για όλα τα παίγνια FindTheSounds για τον χρήστη Postman

```
{
    "payload": [
        {
            "game": {
                "id": " FindTheSound_EASY_Postman_1",
                "difficulty": " EASY",
                "playerName": "Postman",
                "level": 1,
                "maxCompletionTime": 180000
            },
            "solved": false
        }
    ]
}
```

Πίνακας 36 Απάντηση στο request του Πίνακας 35

Το παραπάνω endpoint έχει στόχο μόνο την αναζήτηση όλων των παιγνίων τύπου FindTheSounds. Εάν ο client επιθυμεί περισσότερες πληροφορίες σχετικά με ένα συγκεκριμένο στιγμιότυπο τότε αρκεί να κάνει το request του Πίνακας 37. Η απάντηση που θα λάβει έχει ήδη αναλυθεί στον Πίνακας 26.

```
GET /mci/findTheSounds/FindTheSound_EASY_Postman_1 HTTP/1.1
Host: localhost:8443
X-INFO-PLAYER: Postman
cache-control: no-cache
```

Πίνακας 37 GET request για το παίγνιο με id FindTheSound_EASY_Postman_1

5.4.4 Λύση παιγνίου

Το τελικό βήμα είναι η λύση του παιγνίου. Στο βήμα αυτό, ο client πρέπει να κάνει ένα request σύμφωνα με αυτό του . Για όλα τα requests λύσης της εφαρμογής η δομή είναι η ίδια όπως περιγράφεται στον Πίνακα 17.

```
PUT /mci/findTheSounds/FindTheSound_EASY_Postman_1 HTTP/1.1
Host: localhost:8443
X-INFO-PLAYER: Postman
Content-Type: application/json
cache-control: no-cache
{
      "completionTime": 18000,
      "solution" : {
            "soundId": "Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83",
            "imageId": "Image_6442d418cd584e969a644c87671fdea2"

      }
}
```

Πίνακας 38 Request για την λύση του παιγνίου με id FindTheSound_EASY_Postman_1

Τέλος όταν κάποιος δώσει τη σωστή απάντηση, τότε το service θα επιστρέψει μία απάντηση όπως περιγράφεται στον Πίνακα 39

```
{
    "payload": {
        "images": [
            {
                "id": "Image_9c90b4a254c142edb29c9bfca7287a1c",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\5.jpg"
            },
            {
                "id": "Image_16ba6b21974f411f9b3931b0cafa360c",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\6.jpg"
            },
            {
                "id": "Image_6442d418cd584e969a644c87671fdea2",
                "path":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\images\\11.jpg"
            }
        ],
        "soundId": "Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83",
        "soundPath":
"D:\\WorkBench\\Diplomatiki\\mci\\objects\\sounds\\donkey.mp3",
        "game": {
```

```
            "id": "FindTheSound_EASY_Postman_1",
            "difficulty": "EASY",
            "playerName": "Postman",
            "level": 1,
            "maxCompletionTime": 180000,
            "completionTime": 18000,
            "completedDate": "1577993435032"
        },
        "solved": true
    }
}
```


Πίνακας 39 Απάντηση για την σωστή λύση του παιγνίου με id FindTheSound_EASY_Postman_1

5.4.5 Επαλήθευση λύσης

Επιστρέφοντας στο γραφικό περιβάλλον του Apache Fuseki και αν εκτελέσουμε το **SPARQL** ερώτημα του Πίνακας 27, θα δούμε ότι έχουν δημιουργηθεί νέες τριπλέτες που υποδηλώνουν τη σωστή λύση. Από τον Πίνακα 40 παρατηρούνται ότι έχουν δημιουργηθεί οι εξής συσχετίσεις *<mci:hasCompletionTime>* και *<mci:completedDate>*

subject	predicate	object
<mci:FindTheSound_EASY_Postman_1>	<rdf:type>	<mci:EasyFindTheSound>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasId>	"FindTheSound_EASY_Postman_1"
<mci:FindTheSound_EASY_Postman_1>	<mci:hasLevel>	"1"^^xsd:integer
<mci:FindTheSound_EASY_Postman_1>	<mci:hasDifficulty>	"EASY"
<mci:FindTheSound_EASY_Postman_1>	<mci:maxCompletionTime>	"180000"^^xsd:integer
<mci:FindTheSound_EASY_Postman_1>	<mci:hasPlayer>	"Postman"
<mci:FindTheSound_EASY_Postman_1>	<mci:hasSound>	<mci:Sound_2ec5c2314c2b4748a8a7fb40c8e7fe83>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_9c90b4a254c142edb29c9bfca7287a1c>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_16ba6b21974f411f9b3931b0cafa360c>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasImage>	<mci:Image_6442d418cd584e969a644c87671fdea2>
<mci:FindTheSound_EASY_Postman_1>	<mci:hasCompletionTime>	"18000"^^xsd:integer
<mci:FindTheSound_EASY_Postman_1>	<mci:completedDate>	"1577993435032"

