



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Συστημάτων Μετάδοσης Πληροφορίας και Τεχνολογίας
Υλικών

**Δημιουργία Αλληλεπιδραστικού Γραφικού
Περιβάλλοντος για την οπτική αναπαράσταση των
συσχετίσεων μεταξύ οντολογιών**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

Αθανάσιου Χ. Αποστόλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβρης 2020



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Τομέας Συστημάτων Μετάδοσης Πληροφορίας και Τεχνολογίας
Υλικών

**Δημιουργία Αλληλεπιδραστικού Γραφικού
Περιβάλλοντος για την οπτική αναπαράσταση των
συσχετίσεων μεταξύ οντολογιών**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

Αθανάσιου Χ. Αποστόλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 17η Αυγούστου 2020.

.....
Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

.....
Εμμανουήλ Βαρβαρίγος
Καθηγητής Ε.Μ.Π.

.....
Συμεών Παπαβασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβρης 2020

.....

Αθανάσιος Χ. Αποστόλου

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

Copyright © Αθανάσιος Χ. Αποστόλου

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

δφασδ

γφδς

[illegible]

Λέξεις Κλειδιά

Abstract

δφσαδφ

φσδαφ

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε από τον Μάρτιο έως τον Οκτώβρη του 2020 για την ολοκλήρωση των σπουδών μου στην Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Ο υπεύθυνος τομέας είναι ο Τομέας Επικοινωνιών, Ηλεκτρονικής & Συστημάτων Πληροφορικής υπό την επίβλεψη της καθηγήτριας Θεοδώρας Βρβαρίγου.

Εξαιρετική βοήθεια προήλθε από τον Ευθύμιο Χονδρογιάννη, ο οποίος μου είχε αναθέσει το θέμα της πτυχιακής και ήταν πάντα κατατοπιστικός σε οποιαδήποτε σύνθετα ζητήματα συνάντησα καθ' όλη την διάρκεια και τον ευχαριστώ θερμά.

Κατά την διάρκεια εκπόνησης της διπλωματικής εντρύφησα στις τεχνολογίες του Σημασιολογικού Ιστού και κατανόησα σε βάθος την έννοια της Οντολογίας. Απέκτησα διάφορες γνώσεις για επιστημονικά θέματα που αφορούν την επεξεργασία φαρμακευτικών και κλινικών δεδομένων. Δούλεψα πάνω σε πραγματικά δεδομένα έχοντας την ευκαιρία να συναντήσω ρεαλιστικά προβλήματα και να καταφέρω να τα επιλύσω. Επιπροσθέτως βελτίωσα και εξειδίκευσα τις γνώσεις μου που αφορούν την δημιουργία πραγματικών εφαρμογών τόσο για frontend γραφικών διεπαφών χρηστών, όσο και για backend/servers. Οι γνώσεις αυτές θα συνεισφέρουν θετικά στην αργότερα επαγγελματική μου πορεία.

Πίνακας Περιεχομένων

Περίληψη.....	5
Abstract.....	7
Ευχαριστίες.....	9
1. Εισαγωγή.....	14
2. State of the Art.....	16
2.1. Διαδικτυακές Εφαρμογές.....	16
2.1.1 Τεχνολογίες Ιστού και Javascript.....	16
2.1.2 Vuejs Framework.....	20
2.1.3 Http requests, Ajax και βιβλιοθήκη axios.....	25
2.1.4 Java, Http server και Vertx Framework.....	27
2.2. Σημασιολογικός Ιστός και Οντολογίες.....	31
2.2.1 Σημασιολογικός Ιστός.....	32
2.2.2 Οντολογίες.....	36
2.2.3 Δομή της OWL και βιβλιοθήκη Owlapi.....	38
2.3. Καθορισμός Συσχετίσεων μεταξύ Οντολογιών.....	40
3. Περιγραφή Συστήματος.....	45
3.1. Παρεχόμενες Υπηρεσίες.....	45
3.2. Αρχιτεκτονική Εφαρμογής.....	47
3.2.1 Για τον Client.....	47
3.2.2 Για τον Server.....	48
3.3. Υλοποίηση Επιμέρους Συστημάτων.....	49
3.3.1 Συστήματα Client.....	51
3.3.2 Συστήματα Server.....	54
3.4. Αλληλεπίδραση Επιμέρους Συστημάτων.....	57
4. Εκτέλεση του Συστήματος και Παραδείγματα.....	61
4.1. Αρχικοποίηση και Εκτέλεση του Συστήματος.....	61
4.2. Παραδείγματα Λειτουργίας.....	63
4.2.1 Menu και Home Page.....	63
4.2.2 About Page.....	66
4.2.3 Settings Page.....	67
4.2.4 Visualizer Page.....	67
5. Συμπεράσματα και Αξιολόγηση Συστήματος.....	80
6. Σύνοψη και Μελλοντική Εξέλιξη.....	81
7. Βιβλιογραφικές Αναφορές.....	82

Ευρετήριο Σχημάτων

Figure 1: δενδρική δομή HTML DOM [3].....	17
Figure 2: Event loop of firefox browser [5].....	18
Figure 3: nodejs event loop [6].....	19
Figure 4: Promise states [7].....	20
Figure 5: Reactivity στην VueJs [8].....	23
Figure 6: Vue Instance Life Cycle [9].....	24
Figure 7: Actor model [12].....	29
Figure 8: Vert.x event loop [14].....	30
Figure 9: Vert.x verticles [14].....	31
Figure 10: Vert.x event bus [14].....	31
Figure 11: Αρχιτεκτονική Σημασιολογικού Ιστού.....	33
Figure 12: OWL sublanguages.....	39
Figure 13: Επιμέρους Συστήματα Εφαρμογής.....	50
Figure 14: UML Sequence Diagram μέρος 1ο.....	58
Figure 15: UML Sequence Diagram μέρος 2ο.....	58
Figure 16: Home Page.....	64
Figure 17: Home Page μικρού οριζόντιου μεγέθους.....	65
Figure 18: About Page.....	66
Figure 19: Settings Page.....	67
Figure 20: Visualizer with 1 OWL tab.....	68
Figure 21: Visualizer with 2 OWL tab.....	69
Figure 22: Visualization Statistics.....	70
Figure 23: Visualization Statistics τέλος σελίδας.....	71
Figure 24: Visualization Epoptic View 1.....	72
Figure 25: Visualization Click Element.....	73
Figure 26: Visualization Epoptic View με επιλεγμένο στοιχείο.....	74
Figure 27: Visualization View By Rule, απλός κανόνας.....	75
Figure 28: Visualization View By Rule, σύνθετος κανόνας.....	76
Figure 29: OWL Information.....	77
Figure 30: OWL Classes.....	77
Figure 31: OWL Object Properties.....	78
Figure 32: OWL Data Properties.....	78
Figure 33: OWL Annotation Properties.....	79

Ευρετήριο Πινάκων

Table 1: Συστατικά των οντολογιών.....	37
Table 2: RDF Στοιχεία Σύνταξης βασισμένα στην XML.....	38
Table 3: Οι (Υ)ποχρεωτικές και (Π)ροαιρετικές παράμετροι ενός Κανόνα Αντιστοίχισης...	41
Table 4: Server Error Status Codes.....	56

1. Εισαγωγή

Στις μέρες μας η αποθήκευση και η επεξεργασία της κλινικής και φαρμακευτικής γνώσης αποτελεί ζήτημα με μεγάλο ενδιαφέρον. Η γνώση αυτή αποκτάται από διαφορετικούς οργανισμούς (κλινικές, νοσοκομεία, φαρμακεία, κτλ) και αποθηκεύεται σε διαφορετικά σχήματα και δομές όπως απλά αρχεία ή διάφορες βάσεις δεδομένων. Ωστόσο πέρα από την αποθηκευμένη πληροφορία αυτή καθ' αυτή, μεγάλη σημασία έχει να καθορίσουμε τους όρους που χρησιμοποιούνται για να περιγράψουν αυτή την πληροφορία. Για αυτόν τον λόγο χρησιμοποιούμε Οντολογίες οι οποίες θα εξηγηθούν αναλυτικά στα επόμενα κεφάλαια.

Στην περίπτωση της διπλωματικής μας μελετάμε τέτοια δεδομένα που αφορούν φάρμακα και ασθένειες σε μορφή Οντολογιών που είναι αποθηκευμένα σε αρχεία γλώσσας OWL (Web Ontology Language). Αυτά τα αρχεία περιέχουν πολλές ίδιες έννοιες (π.χ. κάποιο συγκεκριμένο φάρμακο) οι οποίες όμως απεικονίζονται με διαφορετικά μοντέλα σε κάθε περίπτωση. Ο καθορισμός των συσχετίσεων μεταξύ τέτοιων διαφορετικών μοντέλων έχει ιδιαίτερη σημασία, καθώς μπορεί να μας οδηγήσει στην λύση διάφορων προβλημάτων. Τέτοια προβλήματα είναι η ενοποίηση δεδομένων και η απάντηση ερωτημάτων παρά την όποια διαφορετική φύση των μοντέλων μας.

Για τον καθορισμό των συσχετίσεων μεταξύ των οντολογιών υπάρχει αρκετή δουλειά στην βιβλιογραφία όσον αφορά τον εντοπισμό των συσχετίσεων μεταξύ των όρων των οντολογιών αλλά και την έκφραση των κανόνων σε μια μορφή που είναι κατανοητή από τον υπολογιστή. Στην παρούσα διπλωματική εργασία βασιζόμαστε στο εργαλείο *Ontology Alignment Tool* [1] το οποίο πραγματοποιεί αυτοματοποιημένη παραγωγή κανόνων συσχετισμών μεταξύ οντολογιών. Το εργαλείο αυτό θα αναλυθεί στα επόμενα κεφάλαια και θα εξηγηθεί ο τρόπος που προκύπτουν οι κανόνες καθώς και η μορφή τους.

Σκοπός της εργασίας μας είναι η οπτική αναπαράσταση αυτών των κανόνων καθώς και η αναπαράσταση των βασικών όρων των Οντολογιών με τρόπο κατανοητό. Για αυτόν τον σκοπό θα δημιουργήσουμε μια διαδικτυακή εφαρμογή η οποία θα επεξεργάζεται δύο Οντολογίες *owl* καθώς και τους κανόνες συσχέτισης τους οι οποίοι έχουν παραχθεί με το παραπάνω εργαλείο. Το αποτέλεσμα θα είναι σχηματικές αναπαραστάσεις των παραπάνω με δυνατότητα αλληλεπίδρασης του χρήστη για την πλήρη και εύκολη κατανόηση αυτών των συσχετίσεων. Αυτό θα μπορεί να οδηγήσει στην εύρεση λαθών του εργαλείου όπως για παράδειγμα η απουσία κανόνων που θα έπρεπε να υπάρχουν.

Στο κεφάλαιο 2 αναλύουμε τις βασικές θεωρητικές έννοιες που χρειάζονται για την κατανόηση της εργασίας μας καθώς και τις βασικές τεχνολογίες στις οποίες θα βασιστεί η εφαρμογή μας.

Στο κεφάλαιο 3 περιγράφουμε αναλυτικά το σύστημα που υλοποιήσαμε για την αναπαράσταση των συσχετίσεων μεταξύ οντολογιών. Αναλύουμε τα υποσυστήματά του και την σύνδεσή τους μεταξύ τους.

Στο κεφάλαιο 4 θα δείξουμε το σύστημα σε λειτουργία. Θα περιγράψουμε τον τρόπο για να μπορεί εύκολα να το εκτελέσει ο αναγνώστης καθώς και θα δείξουμε παραδείγματα χρήσης του συστήματος.

Τέλος στο κεφάλαιο 5 θα συνοψίσουμε τα συμπεράσματά μας καθώς και θα αξιολογήσουμε το σύστημά μας βάση αντικειμενικών κριτηρίων.

2. State of the Art

Εδώ θα αναλύσουμε τις βασικές έννοιες που χρησιμοποιούνται στην μελέτη μας καθώς και τις κύριες τεχνολογίες στις οποίες θα βασιστεί η εφαρμογή μας.

2.1. Διαδικτυακές Εφαρμογές

Θα εξηγήσουμε τις βασικές αρχές στις οποίες κατασκευάζονται οι διαδικτυακές εφαρμογές και θα αναλύσουμε σε λειτουργικό βαθμό τις τεχνολογίες που θα χρησιμοποιήσουμε.

2.1.1 Τεχνολογίες Ιστού και Javascript

Μια ιστοσελίδα διαδικτύου (*web page*) αποτελείται από αρχεία τα οποία έχουν μια συγκεκριμένη δομή ώστε να μπορούν οι περιηγητές ιστού (*web browsers*) να τα διαβάζουν για να τα αναπαραστήσουν στην οθόνη. Η λειτουργία αυτών των αρχείων περιγράφεται από την προδιαγραφή *html* (*html specification*) [2] . Αυτή η προδιαγραφή ορίζει μια γενικευμένη γλώσσα για την περιγραφή των εγγράφων και των εφαρμογών, καθώς και κάποιες προγραμματιστικές διεπαφές (*api*) για την αλληλεπίδραση με την αναπαράσταση στην μνήμη των πόρων που χρησιμοποιεί αυτή η γλώσσα.

Γενικά ορίζονται δύο βασικές συντάξεις με τις οποίες μπορούν να γραφούν τέτοια αρχεία. Η πρώτη είναι η *HTML* (*HyperText Markup Language*) και η δεύτερη είναι η *XML* (*eXtensible Markup Language*). Δεν θα μπούμε σε λεπτομέρειες περιγραφής τους αλλά και οι 2 γλώσσες που χρησιμοποιούνται για την σύνταξη τέτοιων αρχείων μπορούν να διαβαστούν από όλους τους σύγχρονους περιηγητές.

Η αναπαράσταση στην μνήμη των αντικειμένων που χρησιμοποιούνται στην ιστοσελίδα ή εφαρμογή ιστού μαζί με την προγραμματιστική διεπαφή που ορίζεται για να ελέγχουμε την κατάσταση αυτών των αντικειμένων είναι γνωστή ως *HTML DOM* (*Document Object Model*).

Καταλαβαίνουμε ότι το DOM είναι δύο έννοιες:

- Είναι αρχικά ένα μοντέλο αντικειμένων για ένα έγγραφο html. Για κάθε σελίδα που φορτώνει ο περιηγητής φτιάχνει ένα αντικείμενο DOM. Το αντικείμενο αυτό αποτελείται από όλα τα *html στοιχεία* (*html elements*) οργανωμένα σε δενδρική δομή. Για κάθε τέτοιο στοιχείο υπάρχει η πληροφορία για τις *ιδιότητες* (*properties*), τις *μεθόδους* (*methods*) και τα *γεγονότα* (*events*) που σχετίζονται με αυτό το στοιχείο.
- Είναι επίσης μια προγραμματιστική διεπαφή μέσω της οποίας μπορούμε να επηρεάσουμε και να ελέγξουμε την κατάσταση αυτών των αντικειμένων. Η γλώσσα προγραμματισμού που χρησιμοποιείται από όλους τους περιηγητές για αυτή την λειτουργία είναι η *Javascript*.

Εναποθέτουμε ένα παράδειγμα της δενδρικής δομής των στοιχείων ενός html αντικειμένου:

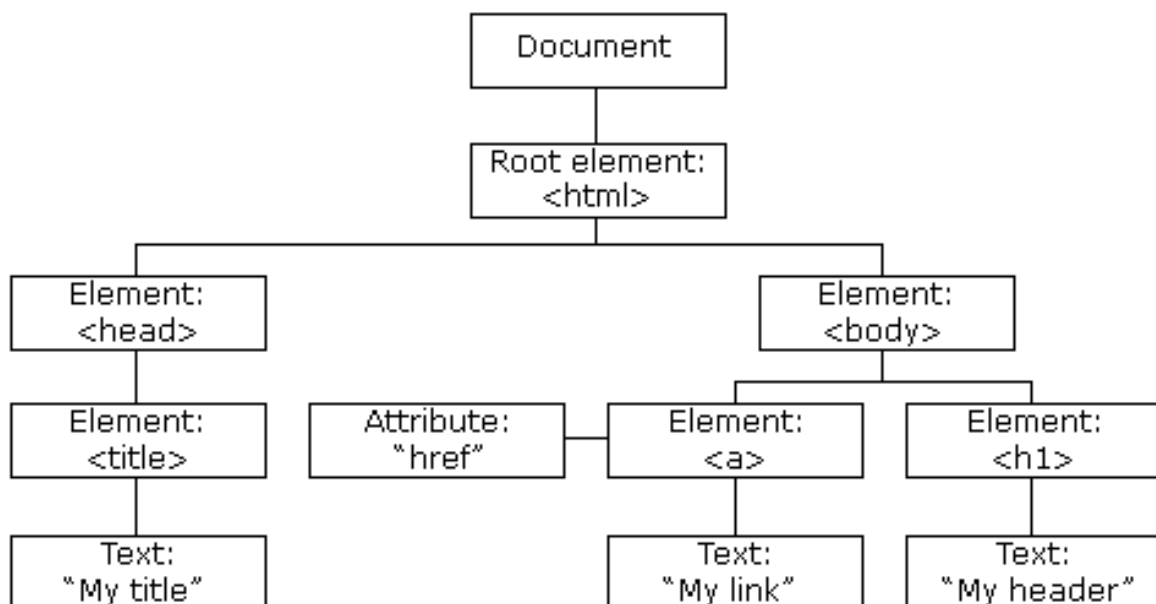


Figure 1: δενδρική δομή HTML DOM [3]

Αναφέραμε ότι η javascript είναι η κύρια γλώσσα επεξεργασίας του DOM. Ο κάθε περιηγητής έχει την δικιά του υλοποίηση javascript για να υποστηρίξει τις απαραίτητες λειτουργίες. Όλες οι υλοποιήσεις τηρούν τις προδιαγραφές για την γλώσσα όπως ορίζονται στο *ECMAScript specification*, τελευταία έκδοση του

οποίου είναι αυτή του 2020 [4]. Ο κώδικας javascript που συμπεριλαμβάνεται από κάποιο αρχείο html εκτελείται από τον περιηγητή κατά την φόρτωση της ιστοσελίδας. Υπάρχουν αρκετές διαφορές από υλοποίηση σε υλοποίηση αλλά για την κατανόηση των επόμενων μας ενδιαφέρουν να εξηγήσουμε κάποιες συμπεριφορές που ορίζονται στο πρότυπο και είναι ίδιες σε κάθε υλοποίηση.

Για διάφορους λόγους η javascript εκτελείται σε μόνο έναν νήμα επεξεργασίας. Αυτό σημαίνει ότι δεν υπάρχει δυνατότητα δημιουργίας πολλαπλών νημάτων από τον επεξεργαστή ώστε να μπορούν να εκτελεστούν πολλές λειτουργίες ταυτόχρονα. Για αυτό τον λόγο η javascript λειτουργεί με ένα μοντέλο ταυτοχρονισμού γνωστό ως *βρόγχος γεγονότων (event loop)*. Οι λεπτομέρειες διαφέρουν ανάλογα με την *μηχανή javascript (javascript engine)* του κάθε περιηγητή, αλλά η λογική είναι ότι υπάρχει μια ουρά μηνυμάτων ή γεγονότων καθένα με τα οποία συνδέονται με μια *συνάρτηση (function)*. Αυτά εκτελούνται κάθε φορά με την σειρά καθώς νέα γεγονότα προστίθενται στην ουρά.

Για καλύτερη κατανόηση δείχνουμε μια γενικευμένη εικόνα επεξήγησης του event loop από την javascript engine του περιηγητή mozilla firefox:

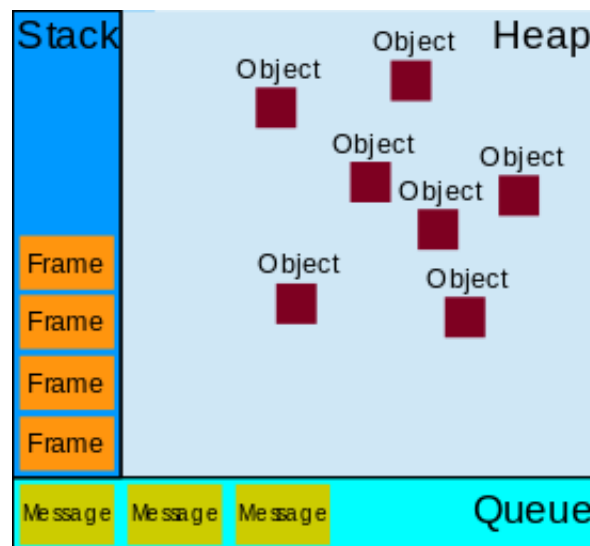


Figure 2: Event loop of firefox browser [5]

καθώς και μια εικόνα των φάσεων που περνάει το event loop του nodejs όπου είναι ένα περιβάλλον εκτέλεσης javascript (*javascript runtime*) βασισμένο στο javascript engine του περιηγητή chrome:



Figure 3: nodejs event loop [6]

Καταλαβαίνουμε ότι αν η συνάρτηση εκτέλεσης κάποιου γεγονότος είναι μεγάλης διάρκειας τότε όλη η ουρά μπλοκάρει και δεν εκτελείται κάποιο άλλο γεγονός μέχρι να τελειώσει το πρώτο. Για αυτό τον λόγο προσπαθούμε να φτιάχνουμε συναρτήσεις με τρόπο ασύγχρονο. Δηλαδή αρχικά τις καταμερίζουμε σε μικρότερα μέρη. Όταν καλούμε κάποια συνάρτηση, τότε αφού τελειώσει το μέρος της που εκτελούνταν, αυτή επιστρέφει τον έλεγχο ώστε να μπορέσει να εκτελεστεί κάποια άλλη συνάρτηση από την ουρά του event loop. Καταλαβαίνουμε ότι επειδή ο έλεγχος επιστρέφει μετά από την εκτέλεση ενός μικρού μέρους της αρχικής συνάρτησης που θέλαμε να υλοποιήσουμε και όχι αφού εκτελεστεί ολόκληρη, θέλουμε κάποιον τρόπο για να καταλαβαίνουμε πότε έχει ολοκληρωθεί όλη η εργασία που θέλαμε να κάνουμε. Στην σύγχρονη javascript η υλοποίηση των ασύγχρονων συναρτήσεων γίνονται μέσω του μηχανισμού των υποσχέσεων

(*Promises*). Ένα promise χρησιμοποιείται για να περιμένουμε κάποια τιμή που ακόμα δεν είναι διαθέσιμη. Μπορεί να έχει 3 καταστάσεις:

- *εκκρεμής (pending)* όταν δεν έχει ολοκληρωθεί ακόμα
- *ολοκληρωμένη (fulfilled)* όταν η εκτέλεση ολοκληρώθηκε και η τιμή είναι διαθέσιμη
- *απορριφθείσα (rejected)* όταν η λειτουργία υπολογισμού της τιμής απέτυχε

Έτσι μπορούμε να προγραμματίσουμε τι θα γίνει σε κάθε περίπτωση και να εκτελούμε ταυτόχρονα υπολογισμούς χωρίς να μπλοκάρουμε το event loop. Παραθέτουμε ένα διάγραμμα των καταστάσεων ενός promise σε προγραμματιστικό επίπεδο:

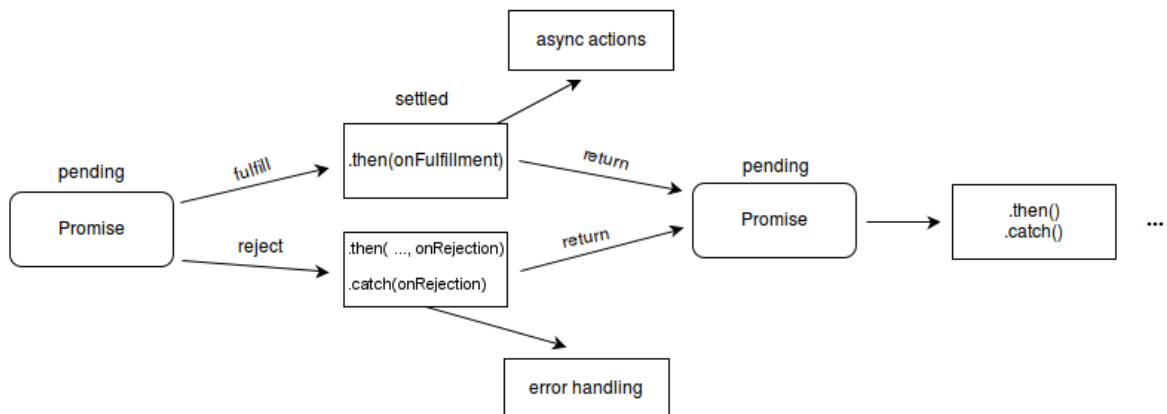


Figure 4: Promise states [7]

2.1.2 Vuejs Framework

Έχουμε εξηγήσει τις βασικές λειτουργίες των διαδικτυακών εφαρμογών και του DOM. Ωστόσο, δεδομένου ότι η βασικές λειτουργίες του σχεδιάστηκαν αρκετά παλιά, παρά τις βελτιώσεις του τα τελευταία χρόνια υπάρχουν ορισμένα μειονεκτήματα. Κύριο μειονέκτημα είναι η κακή απόδοσή του. Έχουμε αναφέρει ότι το DOM κρατάει μια δενδρική δομή των αντικειμένων html. Κάθε φορά που θέλουμε να αλλάξουμε κάποιο html element τότε πρέπει να βρεθεί αυτό το στοιχείο από το δέντρο, να επεξεργαστεί και μετά ο περιηγητής να το απεικονίσει στην οθόνη

(render). Όταν αλλάζουμε πολλά στοιχεία προγραμματιστικά με javascript υπάρχει μεγάλο υπολογιστικό κόστος. Το δεύτερο κύριο μειονέκτημα είναι ότι η προγραμματιστική διεπαφή που ορίζεται για την επεξεργασία του DOM δεν είναι πολύ εύχρηστη. Υπάρχουν βιβλιοθήκες όπως η γνωστή jQuery όπου προσφέρουν λίγο πιο εύχρηστες προγραμματιστικές διεπαφές, ωστόσο αν και έχουν βρει μεγάλη επιτυχία στην δημιουργία απλών ιστοσελίδων όταν πρόκειται για πιο σύνθετες εφαρμογές υστερούν σημαντικά. Για αυτόν τον σκοπό έχουν δημιουργηθεί διάφορα πιο σύνθετα frameworks που προσφέρουν αυξημένες δυνατότητες στους προγραμματιστές.

Ιστορικά τα παλιότερα χρόνια είχαν επικρατήσει τα *frameworks* από την μεριά του διακομιστή (*server side frameworks*). Η λογική είναι ότι οι ιστοσελίδες προσφέρονται από έναν υπολογιστή που έχει ρόλο server. Ωστόσο, πριν οι σελίδες σταλούν στο περιηγητές έχουν επεξεργαστεί με διάφορες *template engines* όπου έχουν δημιουργήσει από πριν το τελικό DOM. Έτσι ελαχιστοποιείται η χρήση javascript και η επεξεργασία του DOM από τους περιηγητές. Αν και αυτή η μέθοδος προτιμάται ακόμα σε πολλές περιπτώσεις, το κύριο μειονέκτημά της είναι η συνεχής εξάρτηση από τον server και οι συνεχείς κλήσεις για καινούριες σελίδες κάθε φορά που υπάρχει ανάγκη για επεξεργασία του DOM.

Ως εναλλακτική λύση έχουν δημιουργηθεί τα *frameworks* από την μεριά του πελάτη (*client side frameworks*). Αυτά προσφέρουν εξελιγμένες δυνατότητες επεξεργασίας του DOM με χρήση javascript χωρίς να χρειάζεται κλήση σε κάποιον server για κάθε σελίδα. Έτσι ο server μπορεί να χρησιμοποιείται αποκλειστικά για να προσφέρει δεδομένα όταν αυτό χρειάζεται χωρίς να έχει κάποιο ρόλο στην επεξεργασία των σελίδων και του DOM. Η διαδικτυακή εφαρμογή χρειάζεται τότε απλά να γίνει διαθέσιμη ως στατικές σελίδες όπου ο περιηγητής μπορεί να κατεβάσει. Η δυνατότητα αυτή της προσφοράς στατικών σελίδων υπάρχει σε πολλούς μικρούς και οικονομικούς από άποψης πόρων servers, αλλά επίσης προσφέρεται πάντα από όλους τους γνωστούς servers που συνήθως χρησιμοποιούνται για δυναμικές σελίδες σε συνδυασμό με κάποια γλώσσα για server side επεξεργασία (π.χ. Apache

ή nginx με γλώσσα php, Tomcat με γλώσσα java, κτλ.). Έτσι υπάρχει ευελιξία στον τρόπο με τον οποίο θα γίνει διαθέσιμη η εφαρμογή στους χρήστες.

Ένα τέτοιο client side javascript framework είναι το VueJs το οποίο θα αναλύσουμε εδώ. Πρόκειται καταρχάς για ένα, όπως αποκαλείται, *προοδευτικό framework* (*progressive framework*). Η έννοια αυτή σημαίνει ότι δεν είναι απαραίτητο να φτιαχτεί μια ολόκληρη ιστοσελίδα ή διαδικτυακή εφαρμογή βάση αυτού, αλλά μπορεί να χρησιμοποιηθεί μόνο για ένα συγκεκριμένο μέρος της εφαρμογής όπου χρειάζονται αυξημένες δυνατότητες επεξεργασίας όταν αυτό κρίνεται απαραίτητο. Η βασική ιδέα που διέπει την VueJs είναι αυτή του *εικονικού DOM* (*Virtual DOM*). Ο προγραμματιστής δεν ασχολείται με το να επεξεργαστεί απευθείας το DOM, αν και εξακολουθεί να υπάρχει αυτή η δυνατότητα. Αντιθέτως, ορίζονται μεταβλητές σε javascript οι οποίες δένονται με συγκεκριμένα elements του DOM (*data binding*). Λέμε ότι τα δεδομένα αυτά που ορίζονται με αυτόν τον τρόπο είναι *αντιδραστικά* (*reactive*). Αυτό σημαίνει ότι το framework παρακολουθεί αυτά τα δεδομένα για αλλαγές. Όταν αυτά αλλάξουν τότε αυτόματα ειδοποιείται η συνάρτηση απεικόνισης (*rendering*) και δημιουργείται το καινούριο πραγματικό DOM με τα νέα δεδομένα. Οι δυνατότητές του είναι αρκετά εξελιγμένες ώστε να γίνονται *rendered* μόνο οι περιοχές όπου άλλαξαν εξοικονομώντας έτσι επεξεργαστικούς πόρους. Επίσης, όταν αλλάζουν πολλές τέτοιες μεταβλητές ταυτόχρονα αλλά στην ίδια φάση λειτουργίας, θα γίνει μόνο μια φορά *render* μετά από όλες τις αλλαγές κάνοντας σαφές το πλεονέκτημα στην απόδοση σε σχέση με την απευθείας επεξεργασία του DOM. Παραθέτουμε μια εικόνα για την καλύτερη κατανόηση της παρακολούθησης αυτών των δεδομένων και του *rendering*:

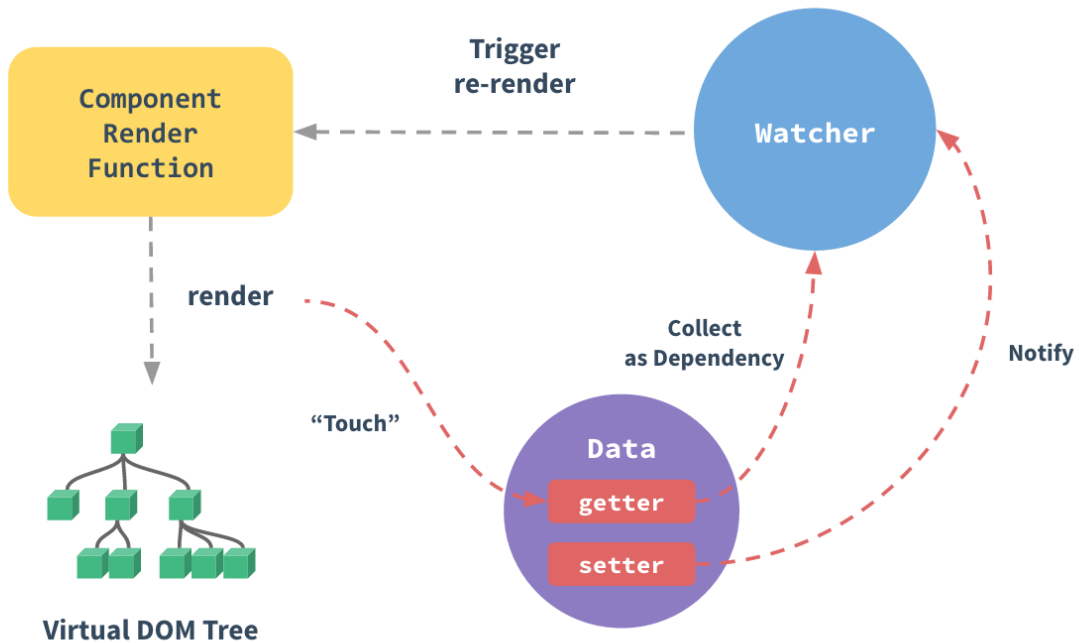


Figure 5: Reactivity στην VueJs [8]

Μια VueJs εφαρμογή οργανώνεται σε ξεχωριστά δομικά στοιχεία (*components*). Κάθε τέτοιο component έχει τον δικό του κώδικα html, την δικό του κώδικα javascript και την δικιά του css για την επεξεργασία της εμφάνισής του. Το αρχικό component είναι το κύριο και κάθε επόμενο προστίθεται σε κάποιο ήδη υπάρχων με σχέση πατέρα παιδιού. Ο πατέρας μπορεί να επικοινωνεί με το παιδί περνώντας του *ιδιότητες (props)*, ενώ το παιδί επικοινωνεί με τον πατέρα στέλνοντάς του γεγονότα (*events*). Κάθε component που δημιουργείται περνάει από διάφορα στάδια ενός κύκλου ζωής όπως φαίνεται στο παρακάτω σχήμα:

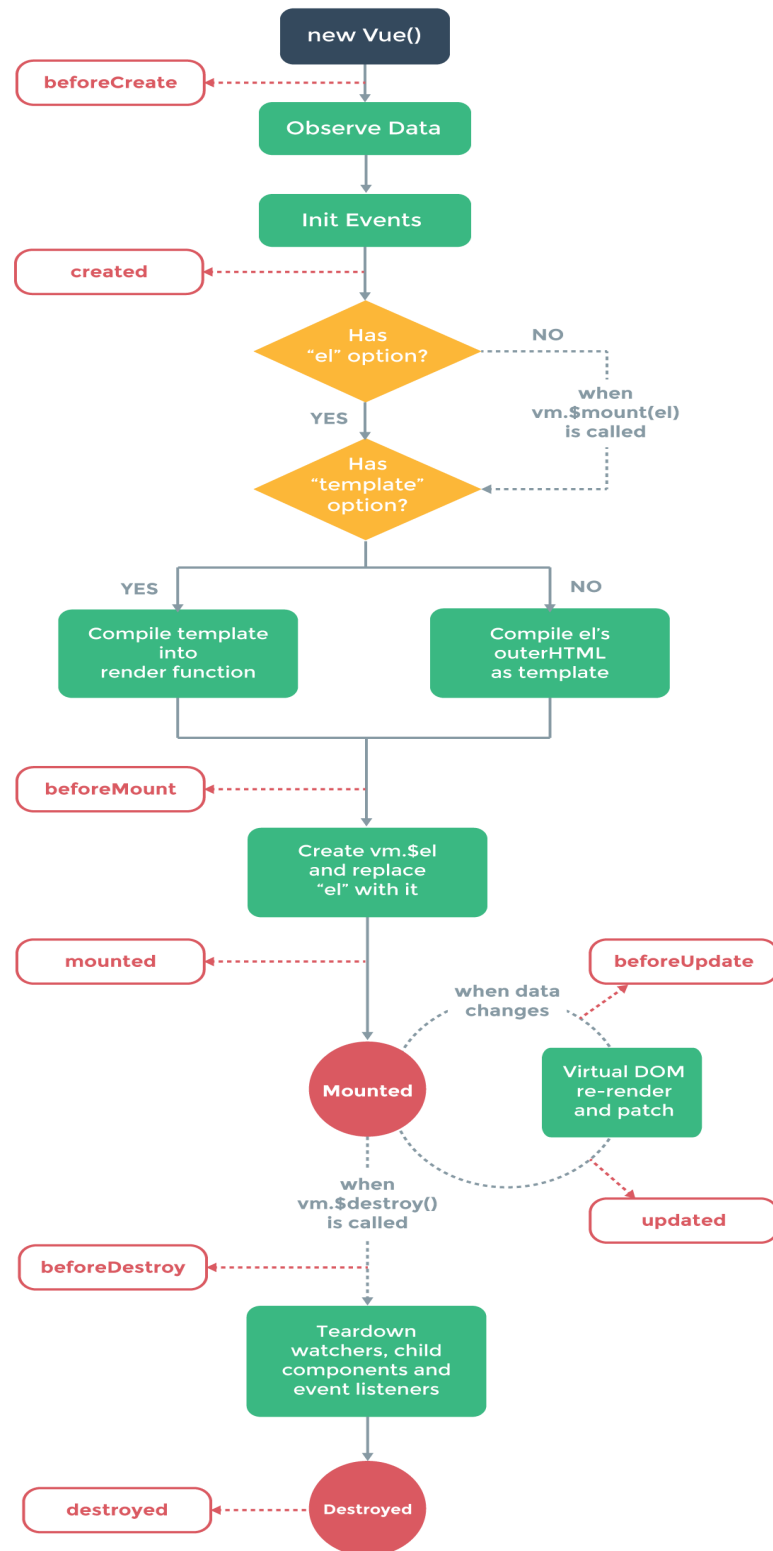


Figure 6: Vue Instance Life Cycle [9]

2.1.3 Http requests, Ajax και βιβλιοθήκη axios

Έχουμε εξηγήσει ότι με τα client side frameworks δεν χρειάζεται να υπάρχει κάποιος server που να επεξεργάζεται τις διαδικτυακές σελίδες με νέα δεδομένα. Ωστόσο, σε πολλές εφαρμογές, όπως θα δούμε και στην δική μας, χρησιμοποιείται ένας server για να επεξεργαστεί και να μας στείλει δεδομένα. Καταλαβαίνουμε ότι θέλουμε έναν τρόπο να μπορούμε να αλληλεπιδράσουμε με έναν server από την μεριά του client αφού έχει φορτωθεί μια σελίδα.

Τα αιτήματα που μας ενδιαφέρουν στην περίπτωση μας είναι τα *http requests*. Το *http (HyperText Transfer Protocol)* είναι ένα πρωτόκολλο δικτύου. Χρησιμοποιείται συνήθως από εφαρμογές για να στέλνουμε ή να λαμβάνουμε δεδομένα από έναν server. Δεν θα αναλύσουμε λεπτομέρειες καθώς την *προδιαγραφή* του (*specification*) μπορεί κανείς να την βρει στο διαδίκτυο [10]. Επιγραφικά αναφέρουμε ότι υποστηρίζει διάφορους μεθόδους ανάλογα με το τι θέλουμε να επιτύχουμε όπως οι μέθοδοι:

- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT
- OPTIONS

Η καλύτερη μέθοδος για να στείλουμε http αιτήματα όπως τα περιγράψαμε παραπάνω ονομάζεται *AJAX (Asynchronous JavaScript and XML)*. Δεν πρόκειται για κάποια συγκεκριμένη υλοποίηση αλλά για μια τεχνική. Ο όρος XML περιλαμβάνεται για ιστορικούς λόγους αλλά δεν η ανάγκη η χρήση της XML. Με αυτή την τεχνική μπορούμε να στέλνουμε http αιτήματα σε κάποιον server ασύγχρονα χωρίς να μπλοκάρουμε το event loop (έχουμε εξηγήσει τη σημασία αυτού στις προηγούμενες ενότητες). Όταν το αίτημα ολοκληρωθεί καλείται κάποιος κώδικας που έχουμε ορίσει ανάλογα με το αν ήταν επιτυχές ή αν απέτυχε αυτό το αίτημα. Οι υλοποιήσεις javascript των περιηγητών διαθέτουν υποστήριξη για αυτή την λειτουργία μέσω του *αντικειμένου XMLHttpRequest (XMLHttpRequest object)*. Δεν θα αναλύσουμε τις λεπτομέρειες αυτού καθώς πρόκειται για μια συγκεκριμένη υλοποίηση της παραπάνω τεχνικής.

Εμάς μας ενδιαφέρει η βιβλιοθήκη *Axios*. Πρόκειται για μια βιβλιοθήκη javascript η οποία έχει δημιουργηθεί για την δημιουργία τέτοιων ασύγχρονων αιτημάτων. Υποστηρίζει πολλές διαφορετικές μεθόδους http και είναι αρκετά ευέλικτο ως προς την παραμετροποίηση αυτών των αιτημάτων. Ο μηχανισμός με τον οποίο πετυχαίνει τα ασύγχρονα αιτήματα χωρίς να μπλοκάρει το event loop είναι αυτός των *Promises* τον οποίο έχουμε εξηγήσει σε προηγούμενη ενότητα. Αυτή η χρήση τους είναι πολύ σημαντική αφού επιτρέπει την εύκολη συνεργασία με άλλες βιβλιοθήκες όπου χρησιμοποιούν *Promises* ή και ακόμα να φτιάχνουμε δικές μας αφαιρετικές συναρτήσεις (abstractions) όπου θα χρησιμοποιούν αυτές την *axios* και θα επιστρέφουν το αντίστοιχο *Promise*, κρύβοντας τις λεπτομέρειες από το υπόλοιπο πρόγραμμά μας.

Τα δεδομένα που θα λαμβάνουμε από τον server θα είναι σε μορφή *JSON* αντί για *XML*. Το *JSON (JavaScript Object Notation)* είναι μια μορφοποίηση για ανταλλαγή δεδομένων. Είναι επίσης ένας τύπος αρχείου όπου αποθηκεύονται δεδομένα με την συγκεκριμένη μορφή. Έχει πολλά πλεονεκτήματα σε σχέση με την *XML*. Τα κυριότερα είναι

- Είναι πιο ευανάγνωστο και μπορεί να διαβαστεί ευκολότερα από τον άνθρωπο.

- Προσθέτει λιγότερους χαρακτήρες για την μορφοποίηση αφού βασίζεται κυρίως στην χρήση αγκύλων. Άρα τα αρχεία έχουν μικρότερο μέγεθος.
- Είναι πιο αποδοτικό στην ανάλυσή του και την επεξεργασία του από υπολογιστές με την χρήση διάφορων βιβλιοθηκών που υπάρχουν.

Το πρότυπο που ορίζεται είναι ανοιχτό και μπορεί κανείς να βρει περισσότερα για την χρήση του στο διαδίκτυο [11].

2.1.4 Java, Http server και Vertx Framework

Έχουμε αναφέρει ότι πολλές διαδικτυακές εφαρμογές χρησιμοποιούν κάποιον server είτε για να προσφέρει ιστοσελίδες είτε για να δέχεται και να στέλνει δεδομένα. Εμείς θα αναφερθούμε στους http servers όπου υλοποιούνται με την γλώσσα προγραμματισμού java και χρησιμοποιούνται από τους clients για να λαμβάνουν και να στέλνουν δεδομένα με τους τρόπους που εξηγήσαμε παραπάνω.

Η java είναι μια γλώσσα προγραμματισμού που έχει σχεδιαστεί κυρίως για να τρέχει σε μια εικονική μηχανή την λεγόμενη *JVM (Java Virtual Machine)*. Ο κώδικας πρώτα μεταγλωτίζεται (compiling) σε αρχεία που περιέχουν java bytecode η οποία είναι μια δυαδική μορφή αναπαράστασης του κώδικα αναγνωρίσιμη από το JVM για να μπορεί να εκτελεστεί. Λόγω της ιστορίας της υπάρχουν πληθώρα από βιβλιοθήκες και πολλές υλοποιήσεις διαφορετικών http servers. Υπάρχουν επίσης πολλές υλοποιήσεις με αρκετές διαφορές τέτοιων εικονικών μηχανών που αναγνωρίζουν java bytecode. Κύριο σημείο ενδιαφέροντος είναι ότι εδώ δεν έχουμε τον περιορισμό του μοναδικού thread ενός προγράμματος που υπάρχει στις υλοποιήσεις javascript των περιηγητών. Άρα ένα πρόγραμμα μπορεί να χρησιμοποιεί πολλά threads για να πετύχει τον σκοπό του.

Ένας http server πρέπει να μπορεί να δέχεται ένα http αίτημα με κάποια από τις http μεθόδους που έχουμε αναφέρει παραπάνω. Κάθε αίτημα θα δέχεται κάποια δεδομένα, θα εκτελεί κάποια συγκεκριμένη εργασία και θα επιστρέφει κάποιο αποτέλεσμα. Στόχος μας είναι να μπορούμε να εξυπηρετήσουμε πολλά τέτοια αιτήματα ταυτόχρονα. Κάθε αίτημα θα πρέπει να εκτελείται απομονωμένα χωρίς τα

ενδιάμεσα αποτελέσματα κάποιας τρέχουσας εργασίας ενός αιτήματος να επηρεάζει τα αποτελέσματα κάποιας άλλης. Θα αναφέρουμε περιληπτικά 3 γνωστά μοντέλα με τα οποία πετυχαίνουμε αυτόν τον ταυτοχρονισμό (concurrency).

- Ο πιο ιστορικός τρόπος να πετύχουμε τον ταυτοχρονισμού είναι με *νήματα* (*threads*). Ένα νήμα είναι απομονωμένο και έχει τον δικό χώρο μνήμης για μεταβλητές αν και μπορεί να διαμοιράζεται πόρους με διάφορους τρόπους. Με αυτό το μοντέλο επιλέγουμε να δημιουργούμε ένα καινούριο νήμα για κάθε καινούριο αίτημα που καλούμε να απαντήσουμε. Έτσι κάθε εργασία που πρέπει να εκτελεστεί είναι απομονωμένη από τις υπόλοιπες. Αναφέρουμε για παράδειγμα ότι ο Tomcat server χρησιμοποιεί αυτό το μοντέλο, αν και υπάρχουν αρκετοί τρόποι παραμετροποίησης του. Μειονέκτημα αυτού του μοντέλου είναι ότι ένας υπολογιστής μπορεί να εκτελεί τόσα παράλληλα νήματα όσα μπορεί να υποστηρίξει ο επεξεργαστής του. Σε κρίσιμους servers με μεγάλο αριθμό αιτημάτων, τα νήματα που δημιουργούνται είναι αρκετές φορές πολλαπλάσια των νημάτων του επεξεργαστή, επιφέροντας μεγάλη μείωση στην απόδοση.
- Ένα άλλο πιο πρόσφατο μοντέλο είναι το *μοντέλο των ηθοποιών* (*actor model*). Σε αυτό μια εργασία γίνεται σε κάποιον actor. Κάθε actor είναι απομονωμένος έχοντας την δικιά του *ιδιωτική κατάσταση* (*private state*) που δεν μπορεί να αλλάξει από κάποιον άλλον. Ο κάθε actor έχει ένα *γραμματοκιβώτιο* (*mailbox*) και επικοινωνούν ανταλλάσσοντας μηνύματα μεταξύ τους. Οι υλοποιήσεις διαφέρουν από σύστημα σε σύστημα αλλά οι actors πάντα χρειάζονται λιγότερους πόρους από τα threads και μπορούν να υπάρχουν σε μεγάλο βαθμό ταυτόχρονα. Μια τέτοια υλοποίηση βασισμένη στο actor model είναι αυτή της βιβλιοθήκης Akka. Δείχνουμε μια εικόνα για καλύτερη κατανόηση αυτού του μοντέλου.

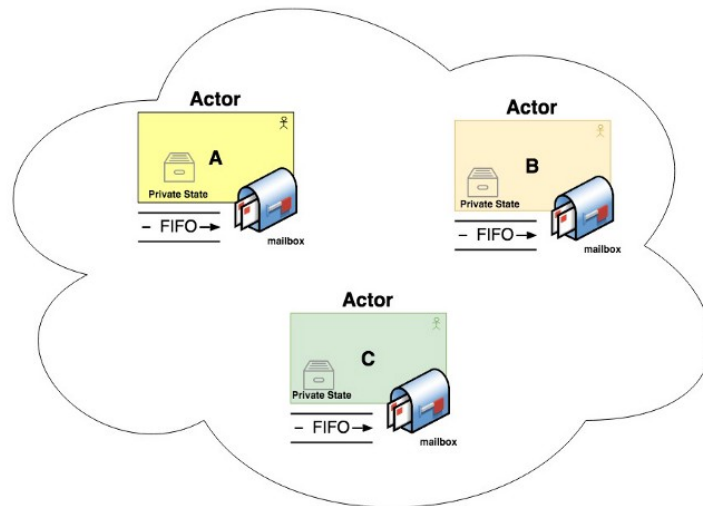


Figure 7: Actor model [12]

- Το τελευταίο μοντέλο που θα αναφέρουμε είναι αυτό του ασύγχρονου προγραμματισμού με κάποιο event loop. Το έχουμε ήδη εξηγήσει καθώς οι υλοποιήσεις javascript βασίζονται σε αυτό το μοντέλο. Πολλές φορές έχει την καλύτερη απόδοση από όλα τα μοντέλα. Ωστόσο, μειονέκτημά του είναι ότι δεν πρέπει ποτέ να μπλοκάρουμε το event loop με συνεχόμενες εργασίες που διαρκούν αρκετή ώρα (π.χ. επαναλήψεις). Έτσι, κάνει τον προγραμματισμό δυσκολότερο αφού πρέπει ο προγραμματιστής να μετατρέπει τις συναρτήσεις του σε ασύγχρονες καθώς και να χρησιμοποιεί όσο μπορεί βιβλιοθήκες με ασύγχρονη λογική.