Πίνακας 40 Τριπλέτες για το στιγμιότυπο παιγνίου με id FindTheSound_EASY_Postman_1

5.5 Επεκτασιμότητα εφαρμογής

Κατά την ανάπτυξη της εφαρμογής αναγνωρίστηκαν κοινά μοτίβα κατά την δημιουργία, λύση και ανάγνωση των διάφορων παιγνίων. Για την δημιουργία ενός στιγμιότυπου θα πρέπει να δημιουργηθούν οι αντίστοιχες συσχετίσεις – περιορισμοί όπως αυτές έχουν προσδιοριστεί στην Οντολογία. Αν υπάρχουν αλλαγές στους περιορισμούς αυτούς, όπως αναφέρθηκε και στην Ενότητα 4.5 Επεκτασιμότητα Οντολογίας, οι αλλαγές αυτές θα πρέπει να γίνουν και σε επίπεδο εφαρμογής.

5.5.1 Δημιουργία δομής της υλοποίησης

Όπως αναφέρθηκε στην υπό-ενότητα 5.2 Αρχιτεκτονική Εφαρμογής, στο project *mciwebapp* θα πρέπει να δημιουργηθούν τα εξής νέα Java πακέτα:

- com.aegean.icsd.mciwebapp.<Όνομα παιγνίου>.controllers
- com.aegean.icsd.mciwebapp. <Όνομα παιγνίου>.interfaces
- com.aegean.icsd.mciwebapp. <Όνομα παιγνίου>.implementations
- com.aegean.icsd.mciwebapp. <Όνομα παιγνίου>.beans

Για την συνεκτικότητα της εφαρμογής οι Java κλάσεις που θα δημιουργηθούν θα πρέπει να τοποθετηθούν στα σωστά Java πακέτα. Με τον τρόπο αυτό διασφαλίζεται η συνεκτικότητα και η επεκτασιμότητα της εφαρμογής.

5.5.2 Δημιουργία Java Beans

Κατά την υλοποίηση ενός νέου παιγνίου πρέπει να δημιουργηθούν τουλάχιστον τέσσερα νέα Java Beans τα οποία περιγράφουν το παίγνιο, την λύση του καθώς και τα HTTP Request και Response του server.

Όπως αναφέρθηκε στην υπό-ενότητα 4.3.1 Game υπάρχουν κοινά **DataProperties** μεταξύ των οντοτήτων των παιγνίων. Για τον λόγο αυτό έχει δημιουργηθεί μία abstract κλάση *BaseGame,* η οποία υλοποιεί ένα βασικό παίγνιο όπως έχει περιγραφεί στην υπό-ενότητα 4.3.1 Game.

Το πρώτο Java Bean πρέπει να περιγράφει το παίγνιο με τέτοιο τρόπο ώστε να μπορεί να γίνει map με την οντότητα που υπάρχει στην οντολογία. Μία οντότητα ενός παιγνίου περιέχει περιορισμούς που συσχετίζονται με άλλες οντότητες. Προκειμένου να απλοποιηθούν οι συσχετίσεις στο επίπεδο της εφαρμογής και να αποφευχθεί το coupling μεταξύ των διαφορετικών Java beans, το Java Bean που περιγράφει ένα παίγνιο έχει ως fields μόνο τα **DataProperties** της οντότητας και πρέπει να κάνει extend την abstract κλάση *BaseGame*.

Το δεύτερο πρέπει να περιγράφει την λύση του παιγνίου. Για παράδειγμα, η λύση για το παίγνιο FindTheSound περιέχει το id του ήχου και το id της επιλεγμένης εικόνας. Το Java Bean της λύσης περιγράφεται ως εξής, στον Πίνακα 41

```
package com.aegean.icsd.mciwebapp.findthesounds.beans;
```

```
public class Solution {
  private String soundId;
  private String imageId;

  public String getSoundId() {
    return soundId;
  }

  public void setSoundId(String soundId) {
    this.soundId = soundId;
  }

  public String getImageId() {
    return imageId;
  }

  public void setImageId(String imageId) {
    this.imageId = imageId;
  }
}
```

Πίνακας 41 Solution Java Bean για το παίγνιο FindTheSound

Τέλος πρέπει να υλοποιηθούν το Request bean, που περιγράφει το request για την λύση του παιγνίου, και το Response bean, το οποίο περιγράφει ένα συγκεκριμένο στιγμιότυπο παιγνίου. Προκειμένου να αποφευχθεί το code duplication, το Java Bean που περιγράφει το request του παιγνίου πρέπει να κάνει extend την κλάση *Request,* η οποία παρέχει τα εξής κοινά fields,

- difficulty, περιγράφει το επίπεδο δυσκολίας
- completionTime, περιγράφει τον χρόνο λύσης.