Εμείς θα επικεντρωθούμε στο Vert.x framework [13]. Πρόκειται για ένα framework που είναι γραμμένο στην γλώσσα Polyglot, η οποία μπορεί να μεταφραστεί σε πολλές άλλες γλώσσες. Έτσι είναι εύκολα διαθέσιμο για όλες τις γλώσσες που μπορούν να τρέξουν στο JVM όπως Java, Scala, Kotlin, κτλ. Έχει εμπνευστεί από την NodeJs που είχαμε αναφέρει ως ένα περιβάλλον για εκτέλεση javascript. Βασίζεται στο μοντέλο του ασύγχρονου προγραμματισμού και μας προσφέρει την δυνατότητα να δημιουργούμε ασύγχρονες συναρτήσεις. Όπως όλες οι υλοποιήσεις αυτού του μοντέλου διαθέτει ένα event loop όπου εκτελεί τα γεγονότα που εισέρχονται σε μια ουρά:

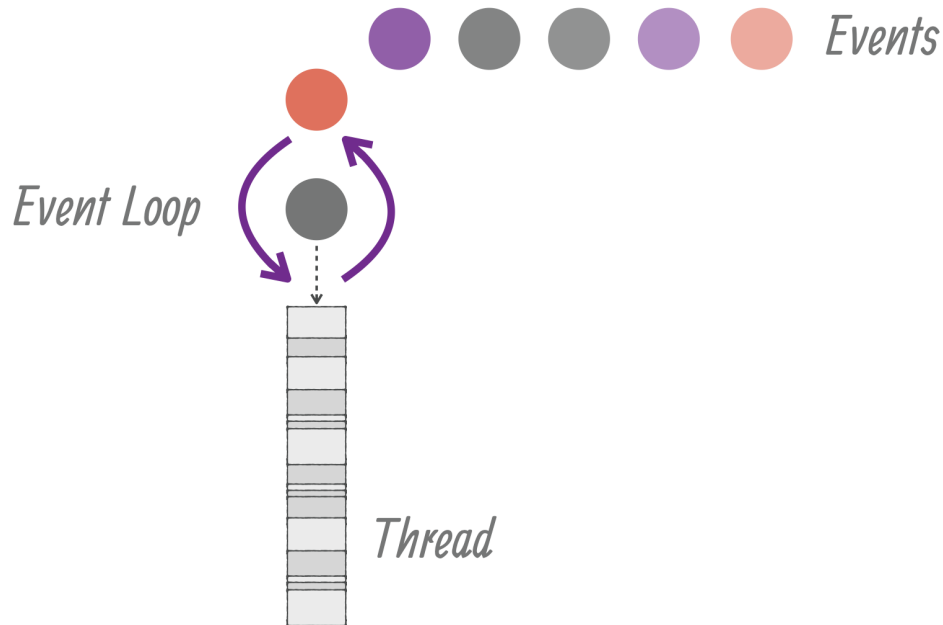


Figure 8: Vert.x event loop [14]

Ισχύουν όλα αυτά που έχουμε εξηγήσει σε προηγούμενες ενότητες για το event loop. Ο τρόπος για να περιμένουμε κάποιο ασύγχρονο αποτέλεσμα έχει την ίδια λογική, απλά οι ορολογίες είναι λίγο διαφορετικές καθώς χρησιμοποιούμε *Promises* και *Futures* για τα οποία δεν θα μπορούμε σε λεπτομέρειες αφού αφορούν την υλοποίηση της προγραμματιστικής διεπαφής.

Κύριο πλεονέκτημα αυτού του framework είναι ότι λόγω του ότι εκτελείται πάνω στο JVM είναι δυνατόν να χρησιμοποιήσει και πολλαπλά νήματα. Ο τρόπος που το πραγματοποιεί αυτό είναι με την έννοια του Verticle. Κάθε Verticle έχει το δικό του event loop και μπορεί να τρέχει σε ξεχωριστό νήμα αν υπάρχει διαθέσιμο στο σύστημα. Συνήθως υπάρχει ένα κύριο (main) Verticle το οποίο κάνει deploy τα υπόλοιπα.

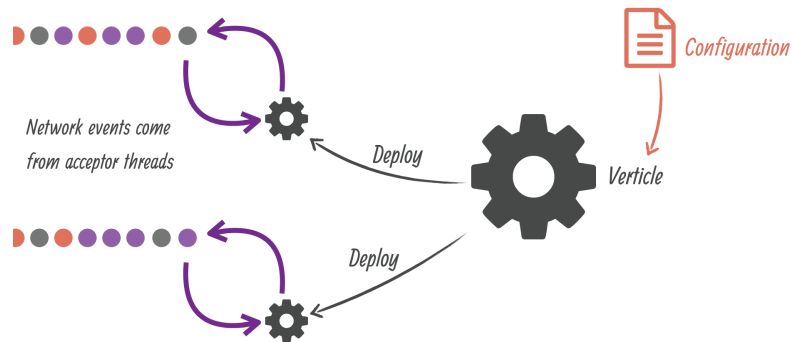


Figure 9: Vert.x verticles [14]

Τα verticles επικοινωνούν μεταξύ τους μέσω ανταλλαγής μηνυμάτων. Όλα τα μηνύματα περνούν από έναν διάδρομο γεγονότων (*event bus*) ο οποίος ελέγχεται από το κύριο Verticle. Ένα παράδειγμα φαίνεται στην παρακάτω εικόνα

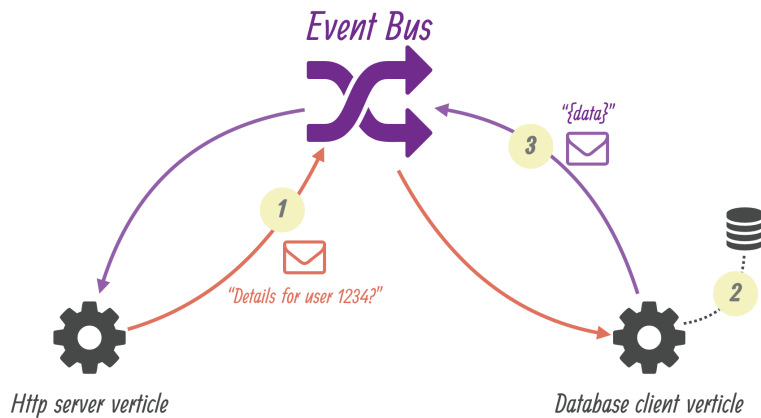


Figure 10: Vert.x event bus [14]

Το framework έχει ενσωματωμένη δυνατότητα δημιουργίας ενός http server όπου λειτουργεί με το ασύγχρονο μοντέλο βάση όλων των παραπάνω.

2.2. Σημασιολογικός Ιστός και Οντολογίες

Θα εξηγήσουμε τις βασικές έννοιες γύρω από τις Οντολογίες

2.2.1 Σημασιολογικός Ιστός

Ιστός των πραγμάτων (Web of things) ή *Ιστός των δεδομένων (Web of Data)*. Η αναζήτηση δεν βασίζεται σε αντιστοίχιση λέξεων αλλά σε σχετικά αντικείμενα και σχέσεις. Η επεξεργασία των δεδομένων δηλ. αναζήτηση και συμπερασμός (reasoning) γίνεται από μηχανές. Ο Σημασιολογικός Ιστός δεν είναι ένας ξεχωριστός Ιστός, αλλά μια επέκταση του τρέχοντος, στον οποίο οι πληροφορίες έχουν σαφώς καθορισμένη σημασία, επιτρέποντας στους υπολογιστές και τους ανθρώπους να συνεργάζονται καλύτερα. Τα πρώτα βήματα για την δόμηση του Σημασιολογικού Ιστού στη δομή του υπάρχοντος Ιστού βρίσκονται ήδη σε εξέλιξη. Στο εγγύς μέλλον, αυτές οι εξελίξεις θα οδηγήσουν σε σημαντική νέα λειτουργικότητα καθώς οι μηχανές γίνονται πολύ καλύτερα προετοιμασμένες να επεξεργαστούν και να «κατανοήσουν» τα δεδομένα που, προς το παρόν, απλώς εμφανίζουν [15] . Η κοινοπραξία WorldWide Web (W3C) είναι μία διεθνής κοινότητα που αναπτύσσει τα *ανοικτά πρότυπα (open standards)* προκειμένου για να εξασφαλίσει τη μακροπρόθεσμη ανάπτυξη του Παγκόσμιου Ιστού (Web).

Η αρχιτεκτονική του σημασιολογικού ιστού απεικονίζεται στο παρακάτω σχήμα. Το πρώτο επίπεδο, URI και Unicode, ακολουθεί τα σημαντικά χαρακτηριστικά του υπάρχοντος WWW. Το Unicode είναι ένα πρότυπο κωδικοποίησης διεθνών συνόλων χαρακτήρων και επιτρέπει τη χρήση όλων των γλωσσών (γραφής και ανάγνωσης) στον ιστό χρησιμοποιώντας μια τυποποιημένη φόρμα. Το *URI (Uniform Resource Identifier)* είναι μια συμβολοσειρά τυποποιημένης φόρμας που επιτρέπει τον μοναδικό προσδιορισμό πόρων (π.χ. έγγραφα). Ένα υποσύνολο του URI είναι το *URL (Uniform Resource Locator)*, το οποίο περιέχει το μηχανισμό πρόσβασης και μια θέση (δικτύου) ενός εγγράφου - όπως <http://www.example.org/>. Ένα άλλο υποσύνολο του URI είναι το URN που επιτρέπει τον εντοπισμό ενός πόρου χωρίς να υποδηλώνει τη θέση του και τα μέσα για την πρόσβαση στην τιμή ή το αντικείμενο - ένα παράδειγμα είναι το <urn:isbn:0-123-45678-9>. Η χρήση του URI είναι σημαντική για ένα κατανεμημένο σύστημα Διαδικτύου καθώς παρέχει κατανοητό προσδιορισμό όλων των πόρων. Μια διεθνής παραλλαγή του URI είναι το *IRI (Internationalized Resource Identifier)* που επιτρέπει τη χρήση χαρακτήρων

Unicode στο αναγνωριστικό (Identifier) και για την οποία ορίζεται μια αντιστοίχιση στο URI.

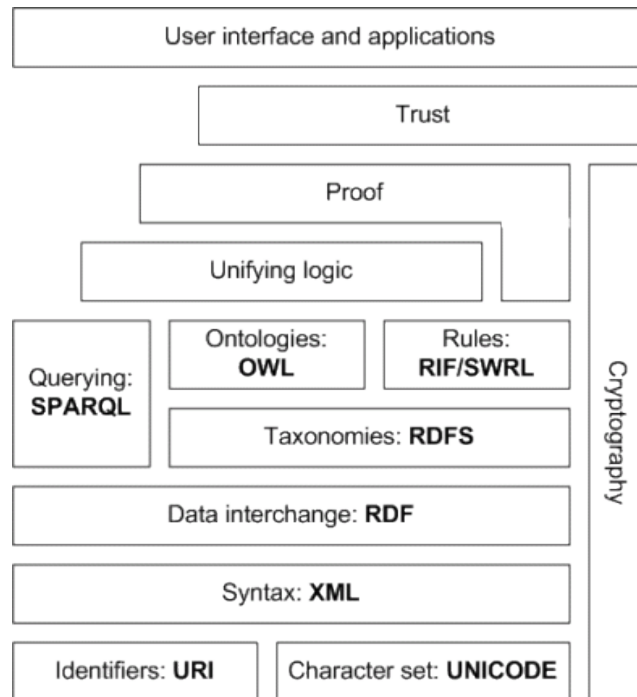


Figure 11: Αρχιτεκτονική Σημασιολογικού Ιστού

Το επίπεδο XML με τους ορισμούς ονόματος χώρου XML (*XML namespace*) και σχήματος XML (*XML schema*) διασφαλίζει ότι υπάρχει μια κοινή σύνταξη που χρησιμοποιείται στον σημασιολογικό ιστό. Το XML είναι μια γλώσσα σήμανσης (*markup*) γενικής χρήσης για έγγραφα που περιέχουν δομημένες πληροφορίες. Ένα έγγραφο XML περιέχει στοιχεία που μπορούν να εμφωλευθούν και μπορούν να έχουν χαρακτηριστικά και περιεχόμενο. Τα ονόματα χώρου XML επιτρέπουν τον καθορισμό διαφορετικών λεξιλογίων σήμανσης (*markup vocabularies*) σε ένα έγγραφο XML. Το σχήμα XML (*XML schema*) χρησιμεύει για την έκφραση σχήματος ενός συγκεκριμένου συνόλου εγγράφων XML.

Μια βασική μορφή αναπαράστασης δεδομένων για το σημασιολογικό Ιστό είναι το *RDF (Resource Description Framework)*. Το RDF είναι ένα πλαίσιο για την αναπαράσταση πληροφοριών σχετικά με τους πόρους σε μορφή γραφήματος. Προοριζόταν κυρίως για την αναπαράσταση μεταδεδομένων (*metadata*) σχετικά με τους πόρους του WWW, όπως ο τίτλος, ο συγγραφέας και η ημερομηνία

τροποποίησης μιας ιστοσελίδας, αλλά μπορεί να χρησιμοποιηθεί για την αποθήκευση οποιωνδήποτε άλλων δεδομένων. Βασίζεται σε τριάδες υποκείμενο-κατηγορήμα-αντικείμενο (subject-predicate-object) που σχηματίζουν γράφο δεδομένων. Όλα τα δεδομένα στον σημασιολογικό ιστό χρησιμοποιούν το RDF ως την κύρια γλώσσα αναπαράστασης. Η σύνταξη του RDF βασίζεται στην XML.

Από μόνο του το RDF χρησιμεύει ως περιγραφή ενός γραφήματος που σχηματίζεται από τριάδες. Ο καθένας μπορεί να ορίσει το λεξιλόγιο (vocabulary) των όρων που χρησιμοποιούνται για πιο λεπτομερή περιγραφή. Για να επιτραπεί η τυποποιημένη περιγραφή των *ταξινομιών (taxonomies)* και άλλων οντολογικών κατασκευών, δημιουργήθηκε ένα *Σχέδιο RDF (RDF Schema ή RDFS)* μαζί με την επίσημη σημασιολογία του στο RDF. Το RDFS μπορεί να χρησιμοποιηθεί για να περιγράψει τις ταξινομίες των κλάσεων (classes) και των ιδιοτήτων (properties) και να τις χρησιμοποιήσει για τη δημιουργία ελαφρών οντολογιών.

Μπορούμε να δημιουργήσουμε πιο λεπτομερείς οντολογίες με την *OWL (Web Ontology Language)*. Η OWL είναι μια γλώσσα περιγραφικής λογικής που συντακτικά ενσωματώνεται στο RDF όπως το RDFS, παρέχοντας επιπλέον τυποποιημένο λεξιλόγιο, και επίσημη σημασιολογία (formal semantics).

Τα RDFS και OWL έχουν καθορισμένη σημασιολογία και αυτή η σημασιολογία μπορεί να χρησιμοποιηθεί για διαδικασίες συλλογιστικής (reasoning) εντός οντολογιών και βάσεων γνώσεων που περιγράφονται χρησιμοποιώντας αυτές τις γλώσσες. Για την παροχή κανόνων πέρα από τις υφιστάμενες δομές αυτών των γλωσσών, έχουν αναδειχτεί δύο πρότυπα - RIF και SWRL.

Για την αναζήτηση δεδομένων RDF καθώς και RDFS και OWL οντολογιών σε βάσεις γνώσεων, υπάρχει η SPARQL (Simple Protocol and RDF Query Language). Η SPARQL είναι γλώσσα τύπου SQL, αλλά χρησιμοποιεί τριάδες (triples) και πόρους (resources) RDF τόσο για αντιστοίχιση τμήματος του ερωτήματος όσο και για την επιστροφή των αποτελεσμάτων του ερωτήματος. Η SPARQL δεν είναι μόνο γλώσσα ερωτημάτων (query language), αλλά είναι επίσης ένα πρωτόκολλο για την πρόσβαση σε δεδομένα RDF.

Όλη η σημασιολογία και οι κανόνες εκτελούνται στα επίπεδα κάτω από την Απόδειξη (Proof) και το αποτέλεσμα θα ελεγχθεί ως προς την αξιοπιστία του. Η επίσημη απόδειξη μαζί με αξιόπιστες εισόδους για την απόδειξη σημαίνει ότι τα αποτελέσματα μπορούν να είναι αξιόπιστα, οπότε εμφανίζονται στο επάνω επίπεδο του παραπάνω σχήματος. Για αξιόπιστες εισόδους, χρησιμοποιούνται μέσα κρυπτογράφησης, όπως *ψηφιακές υπογραφές (digital signatures)* για επαλήθευση της προέλευσης των πηγών. Πάνω από αυτά τα επίπεδα, μπορεί να δημιουργηθούν εφαρμογές με *διεπαφή χρήστη (user interface)*.

Ο Σημασιολογικός Ιστός βασίζεται σε ορισμένες τεχνολογίες.

- *Μεταδεδομένα (Metadata)* - Είναι δομημένες πληροφορίες που περιγράφουν, εξηγούν, εντοπίζουν ή διευκολύνουν με άλλο τρόπο την ανάκτηση, χρήση ή διαχείριση ενός πόρου πληροφοριών. Τα μεταδεδομένα ονομάζονται συχνά δεδομένα για τα δεδομένα ή πληροφορίες για τις πληροφορίες [16].

Τα μεταδεδομένα παρέχουν πληροφορίες που επιτρέπουν την κατανόηση των δεδομένων (π.χ. έγγραφα, εικόνες, σύνολα δεδομένων), έννοιες (π.χ. σχήματα ταξινόμησης) και οντότητες πραγματικού κόσμου (π.χ. άτομα, οργανισμοί, μέρη, πίνακες ζωγραφικής, προϊόντα).

Τύποι μεταδεδομένων:

- Περιγραφικά μεταδεδομένα, περιγράφουν ένα πόρο (resource) με σκοπό ανακάλυψης (discovery) και αναγνώρισης (identification).
- Δομικά μεταδεδομένα, π.χ. μοντέλα δεδομένων (datamodels) και δεδομένα αναφοράς (referencedata).
- Μεταδεδομένα διαχείρισης, παρέχουν πληροφορίες για τη διαχείριση ενός πόρου(resource).

Η διαχείριση των μεταδεδομένων πρέπει να διασφαλίζει:

- Διαθεσιμότητα (Availability): τα μεταδεδομένα πρέπει να είναι προσβάσιμα καταχωρημένα σε ευρετήρια ώστε να μπορούν να βρεθούν.

- Ποιότητα (Quality): τα μεταδεδομένα πρέπει να είναι σταθερής ποιότητας, ώστε οι χρήστες να γνωρίζουν ότι μπορούν να τα εμπιστευθούν.
- Ανθεκτικότητα (Persistence): τα μεταδεδομένα πρέπει να διατηρούνται με την πάροδο του χρόνου.
- Ανοιχτή άδεια χρήσης (OpenLicense): τα μεταδεδομένα πρέπει να είναι διαθέσιμα με άδεια Κοινού Κτήματος (PublicDomain) ώστε να επιτρέπουν την επαναχρησιμοποίησή τους.
- Ορολογίες (Terminologies) - παρέχουν κοινόχρηστα και κοινά λεξιλόγια (vocabularies) ενός πεδίου ορισμού (domain), ώστε οι μηχανές αναζήτησης (searchengines), οι πράκτορες (agents), οι συντάκτες (authors) και οι χρήστες (users) να μπορούν να επικοινωνούν. Όχι καλό, εκτός αν η κάθε μία σημαίνει το ίδιο πράγμα.
- Οντολογίες (Ontologies) - αποτελούν το δομικό στοιχείο του Σημασιολογικού Ιστού. Παρέχουν μια κοινή κατανόηση ενός πεδίου ορισμού (domain) που μπορεί να κοινοποιηθεί σε άτομα και εφαρμογές και θα διαδραματίσει σημαντικό ρόλο στην υποστήριξη της ανταλλαγής πληροφοριών και της ανακάλυψης. Εξετάζονται εκτενώς στο επόμενο κεφάλαιο.

2.2.2 Οντολογίες

Οι οντολογίες αναπτύχθηκαν στην Τεχνητή Νοημοσύνη (Artificial Intelligence) για να ικανοποιήσουν το διαμοιρασμό (sharing) και την επαναλαμβανόμενη χρήση (reuse) της γνώσης. Παρέχουν μία σημασιολογία επεξεργάσιμη από τη μηχανή (machine-processable semantics), πηγών πληροφοριών που μπορούν να είναι επικοινωνήσιμες μεταξύ διαφορετικών πρακτόρων (agents).

Μία οντολογία είναι μια κατηγορηματική (explicit), τυπική (formal), προδιαγραφή μιας διαμοιρασμένης (shared) εννοιολογικής αναπαράστασης (conceptualization), [17]. Ο όρος “κατηγορηματική” (explicit) σημαίνει ότι το είδος των εννοιών που χρησιμοποιούνται, και οι περιορισμοί που αφορούν την χρήση αυτών των εννοιών είναι προσδιορισμένοι με σαφήνεια. Ο όρος “τυπική” (formal) αναφέρεται στο ότι η

οντολογία πρέπει να είναι αναγνώσιμη από τη μηχανή. Ο όρος “διαμοιρασμένη” (shared) αναφέρεται στο ότι η οντολογία πρέπει να αποτυπώνει γνώση κοινής αποδοχής στα πλαίσια της κοινότητας. Τέλος, ο όρος “εννοιολογική αναπαράσταση” (conceptualization) αναφέρεται σε ένα αφηρημένο μοντέλο φαινομένων του κόσμου στο οποίο έχουν προσδιοριστεί οι έννοιες που σχετίζονται με τα φαινόμενα αυτά.

Στον παρακάτω πίνακα παρουσιάζονται τα βασικά συστατικά των οντολογιών:

Table 1: Συστατικά των οντολογιών

Κλάσεις (Classes)	Αναπαραστάσεις συλλογών πόρων (resources), που έχουν συγκεκριμένα, κοινά χαρακτηριστικά.
Άτομα (Individuals)	Στιγμιότυπα (instances) είναι τα μέλη μιας κλάσης.
Ιδιότητες (Attributes)	Οι ιδιότητες, χαρακτηριστικά ή παράμετροι των κλάσεων.
Σχέσεις (Relations)	Οι τρόποι με τους οποίους οι κλάσεις ή τα στιγμιότυπα σχετίζονται μεταξύ τους.
Συναρτήσεις (functions)	Μορφές σχέσεων, που βάσει κάποιων χαρακτηριστικών παράγουν κάποιο νέο χαρακτηριστικό.
Περιορισμοί (Restrictions)	Περιορισμοί στις σχέσεις και στις ιδιότητες των οντολογιών.
Κανόνες (Rules)	Δηλώσεις της μορφής Αν - Τότε .
Αξιώματα (Axioms)	Αναπαριστούν λογικές προτάσεις που θεωρούνται πάντα αληθείς.
Γεγονότα (Events)	Η αλλαγή ιδιοτήτων ή σχέσεων.

2.2.3 Δομή της OWL και βιβλιοθήκη Owlapi

Υπάρχουν αρκετές γλώσσες περιγραφής οντολογιών. Εμείς θα ασχοληθούμε με Οντολογίες που έχουν οριστεί με την γλώσσα *OWL (Web Ontology Language)*. Η OWL είναι μια οντολογική γλώσσα σχεδιασμένη για τον *Σημασιολογικό Ιστό (SemanticWeb)*. Παρέχει μια πλούσια συλλογή τελεστών (*operators*) για τη διαμόρφωση *περιγραφικών εννοιών (concept descriptions)*. Πρόκειται για ένα πρότυπο W3C (*W3C standard*), που προωθεί τη *διαλειτουργικότητα (interoperation)* και το *διαμοιρασμό (sharing)* μεταξύ εφαρμογών και έχει σχεδιαστεί ώστε να είναι συμβατή με τα υπάρχοντα *πρότυπα ιστού (web standards)*.

Όπως έχουμε αναφέρει η σύνταξή της είναι μια επέκταση του RDF και του RDFS τα οποία συντάσσονται με XML με τα παρακάτω βασικά στοιχεία.

Table 2: RDF Στοιχεία Σύνταξης βασισμένα στην XML

rdf:RDF	Ριζικό στοιχείο (rootelement) των εγγράφων RDF. Ορίζει το έγγραφο XML ως έγγραφο RDF. Περιέχει επίσης μια αναφορά στον χώρο ονομάτων RDF (RDF namespace).
rdf:Description	Στοιχείο που περιέχει την περιγραφή του πόρου (resource).
rdf:type	Στιγμιότυπο (instance) του ...
rdf:Bag	Ένα μη ταξινομημένο σύνολο πραγμάτων (container) πόρων (resources). Ενδεχομένως να περιέχει και διπλές τιμές.
rdf:Seq	Ένα ταξινομημένο σύνολο πόρων (resources). Ενδεχομένως να περιέχει και διπλές τιμές.
rdf:Alt	Ορίζει ένα σύνολο εναλλακτικών πόρων από τις οποίες ο χρήστης μπορεί να επιλέξει μόνο μία.
rdf:ID	Υποδεικνύει ένα νέο πόρο (resource).
rdf:about	Αναφορά σε ένα υπάρχοντα πόρο (resource).
rdf:resource	Επιτρέπει σε στοιχεία ιδιοτήτων (propertyelements) να οριστούν

	ως πόροι (resources).
--	-----------------------

Η επέκταση των συντάξεων των XML, RDF και RDFS που ορίζει η OWL γίνεται αντιληπτή από την κεφαλίδα των OWL αρχείων που έχουν την εξής μορφή

```
<rdf:RDF
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
```

Η owl ορίζει 3 υπογλώσσες τις OWL Lite, OWL DL, OWL Full κάθε μια με αυξημένες δυνατότητες σε εκφραστικότητα σε σχέση με την προηγούμενη.

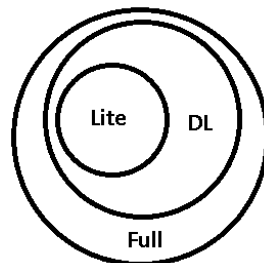


Figure 12: OWL sublanguages

Με το λεξιλόγιου του RDFS μπορούμε να περιγράψουμε κλάσεις (classes) και ιδιότητες (properties). Ωστόσο με την OWL μπορούμε επίσης να περιγράψουμε τις σχέσεις μεταξύ των κλάσεων και των ιδιοτήτων καθώς και τα χαρακτηριστικά τους. Οι κλάσεις και οι ιδιότητες μπορούν να έχουν και σχολιασμούς (annotations).

Η owlapi [18] είναι μια βιβλιοθήκη για java όπου χρησιμοποιείται στην επεξεργασία και την ανάγνωση OWL αρχείων. Προσφέρει διάφορες δυνατότητες και προγραμματιστικές διεπαφές σχετικές με οντολογίες. Εμείς θα επικεντρωθούμε μόνο στο διάβασμα για να μπορούμε να πάρουμε τις κλάσεις και τις ιδιότητες ενός OWL αρχείου μαζί με την επιπλέον πληροφορία που μας ενδιαφέρει. Συγκεκριμένα

μιλάμε για τον OWL/XML αναλυτή (parser) που διαθέτει. Με αυτόν μπορούμε να διαβάζουμε OWL αρχεία σε οποιαδήποτε νόμιμη μορφή έχουν αυτά γραφεί. Μπορούμε να αναλύουμε έτσι τα αξιώματα (axioms) που έχουν οριστεί και να χειριζόμαστε προγραμματιστικά την πληροφορία που μας ενδιαφέρει.

2.3. Καθορισμός Συσχετίσεων μεταξύ Οντολογιών

Έχουμε αναφέρει τα κύρια χαρακτηριστικά των Οντολογιών και συγκεκριμένα αυτών που ορίζονται με την OWL. Κύριος σκοπός της διπλωματικής αυτής είναι να μελετήσουμε τις συσχετίσεις που έχουν μεταξύ τους διαφορετικές τέτοιες οντολογίες. Για αυτόν τον λόγο βασιζόμαστε στο Ontology Alignment Tool [1]. Το εργαλείο αυτό δέχεται 2 οντολογίες owl και παράγει κανόνες αντιστοίχισης (*mapping rules*) τους οποίους εμείς θα χρησιμοποιήσουμε. Οι κανόνες αυτοί προκύπτουν με αυτοματοποιημένο τρόπο, αλλά οι χρήστες του εργαλείου έχουν την δυνατότητα να τους επεξεργαστούν και να προσθαφαιρέσουν κανόνες.

Σε κάθε κανόνα για να αντιστοιχιστούν τα στοιχεία (π.χ. κλάσεις, ιδιότητες, κτλ) της μιας οντολογίας με της άλλης, βασιζόμαστε στην έννοια των *μοτίβων οντολογιών* (*Ontologies Patterns*). Ένα τέτοιο μοτίβο περιγράφει επακριβώς τις παραμέτρους της οντολογίας που συμμετέχει στον κανόνα. Μπορεί να αναφέρεται σε ένα υπάρχων στοιχείο μιας οντολογίας (π.χ. μια κλάση), μπορεί να προσθέτει ένα νέο στοιχείο (π.χ. μια κλάση που θα έχει ρόλο domain σε μια ιδιότητα) ή γενικά να περιέχει έναν οποιοδήποτε συνδυασμών στοιχείων μιας οντολογίας. Κάποιο τέτοιο μοτίβο μπορεί να αποτελείται από άλλα τέτοια μοτίβα. Ορίζονται πολλά τέτοια μοτίβα ανάλογα με την φύση του κανόνα. Για παράδειγμα αναφέρουμε δύο τέτοια:

- Simple Relation Pattern

Το οποίο ορίζει μια σχέση (Relation) ενός συγκεκριμένου Object Property μιας οντολογίας.

- Relation Path Pattern

Με το οποίο ορίζεται μια *Relation* όπου σχηματίζεται από ένα ή περισσότερα *Properties* μιας οντολογίας.

Ο κάθε κανόνας που αντιστοιχίζει δύο οντολογίες έχει συγκεκριμένη μορφή. Κάποια πεδία είναι υποχρεωτικά και περιλαμβάνονται πάντα, ενώ άλλα είναι προαιρετικά και μπορεί να παραλείπονται. Υποχρεωτικά πεδία είναι

- Τα πεδία *entity 1* και *entity 2* τα οποία περιέχουν τα *Ontologies Patterns* που ορίσαμε πιο πάνω για την πρώτη και δεύτερη οντολογία αντίστοιχα. Επίσης πάντα έχει ένα πεδίο *Relation* που
- Το πεδίο *Relation* όπου καθορίζει την συσχέτιση των στοιχείων της πρώτης οντολογίας με την δεύτερη (π.χ. σχέση ισοδυναμίας).

Ορίζονται επίσης τα εξής προαιρετικά πεδία

- Το πεδίο *Transformation* όπου ορίζει κάποιον μετασχηματισμό που λαμβάνει μέρος στις τιμές των ιδιοτήτων της οντολογίας που συμμετέχουν στον κανόνα. Αυτό μπορεί να έχει την μορφή *directTransformation* ή *inverseTransformation* ανάλογα με το είδος του μετασχηματισμού.
- Το πεδίο *Direction* όπου δείχνει για ποια κατεύθυνση (από την 1η στην 2η ή το αντίστροφο) ισχύει η συσχέτιση του κανόνα.
- Το πεδίο *Origin* όπου δείχνει την προέλευση του κανόνα.
- Το πεδίο *Confidence Value* όπου δείχνει το πόσο σίγουρο ήταν το εργαλείο που για την ισχύ του κανόνα που παρήγαγε.
- Το πεδίο *Comments* όπου έχει την περιγραφή του συγκεκριμένου κανόνα.

Αυτά τα βλέπουμε και στον παρακάτω πίνακα που έχει οριστεί από τους δημιουργούς του εργαλείου, όπου στην 2η στήλη το γράμμα Υ δηλώνει υποχρεωτικό πεδίο ενώ το γράμμα Π προαιρετικό:

Table 3: Οι (Υ)ποχρεωτικές και (Π)προαιρετικές παράμετροι ενός Κανόνα Αντιστοίχισης

Παράμετρος	Υ/Π	Περιγραφή
------------	-----	-----------

Entity 1 and 2	Υ	Προσδιορίζει τα συμμετέχοντα στοιχεία της αριστερής και της δεξιάς πλευράς ενός κανόνα αντιστοίχισης
Transformation	Π	Προσδιορίζει τον μετασχηματισμό που λαμβάνει μέρος στις τιμές των ιδιοτήτων (properties) που ορίζονται
Relation	Υ	Προσδιορίζει την σχέση της Entity 1 ως προς την Entity 2 (π.χ. ισοδύναμοι όροι)
Direction	Π	Προσδιορίζει την κατεύθυνση για την οποία η συσχέτιση είναι έγκυρη
Origin	Π	Δείχνει τον τρόπο με τον οποίον ο κανόνας αντιστοίχισης έχει δημιουργηθεί (π.χ. χειροκίνητα από τον χρήστη μέσω του εργαλείου ΟΑΤ)
Confidence Value	Π	Δείχνει πόσο σίγουρο ήταν το ΟΑΤ για τον προτεινόμενο κανόνα αντιστοίχισης όταν αυτός έγινε αποδεκτός από τον χρήστη
Comments	Π	Παρέχει μια περιγραφή αναγνώσιμη από τον άνθρωπο του κανόνα αντιστοίχισης που προσδιορίζεται

Εμείς θα χρησιμοποιούμε τα αποτελέσματα αυτού του εργαλείου τα οποία θα είναι οι παραγόμενοι κανόνες συσχέτισης όπως τους περιγράψαμε παραπάνω σε ένα αρχείο JSON.

Ως πρώτο παράδειγμα δείχνουμε έναν τέτοιο κανόνα που υποδεικνύει μια ισοδυναμία μιας κλάσης της πρώτης οντολογίας με μια κλάση της δεύτερης.

```
{
  "entity1": {
    "pid": "SimpleClassPattern",
    "classuri":
      "http://www.semanticweb.org/ntua/iccs/harmonicss/cohort#Domain-002-Term-001",
    "classname": "hispanic"
  },
  "entity2": {
    "pid": "SimpleClassPattern",
    "classuri":
      "http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#ETHN-01",
    "classname": "Latin"
  },
  "directTransformation": null,
  "inverseTransformation": null,
  "relation": "Equivalent",
  "direction": "",
  "comments": ""
}
```

Βλέπουμε ότι ορίζεται το συγκεκριμένο Ontology Pattern για κάθε στοιχείο της κάθε οντολογίας όπως το έχουμε περιγράψει. Στο πεδίο Relation φαίνεται η σχέση ισοδυναμίας (Equivalent) των κλάσεων που ορίστηκαν για την Entity 1 και Entity 2.

Ως δεύτερο παράδειγμα δείχνουμε έναν πιο σύνθετο κανόνα

```
{
  "entity1": {
    "pid": "PropertiesCollectionPattern",
    "proparray": [{
      "pid": "SimplePropertyPattern",
      "propertyuri":
"http://www.semanticweb.org/ntua/iccs/harmonicss/cohort#Parameter-006",
      "propertyname": "CONSTITUTIONAL DOMAIN ACTIVITY",
      "index": 0,
      "valuerange":
"http://www.semanticweb.org/ntua/iccs/harmonicss/cohort#Domain-003"
    }]
  },
  "entity2": {
    "pid": "ClassWithPropertiesRestrictionPattern",
    "cls": {
      "pid": "SimpleClassPattern",
      "classuri":
"http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/reference-
model#ESSDAI-Domain-AL",
      "classname": "ESSDAI Domain Activity Level"
    },
    "proparray": [{
      "pid": "SimpleRelationPattern",
      "relationuri":
"http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/reference-
model#quest-ESSDAI-Domain-CV",
      "relationname": "ESSDAI Domain Coded Value",
      "index": 0
    }],
    {
      "pid": "SimpleRelationPattern",
      "relationuri":
"http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/reference-
model#activity-Level-CV",
      "relationname": "Activity Level Coded Value",
      "index": 1
    }
  ]
},
  "directTransformation": {
    "uri": "CLASS:
ntua.iccs.harmonicss.cohort.trans.quest.ESSDAIDomainActivityLevel",
    "arguments": [{
      "argname": "ESSDAI Domain",
```

```
    "argvalue":  
    "http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#ES  
SDAI-01"  
    }],  
    "description": "Provides the values of the parameters specified in the  
right side taking into account the patient data in the given fields as  
well as additional parameters provided."  
    },  
    "inverseTransformation": null,  
    "relation": "Linked With",  
    "direction": "From Ontology 1 to Ontology 2",  
    "comments": ""  
}
```

Όπως βλέπουμε αυτός ο κανόνας αντιστοιχίζει πολλές ιδιότητες (Properties) από την Entity 1 με μία κλάση και πολλές ιδιότητες από την Entity 2. Ο κανόνας ορίζει κάποιον μετασχηματισμό με το πεδίο `directTransformation`. Εκεί βλέπουμε ότι ορίζονται πολλές παράμετροι (arguments) όπου η κάθε μια έχει κάποιο όνομα και κάποια τιμή καθώς και μια περιγραφή του κανόνα. Η σχέση των στοιχείων των οντολογιών που συσχετίστηκαν ορίζεται ως σχέση σύνδεσης (Linked With) και βλέπουμε ότι ο κανόνας έχει ισχύ μόνο από την κατεύθυνση της πρώτης οντολογίας προς την δεύτερη.

3. Περιγραφή Συστήματος

Εδώ θα περιγράψουμε το σύστημα που δημιουργήσαμε για την αναπαράσταση των κανόνων συσχετίσεων (Mapping Rules) και τις οντολογίες (Ontologies) που συνδέουν όπως εξηγήθηκαν στην προηγούμενη ενότητα. Το σύστημα που φτιάξαμε το ονομάσαμε *OMV (Ontologies Mapping Visualizer)* και τα χαρακτηριστικά του περιγράφονται στις επόμενες ενότητες.

3.1. Παρεχόμενες Υπηρεσίες

Σκοπός της εφαρμογής είναι η οπτική αναπαράσταση των κανόνων αντιστοίχισης δύο διαφορετικών Οντολογιών. Δεδομένου του ότι η πληροφορία που περιέχεται στις Οντολογίες καθώς και στην αντιστοίχιση είναι αρκετά μεγάλη, δεν είναι δυνατόν να δειχτεί με μια απλή όψη. Για αυτό τον λόγο δημιουργούμε έναν αλληλεπιδραστικό γραφικό περιβάλλον (GUI) το οποίο δίνει την δυνατότητα στον χρήστη να αλληλεπιδρά με τα διάφορα γραφικά αντικείμενα και να βλέπει περισσότερες πληροφορίες ανάλογα με την εστίαση του ενδιαφέροντός του. Οι βασικές χρήσεις που μπορεί να αξιοποιήσει ο χρήστης φαίνονται πολύ συνοπτικά στο παρακάτω *UML διάγραμμα Περιπτώσεων Χρήσης (UML Use Case Diagram)*.



Θα περιγράψουμε με λίγα λόγια τις λειτουργίες αυτές οι οποίες θα δειχθούν πιο διεξοδικά με παραδείγματα στο Κεφάλαιο 4:

- Βασική λειτουργία είναι η δημιουργία μια νέας οπτικοποίησης που την ονομάζουμε Visualization. Για αυτή την λειτουργία το σύστημα μας διαθέτει δύο αναλυτές (Parsers) υπό την γενική έννοια, οι οποίοι επεξεργάζονται δύο OWL αρχεία και ένα αρχείο JSON με τους Mapping Rules που συνδέουν τις οντολογίες αυτές.
- Από το Visualization που έχουμε δημιουργήσει ο χρήστης μπορεί να δει τις πληροφορίες της κάθε OWL οντολογίας (ενέργεια Show OWL Info). Οι πληροφορίες που φαίνονται αφορούν τα βασικά στοιχεία της κάθε οντολογίας που είναι οι κλάσεις (Classes), ιδιότητες αντικειμένων (Object Properties), ιδιότητες δεδομένων (Data Properties), ιδιότητες σχολιασμών (Annotation Properties) και οι σχολιασμοί (Annotations) του κάθε στοιχείου.
- Από το Visualization ο χρήστης μπορεί να δει επίσης κάποια στατιστικά που αφορούν τον σύνολο των κανόνων που έχουν οριστεί και των στοιχείων που περιέχουν οι Οντολογίες. Επίσης φαίνονται στατιστικά για τους κανόνες που συμμετέχουν οι κλάσεις ανώτερων επιπέδων της δεύτερης Οντολογίας, επειδή πολλές φορές χρησιμοποιείται ως μοντέλο προς αναφορά (Reference Model).
- Τους κανόνες αντιστοίχισης (Mapping Rules) μπορούμε να τους δούμε στην εποπτική όψη (Eroptic View). Εκεί ο χρήστης μπορεί να πάρει μια γενική εικόνα για το ποια στοιχεία της κάθε οντολογίας συμμετέχει στους κανόνες και να αλληλεπιδράσει μαζί τους
- Επίσης στην όψη ανά κανόνα (By Rule View) ο χρήστης μπορεί να δει τον κάθε κανόνα ξεχωριστά μαζί με τα στοιχεία κάθε οντολογίας που συνδέει και όλες τις λεπτομέρειες που ορίζονται σε αυτόν.
- Η τελευταία σημαντική χρήση της εφαρμογής μας είναι η αλλαγή των ρυθμίσεων για την προσαρμογή του περιβάλλοντος στις ανάγκες του χρήστη.

Οι δυνατές αλλαγές στην παρούσα έκδοση είναι περιορισμένες και αφορούν την λειτουργική χρήση του συστήματός μας, αλλά στο μέλλον μπορούν να προστεθούν κι άλλες όπως η προσαρμογή της εμφάνισης του περιβάλλοντος.

3.2. Αρχιτεκτονική Εφαρμογής

Εδώ θα μιλήσουμε για την αρχιτεκτονική του συστήματός μας. Το μοντέλο που ακολουθούμε είναι το λεγόμενο μοντέλο Πελάτη – Εξυπηρετητή (Client – Server model). Σε αυτό, ο χρήστης αλληλεπιδρά με την εφαρμογή Client ή οποία λέγεται και front end. Αυτή η εφαρμογή επικοινωνεί μέσω αιτημάτων με έναν άλλον υπολογιστή ο οποίος απαντά αυτά τα ερωτήματα και λέγεται Server ή back end.

3.2.1 Για τον Client

Ο client υλοποιείται ως μια διαδικτυακή εφαρμογή με το client side framework VueJs. Αυτό μας επιτρέπει να λειτουργεί ανεξάρτητα από τον server. Η διανομή του client μπορεί να γίνει από έναν αδύναμο και οικονομικό εξυπηρετητή στατικών σελίδων ή από προσαρμογή του server μας ώστε να προσφέρει και στατικές σελίδες εκτός από το Api. Ο κώδικας του client γράφεται ανεξαρτήτως του server με μόνη υποχρέωση να γνωρίζει το api και να ακολουθεί τις όποιες αλλαγές του ώστε να μπορεί να επικοινωνεί χωρίς προβλήματα με τον server.

Ο client στέλνει πάντα αιτήματα πρωτοκόλλου http (http requests). Τα αιτήματα αυτά χρησιμοποιούν κυρίως τις μεθόδους GET, POST, PUT, DELETE του http που έχουμε αναφέρει στο κεφάλαιο 2. Το σώμα των μηνυμάτων (body) κωδικοποιείται σε JSON όπου αυτό είναι δυνατόν. Στην περίπτωση που χρειαζόμαστε να στείλουμε αρχεία, τότε αυτά κωδικοποιούνται με την δυαδική μορφοποίηση form-data. Σε αυτή την επιλογή δεν κωδικοποιούνται χαρακτήρες, αλλά στέλνονται αυτούσια τα αρχεία ως δυαδικά, ώστε να μπορεί να τα διαβάσει και να τα επεξεργαστεί ο server. Να σημειωθεί ότι όταν στέλνονται αρχεία πρέπει υποχρεωτικά να χρησιμοποιηθεί η μέθοδος POST.

3.2.2 Για τον Server

Ο server υλοποιείται ως μια ασύγχρονη εφαρμογή σε java με το framework Vert.x. Σκοπός του είναι η δημιουργία ενός http server που εξυπηρετεί τα αιτήματα του Client. Έχει την δυνατότητα να προσφέρει και στατικές σελίδες για την περίπτωση που θέλουμε να ενσωματώσουμε τον client μέσω στον server και να διανέμεται μέσω αυτού. Ωστόσο εμείς θα επικεντρωθούμε στην βασική λειτουργία του όπου είναι αυτή μιας διεπαφής για χρήση από τον client. Αυτή η διεπαφή λέμε ότι είναι ένα REST api. Ο όρος *REST (Representational State Transfer)* είναι κάποιες προδιαγραφές και παραδοχές για την δημιουργία ενός client-server συστήματος, ο ορισμός του οποίου μπορεί να βρεθεί στο διαδίκτυο [19]. Στις μέρες μας πολλές από αυτές τις προδιαγραφές θεωρούνται πολλές φορές ξεπερασμένες και έτσι γίνεται διαχωρισμός σε συστήματα RESTful όπου τηρούν πλήρως τις προδιαγραφές και απλά REST όπου τηρούν κάποιες από αυτές. Εμείς θα αναφέρουμε ακριβώς ποιες προδιαγραφές τηρεί ο server μας, κάποιες από τις οποίες στηρίζονται στο REST, ενώ κάποιες άλλες είναι δικές μας.

- Μοντέλο client-server

Όπως έχουμε πει η υλοποίηση του client και του server είναι διαχωρισμένες με τον server να μην γνωρίζει λεπτομέρειες για την εμφάνιση και την απεικόνιση του γραφικού περιβάλλοντος.

- Http Server

Ο server μας υποστηρίζει το πρωτόκολλο http και απαντάει http requests. Σαφώς αναλόγως με την παράταξη (deployment) του server χρησιμοποιείται το https που διασφαλίζει την ασφαλή σύνδεση http.

- JSON respond

Ο server μας πάντα απαντάει με σώμα (body) μορφής JSON. Αυτό μας επιτρέπει εύκολο χειρισμό της απάντησης (response) από τον client με όμοιο τρόπο για διαφορετικά requests.

- Cache (προσωρινή μνήμη)

Ο server μας διαθέτει τρόπους χρησιμοποίησης προσωρινής μνήμης (caching) για να απαντάει πιο αποδοτικά συνεχόμενα όμοια requests.

- Stateless

Τέλος η πιο σημαντική προδιαγραφή είναι ότι ο server μας δεν κρατάει πληροφορίες κατάστασης για τον client ή είναι όπως λέμε Stateless. Αυτό πρακτικά σημαίνει ότι κάθε request πρέπει να έχει όλη την απαραίτητη πληροφορία για να απαντηθεί. Δεν επιτρέπεται να έχει κρατήσει ο server πληροφορία από προηγούμενο request ενός client ώστε να απαντήσει κάποιο επόμενο. Επίσης δεν επιτρέπονται διάφοροι μηχανισμοί για sessions αφού χρειάζονται ο server να κρατάει πληροφορία για κάποιον συγκεκριμένο client. Αυτή η προϋπόθεση είναι από τις βασικές του REST και αν μπορεί να φαίνεται ότι οδηγεί στο να στέλνεται περίσσεια πληροφορία σε πολλά requests, τελικά οδηγεί σε πιο σταθερά συστήματα με αποφυγή πολλών λαθών εξαιτίας περίπλοκων δομών.

3.3. Υλοποίηση Επιμέρους Συστημάτων

Εδώ θα μιλήσουμε για τα επιμέρους υποσυστήματα και εξαρτήματα (components) σε γενικευμένη μορφή με τα οποία έχει δημιουργηθεί το σύστημά μας. Για κάθε τέτοιο υποσύστημα θα αναφέρουμε τις τεχνολογίες με τις οποίες έχει δημιουργηθεί. Τα υποσυστήματα φαίνονται στο παρακάτω γενικευμένο σχεδιάγραμμα, όπου τα βελάκια δείχνουν την σύνδεση ή εξάρτηση των εξαρτημάτων. Οι διακεκομμένες γραμμές και τα διακεκομμένα κουτάκια δηλώνουν υπο εξαρτήματα που περιλαμβάνονται σε αυτά που τα συνδέει αλλά η σημαντικότητά τους τα κάνει ενδιαφέροντα να αναφερθούν ξεχωριστά.