Επίσης το Java Bean που περιγράφει το στιγμιότυπο του παιγνίου πρέπει να κάνει extend την generic κλάση ServiceResponse, με type το Java Bean που περιγράφει την οντότητα.

5.5.3 Δημιουργία Controller

Το νέο παίγνιο θα πρέπει να γίνει διαθέσιμο μέσω του HTTP πρωτοκόλλου. Αυτό μπορεί να γίνει με την δημιουργία ενός νέου RestController (Spring Framework, n.d.). Για να υποδηλώσουμε μία Java κλάση ως RestController αρκεί να χρησιμοποιήσουμε το Spring Java annotation @RestController. Η εφαρμογή κρατά απομονωμένα τα HTTP Verbs για κάθε παίγνιο μέσω διαφορετικού URL context path. Για τον λόγο αυτό είναι υποχρεωτική η δημιουργία ενός νέου context path με όνομα, κατά προτίμηση το όνομα του νέου παιγνίου. Η δημιουργία ενός νέου context path γίνεται χρησιμοποιώντας το Spring Java annotation @RequestMapping (Spring Framework, n.d.).

Τα HTTP verbs που πρέπει να υλοποιηθούν είναι συγκεκριμένα και παρουσιάζονται στον Πίνακα 1. Επίσης θα πρέπει να ληφθεί υπόψιν η υποχρεωτική ύπαρξη του HTTP Request Header *"*X-INFO-PLAYER*"*. Η κεφαλίδα αυτή μεταφέρει το όνομα του παίκτη και χρησιμοποιείται για την δημιουργία στιγμιότυπων παιγνίων, καθώς και για την υποστήριξη πολλαπλών χρηστών. Τέλος τα endpoints που δημιουργούνται μέσα στον *RestController* θα πρέπει να κάνουν παράγουν responses και να καταναλώνουν requests με την μορφή *JSON*.

Τα παραπάνω εμφανίζονται σαν παράδειγμα στον Πίνακα 42 , η οποία εμφανίζει την υλοποίηση του *RestController* για το παίγνιο FindTheSound

```
/**
 * https://localhost:8443/mci/findTheSounds
 */
@RestController
@RequestMapping("findTheSounds")
public class FindTheSoundController {

  private static final Logger LOGGER = LogManager.getLogger(FindTheSoundController.class);

  @Autowired
  private IFindTheSoundSvc findTheSoundSvc;

  @GetMapping(value = "",
    produces = MediaType.APPLICATION_JSON_VALUE)
  @ResponseStatus(HttpStatus.OK)
  public Response<List<ServiceResponse<FindTheSound>>> getGames(@RequestParam(name = "difficulty", required = false) String difficulty,
                                                                @RequestParam(name = "completed", required = false) Boolean completed,
                                                                @RequestHeader("X-INFO-PLAYER") String player) throws MciException {
    List<ServiceResponse<FindTheSound>> responses = findTheSoundSvc.getGames(player, FindTheSound.class);
    List<ServiceResponse<FindTheSound>> filtered = FilterResponse.by(responses, difficulty, completed);
    return new Response<>(filtered);
  }

  @PostMapping(consumes = MediaType.APPLICATION_JSON_VALUE,
          produces = MediaType.APPLICATION_JSON_VALUE)
  @ResponseStatus(HttpStatus.CREATED)
  public Response<FindTheSoundResponse> createGame(@RequestBody FindTheSoundRequest req,
                                                   @RequestHeader("X-INFO-PLAYER") String player) throws MciException {
    LOGGER.info("createRecall request received");
    Difficulty dif = Difficulty.valueOf(req.getDifficulty().toUpperCase());
    FindTheSoundResponse resp = findTheSoundSvc.createGame(player, dif, FindTheSound.class);
    return new Response<>(resp);
```

```
  }

  @GetMapping(value = "/{id}",
          produces = MediaType.APPLICATION_JSON_VALUE)
  @ResponseStatus(HttpStatus.OK)
  public Response<FindTheSoundResponse> getGame(@PathVariable("id") String id,
                                        @RequestHeader("X-INFO-PLAYER")    String
player) throws MciException {
    FindTheSoundResponse     resp     =     findTheSoundSvc.getGame(id,     player,
FindTheSound.class);
    return new Response<>(resp);
  }

  @PutMapping(value = "/{id}",
          consumes = MediaType.APPLICATION_JSON_VALUE,
          produces = MediaType.APPLICATION_JSON_VALUE)
  @ResponseStatus(HttpStatus.OK)
  public Response<FindTheSoundResponse> solveGame(@PathVariable("id") String id,
                                          @RequestBody FindTheSoundRequest req,
                                          @RequestHeader("X-INFO-PLAYER")  String
player) throws MciException {
    FindTheSoundResponse   response   =   findTheSoundSvc.solveGame(id,   player,
req.getCompletionTime(), req.getSolution(), FindTheSound.class);
    return new Response<>(response);
  }
}
```