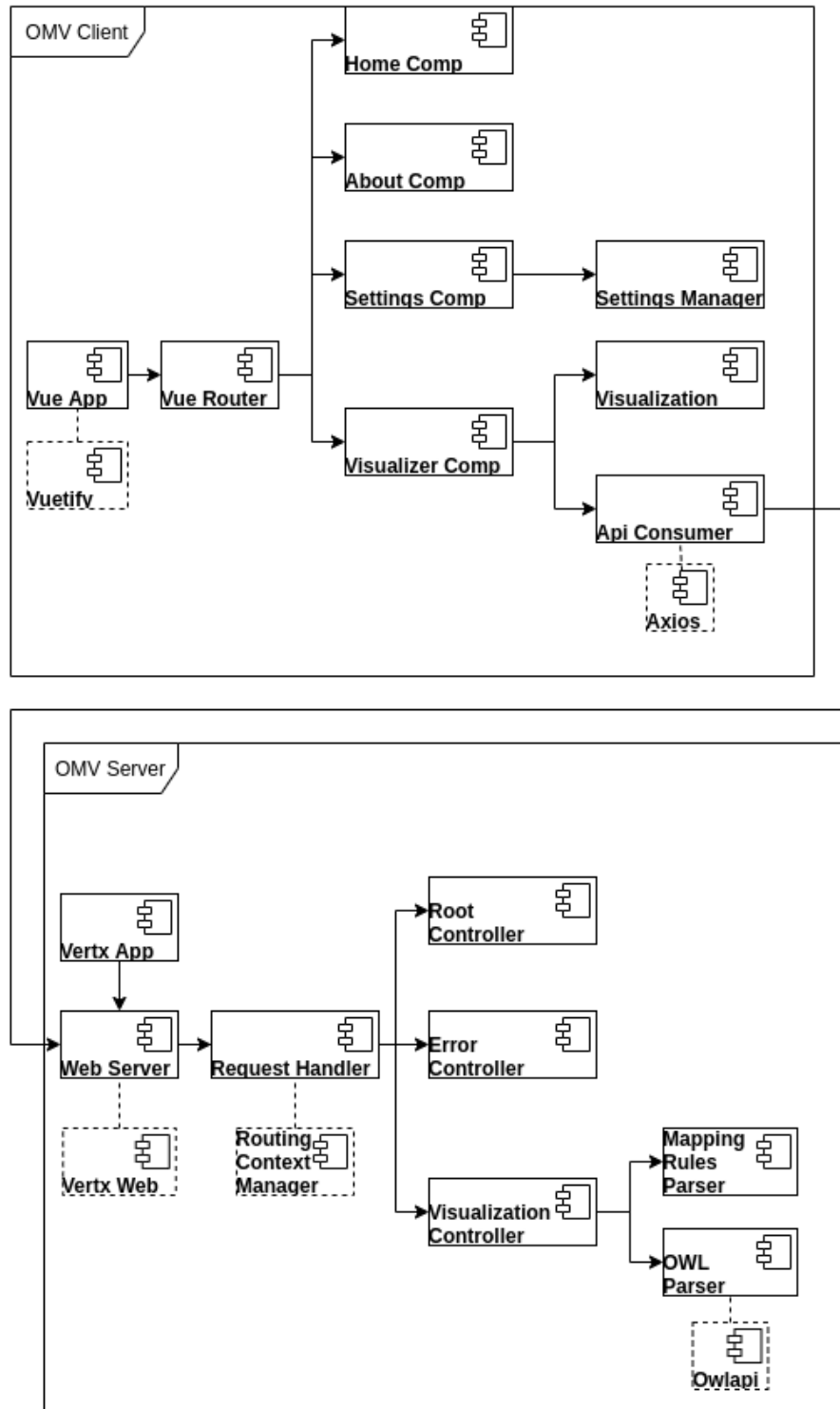


Figure 13: Επιμέρους Συστήματα Εφαρμογής

3.3.1 Συστήματα Client

Ο client μας υλοποιείται με διαδικτυακές τεχνολογίες και το client-side framework VueJS όπως τα έχουμε περιγράψει στο κεφάλαιο 2. Η γλώσσα που χρησιμοποιούμε για τα στοιχεία που αλληλεπιδρούν με την VueJS είναι η javascript. Ωστόσο, για δικές μας κλάσεις που έχουν να κάνουν με τα δεδομένα χρησιμοποιούμε την γλώσσα Typescript. Η typescript είναι μια γλώσσα βασισμένη σε javascript όπου προσφέρει κάποιες παραπάνω δυνατότητες όπως ο ορισμός τύπων μεταβλητών με σκοπό την διευκόλυνση του προγραμματιστή. Η typescript μεταφράζεται σε javascript την ώρα του χτισίματος της εφαρμογής (building) από τον compiler της (πιο σωστά λέγεται και transpiler αφού μεταφράζει μια γλώσσα σε κάποια άλλη και όχι σε binary code). Άρα τελικά το περιβάλλον εκτέλεσης της (runtime) είναι ακριβώς το ίδιο με αυτό της javascript και ισχύουν όλα αυτά που έχουμε αναλύσει στο Κεφάλαιο 2.

Vue App

Για να τρέξει η εφαρμογή μας θέλουμε να φορτωθεί το αρχικό html αρχείο που έχει όνομα index.html. Για αυτό τον λόγο ρυθμίζουμε πάντα τον στατικό server που διανέμει τον client μας να ανακατευθύνει πάντα όλα τα requests σελίδας του browser σε αυτό το αρχείο. Για να τρέξει το Vue application χρειαζόμαστε ένα αρχικό vue component το οποίο το λέμε και component ρίζας (root component). Αυτό τοποθετείται (mount) σε ένα html element που υπάρχει στο index.html μέσω javascript και αρχίζει η εφαρμογή μας. Πλέον μπορούν να δημιουργηθούν άλλα components και να εκμεταλλευτούμε όλες τις δυνατότητες της VueJS που έχουμε αναλύσει στο κεφάλαιο 2.

Vuetify

Αξίζει να αναφερθεί ξεχωριστά το Vuetify το οποίο είναι ένα framework βασισμένο σε VueJS που υλοποιεί το Material Design. Το Material Design προσδιορίζει κάποιες βασικές προδιαγραφές που δημιουργήθηκαν από την Google για την κατασκευή γραφικών εφαρμογών με τέτοιο τρόπο ώστε να είναι ευανάγνωστες από τον χρήστη καθώς και να μπορούν να χρησιμοποιηθούν εύκολα από οθόνες αφής. Το vuetify μας προσφέρει έτοιμα components που τηρούν αυτές τις προδιαγραφές και μπορούμε να τα χρησιμοποιήσουμε και να τα ελέγξουμε μέσω των τρόπων που

μας επιτρέπει η VueJS, όπως μέσω ιδιοτήτων (props). Επίσης μπορούμε να τα επεκτείνουμε και να δημιουργήσουμε δικά μας components βασισμένα στα components του vuexify ώστε να μπορούμε να επαναχρησιμοποιούμε λειτουργίες τους χωρίς να τις υλοποιούμε εξ αρχής. Όλα τα components που έχουμε φτιάξει στην εφαρμογή μας χρησιμοποιούν vuexify components.

Vue Router

Το VueJS μας επιτρέπει να επεξεργαζόμαστε τις σελίδες με προηγμένο τρόπο μέσω javascript. Ωστόσο, είναι πολλές φορές θεμιτό να υπάρχει μια περιήγηση στις διάφορες υποσελίδες μέσω ενός URL που δίνουμε στον browser (π.χ. για να κάνουμε bookmark μια συγκεκριμένη υποσελίδα). Η VueJS διαθέτει ένα πρόσθετο (plugin) που ονομάζεται vue-router. Αυτό εκμεταλλεύεται το HISTORY api της html που της επιτρέπει να ελέγχει και να διαβάσει το τρέχον URL που υπάρχει στον browser. Έτσι, μπορούμε να ορίσουμε διαδρομές που μεταφερόμαστε μέσω του URL παρόλο που δεν γίνονται αιτήματα σε κάποιον server για νέες σελίδες αλλά όλα τρέχουν μέσω της αρχικής index.html. Για να μπορεί να λειτουργήσει σωστά το vue router χρειάζεται η ανακατεύθυνση που περιγράψαμε στο Vue App component ώστε όταν για παράδειγμα δίνουμε απευθείας κάποιο URL στον browser μας, ο στατικός server να μας στέλνει στο index.html και από εκεί αφού φορτωθεί η εφαρμογή μας να μπορεί να μας στείλει το vue router στην υποσελίδα που θέλουμε. Οι υποσελίδες που χρησιμοποιούμε στην εφαρμογή μας είναι οι */home*, */about*, */settings*, */visualizer*, κάθε μία από τις οποίες ορίζεται με έναν καινούριο vue component.

Home Comp

Είναι ένα Vue component που φτιάχνει την υποσελίδα */home*. Περιέχει σύντομη περιγραφή της εφαρμογής και κάποια links χωρίς να αλληλεπιδρά με κάποιο άλλο σύστημα.

About Comp

Είναι ένα Vue component που φτιάχνει την υποσελίδα */about*. Δείχνει σύντομες οδηγίες χρήσης, πληροφορίες για τον δημιουργό και πληροφορίες για την εφαρμογή (όπως π.χ. η έκδοση) οι οποίες ορίζονται είτε κατά την ώρα του χτισίματος (building) ή από το περιβάλλον εκτέλεσης (runtime).

Settings Comp

Είναι ένα Vue component που φτιάχνει την υποσελίδα */settings*. Περιλαμβάνει κάποια πεδία για να μπορεί να αλλάξει ο χρήστης τις ρυθμίσεις τις εφαρμογής. Για την εφαρμογή των αλλαγών το σύστημα αλληλεπιδρά με το υποσύστημα Settings Manager.

Settings Manager

Είναι υπεύθυνος για το διάβασμα των ρυθμίσεων της εφαρμογής καθώς και για την εφαρμογή των αλλαγών τους. Προσφέρει δυνατότητα επαναφοράς (reset) των ρυθμίσεων στις προκαθορισμένες (default) που έχουν οριστεί από τον προγραμματιστή. Οι ρυθμίσεις που αλλάζουν αποθηκεύονται στο Local Storage, το οποίο είναι ένας μηχανισμός που προσφέρει το περιβάλλον διαδίκτυακών εφαρμογών για αποθηκευτικό χώρο των διάφορων ιστοσελίδων. Το Local Storage ελέγχεται μέσω javascript και βρίσκεται τοπικά σε κάθε browser που επισκέπτεται την εφαρμογή.

Visualizer Comp

Είναι ένα Vue component που φτιάχνει την υποσελίδα */visualizer*. Εκεί ο χρήστης μπορεί να επιλέξει τα αρχεία OWL και το αρχείο JSON με τα Mapping Rules για να δημιουργήσει μια νέα οπτικοποίηση (Visualization). Αλληλεπιδρά με τα υποσυστήματα Api Consumer και Visualization. Συγκεκριμένα καλεί την συνάρτηση του Api Consumer ώστε να πάρει κάποιο αποτέλεσμα από τον server και στην συνέχεια το περνάει στο υποσύστημα Visualization για να γίνει η οπτικοποίηση που θέλουμε.

Api Consumer

Είναι το υποσύστημα που αναγνωρίζει το Api του server για να του στέλνει και να λαμβάνει δεδομένα. Ξέρει για κάθε αντικείμενο που θέλουμε να πάρουμε ποια δεδομένα χρειάζεται να στείλουμε, σε ποια κωδικοποίηση και ποια άκρη (endpoint) του server θα καλέσουμε. Χρησιμοποιεί την βιβλιοθήκη Axios.

Axios

Είναι η βιβλιοθήκη που χρησιμοποιείται από τον Api Consumer για να κάνει http requests. Λειτουργεί με τον τρόπο και τους μηχανισμούς που εξηγήσαμε στο

κεφάλαιο 2, δίνοντας την δυνατότητα στο σύστημα να προσαρμόζει διαφορετικά το γραφικό περιβάλλον ανάλογα με την επιτυχία ή όχι του αιτήματος.

Visualization

Δέχεται κάποιο αποτέλεσμα οπτικοποίησης το οποίο έχει ανακτηθεί από τον server. Σκοπός του είναι να δείξει πληροφορίες για τις οντολογίες και για τους κανόνες αντιστοίχισης που δόθηκαν. Διαθέτει tabs για την αλλαγή της όψης που θέλει να δει ο χρήστης και δημιουργεί αλληλεπιδραστικά components που αντιδρούν στα clicks του ποντικιού για να δείξουν περισσότερες πληροφορίες. Για την αναπαράσταση χρησιμοποιούνται ο διανυσματικός τύπος εικόνας SVG (Scalable Vector Graphics). Αυτός ο τύπος είναι βασισμένος σε XML και υποστηρίζεται από όλους τους σύγχρονους browsers. Με αυτόν φτιάχνουμε εικόνες σε δενδρική δομή για τα elements της κάθε οντολογίας καθώς και γραμμές και άλλα γραφικά στοιχεία για τους κανόνες αντιστοίχισης.

3.3.2 Συστήματα Server

Ο server ακολουθεί τις αρχές REST όπως τις έχουμε περιγράψει και υλοποιείται με την χρήση του Vert.x framework που έχουμε εξηγήσει στο κεφάλαιο 2.

Vertx App

Η εφαρμογή μας όλη τρέχει σε ένα κύριο Vertx Verticle προσφέροντάς μας την δυνατότητα να προγραμματίσουμε με ασύγχρονο τρόπο. Κύριος σκοπός είναι η δημιουργία ενός Web Server.

Web Server

Πρόκειται για έναν http server που μπορεί να δέχεται και να απαντάει http requests. Τρέχει σε κάποια προκαθορισμένη θύρα (port) η οποία μπορεί να αλλαχτεί από τον τρόπο εκκίνησης του server. Ο υλοποίησή του διατίθεται από το Vertx και αλληλεπιδρά με την επιπρόσθετη βιβλιοθήκη vertx-web για ευκολότερη χρήση του.

Vertx Web

Αν και το Vertx διαθέτει δυνατότητες δημιουργίας ενός http server, ο έλεγχός του προγραμματιστικά είναι λίγο δύσχρηστος. Για αυτόν τον λόγο υπάρχει η επιπλέον βιβλιοθήκη vertx-web η οποία δημιουργεί έναν δρομολογητή (router). Σε αυτόν μπορούμε να καθορίσουμε διαδρομές ανάλογα με την άκρη του server (endpoint)

στην οποία γίνεται κάποιο http request. Αλληλεπιδρά με το σύστημα Request Handler.

Request Handler

Για κάθε αίτημα που γίνεται σε ένα endpoint, ο router ορίζει μια συνάρτηση που θα κληθεί να εκτελεστεί. Αυτή λέγεται χειριστής αιτήματα (Request Handler) και είναι ξεχωριστή για κάθε αίτημα. Βασικό εργαλείο που χρησιμοποιεί είναι ο Routing Context Manager.

Routing Context Manager

Είναι ένα αντικείμενο που δημιουργείται από το router του vertx-web όταν γίνεται κάποιο request. Αυτό είναι υπεύθυνο για όλες τις πληροφορίες που σχετίζονται με το request. Συγκεκριμένα μπορούμε να διαβάσουμε από αυτό τα δεδομένα που έστειλα ο client αναλόγως με την μορφή τους. Τον χρησιμοποιούμε επίσης για να δημιουργήσουμε την απάντηση που θα στείλουμε πίσω στον client. Διαθέτει διάφορους μεθόδους για να ελέγχουμε αν η μορφοποίηση των δεδομένων που πήραμε είναι αυτή που θέλουμε καθώς και για να ανακατευθύνουμε το αίτημα σε κάποιον άλλον handler αν αυτό είναι επιθυμητό. Για να απαντήσουμε με το επιθυμητό αποτέλεσμα καλείται ο αντίστοιχος Controller που κάνει την επιθυμητή εργασία και επιστρέφει το αποτέλεσμα, πάντα σε μορφή JSON.

Root Controller

Καλείται από τον request handler του endpoint `/api`. Είναι το υποσύστημα που είναι υπεύθυνο για να δείξει κάποιες πληροφορίες σχετικά με τον server, όπως την έκδοσή του (version) και τον χρόνο λειτουργίας του (uptime).

Error Controller

Καλείται συνήθως ως ανακατεύθυνση από κάποιον handler ή κάποιον controller όταν υπάρχει κάποιο σφάλμα στην επιθυμητή εργασία. Η απάντησή του έχει κάποιο status code που είναι ένας αριθμός που δείχνει τον τύπο σφάλματος και ένα body με την περιγραφή του σφάλματος. Τα status code μαζί με την περιγραφή και το body τους που έχουμε ορίσει φαίνονται στον παρακάτω πίνακα.

Table 4: Server Error Status Codes

Status	Body	Περιγραφή
550	Το μήνυμα σφάλματος συστήματος. Εξαρτάται από το συγκεκριμένο σφάλμα.	Καλείται όταν υπάρχει κάποιο σφάλματος συστήματος.
500	Το μήνυμα σφάλματος συστήματος. Εξαρτάται από το συγκεκριμένο σφάλμα.	Καλείται όταν υπάρχει κάποιο σφάλματος συστήματος για το οποίο δεν έχουμε καθορίσει τρόπο να αντιμετωπιστεί
422	Προσαρμοσμένο μήνυμα ανάλογα με την περίπτωση σφάλματος στην χρήση της εφαρμογής.	Είναι σφάλμα που το ορίζουμε εμείς και στέλνεται όταν ο χρήστης έχει κάνει κάποιο λάθος στην λειτουργία όπως για παράδειγμα να στείλει δεδομένα λάθος οντολογιών.
405	"Requested URL exists, but method is not allowed"	Καλείται όταν η διαδρομή (endpoint) που δόθηκε υπάρχει αλλά δεν υποστηρίζεται η συγκεκριμένη HTTP μέθοδος
404	"Requested URL not found. All api URLs start with /api/"	Καλείται όταν η διαδρομή δεν υπάρχει.
403	"Permission Denied"	Καλείται όταν ο χρήστης δεν έχει δικαιώματα για κάποια λειτουργία
400	"Bad Request"	Καλείται αν υπάρχει η διαδρομή αλλά το απαιτούμενο body του request είναι άδαιο.

Visualization Controller

Καλείται από τον request handler του endpoint `/api/visualization`. Είναι υπεύθυνος για την δημιουργία μιας νέας οπτικοποίησης. Λαμβάνει τα αρχεία OWL και το JSON με τους Mapping Rules μορφοποιημένα ως form data. Θα τα αναλύσει και θα επιστρέψει ένα JSON response με όλη την πληροφορία που χρειάζεται ο client για να μπορεί να αναπαραστήσει. Έχει την δυνατότητα να λάβει μόνο ένα OWL αρχείο και Mapping Rules και να χρησιμοποιήσει ένα προκαθορισμένο αρχείο που έχουμε ορίσει ως Reference Model ως το δεύτερο. Για την άντληση της πληροφορίας αλληλεπιδρά με τα συστήματα Mapping Rules Parser και OWL Parser.

Mapping Rules Parser

Αυτό το σύστημα διαβάζει το JSON αρχείο που περιέχει τους Mapping Rules και οργανώνει την πληροφορία. Η ανάγνωση γίνεται μέσω του αντικειμένου JsonObject

που διαθέτει το vertx framework και μας επιτρέπει να καταλαβαίνουμε τα επιμέρους πεδία και να διαβάζουμε τις τιμές. Η πληροφορία οργανώνεται σε 2 λίστες από κανόνες, 1 για αυτούς που αφορούν μόνο κλάσεις και 1 για αυτούς που συμμετέχουν και ιδιότητες (Properties) οντολογιών. Ο κάθε κανόνας περιέχει όλη την πληροφορία για να μπορεί να αναπαρασταθεί από έναν client.

OWL Parser

Αυτό το σύστημα διαβάζει και αναλύει την πληροφορία που περιέχεται στα OWL αρχεία. Οργανώνει τα στοιχεία Classes, Object Properties, Data Properties σε μια δενδρική δομή για το καθένα όπου ξεκινάμε από το κορυφαίο στοιχείο και όλα τα άλλα έχουν σχέση πατέρα-παιδιών. Επίσης οργανώνουμε σε μια λίστα τα Annotation Properties που περιέχει η οντολογία. Για κάθε στοιχείο που διαβάσαμε βρίσκουμε και τα Annotations που το περιγράφουν. Βρίσκουμε επιπλέον και άλλες πληροφορίες που περιγράφουν την οντολογία μας. Χρησιμοποιεί την βιβλιοθήκη Owlapi.

Owlapi

Έχουμε εξηγήσει τις δυνατότητες αυτής της βιβλιοθήκης στο κεφάλαιο 2.

Χρησιμοποιούμε μόνο τις δυνατότητες ανάγνωσης αυτής ώστε να ανακτήσουμε τις πληροφορίες που χρειαζόμαστε για τις οντολογίες.

3.4. Αλληλεπίδραση Επιμέρους Συστημάτων

Έχουμε εξηγήσει πως έχουν υλοποιηθεί τα επιμέρους συστήματα. Τώρα θα περιγράψουμε πως αυτά αλληλεπιδρούν μεταξύ τους χρονικά σε κάποια ενέργεια του χρήστη. Για τον λόγο αυτό δείχνουμε το UML διάγραμμα ακολουθίας (sequence UML diagram). Για να δείξουμε όλες τις βασικές λειτουργίες χρησιμοποιούμε το σενάριο όπου ο χρήστης τοποθετεί απευθείας το URL στον browser του και βάζει κάποια input για να φτιάξει μια οπτικοποίηση (Visualization). Για να χωρέσουν στο διάγραμμα όλα τα συστήματα, δείχνουμε δύο διαγράμματα, όπου στο 1ο ο Server φαίνεται σαν ενιαίο σύστημα και στο 2ο δείχνουμε τα υπο συστήματα του Server.

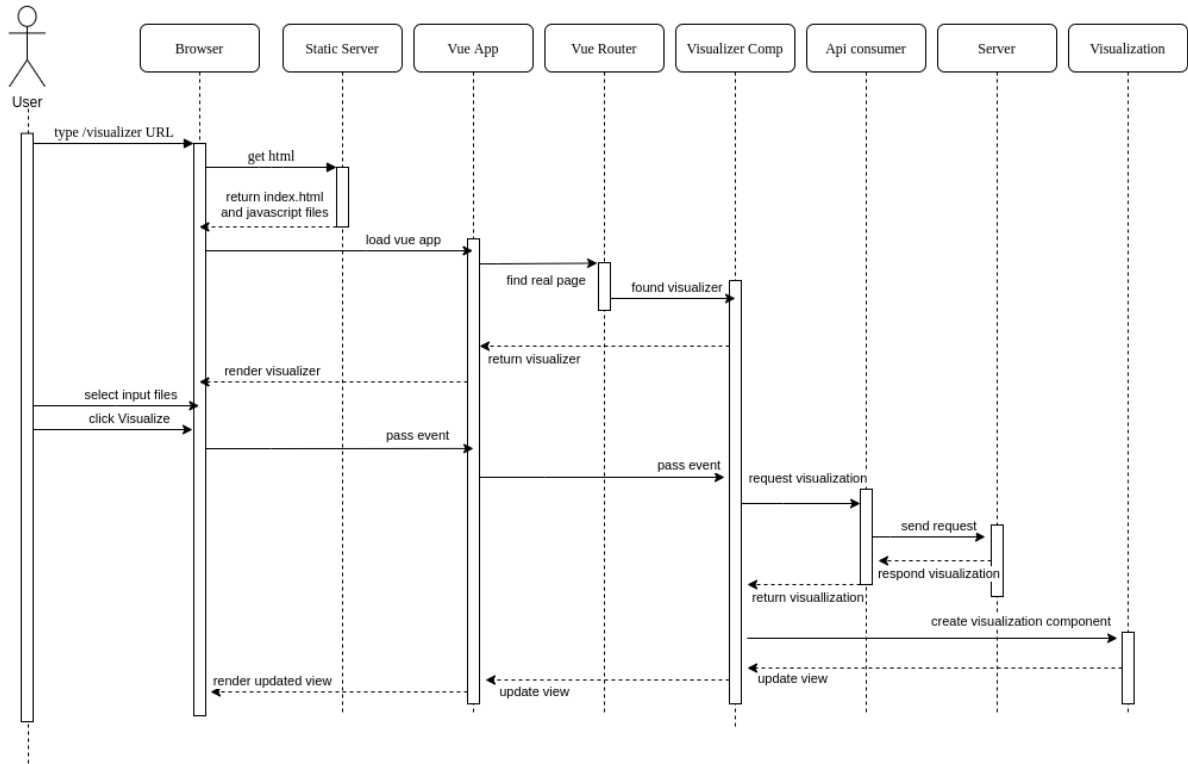


Figure 14: UML Sequence Diagram μέρος 1ο

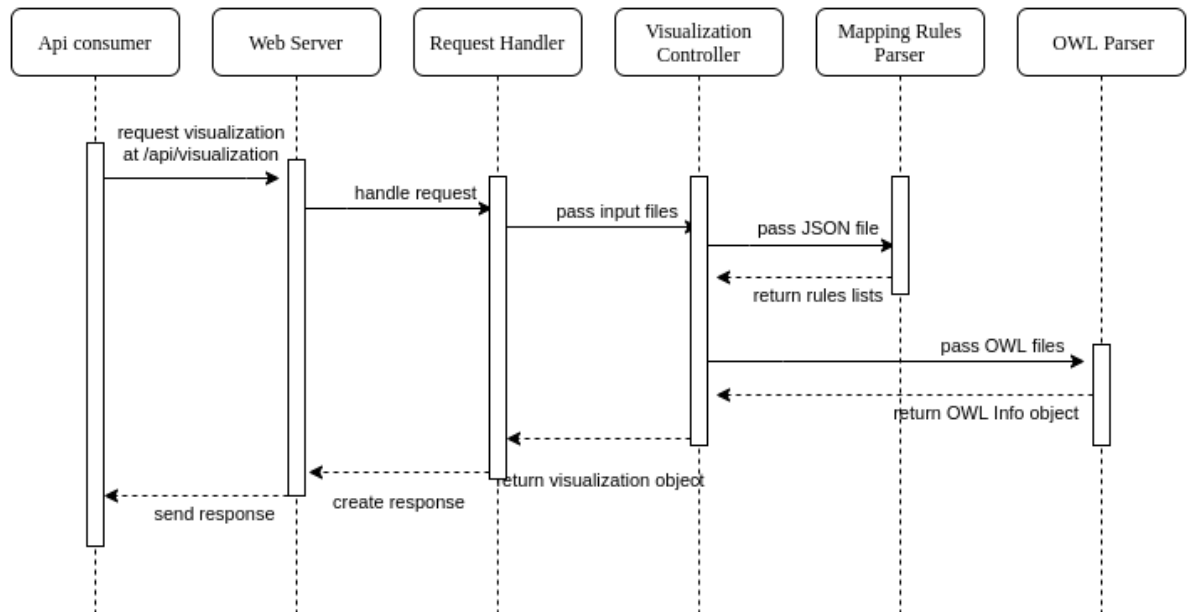


Figure 15: UML Sequence Diagram μέρος 2ο

Περιγράφουμε τώρα την διαδικασία που φαίνεται στο σχήμα χρονικά, δηλαδή όπως γίνεται η κάθε ενέργεια με την σειρά. Θεωρούμε ότι έχουμε τοποθετήσει τον client σε έναν ξεχωριστό μικρό στατικό server καθώς και ότι τρέχουμε τον κύριο server μας σε κάποιο μηχάνημα. Αρχικά ο χρήστης τοποθετεί το URL στον browser του που έχει την μορφή *APP-DOMAIN/visualizer*, Όπου το APP-DOMAIN είναι το domain namespace του στατικού server που προσφέρει τον client. Τότε ο browser ζητάει από τον server να φέρει την ιστοσελίδα που σχετίζεται σε εκείνο το path. Όπως έχουμε εξηγήσει ωστόσο, για να λειτουργήσει το Vue Router ο server έχει ρυθμιστεί να ανακατευθύνει το αίτημα πάντα στο *index.html*. Έτσι επιστρέφει στον browser αυτό το αρχείο μαζί με όλα τα αρχεία javascript που έχουν συμπεριληφθεί σε αυτό. Ο browser το φορτώνει και τρέχει την προγραμματισμένη javascript όπου φορτώνει την εφαρμογή Vue και περνάει το path που είχε δώσει ο χρήστης στο Vue Router. Το Vue router βρίσκει το component που συνδέεται με αυτό το path και το επιστρέφει στο Vue App όπου το δημιουργεί (create) και στέλνει το ανανεωμένο DOM στον browser ο οποίος το εμφανίζει (rendering). Σημειώνουμε ότι η ίδια διαδικασία ακολουθείται για όλες τις σελίδες που έχουμε ορίσει με το vue router όταν ο χρήστης δίνει κάποιο URL. Ωστόσο, αφού φορτωθούν την πρώτη φορά, αν ο χρήστης αλλάζει σελίδα μέσω του μενού της εφαρμογής και όχι δίνοντας κάποιο URL τότε δεν ξαναστέλνεται αίτημα στον στατικό server και απλά η το vue router επιστρέφει απευθείας την σελίδα που ήδη έχει κατέβει στον browser.

Ο χρήστης τώρα στην σελίδα Visualizer δίνει τα αρχεία OWL και τα Mapping Rules που τα συνδέει και πατάει το κουμπί Visualize. Το event αυτό στέλνεται από τον browser στο Vue App όπου αυτό το στέλνει στο Visualizer Comp και με την σειρά του καλεί το Api Consumer. Αυτός φτιάχνει ένα http request όπως το έχουμε περιγράψει και το στέλνει στην διεύθυνση του κύριου server στο endpoint */api/visualization*. Ο web server καλεί τον αντίστοιχο request handler και αφού υπάρχει αυτό με την σειρά του καλεί τον Visualization Controller. Εκείνος χρησιμοποιεί τον Mapping Rules Parser και τον OWL Parser για να επεξεργαστεί τα αρχεία και να επιστρέψει κάποιο Visualization object ή κάποιο error. Αυτά στέλνονται πίσω στον request handler ο οποίος φτιάχνει ένα http response με το οποίο ο web server απαντάει στον Api Consumer.

Το Visualizer Comp έχει τώρα το αποτέλεσμα από τον Api Consumer και αν ήταν επιτυχές τότε το περνάει ως Prop στο Visualization component και το αρχικοποιεί. Αυτό αλλάζει το Virtual DOM του όπως το έχουμε εξηγήσει και ειδοποιεί το Vue για τις αλλαγές. Το Vue App ενεργοποιεί με την σειρά του το rendering του browser μόνο για τα σημεία της ιστοσελίδας που έχουν μεταβληθεί. Έτσι ο χρήστης βλέπει την οπτικοποίηση στην οθόνη του.

4. Εκτέλεση του Συστήματος και Παραδείγματα

Θα περιγράψουμε πώς μπορεί κανείς κάποιος να αρχικοποιήσει και να εκτελέσει την εφαρμογή. Στην συνέχεια θα δείξουμε παραδείγματα εκτέλεσης μαζί με εικόνες για την πλήρη κατανόηση της πληροφορίας που μπορεί να καλύψει το σύστημά μας.

4.1. Αρχικοποίηση και Εκτέλεση του Συστήματος

Θα περιγράψουμε πώς μπορούμε να τρέξουμε την εφαρμογή μας αρχίζοντας από την αρχή. Λαμβάνουμε πρώτα τον κώδικα ο οποίος είναι διαθέσιμος και σε ένα αποθετήριο κώδικα (repository) στο github.com όπου χρησιμοποιεί το πρωτόκολλο git. Συγκεκριμένα βρίσκεται στην διεύθυνση

<https://github.com/ThanosApostolou/omv>

Για να μπορέσουμε να χτίσουμε και να τρέξουμε την εφαρμογή χρειαζόμαστε να έχουμε κάποιο *Java SDK* (το οποίο είναι το περιβάλλον προγραμματισμού για java) μεγαλύτερης ή ίσης έκδοσης από την 8. Επίσης χρειαζόμαστε εγκατεστημένο *nodejs* και *npm* όπου πρόκειται για ένα περιβάλλον εκτέλεσης (runtime) javascript και έναν διαχειριστή πακέτων (package manager) για javascript ο οποίος χρησιμοποιείται για την αυτόματη εύρεση βιβλιοθηκών που έχουμε συμπεριλάβει στην εφαρμογή μας ως εξαρτήσεις (dependencies).

Εκτέλεση τοπικά

Για να εκτελέσουμε την εφαρμογή τοπικά ανήγουμε κάποιο τερματικό (terminal) ανάλογα με το λειτουργικό σύστημα που χρησιμοποιούμε στον φάκελο *omv-server*. Από εκεί τρέχουμε το αντίστοιχο *gradlew* αρχείο (script) ανάλογα με το σύστημά μας με την παράμετρο *run*. Για παράδειγμα από linux ή Mac OS τρέχουμε “./gradlew run” ενώ από windows “gradlew.bat run”. Αυτό χρησιμοποιεί τον ενσωματωμένο *gradle* που είναι ένα build system και package manager για java για να φέρει όλα τα dependencies, να κάνει compile τον server και να τον τρέξει με προγραμματιστικό τρόπο (development mode). Ο server τώρα αρχίζει να τρέχει στην θύρα (port) 8080 και μπορούμε να το επιβεβαιώσουμε δίνοντας σε έναν browser την διεύθυνση

`"localhost:8080/api"`, όπου θα πρέπει να εμφανιστούν κάποια βασικά στοιχεία του server όπως η έκδοσή του σε μορφή JSON.

Για τον client ανοίγουμε ένα 2ο τερματικό στον φάκελο `omv-client`. Από εκεί τρέχουμε την εντολή `"npm install"` για να φέρει τοπικά ο `npm package manager` όλα τα `dependencies` που χρησιμοποιεί η εφαρμογή μας. Μετά εκτελούμε `"npm run serve"` όπου αυτή η εντολή χρησιμοποιεί ένα εργαλείο που λέγεται `webpack` ώστε να τρέξει έναν τοπικό στατικό server ο οποίος προσφέρει τον client μας. Αυτός ο `webpack dev server` τρέχει στο 1ο port που βρει διαθέσιμο αρχίζοντας από το 8080 και αυξάνοντας κατά 1. Οπότε αφού ήδη τρέχουμε τον server στο 8080, θα πρέπει να χρησιμοποιήσει το 8081 (αν δεν τρέχουμε κάποια άλλη εφαρμογή που να το έχει δεσμεύσει στον υπολογιστή μας). Δίνοντας την διεύθυνση `"localhost:8081"` σε κάποιον browser θα πρέπει να δούμε το γραφικό περιβάλλον της εφαρμογής μας.

Χτίσιμο και Διανομή (building and distribution)

Περιγράφουμε την διαδικασία για να τρέξουμε την εφαρμογή μας όπως λέμε σε τρόπο παραγωγής (`production mode`), δηλαδή για να είναι έτοιμη προς χρήση.

Ανοίγουμε τερματικό στον φάκελο `omv-server` και εκτελούμε την εντολή `./gradlew shadowJar` ή `gradlew.bat shadowJar` ανάλογα με το σύστημα. Αυτή η εντολή χρησιμοποιεί πάλι τον `gradle` αλλά τώρα φέρνει όλα τα `dependencies`, τα κάνει `compile` μαζί με την εφαρμογή μας και φτιάχνει ένα αρχείο `jar`. Αυτό το αρχείο μπορεί να διαβαστεί και να εκτελεστεί από ένα `Java Virtual Machine` και είναι όπως λέμε ένα παχύ `jar` (`fat jar`), δηλαδή εμπεριέχει ενσωματωμένα όλα τα `dependencies` και τις βιβλιοθήκες που χρειάζεται η εφαρμογή μας. Το αρχείο που δημιουργήθηκε βρίσκεται στην τοποθεσία `"omv-server/build/libs/omv-server-VERSION-fat.jar"` όπου `VERSION` είναι η τρέχουσα έκδοση που έχει οριστεί από τον προγραμματιστή. Μπορούμε να το βάλουμε πλέον σε έναν υπολογιστή που θα θέλουμε να τρέχει τον server μας και να το εκτελέσουμε με την εντολή `"java -Dport=PORT -jar omv-server-VERSION-fat.jar"` όπου `PORT` είναι το port που θέλουμε να τρέχει ο server μας και θα είναι το 8080 αν παραληφθεί.

Για τον client αλλάζουμε πρώτα στο αρχείο `omv-client/src/SettingsManager.ts` στην μέθοδο `init()` την γραμμή μετά το `else` όπου ορίζει τον `defaultServer` ώστε να έχει την τιμή του domain ή της IP (μαζί με το port αν είναι διαφορετικό του 80 για http ή 443 για https) του μηχανήματος όπου έχουμε τρέξει τον server. Μετά ανοίγουμε τερματικό στον φάκελο `omv-client` και τρέχουμε την εντολή `"npm install"` για να φέρουμε τα dependencies και μετά `"npm run build"`. Με την δεύτερη εντολή χρησιμοποιούμε πάλι τον webpack αλλά όχι για να φτιάξουμε έναν dev server όπως κάναμε στην τοπική εκτέλεση, αλλά για να φτιάξουμε τον φάκελο `omv-client/dist` ο οποίος περιέχει όλη την τελική εφαρμογή μας συμπεριλαμβανομένων όλων των dependencies που χρησιμοποιούνται. Τώρα αρκεί να διαθέσουμε αυτόν τον φάκελο σε κάποιον απλό server που να μπορεί να προσφέρει στατικές σελίδες και να τον ρυθμίσουμε να κάνει ανακατεύθυνση (redirect) όλα τα requests στο `index.html`. Αυτό μπορεί να γίνει πολύ εύκολα μέσω έτοιμων servers όπως `apache` ή `tomcat`.

4.2. Παραδείγματα Λειτουργίας

Θα δείξουμε την λειτουργία της εφαρμογής από την μεριά του χρήστη περιγράφοντας κάθε βασική υποσελίδα και τις βασικές λειτουργίες της. Οι εικόνες που θα δείξουμε θα είναι από τοπική εκτέλεση στον υπολογιστή μας. Υπενθυμίζουμε από το κεφάλαιο 3, ότι ενώ σε κάθε σελίδα θα αλλάζει το URL στον browser, αίτημα για νέα σελίδα στον στατικό server γίνεται μόνο την 1η φορά που χρειαζόμαστε να κατεβάσουμε μια σελίδα. Τις υπόλοιπες αρκεί το Vue Router να φορτώνει την αντίστοιχη σελίδα.

4.2.1 Menu και Home Page

Είναι η πρώτη σελίδα και βρίσκεται στο path `"/home"`. Επίσης σε εκείνο το path γινόμαστε `redirect` από το vue router όταν φορτώνουμε μόνο το domain ή την ip στις σελίδας με το port δηλαδή από το path `"/"`.

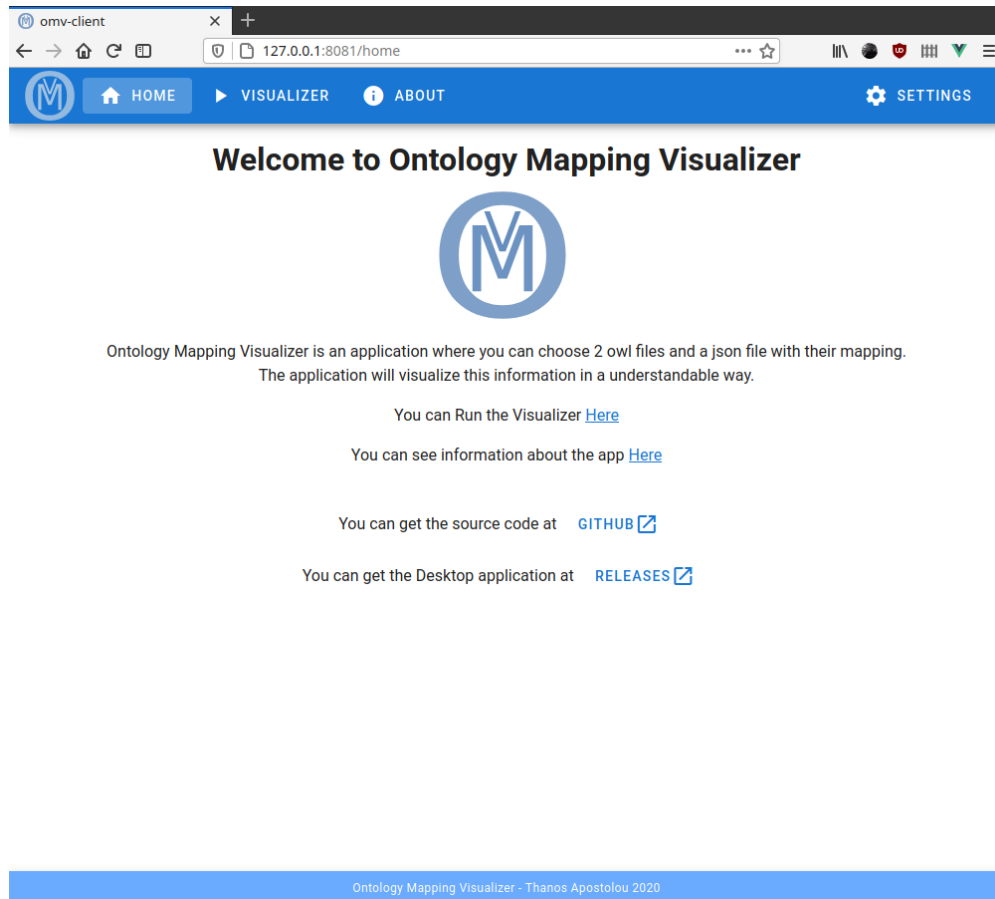


Figure 16: Home Page

Όπως βλέπουμε περιέχει μια απλή περιγραφή για την εφαρμογή μας με κάποιους συνδέσμους (links) όπως για το που μπορούμε να βρούμε τον κώδικα της εφαρμογής. Η οριζόντια μπάρα για να περιηγούμαστε στις σελίδες είναι ενιαία για όλες και απλά αλλάζει κάθε φορά το χρώμα της σελίδας στην οποία βρισκόμαστε. Η εφαρμογή όλη έχει δημιουργεί ώστε να έχει όσο το δυνατόν responsive design δηλαδή να προσαρμόζεται στο μέγεθος της οθόνης. Οπότε βλέπουμε ότι μικραίνει το οριζόντιο μέγεθος του παραθύρου τότε τα στοιχεία που υπάρχουν στην οριζόντια μπάρα μετατρέπονται σε ένα μενού.

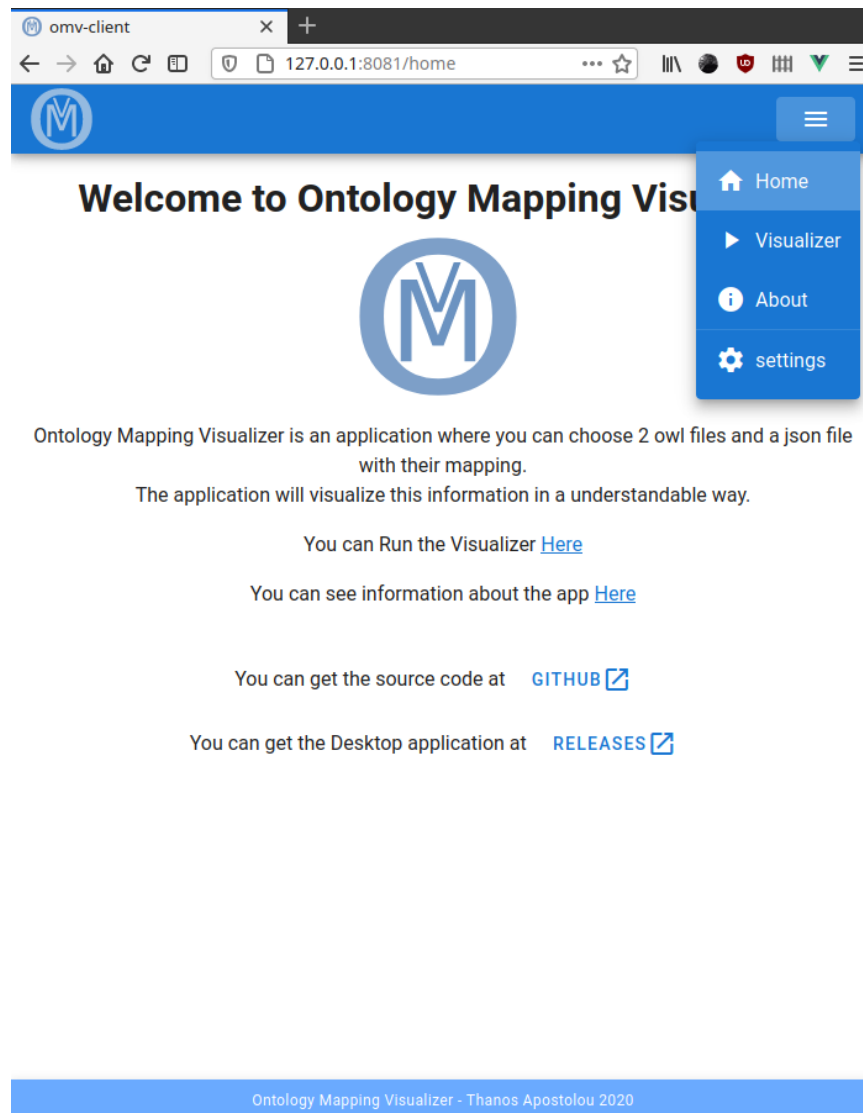


Figure 17: Home Page μικρού οριζόντιου μεγέθους

Επίσης το footer που βλέπουμε με μπλε χρώμα στο κάτω κάτω της σελίδας είναι ενιαίο για κάθε σελίδα. Η χρήση του βοηθάει στο να καταλαβαίνει εύκολα ο χρήστης το τέλος της εφαρμογής, σε αντίθεση με την περίπτωση που υπάρχει κι άλλο διαθέσιμο περιεχόμενο που μπορεί να εμφανιστεί με scroll.

4.2.2 About Page

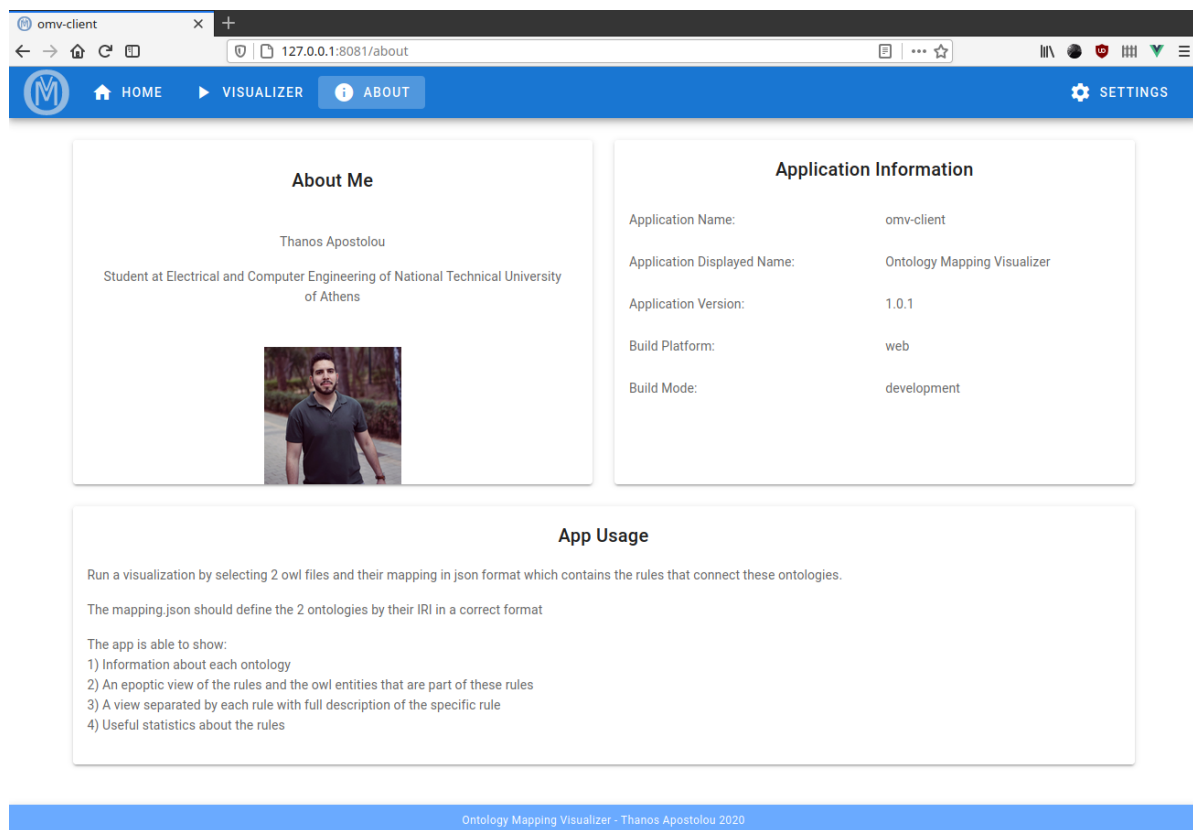


Figure 18: About Page

Όπως βλέπουμε δείχνει κάποιες πληροφορίες για τον δημιουργό της εφαρμογής, πληροφορίες για το περιβάλλον και την έκδοση καθώς και μια πολύ σύντομη περιγραφή για τα βήματα λειτουργίας της.