Πίνακας 42 RestController για το παίγνιο FindTheSound

5.5.4 Δημιουργία Interface

Όπως αναφέρθηκε στην υπό-ενότητα 5.2.1 Δομή Service, για κάθε παίγνιο που υλοποιείται στην εφαρμογή, πρέπει να υπάρχει ένα Java Interface το οποίο μπορεί να χρησιμοποιηθεί ως ένα abstract layer για το implementation του. Οι μέθοδοι του Interface πρέπει να είναι παρόμοιες με τις λειτουργικότητες των HTTP Verbs. Καθώς όλα τα παίγνια υλοποιούν τις ίδιες λειτουργικότητες, έχει δημιουργηθεί το generic interface του Πίνακα 43. Τα types που δέχεται το interface έχουν αναπτυχθεί στην υπό-ενότητα 5.5.2 Δημιουργία Java Beans

```
public interface IGameService<T extends BaseGame, R extends ServiceResponse<T>> {

  List<ServiceResponse<T>> getGames(String playerName, Class<T> gameClass) throws
MciException;

  R createGame(String playerName, Difficulty difficulty, Class<T> gameClass) throws
MciException;

  R getGame(String id, String player, Class<T> gameClass) throws MciException;

  R solveGame(String id, String player, Long completionTime, Object solution,
Class<T> gameClass) throws MciException;
```

```
}
```

Πίνακας 43 Interface IGameService

5.5.5 Δημιουργία Service

Στην υπό-ενότητα 5.5.4 Δημιουργία Interface αναφέρθηκε ότι πρέπει να δημιουργήσουμε ένα νέο Interface. Στη συνέχεια θα πρέπει να υλοποιηθεί το Interface αυτό υπό την μορφή ενός Service. Για να δηλωθεί μία Java κλάση ως Spring Service αρκεί να χρησιμοποιηθεί το Java annotation @Service (Spring Framework, n.d.). Στη συνέχεια η κλάση αυτή πρέπει να κάνει implement το Interface που δημιουργήθηκε στην υπό-ενότητα 5.5.4 Δημιουργία Interface.

Κατά την υλοποίηση των παιγνίων παρατηρήθηκαν κοινές λειτουργίες μεταξύ τους. Αυτό οδήγησε στην δημιουργία της abstract κλάσης AbstractGameSvc, η οποία υλοποιεί τις μεθόδους του Interface της υπό-ενότητας 5.5.4 Δημιουργία Interface. Ως abstract κλάση έχει τις παρακάτω abstract μεθόδους που πρέπει να υλοποιηθούν από τις κλάσεις-παιδία:

- void handleDataTypeRestrictions(String fullName, T toCreate): Η μέθοδος αυτή έχει ως παραμέτρους το πλήρες όνομα του παιγνίου, το οποίο αποτελείται από το επίπεδο δυσκολίας και το όνομα του παιγνίου όπως για παράδειγμα EasyFindTheSound, και την κλάση του παιγνίου που χειρίζεται το service. Η κλάση του παιγνίου όπως αναφέρθηκε στην υπό-ενότητα 5.5.2 Δημιουργία Java Beans πρέπει να επεκτείνει την κλάση *BaseGame*. Ο σκοπός της μεθόδου αυτής είναι να θέσει τιμές στα fields της κλάσης του παιγνίου. Στην υπό-ενότητα 5.6.2 Δημιουργία Java Beans είδαμε ότι τα fields αυτά αποτελούν τα **DataProperties** της οντολογίας του παιγνίου.
- void handleObjectRestrictions(String fullName, T toCreate): Η μέθοδος αυτή έχει ως παραμέτρους το πλήρες όνομα του παιγνίου, το οποίο αποτελείται από το επίπεδο δυσκολίας και το όνομα του παιγνίου όπως για παράδειγμα EasyFindTheSound, και την κλάση του παιγνίου που χειρίζεται το service. Ο σκοπός της μεθόδου αυτής είναι να δημιουργήσει όλες τις τριπλέτες στην βάση γνώσης οι οποίες συσχετίζουν δύο διαφορετικές οντότητες. Οι συσχετίσεις αυτές είναι τα **ObjectProperties** με **domain** την οντότητα του παιγνίου.
- boolean isValid(Object solution): Η μέθοδος αυτή ελέγχει αν το αντικείμενο solution είναι ένα ορθό αντικείμενο και ελέγχει αν τα fields του αντικειμένου έχουν τιμή.
- boolean checkSolution(T game, Object solution): Η μέθοδος αυτή δέχεται το Java Bean του παιγνίου και το αντικείμενο της λύσης. Ο σκοπός της μεθόδου αυτής είναι ο έλεγχος του αντικειμένου της λύσης ενάντια στην βάση γνώσης. Η υλοποίηση της μεθόδου αυτής θα πρέπει να περιλαμβάνει ερωτήσεις στην βάση γνώσης με βάση την λογική κάθε παιγνίου αλλά και των αντικειμένων που χρησιμοποιούνται.
- R toResponse(T game): Η μέθοδος αυτή παίρνει σαν παράμετρο το Java Bean του παιγνίου και το μετατρέπει στο Response Java Bean (5.5.2 Δημιουργία Java Beans). Το αποτέλεσμα της μεθόδου αυτής θα είναι η απάντηση που θα σταλεί στον client.

Για την δημιουργία ενός Service παιγνίου μπορεί να χρησιμοποιηθεί είτε η abstract κλάση AbstractGameSvc όπως φαίνεται στον Πίνακα 44, είτε με την υλοποίηση του Interface που δημιουργήθηκε στην υπό-ενότητα 5.5.4 Δημιουργία Interface

```
@Service
public class FindTheSoundSvc extends AbstractGameSvc<FindTheSound, FindTheSoundResponse>
implements IFindTheSoundSvc
```

Πίνακας 44 Παράδειγμα χρήσης της κλάσης AbstractGameSvc

6

Συμπεράσματα

Στην εργασία αυτή μελετήθηκαν οι οντολογίες και πως αυτές μπορούν να χρησιμοποιηθούν για την διευκόλυνση δημιουργίας cognitive παιγνίων. Επίσης, χρησιμοποιώντας το SPARQL Protocol καθώς και αρχιτεκτονική των microservices είναι εφικτή η δημιουργία ενός Web API το οποίο θα επιτρέπει την χρήση της οντολογίας απομακρυσμένα. Η εφαρμογή που έχει δημιουργηθεί για την επικοινωνία με τους clients χρησιμοποιεί απλό HTTP πρωτόκολλο και οι απαντήσεις είναι μορφοποιημένες σε JSON. Καθώς ακολουθεί την REST αρχιτεκτονική τα endpoints της εφαρμογής έχουν την μορφή ενός resource, πράγμα που καθιστά εύκολη την χρήση τους από πιθανούς clients.

Ως δυνατότητα αναφέρεται η χρήση της εφαρμογής από πραγματικούς clients, όπως κάποιες Android εφαρμογές, ή εφαρμογές βασισμένες στο διαδίκτυο. Οι εφαρμογές αυτές μπορούν εύκολα να καταναλώσουν τις υπηρεσίες που προσφέρει το Web API, καθώς και οι δύο γνωρίζουν πώς να καταναλώνουν JSON απαντήσεις.

Σχετικά με το Web API που αναπτύχθηκε στα πλαίσια της εργασίας αυτής, σαν μελλοντική εργασία θα μπορεί να είναι η ενσωμάτωση ενός login μηχανισμού. Ο μηχανισμός αυτός θα

επιτρέπει την δημιουργία προφίλ του τελικού χρήστη. Έχοντας ένα ενεργό προφίλ τότε μπορεί να επιτευχθεί personalization στις υπηρεσίες που προσφέρει το Web API. Όπως έχει αναφερθεί στην αρχή της εργασίας αυτής, τα άτομα που πάσχουν από κάποιο είδος άνοιας μπορούν να έχουν μία γνωστική λειτουργία σε πιο δυσχερή κατάσταση από κάποια άλλη. Η δημιουργία ενός προφίλ θα επιτρέψει την αναγνώριση μίας τέτοιας κατάστασης και θα προτείνει στον χρήστη συγκεκριμένα παίγνια που στοχεύουν στην λειτουργία αυτή. Εκτός της δημιουργίας προφίλ, νέες οντότητες πρέπει να δημιουργηθούν στην οντολογία οι οποίες να ομαδοποιούν τα υπάρχοντα παίγνια ανάλογα με την γνωστική λειτουργία που στοχεύουν, καθώς επίσης και οντότητες που αφορούν τις πληροφορίες του χρήστη.