4.2.3 Settings Page

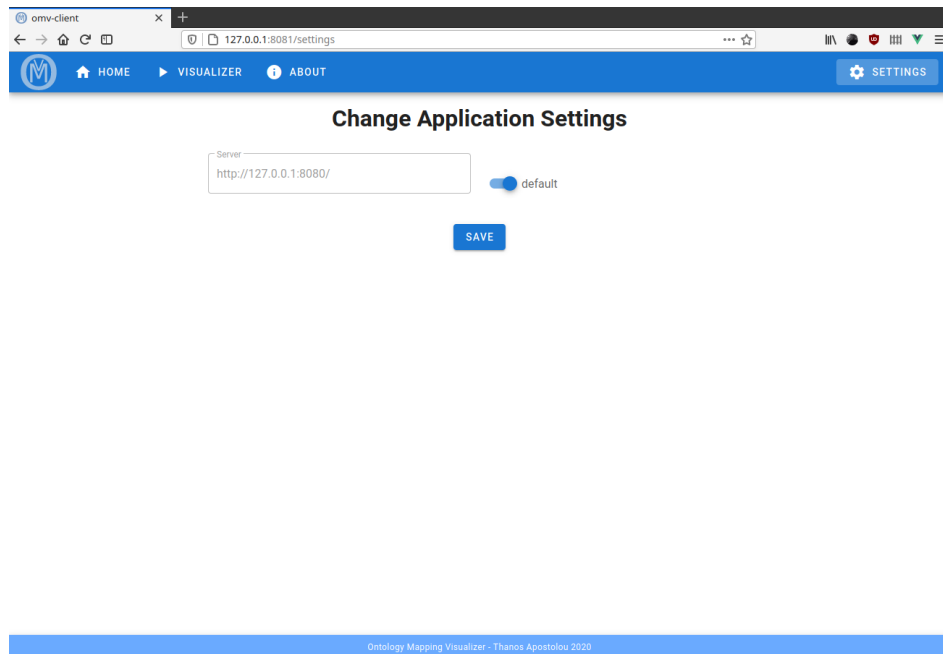


Figure 19: Settings Page

Περιλαμβάνει τις ρυθμίσεις της εφαρμογής, όπου δίνουν τη δυνατότητα στον χρήστη να προσαρμόσει την εφαρμογή. Κάθε ρύθμιση που περιλαμβάνεται έχει έναν διακόπτη (switch) για να μπορεί ο χρήστης εύκολα να επαναφέρει την προκαθορισμένη (default) τιμή της. Στην παρούσα έκδοση περιλαμβάνεται μόνο το πεδίο αλλαγής του server, για να έχει την ευχέρεια ο χρήστης να χρησιμοποιήσει κάποιο άλλον μηχανήμα ως κύριο server. Σε μελλοντικές εκδόσεις μπορεί να προστεθούν επιπλέον επιλογές όπως το μέγεθος των σχημάτων, το θέμα εμφάνισης και άλλα.

4.2.4 Visualizer Page

Είναι η σελίδα όπου περιέχει την βασική λειτουργία της εφαρμογής μας. Όπως βλέπουμε στις παρακάτω εικόνες διαθέτει δύο βασικές επιλογές προσβάσιμες με καρτέλες (tabs). Η πρώτη είναι να δώσει ο χρήστης 1 OWL αρχείο και ένα JSON αρχείο με τους Mapping Rules και η οπτικοποίηση θα γίνει θεωρώντας ως 2ο αρχείο το αρχείο “*HarmonicSS-Reference-Model+Vocabularies-v.0.9.3.owl*” που έχει ως IRI το “*http://www.semanticweb.org/ntua/iccs/harmonicss/terminology*” και το συμπεριλαμβάνει ο server μας αφού χρησιμοποιείται πολλές φορές ως μια

Οντολογία αναφοράς (reference). Η δεύτερη επιλογή είναι να επιλέξει ο χρήστης 2 OWL αρχεία και ένα JSON αρχείο με τους Mapping Rules για τα οποία θα γίνει η οπτικοποίηση.

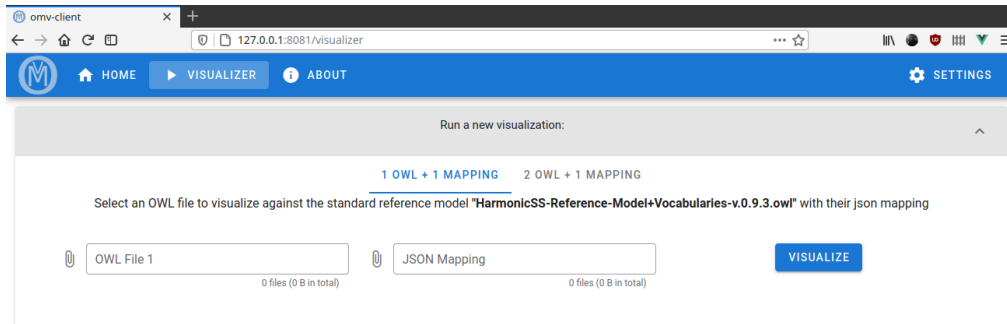


Figure 20: Visualizer with 1 OWL tab

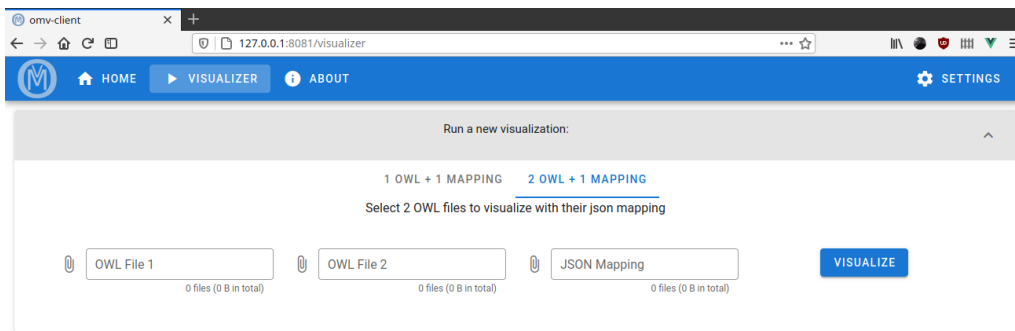


Figure 21: Visualizer with 2 OWL tab

Επιλέγουμε ένα αρχείο οντολογίας που έχουμε με όνομα “*AOUD-UD DATASET FINAL FOR HARMONIZATION-Metadata_MOD.xlsx.owl*” που έχει IRI “*http://www.semanticweb.org/ntua/iccs/harmonicss/cohort*” και το αντίστοιχο αρχείο με τα JSON Mapping Rules όπου έχει τους κανόνες αντιστοίχισής του με την reference Οντολογία και πατάμε το κουμπί VISUALIZE. Παίρνουμε την παρακάτω εικόνα:

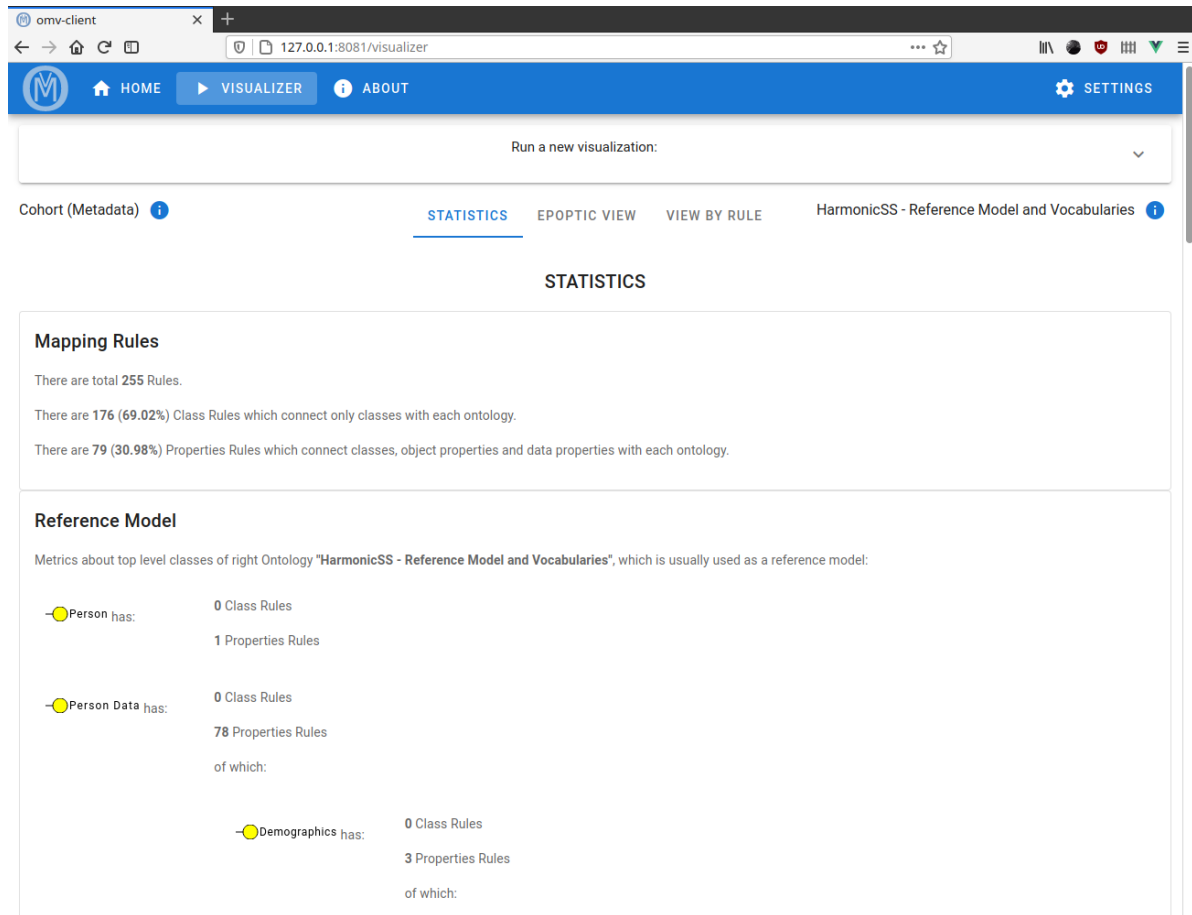


Figure 22: Visualization Statistics

Η εφαρμογή μας έστειλε τα αρχεία στον server και πήρε την απάντηση. Η απάντηση εδώ δεν είχε κάποιο σφάλμα και περιείχε όλη την πληροφορία που χρειαζόμαστε για την οπτικοποίηση (Visualization). Βλέπουμε ότι τα πεδία επιλογής αρχείων κρύφτηκαν αυτόματα για εξοικονόμηση χώρου σε ένα όπως λέγεται expansion panel όπου ο χρήστης μπορεί να το ανοίξει για να κάνει μια νέα οπτικοποίηση πάλι. Υπάρχουν 3 βασικά tabs όπου μας δίνουν διαφορετικές πληροφορίες για τους Mapping Rules και τις οντολογίες. Το πρώτο, το οποίο επιλέγεται εξορισμού είναι το tab "Statistics". Σε αυτό βλέπουμε κάποια στατιστικά για τους κανόνες οι οποίοι είναι χωρισμένοι σε αυτούς που αφορούν μόνο κλάσεις (Owl Classes) και σε αυτούς που εμπεριέχουν και ιδιότητες (Owl Properties). Στη συνέχεια βλέπουμε στατιστικά για τους κανόνες που συμμετέχουν οι κορυφαίες κλάσεις της reference Οντολογίας μαζί με τα παιδιά τους μέχρι 2 επίπεδα. Προς το τέλος της σελίδας φαίνεται πληροφορία για το πόσα βασικά elements όπου έχουμε ορίσει τα Classes, Object Properties,

Data Properties και Annotation Properties περιέχει κάθε οντολογία όπως φαίνεται στην επόμενη εικόνα.

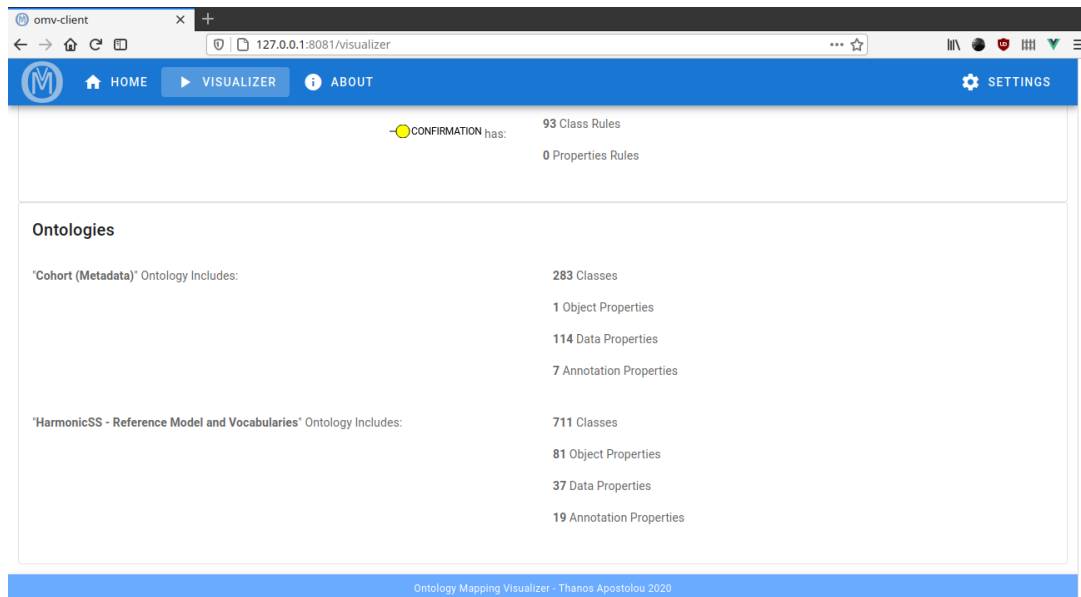


Figure 23: Visualization Statistics τέλος σελίδας

Στο tab “Eroptic View” παίρνουμε μια εποπτική εικόνα για τους κανόνες και την σύνδεσή τους με τα στοιχεία της κάθε οντολογίας.

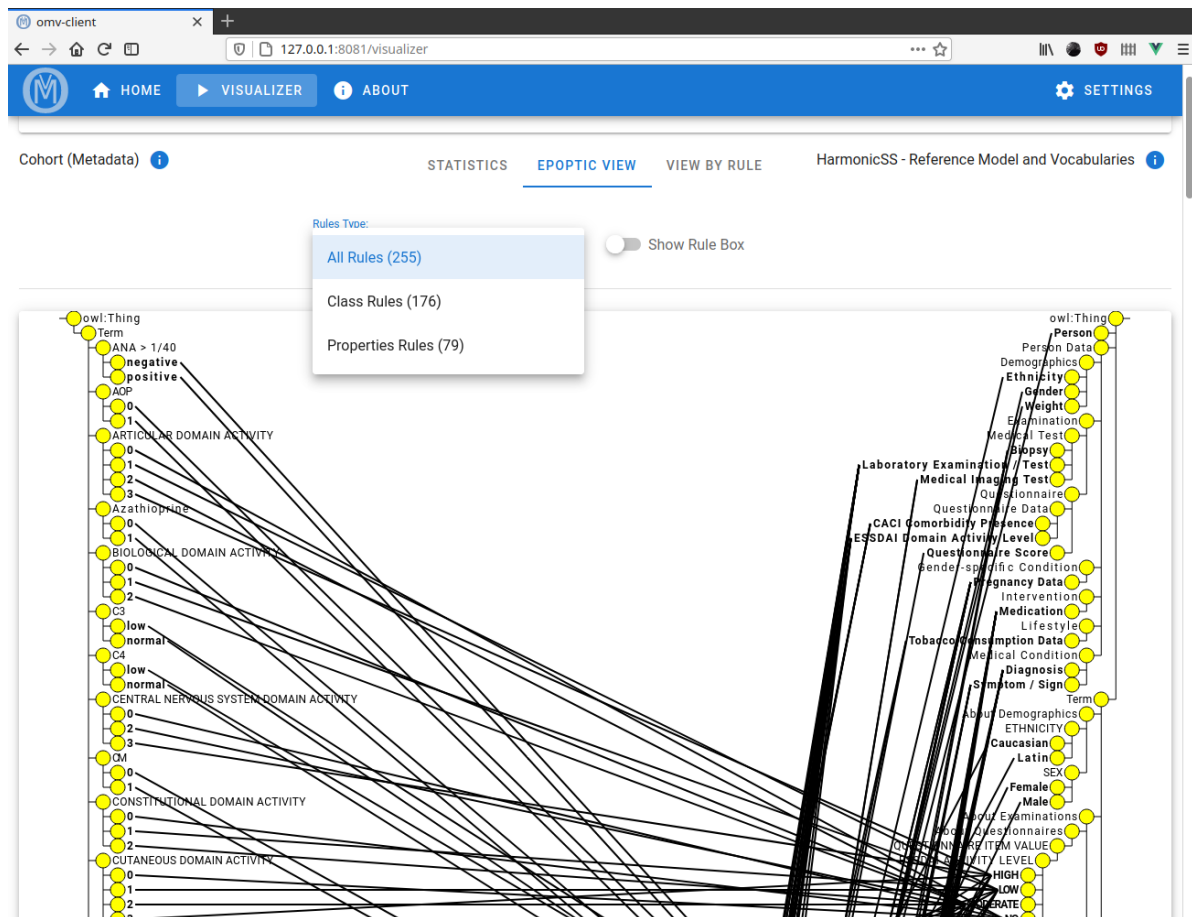


Figure 24: Visualization Epopic View 1

Εξορισμού φαίνονται όλοι οι κανόνες αντιστοίχισης και μόνο τα στοιχεία των κλάσεων που συμμετέχουν σε αυτούς με έντονα (bold) γράμματα καθώς και η οντολογία γονέας για την κατανόηση του δέντρου. Ενεργοποιούμε τον διακόπτη “Show Rule Box” και αυτό μας εμφανίζει ένα γκρι κουτί στην μέση όπου έχει την έννοια του κανόνα και τώρα τα στοιχεία των κλάσεων ενώνονται με τα κουτιά αντί για απευθείας μεταξύ τους. Υπάρχει δυνατότητα ταξινόμησης των κουτιών βάση της αριστερής ή της δεξιάς οντολογίας. Όταν κάνουμε κλικ σε κάποια οντολογία μας εμφανίζεται ένα νέο ενσωματωμένο παράθυρο όπου δείχνει πληροφορίες αυτού του στοιχείου. Οι πληροφορίες του στοιχείου περιέχουν το όνομα το IRI, τα Annotations του και σε πόσους κανόνες συμμετέχει αυτό το στοιχείο μόνο του καθώς και μαζί με τα παιδιά του. Επίσης εμφανίζονται τότε μόνο οι κανόνες όπου συμμετέχουν σε αυτό το στοιχείο και στα παιδιά του όπου χρωματίζονται ως μπλε. Αν ξανακάνουμε κλικ τότε ξαναφαίνονται όλοι οι κανόνες όπως πριν κάνουμε το 1ο κλικ. Επιλέγουμε

να μας εμφανιστούν μόνο οι “Properties Rules”, τσεκάρουμε το “*Show Rule Box*” και κάνουμε κλικ στην Data Property “*About .. About Demographics + Smoking Status & Pregnancies*” παίρνοντας την εξής εικόνα:

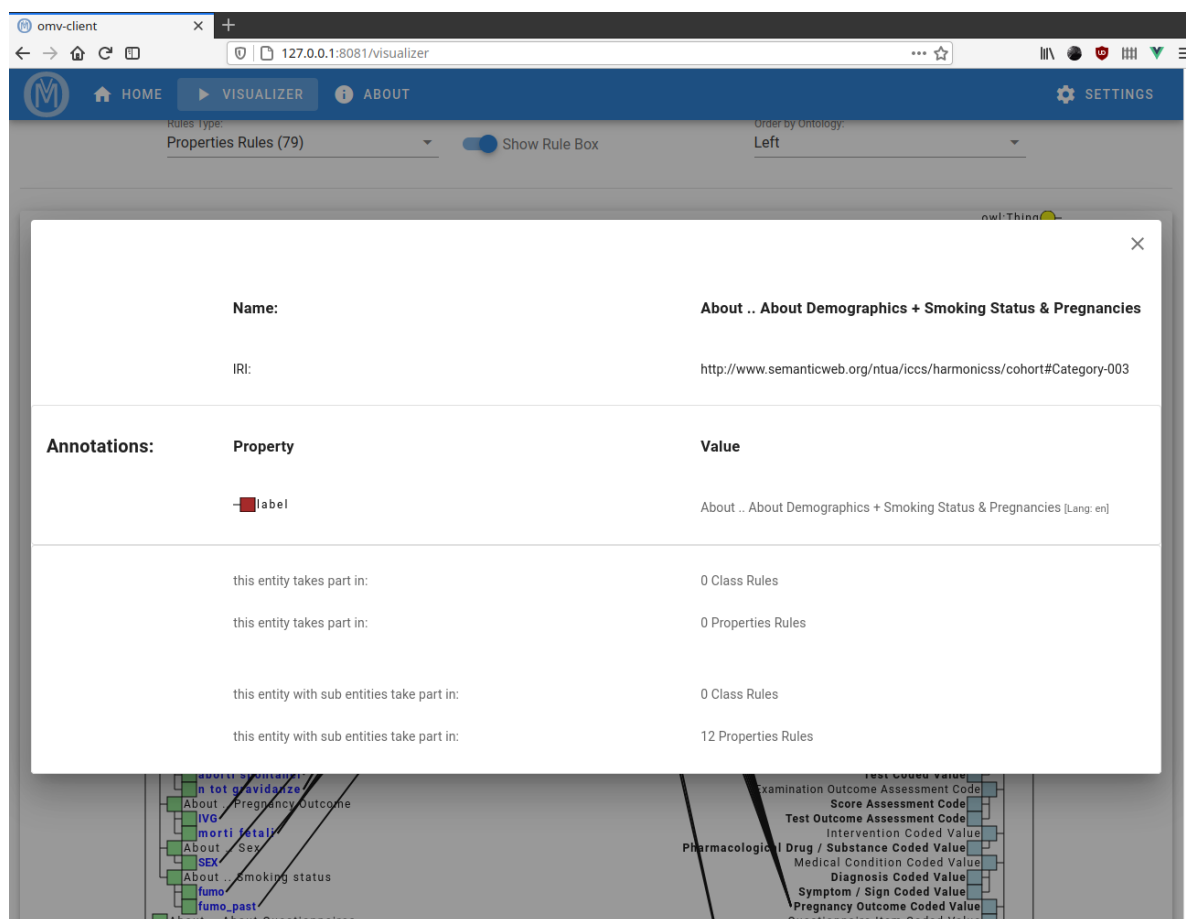


Figure 25: Visualization Click Element

Αφού κλείσουμε τον διάλογο βλέπουμε αυτό που περιγράψαμε

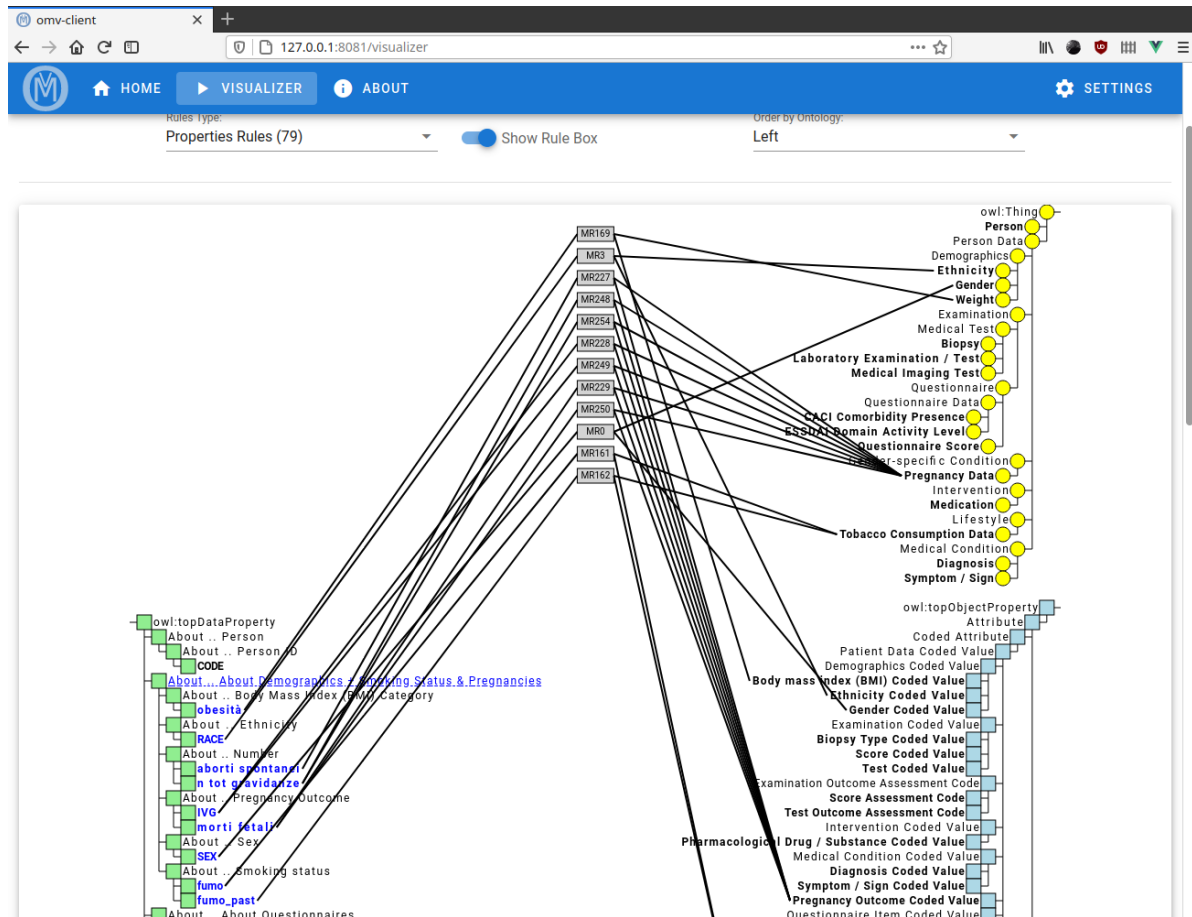


Figure 26: Visualization Eropic View με επιλεγμένο στοιχείο

Το τρίτο tab μας δίνει την όψη ανά κανόνα (View By Rule). Σε αυτήν μπορούμε να επιλέγουμε πάλι να δούμε όλες τους κανόνες, μόνο αυτούς που αφορούν Classes ή αυτούς που περιέχουν και Properties και να τους ταξινομήσουμε ανάλογα με την σειρά που τους συναντάμε είτε από τα στοιχεία της αριστερής οντολογίας είτε από της δεξιάς. Κάθε κανόνας φαίνεται ξεχωριστά και βλέπουμε όλες τις πληροφορίες που περιέχουν όπως τις περιγράψαμε στην ενότητα 2.3. Για κάθε κανόνα έχει οριστεί ένας Mapping Rule Number όπου δείχνει τον αύξοντα αριθμό που τον συναντήσαμε στο αρχείο με τους Mapping Rules.