Ο κλάδος που ασχολείται με τις οντολογίες είναι σχετικά νέος και πολλά υποσχόμενος. Μέχρι στιγμής βρίσκει μεγαλύτερη εφαρμογή σε ιατρικές εφαρμογές αλλά η ανάγκη για την επέκταση μίας βάσης γνώσης και για την δημιουργία συμπερασμάτων εμφανίζεται σε όλους τους επιχειρηματικούς κλάδους. Ας μην ξεχνάμε ότι μία οντολογία μπορεί να χρησιμοποιήσει οντότητες μίας άλλης, το οποίο σημαίνει διαμοιρασμό της γνώσης και αυτό με τη σειρά του σημαίνει εμπλουτισμό της ήδη υπάρχουσας πληροφορίας. Αυτό σε συνδυασμό με machine learning τεχνολογίες μπορούν να οδηγήσουν στην δημιουργία έξυπνων υπηρεσιών, οι οποίες είναι ικανές λογικού συμπερασμού.

Παράρτημα Ι - Repositories

Έχουν δημιουργηθεί 2 Git repositories στην πλατφόρμα Github τα οποία περιέχουν την οντολογία και τον κώδικα για τον server. Τα URL είναι τα παρακάτω:

1. Server : https://github.com/Binarios/Mci
2. Ontology: https://github.com/Binarios/MciOntology

Βιβλιογραφία

W3C SPARQL Working Group. (2013, March 21). *SPARQL 1.1 Overview*. Ανάκτηση από W3C: https://www.w3.org/TR/sparql11-overview/

Abburu, S. (2012). A Survey on Ontology Reasoners and Comparison. *International Journal of Computer Applications, 57*(17).

Alloni, A., Sinforiani, E., Zucchella, C., Sandrini, G., Bernini, S., Cattani, B., . . . Pistarini, C. (2017). Computer-based cognitive rehabilitation: the CoRe system. *Disability and Rehabilitation , 39*(4), 407-417. doi:https://doi.org/10.3109/09638288.2015.1096969

Antoniou, G., & van Harmelen, F. (n.d.). Web Ontology Language: OWL. Στο S. Staab, & R. Studer, *Handbook on Ontologies* (σσ. 91-110). Springer.

Apache. (2019). *https://jena.apache.org/documentation/ontology/*. Ανάκτηση από Apache Jena: https://jena.apache.org/documentation/ontology/

Apache. (2019). *TDB*. Ανάκτηση από Apache Jena: https://jena.apache.org/documentation/tdb/

Baader, F. (2002). Basic description logics. Στο *In[11* (σσ. 43-95).

Baader, F., & Nutt, W. (2003). Basic Description Logics. Στο F. Baader, D. Calvanese, D. McGuinness, P. P. Schneider, & D. Nardi, *The Description Logic Handbook: Theory, Implementation and Applications* (σσ. 47-50). Cambridge university press.

Baader, F., Horrocks, I., & Sattler, U. (2010). Description Logics. Στο S. Staab, & R. Studer, *Handbook On Ontologies* (σσ. 21-44). Springer.

Bakshi, K. (2017). Microservices-based software architecture and approaches. *In 2017 IEEE Aerospace Conference* (σσ. 1-8). IEEE.

Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., & Stein, L. A. (2004, February 10). *OWL Web Ontology Language*. Ανάκτηση από W3C: https://www.w3.org/TR/2004/REC-owl-ref-20040210/#Sublanguage-def

Beckett, D., & Broekstra, J. (2013, March 21). *SPARQL Query Results XML Format (Second Edition)*. Ανάκτηση από W3C: https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/

Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G., & Machina, L. (2014, February 25). *Turtle*. Ανάκτηση από W3: https://www.w3.org/TR/turtle/

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American, 284*(5), 28-37.

Bock, J., Ji, Q., Haase, P., & Volz, R. (2008, June 1). *Benchmarking OWL Reasoners.* Tenerife: In ARea2008-Workshop on Advancing Reasoning on the Web: Scalability and Commonsense.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., & Yergeau, F. (2008, November 26). *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Ανάκτηση από w3: https://www.w3.org/TR/2008/REC-xml-20081126/

Chartomatsidis, M., & Goumopoulos, C. (2019). A Balance Training Game Tool for Seniors Using Microsoft Kinect and 3D Worlds. *Proceedings of the 5th International Conference on Information and Communication Technologies for Ageing Well and e-Health* (pp. 135-145). Herakleio, Greece: SCITEPRESS.

Clark, K. G., Feigenbaum, L., & Torres, E. (2013, March 21). *SPARQL 1.1 Query Results JSON Format*. Ανάκτηση από W3C: https://www.w3.org/TR/2013/REC-sparql11-results-json-20130321/

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. Στο N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, & L. Safina, *Present and Ulterior Software Engineering* (σσ. 195-216).

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Phillipe, C., Krogdahl, P., . . . Newling, T. (2004). *Patterns: Service-Oriented Architecture and Web Services.* IBM.

E-Prime. (n.d.). Ανάκτηση από E-Prime: https://pstnet.com/products/e-prime/

Feigenbaum, L., Williams, G. T., Clark, K. G., & Torres, E. (2013, March 21). *SPARQL 1.1 Protocol*. Ανάκτηση από W3C: https://www.w3.org/TR/sparql11-protocol/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Irvine: UNIVERSITY OF CALIFORNIA.

Gandon, F., & Schreiber, G. (2014, February 25). *RDF 1.1 XML Syntax.* Ανάκτηση από W3: https://www.w3.org/TR/rdf-syntax-grammar/

Gao, S. S., Sperberg-McQueen, C., & Thompson, H. S. (2012, April 5). *W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures.* Ανάκτηση από W3: https://www.w3.org/TR/xmlschema11-1/

Garcia Marin, J. A., Navarro, K. F., & Lawrence, E. (2011). Serious Games to Improve the Physical Health of the Elderly: A Categorization Scheme. *International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services.* Barcelona.

Gavalas, D., Nicopolitidis, P., Kameas, A., Goumopoulos, C., Bellavista, P., Lambrinos, L., & and Guo, B. (2017). Smart Cities: Recent Trends, Methodologies, and Applications. *Wireless Communications and Mobile Computing*, 7090963.

Goumopoulos, C., & Kameas, A. (2009). Smart objects as components of UbiComp applications. *International Journal of Multimedia and Ubiquitous Engineering*, 1-20.

Goumopoulos, C., Papa, I., & Stavrianos, A. (2017). Development and Evaluation of a Mobile Application Suite for Enhancing the Social Inclusion and Well-Being of Seniors. *Informatics*, 15.

Green, S., & Bavelier, D. (2004). The Cognitive Neuroscience of Video Games. Στο *Digital Media:Transformations in Human Communication* (σσ. 211-223). Messaris & Humphreys, Eds.

Gruber, T. R. (1993, June). A translation approach to portable ontology specifications. *Knowledge Acquisition, 5*(2), 199-220. doi:https://doi.org/10.1006/knac.1993.1008

Gustafson, L. (1996, December). What is dementia? *Acta Neurologica Scandinavica, 94*(s168), 22-24. Ανάκτηση από Acta Neurologica ScandinavicaVolume 94, Issue s168: https://doi.org/10.1111/j.1600-0404.1996.tb00367.x

Hähnle, R. (2001). Tableaux and Related Methods. Στο A. Robinson, & A. Voronkov, *Handbook of Automated Reasoning* (Τόμ. 1, σσ. 101-178).

Horridge, M., & Patel-Schneider, P. (2008, November 28). *Manchester Syntax*. Ανάκτηση από w3.org: https://www.w3.org/2007/OWL/draft/ED-owl2-manchester-syntax-20081128/

Horrocks, I. (2002). DAML+OIL: a Description Logic. *Data Engineering*, 4-9.

Horrocks, I., Patel-Schneider, P. F., & van Harmelen, P. (2003). From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Journal of Web Semantics*, 7-26.

Horrocks, I., Sattler, U., & Tobies, S. (n.d.). Reasoning with Individuals for the Description Logic SHIQ. *International Conference on Automated Deduction* (σσ. 482-496). Springer.

Lacy, L. W. (2005). *Owl: Representing Information Using the Web Ontology Language.* Trafford.

Leonardi, G., Panzarasa, S., & Quaglini, S. (2011). Ontology-based automatic generation of computerized cognitive exercises. *Studies in Health Technology and Informatics, 169*, 779-783.

Luciano, G., Mariano, A., Giacinto, B., Malena, F., Francisco, I., & Lisa, P. (2006). Cognition, technology and games for the elderly: An introduction to ELDERGAMES Project . *PsychNology, 4*(3), 285-308.

Matentzoglu, N., Leo, J., Hudhra, V., Parsia, B., & Sattler, U. (2015). A Survey of Current, Stand-alone OWL Reasoners. Στο *ORE* (σσ. 68-79).

McCallum, S., & Boletsis, C. (2013). Dementia Games: A Literature Review. *International Conference on Serious Games Development and Applications* (σσ. 15-27). Springer, Berlin, Heidelberg. doi:https://doi.org/10.1007/978-3-642-40790-1_2

McGuiness, D. L., & van Harmelen, F. (2004, February 10). *OWL Web Ontology Language*. Ανάκτηση από W3C: https://www.w3.org/TR/owl-features/

Nardi, D., & Brachman, R. J. (2003). An Introduction to Description Logics. Στο F. Baader, D. Calvanese, D. McGuinness, & P. P. Schneider, *The Description Logic Handbook: Theory, Implementation and Applications* (σσ. 1-40). Cambridge University Press.

Newman, S. (2016). *Building Microservices: Designing Fine-Grained Systems.* O'Reilly.

Pan, J. Z. (2010). Resource Description Framework. Στο S. Staab, & R. Studer, *Handbook On Ontologies* (σσ. 71-90). Springer.

Papatheodoropoulos, C. (2015). *Νόσος του Alzheimer.*

Parsia, B., & Sirin, E. (2004). Pellet: An owl dl reasoner. *Third international semantic web conference*, *18.*

Pautasso, C. (2013). RESTful Web Services: Principles, Patterns, Emerging Technologies. Στο A. Bouguettaya, Q. Z. Sheng, & F. Daniel, *Web Services Foundations* (σσ. 31-51). Springer.

Pautasso, C., Wilde, E., & Alarcon, R. (2014). *REST: Advanced Research Topics and Practical Applications.* Springer.

Pérez, J., Arenas, M., & Gutierrez, C. (2006). Semantics and Complexity of SPARQL. *International Semantic Web Conference* (σσ. 30-43). Springer.

Poli, R., & Obrst, L. (2010). The Interplay Between Ontology as Categorial Analysis and Ontology as Technology. Στο *Theory and applications of Ontology: Computer applications* (σσ. 1-26). New York: Springer.

Prasanna, D. R. (2009). *Dependency Injection.* KU Digital Collections.

Prud'hommeaux, E., & Seaborne, A. (2008, January 15). *SPARQL Query Language for RDF*. Ανάκτηση από W3C: https://www.w3.org/TR/rdf-sparql-query/

Quaglini, S., Panzarasa, S., Giorgiani, T., Zucchella, C., Bartolo, M., Sinforiani, E., & Sandrini, G. (2009). Ontology-Based Personalization and Modulation of Computerized Cognitive Exercises. *Springer, 5651*, 240-244. doi:https://doi.org/10.1007/978-3-642-02976-9_34

Sattler, U. (1996). A concept language extended with different kinds of transitive roles. *Annual Conference on Artificial Intelligence* (σσ. 333-345). Springer.

Schmidt-Schauβ, M., & Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 1-26.

Shearer, R., Motik, B., & Horrocks, I. (n.d.). HermiT: A Highly-Efficient OWL Reasoner. *Owled*.

Simancík, F., Kazakov, Y., & Horrocks, I. (2011). Consequence-Based Reasoning beyond Horn Ontologies. *Twenty-Second International Joint Conference on Artificial Intelligence.*

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007, June). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics, 5*(2), 51-53. doi:https://doi.org/10.1016/j.websem.2007.03.004

Spring Framework. (n.d.). *Annotation Type RequestMapping*. Ανάκτηση από Spring Framework: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html

Spring Framework. (n.d.). *Annotation Type RestController*. Ανάκτηση από Spring Framework: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html

Spring Framework. (n.d.). *Annotation Type Service*. Ανάκτηση από Spring Framework: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Service.html

Tong, T., Chan, J. H., & Chignell, M. (2017). Serious Games for Dementia. *WWW '17 Companion Proceedings of the 26th International Conference on World Wide Web Companion* (σσ. 1111-1115). International World Wide Web Conferences Steering Committee. doi:https://doi.org/10.1145/3041021.3054930

Tsarkov, D., & Horrocks, I. (2006). FaCT++ Description Logic Reasoner: System Description. *International Joint Conference on Automated Reasoning*, (σσ. 292-297).

W3C OWL Working Group. (2012, December 11). *OWL 2 Web Ontology Language*. Ανάκτηση από W3C: https://www.w3.org/TR/owl2-overview/