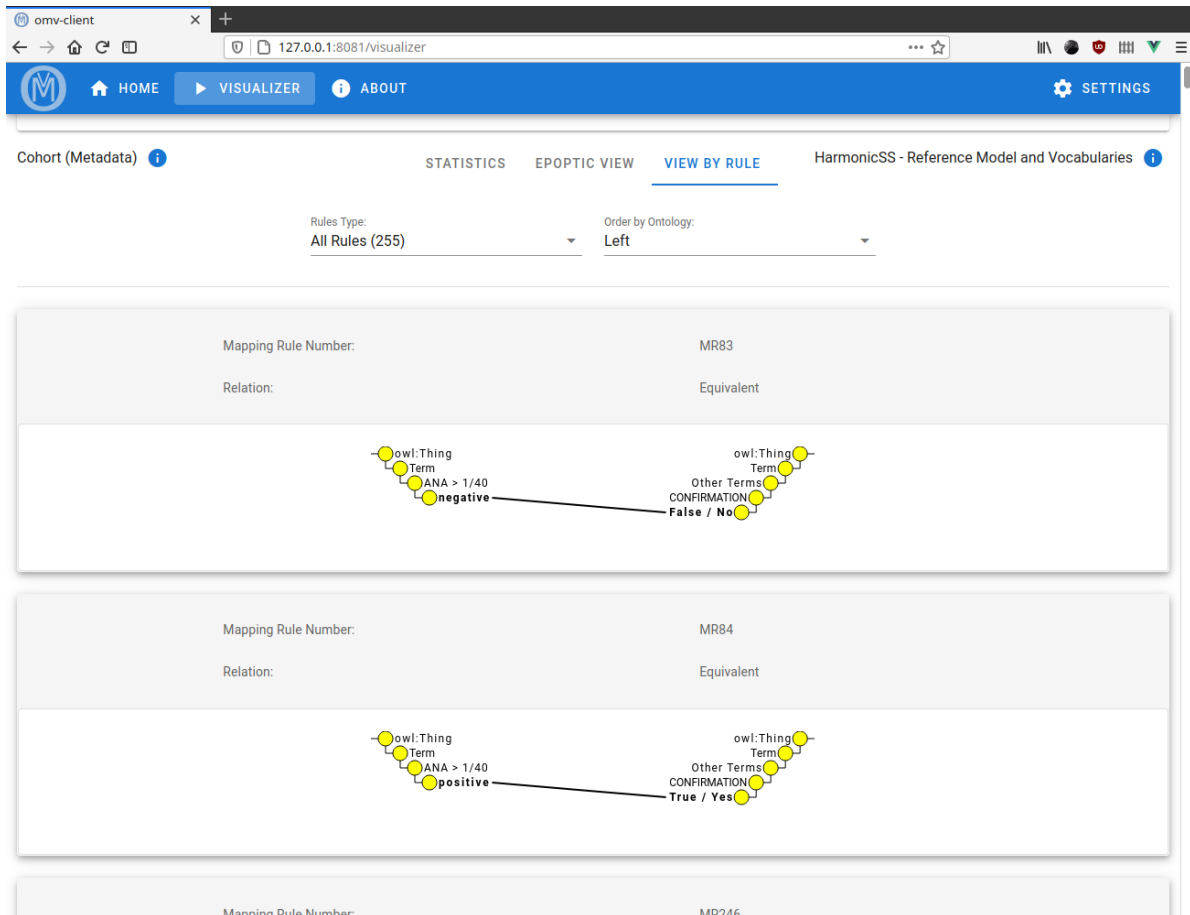


Figure 27: Visualization View By Rule, απλός κανόνας

Βλέπουμε ότι κάθε κανόνας δείχνει μόνο τα στοιχεία που συνδέει αυτός μαζί με τους γονείς τους στο δέντρο τους. Τα στοιχεία αυτά μπορούν να γίνουν κλικ και να μας δώσουν τις πληροφορίες τους με το ίδιο ενσωματωμένο παράθυρο όπως στην εποπτική όψη. Εδώ βέβαια δεν χρωματίζονται τα στοιχεία και δεν μένουν επιλεγμένα αφού δεν υπάρχει λόγος για τέτοια χρήση. Από κάθε κανόνα φαίνονται μόνο τα πεδία που δεν έχουν άδειες τιμές.

Όταν έχουμε έναν πιο σύνθετο κανόνα τότε φαίνεται πάντα το γκρι κουτί για σωστή κατανόηση. Επίσης φαίνεται ο μετασχηματισμός (Transformation) αν υπάρχει με όλα τα arguments που έχει. Στην περίπτωση που ένα argument έχει κάποιο IRI κάποιου στοιχείου οντολογίας για τιμή, τότε σχηματίζεται αυτό ως ένα μονό στοιχείο με κατεύθυνση είτε από δεξιά είτε από αριστερά για να φαίνεται από ποια οντολογία προέρχεται. Κάποιες Properties που συμμετέχουν σε κανόνες έχουν πολλές φορές

ένα πεδίο “*Parameter Value*” στα Annotations τους. Αυτά έχουν τιμή κάποιες κλάσης από την οντολογία και αυτές φαίνονται ακριβώς κάτω από το τέλος του Transformation. Όταν πατάμε την επιλογή “*Show Relevant Rules*” τότε εμφανίζονται όλοι οι κανόνες κλάσεων, αν υπάρχουν, που συμμετέχουν τα παιδιά της συγκεκριμένης κλάσης που έχει οριστεί ως *Parameter Value*. Αυτά φαίνονται ξεκάθαρα στην παρακάτω εικόνα:

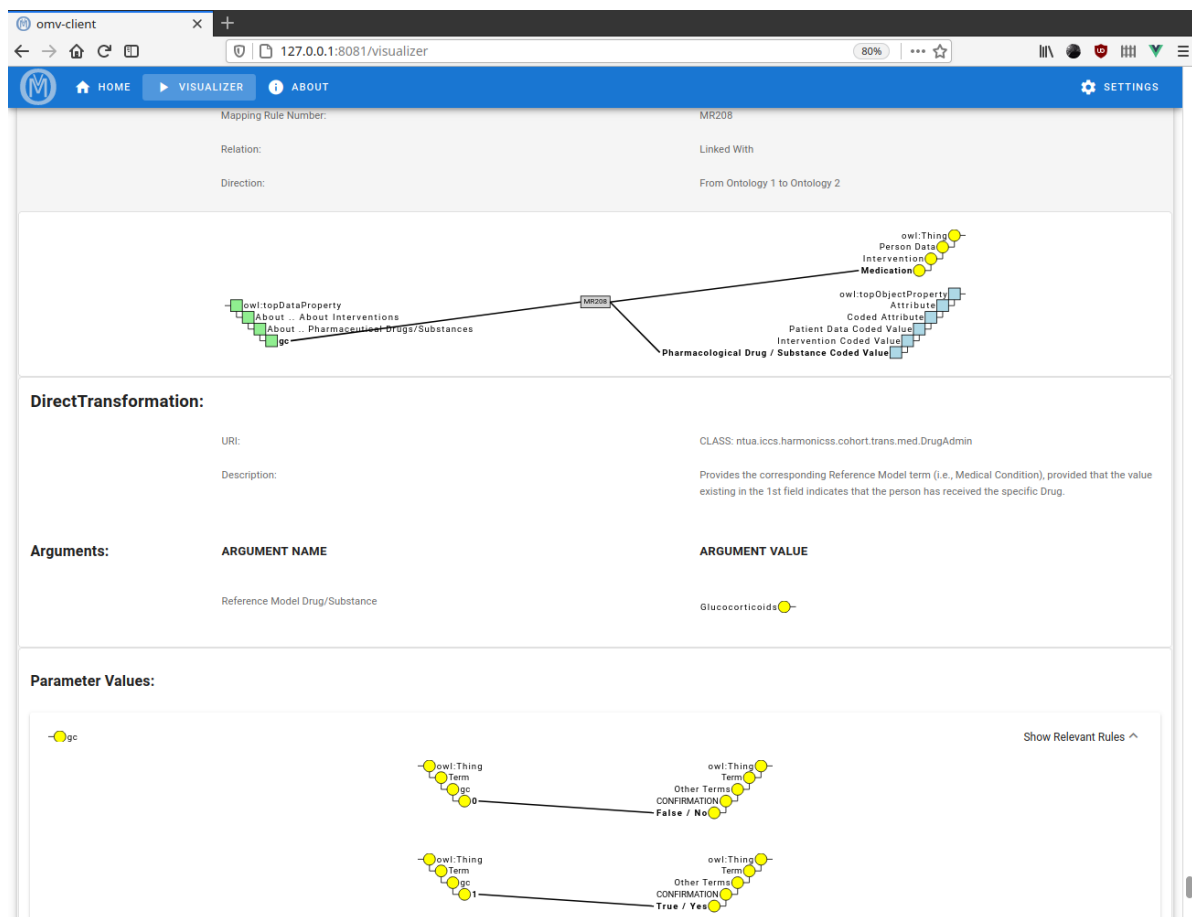



Figure 28: Visualization View By Rule, σύνθετος κανόνας

Τέλος αναφέρουμε ότι ανεξάρτητα από τα tabs των όψεων των κανόνων, δίπλα από κάθε οντολογία φαίνεται ένα κουμπί με την εικόνα . Όταν κάνουμε κλικ σε αυτό μας εμφανίζεται ένα ενσωματωμένο παράθυρο όπου περιέχει πληροφορίες για την Οντολογία, τις Classes, Object Properties, Data Properties σε δενδρική εμφάνιση και τις Annotation Properties σε εμφάνιση λίστας. Εδώ φαίνονται όλα τα στοιχεία ανεξαρτήτως κανόνων τα οποία μάλιστα μπορούν να γίνουν κλικ με την σειρά τους

ώστε να δώσουν σε νέο ενσωματωμένο παράθυρο της πληροφορίες τους όπως τις έχουμε δει στην περίπτωση όψεων των κανόνων. Αυτά φαίνονται συνοπτικά για την reference Οντολογία στις παρακάτω εικόνες.

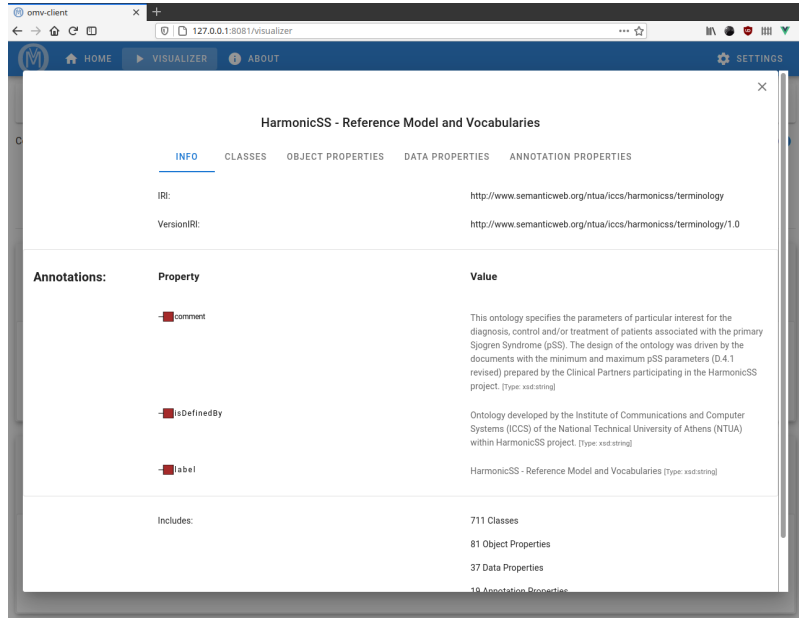


Figure 29: OWL Information

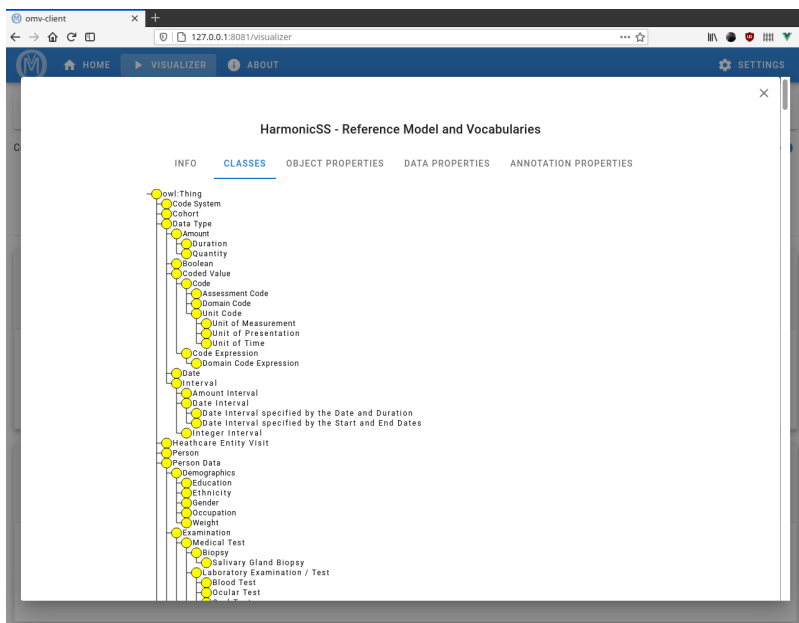


Figure 30: OWL Classes

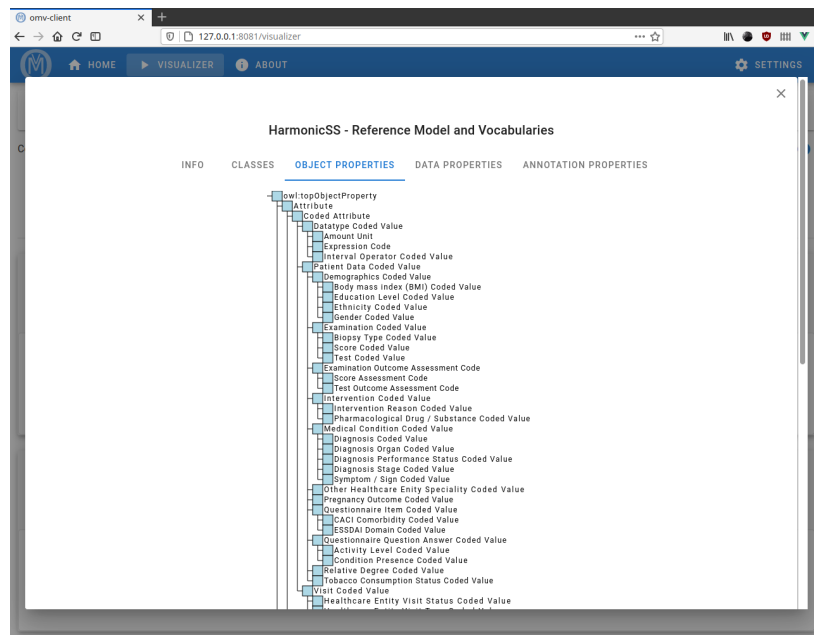


Figure 31: OWL Object Properties

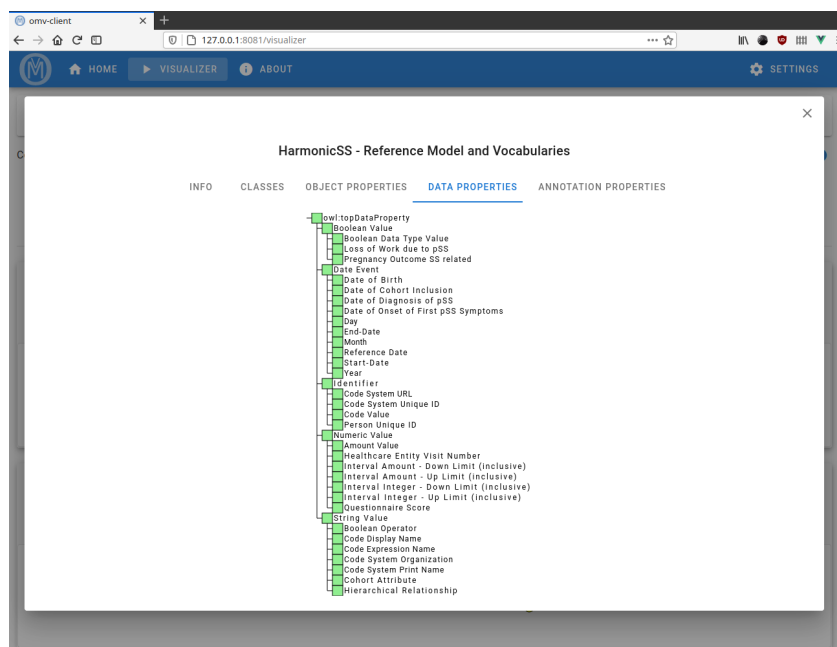


Figure 32: OWL Data Properties

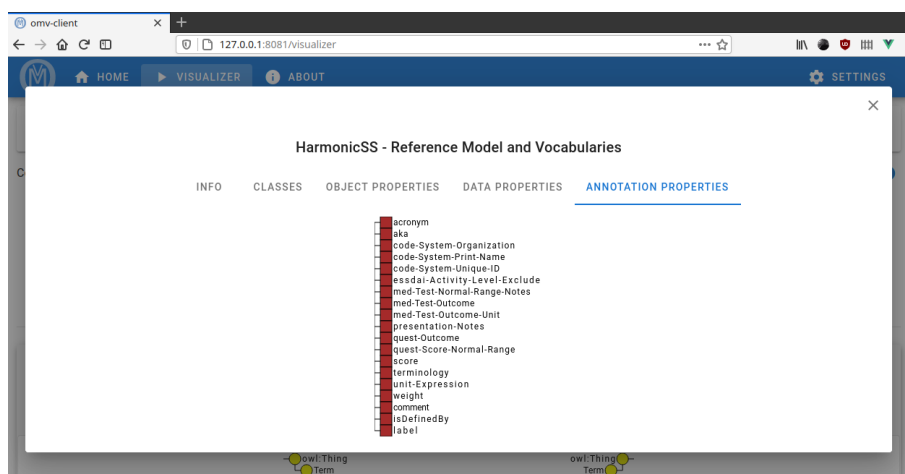


Figure 33: OWL Annotation Properties

5. Συμπεράσματα και Αξιολόγηση Συστήματος

Δφασδ

δασφς

6. Σύνοψη και Μελλοντική Εξέλιξη

Δφασδ

7. Βιβλιογραφικές Αναφορές

- [1]: Efthymios Chondrogiannis, Vassiliki Andronikou, Efstathios Karanastasis, Theodora Varvarigou, Ontology Alignment Tool, , Ανακτήθηκε από <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.664.4075&rep=rep1&type=pdf>
- [2]: WHATWG, HTML Living Standard, 2020, Ανακτήθηκε από <https://html.spec.whatwg.org/print.pdf>
- [3]: , The HTML DOM, , Ανακτήθηκε από https://www.w3schools.com/whatis/whatis_html5dom.asp
- [4]: , ECMAScript Specification, 2020, Ανακτήθηκε από <https://www.ecma-international.org/ecma-262/>
- [5]: , Concurrency model and the event loop, , Ανακτήθηκε από <https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop>
- [6]: , The Node.js Event Loop, , Ανακτήθηκε από <https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/>
- [7]: , Promise, , Ανακτήθηκε από https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
- [8]: , Reactivity in VueJs, , Ανακτήθηκε από <https://vuejs.org/v2/guide/reactivity.html>
- [9]: , The Vue Instance, , Ανακτήθηκε από <https://vuejs.org/v2/guide/instance.html>
- [10]: , Hypertext Transfer Protocol -- HTTP/1.1, , Ανακτήθηκε από <https://www.ietf.org/rfc/rfc2616.txt>
- [11]: , Introducing JSON, , Ανακτήθηκε από <https://www.json.org/json-en.html>
- [12]: , Actor Model in Nutshell, , Ανακτήθηκε από <https://medium.com/@KtheAgent/actor-model-in-nutshell-d13c0f81c8c7>
- [13]: , Vertx, , Ανακτήθηκε από <https://vertx.io/>
- [14]: Julien Ponge, Thomas Segismont, Julien Viet, A gentle guide to asynchronous programming with Eclipse Vert.x for Java developers, 2019, Ανακτήθηκε από <https://vertx.io/docs/guide-for-java-devs/guide-for-java-devs.pdf>
- [15]: Berners-Lee, T., Hendler, J. & Lassila, , 2001, Ανακτήθηκε από <https://web.archive.org/web/20081114135540/http://www.sciam.com/article.cfm?id=the-semantic-web&print=true>.
- [16]: Riley, J., Understanding Metadata Baltimore: National Information Standards Organization (NISO), 2017, Ανακτήθηκε από https://groups.niso.org/apps/group_public/download.php/17446/Understanding%20Metadata.pdf
- [17]: Gruber T.R., A Translation Approach to Portable Ontology Specification, Knowledge Acquisition, 1993
- [18]: , Owlapi Wiki, , Ανακτήθηκε από <https://github.com/owlcs/owlapi/wiki>
- [19]: Roy Thomas Fielding, Representational State Transfer, 2000, Ανακτήθηκε από https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm