ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΕΠΙΣΤΗΜΗ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Διπλωματική Εργασία
Μεταπτυχιακού Διπλώματος Ειδίκευσης

«Ανάκτηση Πληροφορίας με Οντολογίες»

Σταυρουλάκης Δημήτριος

Επιβλέπων: Καλαμπούκης Θεόδωρος

Εξωτερικός Αξιολογητής: Ανδρουτσόπουλος Ίων

ΑΘΗΝΑ, ΙΟΥΛΙΟΣ 2005

Πρόλογος

Η παρούσα εργασία ασχολείται με την καινούρια κατεύθυνση που αρχίζει να λαμβάνει το διαδίκτυο, η οποία είναι γνωστή ως Σημασιολογικός Ιστός. Στον χώρο αυτό υπάρχει η έννοια των οντολογιών, η οποία ορίζεται ως η τυπική προδιαγραφή μιας εννοιοποίησης. Σε μία οντολογία περιέχονται κλάσεις, στιγμιότυπα των κλάσεων αυτών, που ονομάζονται οντότητες και ιδιότητες, οι οποίες έχουν τον ρόλο της συσχέτισης δύο κλάσεων. Χάρη στο χαρακτηριστικό τους αυτό οι οντολογίες έχουν την ικανότητα να αναπαριστούν γνώση και να υποστηρίζουν την επικοινωνία μεταξύ εφαρμογών και την αυτόματη εξαγωγή συμπερασμάτων.

Η εφαρμογή που καλούμε να αναπτύξουμε βασίζεται κατά πολύ στην διπλωματική εργασία της Γιγουρτσή Ευθυμίας Μαρίας και έχει να κάνει με την υποστήριξη με οντολογίες του συστήματος ανάκτησης πληροφορίας, το οποίο έχει υλοποιήσει.

Σε ένα δευτερεύον επίπεδο επιχειρούμε τη βελτίωση του συστήματος με την αναζήτηση ενός καλύτερου προγράμματος crawler, το οποίο θα δημιουργήσει μια καλύτερη βάση δεδομένων (κειμένων) για το σύστημα, συνεπώς θα βελτιώσει και τις επιδόσεις του.

Στο πρώτο κεφάλαιο παραθέτουμε γενικές πληροφορίες για τις οντολογίες, τι είναι, που χρησιμεύουν και μια γενική μεθοδολογία για την σωστή ανάπτυξή τους. Επίσης βλέπουμε κάποια παραδείγματα οντολογιών.

Στο δεύτερο κεφάλαιο ασχολούμαστε με τις γλώσσες στις οποίες μπορεί κανείς να συντάξει μια οντολογία και επιλέγουμε την καταλληλότερη, την οποία θα χρησιμοποιήσουμε για τη δική μας οντολογία.

Στο τρίτο κεφάλαιο μελετάμε τα προγράμματα συγγραφής οντολογιών, τα οποία μας προσφέρουν πολλές ευκολίες κατά τη διαδικασία συγγραφής μιας οντολογίας με κυριότερη τη γραφική διεπαφή χρήστη. Με άλλα λόγια μας απαλλάσσουν από την ανάγκη να θυμόμαστε τις λεπτομέρειες της γλώσσας στην οποία συντάσσουμε.

Στο τέταρτο κεφάλαιο περιγράφουμε την εφαρμογή την οποία υλοποιήσαμε. Αρχικά βλέπουμε μια μελέτη των υπαρχόντων αυτή τη στιγμή crawlers και επιλέγουμε τον καταλληλότερο από αυτούς ως εργαλείο για τη βελτίωση της βάσης δεδομένων. Στη συνέχεια παραθέτουμε την οντολογία, την οποία δημιουργήσαμε για να χρησιμοποιήσουμε στο σύστημα ερωταποκρίσεων και την εφαρμογή που χρειάστηκε να υλοποιήσουμε για να πετύχουμε την ενσωμάτωση της οντολογίας στο σύστημα.

Τέλος, στο πέμπτο κεφάλαιο εξετάζουμε τους τρόπους βελτίωσης του συστήματος, αρχικά με τη χρήση του crawler και στη συνέχεια με την περαιτέρω χρήση της οντολογίας.

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Θεόδωρο Καλαμπούκη για τις πολύτιμες συμβουλές του και την καθοδήγησή του κατά τη διάρκεια της εκπόνησης της εργασίας μου, καθώς και τον κ. Ίωνα Ανδρουτσόπουλο, που αποδέχθηκε τον ρόλο του εξωτερικού αξιολογητή. Θα ήθελα επίσης να ευχαριστήσω τους υπευθύνους του εργαστηρίου Τεχνολογιών Εκπαίδευσης από Απόσταση, Ιδεατής Πραγματικότητας και Πολυμέσων, Αντωνία Κυριακοπούλου, Παναγιώτη Μαρκόπουλο, Χρήστο Αμανατίδη, Γιώργο Κουρούπα, για όλους τους απαραίτητους πόρους που μου παρείχαν, καθώς και για την βοήθειά τους, όποτε την χρειάστηκα. Τέλος, θα ήθελα να ευχαριστήσω τον κ. Ιωάννη Ζωριό και τον συνάδελφο Σπύρο Καλλώνη για τις συμβουλές τους σχετικά με τις οντολογίες.

Περιεχόμενα

Κεφάλαιο 1ο: Οντολογίες

1.1 Εισαγωγή

Η πληροφορία στις μέρες μας έχει αναχθεί σε ένα από τα σπουδαιότερα αγαθά. Καθώς όμως μεγαλώνει η χρησιμότητα αλλά και η χρήση της, ταυτόχρονα μεγαλώνει και ο όγκος της άρα και η ανάγκη για σωστή διαχείρισή της. Οι άνθρωποι, οι οργανισμοί και τα συστήματα λογισμικού πρέπει να επικοινωνούν μεταξύ τους. Ωστόσο, λόγω διαφορετικών αναγκών και γνωσιακού επιπέδου, μπορεί να υπάρχουν διαφορετικές αντιλήψεις και όψεις για το ίδιο ουσιαστικά θέμα. Μπορεί να χρησιμοποιείται διαφορετική ορολογία, διαφορετική ή επικαλυπτόμενη μεθοδολογία και δομή. Συνεπώς η έλλειψη μίας *κοινής αντίληψης* μπορεί να οδηγήσει σε *φτωχή επικοινωνία* μεταξύ των ανθρώπων και των οργανισμών αυτών.

Όταν πρόκειται για την ανάπτυξη ενός συστήματος IT (Information Tecnology), αυτή η έλλειψη μιας κοινής αντίληψης οδηγεί σε δυσκολίες στην ανάλυση των απαιτήσεων, συνεπώς και στον καθορισμό των *προδιαγραφών* του συστήματος. Από την άλλη πλευρά, ετερόκλητες μέθοδοι μοντελοποίησης, γλώσσες και εργαλεία λογισμικού περιορίζουν σημαντικά την *διαλειτουργικότητα* (inter-operability) και την δυνατότητα *επαναχρησιμοποίησης* (re-use) και *διαμοιρασμού* (sharing). Ως αποτέλεσμα των προβλημάτων αυτών συχνά έχουμε *σπατάλη* κόπου και χρόνου στην προσπάθεια για την «εφεύρεση του τροχού».

Ο τρόπος να αντιμετωπίσουμε τα προβλήματα αυτά είναι να μειώσουμε ή να εξαλείψουμε την σημασιολογική και ονοματολογική σύγχυση και να καταλήξουμε σε μία *κοινή αντίληψη*. Μια τέτοια αντίληψη μπορεί να λειτουργήσει ως ένα *ενωτικό πλαίσιο* για τις διαφορετικές όψεις και να αποτελέσει την βάση για:

- **Επικοινωνία** μεταξύ *ανθρώπων* με διαφορετικές ανάγκες και προοπτικές, οι οποίες προκύπτουν από τα διαφορετικά τους γνωσιακά επίπεδα,

- **Διαλειτουργικότητα** μεταξύ *συστημάτων*, η οποία επιτυγχάνεται μέσω της μετάφρασης μεταξύ διαφορετικών μεθόδων μοντελοποίησης, γλωσσών και εργαλείων λογισμικού,

- **Πλεονεκτήματα τεχνολογίας λογισμικού**. Συγκεκριμένα:

    - *Επαναχρησιμοποίηση*: η κοινή αντίληψη είναι η βάση για μία επίσημη κωδικοποίηση των σημαντικών οντοτήτων, ιδιοχαρακτηριστικών, διαδικασιών και των μεταξύ τους συσχετίσεων στο πεδίο που μας ενδιαφέρει. Αυτή η επίσημη αναπαράσταση μπορεί να είναι (ή να γίνει μέσω αυτόματης μετάφρασης) ένα επαναχρησιμοποιήσιμο και διαμοιραζόμενο στοιχείο ενός συστήματος λογισμικού.

    - *Αξιοπιστία*: Μια επίσημη αναπαράσταση καθιστά επίσης δυνατή την αυτοματοποίηση ελέγχων συνάφειας, οδηγώντας σε πιο αξιόπιστο λογισμικό.

    - *Προδιαγραφές*: η κοινή αντίληψη μπορεί να βοηθήσει στην διαδικασία ανάλυσης απαιτήσεων άρα και στον καθορισμό των προδιαγραφών για ένα σύστημα IT. Αυτό ισχύει ιδιαίτερα στην περίπτωση που οι απαιτήσεις συμπεριλαμβάνουν διαφορετικές ομάδες, οι οποίες χρησιμοποιούν διαφορετική ορολογία για το ίδιο πεδίο ή για πολλαπλά πεδία.

Στο σημείο αυτό εισάγουμε την έννοια της οντολογίας, η οποία είναι και η λύση στο πρόβλημά μας. «Οντολογία» είναι ο όρος που αναφέρεται στην κοινή αντίληψη ενός πεδίου ενδιαφέροντος και που μπορεί να χρησιμοποιηθεί ως το ενωτικό πλαίσιο που αναφέραμε μέχρι τώρα.

Μία οντολογία απαραίτητα περιλαμβάνει ένα είδος θεώρησης του κόσμου, από την οπτική γωνία ενός δεδομένου πεδίου. Η οπτική αυτή συχνά περιγράφεται ως ένα σύνολο εννοιών (π.χ. οντοτήτων, ιδιοχαρακτηριστικών, διαδικασιών), ορισμών αυτών και μεταξύ τους

συσχετίσεων. Αυτό αναφέρεται στην βιβλιογραφία ως *εννοιοποίηση* (conceptualization).

Αυτή η εννοιοποίηση μπορεί να είναι αυτονόητη για κάποιο άτομο ή λογισμικό πακέτο, αλλά μη προφανής. Ο όρος 'οντολογία' μπορεί να χρησιμοποιηθεί για να περιγράψει μια τέτοια εννοιοποίηση, αλλά η συνηθισμένη χρήση του είναι μία *σαφής* αναπαράσταση μίας εννοιοποίησης.

Μία οντολογία μπορεί να έχει διάφορες μορφές, αλλά θα περιέχει οπωσδήποτε ένα λεξικό όρων και προδιαγραφές των εννοιών τους (δηλαδή ορισμούς). Το λεξικό αυτό και οι έννοιες μπορούν να εκφραστούν με διάφορους τρόπους, όσον αφορά την τυπικότητα:

- *Μη-τυπική έκφραση*: ελεύθερη έκφραση σε φυσική γλώσσα
- *Ημι-μη-τυπική έκφραση*: έκφραση σε μία περιορισμένη και δομημένη μορφή φυσικής γλώσσας, κάτι που αυξάνει σημαντικά την σαφήνεια
- *Ημι-τυπική έκφραση*: έκφραση σε μία τεχνητή τυπικά ορισμένη γλώσσα
- *Αυστηρά τυπική έκφραση*: διεξοδικά καθορισμένοι όροι με τυπική σημασιολογία, θεωρήματα και αποδείξεις ιδιοτήτων, όπως είναι η ορθότητα και η πληρότητα.

Στη συνέχεια θα εξετάσουμε λεπτομερέστερα τις οντολογίες, τις εφαρμογές τους και τις μεθοδολογίες ανάπτυξής τους.

1.2 Τι είναι οντολογία

1.2.1 Ιστορική αναδρομή

Ο όρος 'οντολογία' έχει χρησιμοποιηθεί εδώ και πολλά χρόνια. Για την ακρίβεια η πρώτη της αναφορά χρονολογείται στο 1721 και ορίζεται ως (1) *ένας κλάδος της μεταφυσικής που ασχολείται με τη φύση και τις*

συσχετίσεις των όντων και (2) *μία συγκεκριμένη θεωρία σχετικά με τη φύση των όντων ή τα υπαρκτά είδη*. Αυτοί οι ορισμοί παρέχουν μια αφαιρετική φιλοσοφική έννοια της οντολογίας. Ωστόσο και η έννοια της μαθηματικής και τυπικής οντολογίας έχει διαχωριστεί και αυτή εδώ και πολύ καιρό, περίπου από το 1900. Μέχρι πρόσφατα οι οντολογίες έχουν παραμείνει ένα πεδίο ακαδημαϊκού ενδιαφέροντος για φιλοσόφους, γλωσσολόγους, ερευνητές αναπαράστασης γνώσης κι αυτό συμβαίνει καθ' όλη τη διάρκεια της πολύχρονης ιστορίας τους.

Τα τελευταία χρόνια όμως, οι οντολογίες έχουν κερδίσει το ενδιαφέρον και την αποδοχή από κοινό με υπολογιστικό ενδιαφέρον και γνωσιακό επίπεδο. Μερικά από τα πεδία που οι οντολογίες βρίσκουν εφαρμογή είναι η αναπαράσταση γνώσης, η ποιοτική μοντελοποίηση, η τεχνολογία γλωσσών, ο σχεδιασμός βάσεων δεδομένων, η ανάκτηση πληροφορίας, η διαχείριση γνώσης, η βιβλιοθηκονομία, το ηλεκτρονικό εμπόριο, ο σημασιολογικός ιστός.

1.2.2 Ορισμός

Περιορίζοντας το ενδιαφέρον μας για τις οντολογίες στον ιστό και στην ανάκτηση πληροφορίας, στην υπάρχουσα βιβλιογραφία είναι πιθανό να βρούμε πολλούς ορισμούς για τον όρο 'οντολογία'. Αυτός που αναφέρεται πιο συχνά είναι ο ορισμός που προτάθηκε από τον Gruber και είναι ο εξής:

Μία οντολογία είναι μία τυπική, ρητή προδιαγραφή μίας διαμοιραζόμενης εννοιοποίησης.

Μία ανάλυση του ορισμού αυτού αναφέρει ότι 'εννοιοποίηση' είναι ένα αφαιρετικό μοντέλο ενός φαινομένου, η λέξη 'τυπική' σημαίνει μία ακριβή μαθηματική περιγραφή, ο όρος 'ρητή' εκφράζει την ακρίβεια των εννοιών και των σαφώς ορισμένων συσχετίσεων μεταξύ τους και τέλος, η λέξη 'διαμοιραζόμενη' δηλώνει την ύπαρξη μιας συμφωνίας μεταξύ των χρηστών της οντολογίας.

Ο ορισμός αυτός είναι αρκετά γενικός, ωστόσο οι οντολογίες μπορούν επίσης να οριστούν ως προς συγκεκριμένες έννοιες. Για

παράδειγμα, όσον αφορά τους agents, στην τεχνητή νοημοσύνη αναφέρεται ότι *μία οντολογία είναι μία επίσημη περιγραφή των εννοιών και των συσχετίσεων που μπορούν να υπάρξουν σε μία κοινότητα από agents.*

Επίσης η σπουδαιότητα των όρων μπορεί να διαφανεί στον ακόλουθο ορισμό: *μία οντολογία είναι ένα ιεραρχικά δομημένο σύνολο όρων, με σκοπό να περιγράψει έναν τομέα και το οποίο μπορεί να χρησιμοποιηθεί ως ένα αρχικό θεμέλιο για μία βάση γνώσης.*

Μία αρκετά περιληπτική αλλά και περιεκτική περιγραφή των οντολογιών, η οποία βρίσκεται στην λίστα ηλεκτρονικού ταχυδρομείου SRKB (Shared Reusable Knowledge Bases) είναι η παρακάτω:

Οι οντολογίες είναι συμφωνίες για διαμοιραζόμενες εννοιοποιήσεις. Οι διαμοιραζόμενες εννοιοποιήσεις περιλαμβάνουν εννοιολογικά πλαίσια για μοντελοποίηση γνώσης, συγκεκριμένου περιεχομένου πρωτόκολλα για επικοινωνία μεταξύ agents και συμφωνίες σχετικά με την αναπαράσταση θεωριών για συγκεκριμένα πεδία. Όσον αφορά τον διαμοιρασμό γνώσης, οι οντολογίες ορίζονται ως ορισμοί λεξιλογίου αναπαράστασης. Μια πολύ απλή περίπτωση θα ήταν μία ιεραρχία τύπων, στην οποία καθορίζονται οι κλάσεις και οι συσχετίσεις ταξινόμησης. Επίσης σχήματα σχεσιακών βάσεων δεδομένων μπορούν να λειτουργήσουν ως οντολογίες, καθορίζοντας τις συσχετίσεις που μπορούν να υπάρξουν σε μία διαμοιραζόμενη βάση δεδομένων και τους περιορισμούς ακεραιότητας που πρέπει να ισχύουν για αυτές.

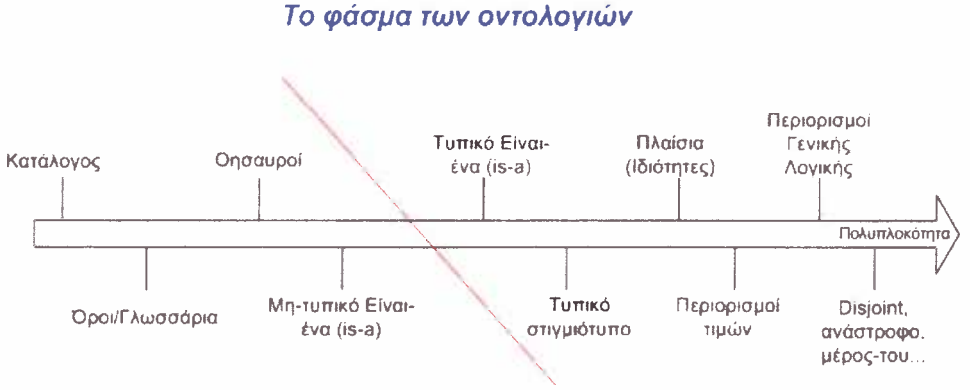
Υπάρχουν και αρκετοί ακόμα ορισμοί, οι οποίοι μάλιστα είναι νεότεροι αλλά δεν θα τους αναφέρουμε, πάρα μόνο θα εστιάσουμε στα βασικά στοιχεία των οντολογιών, όπως αυτά εξάγονται από όλους τους ορισμούς που έχουν δοθεί για αυτές.

- Οι οντολογίες χρησιμοποιούνται για να περιγράψουν ένα συγκεκριμένο πεδίο
- Οι όροι και οι συσχετίσεις είναι σαφώς ορισμένοι στο πεδίο αυτό

- Υπάρχει ένας μηχανισμός οργάνωσης των όρων (συνήθως μία ιεραρχική δομή, καθώς και συσχετίσεις IS-A, HAS-A)
- Υπάρχει μια συμφωνία μεταξύ των χρηστών μιας οντολογίας, ούτως ώστε το νόημα των όρων να χρησιμοποιείται με συνέπεια.

1.2.3 Το φάσμα των οντολογιών

Στη συνέχεια θα εξετάσουμε το φάσμα των οντολογιών, όπως αυτό παρουσιάστηκε στο συνέδριο AAAI '99 ώστε να κατανοήσουμε πλήρως τι ακριβώς μπορεί να είναι μία οντολογία (Εικόνα 1).



Εικόνα 1: Το φάσμα των οντολογιών. Ό,τι θεωρείται οντολογία (με την έννοια της αυτόματης εξαγωγής συμπερασμάτων) βρίσκεται στα δεξιά της κάθετης γραμμής.

Η απλούστερη μορφή μιας πιθανής οντολογίας είναι ένα ελεγχόμενο λεξιλόγιο (vocabulary), δηλαδή μία πεπερασμένη λίστα όρων. Σε αυτήν την κατηγορία ανήκουν όλων των ειδών οι κατάλογοι. Οι κατάλογοι έχουν τη δυνατότητα να παρέχουν μία ρητή ερμηνεία για κάθε όρο.

Παρόμοιο με το λεξιλόγιο είναι το ονοματολόγιο (glossary), το οποίο είναι μια πεπερασμένη λίστα όρων και εννοιών. Οι έννοιες καθορίζονται ως δηλώσεις σε φυσική γλώσσα. Αυτό παρέχει ένα είδος σημασιολογίας καθώς οι άνθρωποι έχουν τη δυνατότητα να διαβάζουν τις δηλώσεις αυτές και να

τις ερμηνεύουν. Συνήθως όμως αυτές δεν είναι ρητά ορισμένες, οπότε οι προδιαγραφές αυτές δεν είναι κατάλληλες για επεξεργασία από υπολογιστές.

Οι θησαυροί όρων (thesaurus) παρέχουν κάποια επιπλέον σημασιολογία, όσον αφορά τις συσχετίσεις μεταξύ των όρων. Για παράδειγμα παρέχουν πληροφορία σχετικά με τα συνώνυμα των όρων. Σε πολλές περιπτώσεις οι θησαυροί μπορούν να διερμηνευτούν με σαφήνεια από computer agents, αλλά τυπικά δεν παρέχουν κάποια ιεραρχική δομή.

Οι πρώτες προδιαγραφές ιεραρχιών όρων που παρουσιάστηκαν στον ιστό δεν είχαν ακριβή ιεραρχία is-a (είναι-ένα). Για παράδειγμα μία ταξονομία του Yahoo αναφέρει ότι η κατηγορία «Ενδυμασία» περιλαμβάνει την υποκατηγορία «Γυναίκα» η οποία περιλαμβάνει τις υποκατηγορίες «Φορέματα» και «Αξεσουάρ». Είναι προφανές ότι το φόρεμα δεν είναι γυναίκα, έτσι η (ρητά ορισμένη κατά τα άλλα) ιεραρχία δεν είναι μια ακριβής is-a ιεραρχία. Έτσι ενώ θα μπορούσαμε να τη θεωρήσουμε ένα είδος οντολογίας, στην πράξη παρουσιάζονται προβλήματα με τέτοιου είδους ταξονομίες. Αυτό είναι που φαίνεται και στο σχήμα σαν οντολογία άτυπης μορφής is-a.

Στη συνέχεια έχουμε τις αυστηρά καθορισμένες ιεραρχίες υποκλάσεων. Στα συστήματα αυτά αν η κλάση Α είναι υπερκλάση της Β, τότε αν ένα αντικείμενο είναι στιγμιότυπο της Β, θα είναι και στιγμιότυπο της Α. Αυτού του είδους οι ιεραρχίες είναι απαραίτητες για να χρησιμοποιήσουμε την κληρονομικότητα. Το επόμενο σημείο του φάσματος των οντολογιών είναι οι επίσημες συσχετίσεις στιγμιότυπων. Κάποια σχήματα κατηγοριοποίησης περιλαμβάνουν μόνο ονόματα κλάσεων, ενώ κάποια άλλα περιλαμβάνουν και βασικό ανεξάρτητο περιεχόμενο. Τα συγκεκριμένα σχήματα περιλαμβάνουν και τα στιγμιότυπα.

Το επόμενο σημείο είναι τα Πλαίσια. Σε αυτό οι κλάσεις περιλαμβάνουν πληροφορίες για τις ιδιότητες. Οι ιδιότητες είναι ιδιαίτερα χρήσιμες όταν καθορίζονται σε ένα γενικότερο επίπεδο της κλάσης και στη συνέχεια κληρονομούνται από τις κατάλληλες υποκλάσεις και στιγμιότυπα.

Μία πιο εκφραστική κατηγορία στο φάσμα είναι οι περιορισμοί τιμών (value restrictions). Εδώ μπορούμε να τοποθετήσουμε περιορισμούς στις ιδιότητες των κλάσεων. Για παράδειγμα η ιδιότητα «Μισθός» θα μπορεί να πάρει τιμές μόνο αριθμούς ή αριθμούς συγκεκριμένου πεδίου τιμών.

Καθώς προχωράμε παραπέρα στο φάσμα, οι οντολογίες πρέπει να αναπαριστούν όλο και περισσότερη πληροφορία. Για παράδειγμα αν έχουμε μια ιδιότητα που βασίζεται σε μια μαθηματική εξίσωση, μπορεί να θέλουμε να τις δώσουμε τιμές από άλλες ιδιότητες. Σε κάποιες γλώσσες επιτρέπεται να κάνουμε αυθαίρετα λογικές δηλώσεις ή να ορίζουμε περιορισμούς σε λογική πρώτης τάξης ανάμεσα σε όρους, ακόμα και να ορίζουμε πιο λεπτομερείς συσχετίσεις όπως είναι οι disjoint κλάσεις, οι ανάστροφες συσχετίσεις κ.α.

Για να είναι κάτι οντολογία θα θεωρήσουμε ότι πρέπει να έχει τις ακόλουθες ιδιότητες. Οι πρώτες τρεις που θα αναφέρουμε ορίζουν τις απλές οντολογίες:

- Πεπερασμένο, ελεγχόμενο (και επεκτάσιμο) λεξιλόγιο
- Σαφή ερμηνεία των κλάσεων και των συσχετίσεων των όρων
- Αυστηρές, ιεραρχικές συσχετίσεις υποκλάσης μεταξύ των κλάσεων

Θεωρούμε τις παρακάτω ιδιότητες τυπικές αλλά όχι υποχρεωτικές:

- Προδιαγραφές ιδιοτήτων ανά κλάση
- Καταμέτρηση οντοτήτων στην οντολογία
- Προδιαγραφές περιορισμών τιμών ανά κλάση

Τέλος, οι παρακάτω ιδιότητες μπορεί να είναι επιθυμητές, χωρίς όμως να είναι συνηθισμένες, ούτε υποχρεωτικές:

- Προδιαγραφές των disjoint κλάσεων

- Προδιαγραφές των αυθαίρετων λογικών συσχετίσεων μεταξύ των όρων
- Διακεκριμένες συσχετίσεις όπως η αναστροφή και το μέρος – όλο (part – whole)

Η γραμμή στην Εικόνα 1 είναι σχεδιασμένη έτσι ώστε καθετί που βρίσκεται στα δεξιά της πληροί τουλάχιστον τις τρεις πρώτες προϋποθέσεις, οπότε και μπορεί να αποκαλείται οντολογία.

1.3 Χρήσεις των οντολογιών

1.3.1 Τομείς ενδιαφέροντος

Οι οντολογίες μπορούν να κατηγοριοποιηθούν ανάλογα με τον τομέα στον οποίο χρησιμοποιούνται. Στη βιβλιογραφία υπάρχουν πολλές απόψεις σχετικά με τους σκοπούς που επιτελούν οι οντολογίες. Από μια σκοπιά υψηλού επιπέδου μπορούμε να δούμε τις οντολογίες ως ένα μέσο επαναχρησιμοποίησης. Από μια άλλη πλευρά οι οντολογίες εξαρτώνται από το λογισμικό με το οποίο θα χρησιμοποιηθούν, είτε πρόκειται να χρησιμοποιηθεί από μια μικρή ομάδα χρηστών είτε από μια μεγάλη κοινότητα. Επίσης, οι οντολογίες μπορεί να είναι κυρίως ένα μέσο οργάνωσης μιας βάσης γνώσης ή ακόμα κι ένα μέρος μιας βάσης γνώσης. Κάποιοι άλλοι τις θεωρούν ένα διαγλωσσικό εργαλείο σχεδιασμένο για μια συγκεκριμένη κάθε φορά εφαρμογή.

Μια άλλη σημαντική χρήση των οντολογιών είναι η ενσωμάτωση διαφορετικών πεδίων σε ένα συνεκτικό δίκτυο. Αυτό εμφανίζεται στην τεχνολογία επιχειρηματικών διαδικασιών, σε κατανεμημένες αρχιτεκτονικές πολλαπλών agents και στην ταυτόχρονη (concurrent) μηχανική και σχεδιασμό.

Σύμφωνα με όλα αυτά χωρίζουμε τον χώρο χρήσεων των οντολογιών στις εξής κατηγορίες:

- **Επικοινωνία**. Οι οντολογίες, όπως έχουμε αναφέρει, μειώνουν την εννοιολογική σύγχυση παρέχοντας ένα ενωτικό πλαίσιο μέσα σε έναν οργανισμό. Έτσι καθιστούν δυνατή μια κοινή κατανόηση και επικοινωνία μεταξύ ατόμων με διαφορετικές ανάγκες και απόψεις, οι οποίες οφείλονται στο διαφορετικό υπόβαθρο του καθένα.

    - *Κανονικοποιημένο μοντέλο*. Πιο συγκεκριμένα, οι οντολογίες βοηθάνε στη δημιουργία ενός κανονικοποιημένου μοντέλου ενός συστήματος, το οποίο παρέχει μια σημασιολογία για το σύστημα και ένα εύκολα επεκτάσιμο μοντέλο, που επιτρέπει σημασιολογικούς μετασχηματισμούς μεταξύ διαφορετικών εννοιών.

    - *Δίκτυο συσχετίσεων*. Επίσης μπορούμε να δημιουργήσουμε ένα δίκτυο συσχετίσεων, και στη συνέχεια να το παρακολουθούμε και να το διατρέχουμε. Ένα τέτοιο δίκτυο είναι ρητά καθορισμένο σε ένα σύστημα, αλλά συχνά οι χρήστες έχουν διαφορετικές θεωρήσεις. Στην περίπτωση αυτή οι οντολογίες θα χρησιμεύσουν για να δηλώσουν τους λογικούς δεσμούς μεταξύ των στοιχείων κάθε μοντέλου σε ένα σύστημα.

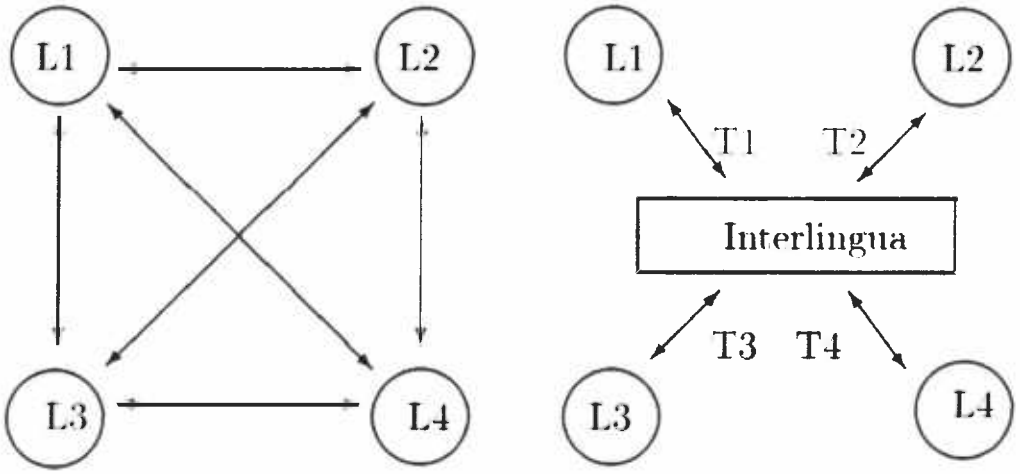
    - *Συνοχή και έλλειψη ασάφειας*. Ένα βασικό χαρακτηριστικό των οντολογιών είναι ότι παρέχουν συνοχή και έλλειψη ασάφειας. Ακόμα και στην περίπτωση που ο χρήστης χρησιμοποιεί διαφορετική οντολογία από το λογισμικό, θα πρέπει να υπάρχει ένα περιβάλλον που αναπαριστά τις διαφορετικές έννοιες των όρων και πώς αυτές αντιστοιχούνται.

    - *Ολοκλήρωση των διαφορετικών απόψεων των χρηστών*. Σε ένα σύστημα με πολλούς επικοινωνούντες agents είναι ζωτικής σημασίας να υπάρχει ολοκλήρωση

των διαφορετικών απόψεων των χρηστών. Για παράδειγμα σε μία εταιρεία άλλη οπτική έχει ένας εργαζόμενος για την εταιρεία, για το σκοπό της, για τη λειτουργία της και άλλη ένας υψηλά ιστάμενος. Μια οντολογία μπορεί να βοηθήσει στην επίτευξη της ολοκλήρωσης αυτής, παρέχοντας μια κοινή συμφωνία μεταξύ των διαφορετικών χρηστών.

- **Διαλειτουργικότητα (Interoperability)**. Σε πολλές εφαρμογές των οντολογιών εμφανίζεται το θέμα της διαλειτουργικότητας, κατά το οποίο έχουμε διαφορετικούς χρήστες που θέλουν να ανταλλάξουν δεδομένα είτε που χρησιμοποιούν διαφορετικά εργαλεία λογισμικού. Ένα περιβάλλον που παρέχει την απαραίτητη ολοκλήρωση των διαφορετικών αυτών εργαλείων, συνήθως είναι απαραίτητο σε τομείς όπως η επιχειρηματική μοντελοποίηση και οι αρχιτεκτονικές πολλαπλών agents.

    - *Οι οντολογίες ως διαγλωσσικά εργαλεία*. Για να συμβάλλουν στην διαλειτουργικότητα οι οντολογίες μπορούν να χρησιμοποιηθούν ως μέσα μετάφρασης μεταξύ διαφορετικών γλωσσών και αναπαραστάσεων. Μια πρόταση είναι να έχουμε μια οντολογία που λειτουργεί σαν διαγλωσσικό εργαλείο. Αν είχαμε *n* διαφορετικές οντολογίες θα χρειαζόμασταν $O(n^2)$ μεταφραστές αλλά με την κεντρική διαγλωσσική οντολογία ο αριθμός αυτός μειώνεται σε $O(n)$ (Εικόνα 2).



Εικόνα 2: Με τη χρήση μιας οντολογίας ως διαγλωσσικό εργαλείο χρειαζόμαστε μόνο $O(n)$ μεταφραστές αντί για $O(n^2)$, για τη μετάφραση από τη γλώσσα L_i στη γλώσσα L_j.

- *Διαστάσεις διαλειτουργικότητας*. Εκτός από τα εργαλεία πρέπει να κάνουμε και κάποιες άλλες διακρίσεις. Πρέπει να λάβουμε υπόψιν μας τις σχέσεις μεταξύ των χρηστών που διαμοιράζονται εργαλεία και δεδομένα. Είναι σημαντικό οι οντολογίες και τα εργαλεία που χρησιμοποιούνται από διαφορετικούς χρήστες και agents στην ίδια εταιρεία να είναι διαμοιράσιμες και επαναχρησιμοποιήσιμες μεταξύ των διαφορετικών αυτών οντοτήτων.

  - Εσωτερική διαλειτουργικότητα: όλα τα συστήματα που απαιτούν διαλειτουργικότητα, βρίσκονται υπό τον άμεσο έλεγχο μιας οργανωτικής μονάδας.

  - Εξωτερική διαλειτουργικότητα: υπάρχει μια οργανωτική μονάδα, η οποία επιθυμεί να απομονωθεί από αλλαγές που γίνονται σε αυτήν από το εξωτερικό περιβάλλον.

- Ολοκλήρωση οντολογιών μεταξύ πεδίων: ένα άλλο στοιχείο της διαλειτουργικότητας είναι η ολοκλήρωση οντολογιών διαφορετικών πεδίων, με σκοπό την υποστήριξη μιας λειτουργίας.

- Ολοκλήρωση οντολογιών μεταξύ εργαλείων: από την άλλη, μπορεί να θέλουμε να ολοκληρώσουμε διαφορετικές οντολογίες στο *ίδιο* πεδίο, που όμως χρησιμοποιούνται από διαφορετικά εργαλεία. Έτσι για να έχουμε διαλειτουργικότητα πρέπει να έχουμε μια κοινή οντολογία για όλα τα εργαλεία, πράγμα που αποτελεί την μεγαλύτερη πρόκληση στην χρήση των οντολογιών αφού τα εργαλεία δεν είναι υποχρεωμένα να υποστηρίζουν την ολοκλήρωση των οντολογιών.

- **Τεχνολογία συστημάτων**. Όσον αφορά τα συστήματα λογισμικού οι οντολογίες μπορούν να συμβάλουν στους εξής τομείς:

    - *Προδιαγραφές*. Μια κοινή κατανόηση του προβλήματος μπορεί να βοηθήσει στον καθορισμό των προδιαγραφών ενός συστήματος. Η οντολογία μπορεί να βοηθήσει σε διαφορετικό βαθμό ανάλογα με τον βαθμό τυπικότητας και αυτοματισμού που χρησιμοποιήθηκαν στον σχεδιασμό του συστήματος. Σε μια μη-τυπική προσέγγιση οι οντολογίες μπορούν να βοηθήσουν στην διαδικασία ανάλυσης των απαιτήσεων του συστήματος και στην διαδικασία κατανόησης των συσχετίσεων μεταξύ των συστατικών του. Στην τυπική προσέγγιση οι οντολογίες παρέχουν μια ρητή προδιαγραφή του συστήματος, το οποίο μας επιτρέπει να κατανοήσουμε για ποιον σκοπό είναι σχεδιασμένο το σύστημα και όχι πώς επιτελείται η λειτουργικότητα αυτή.

- *Αξιοπιστία.* Οι μη-τυπικές οντολογίες μπορούν να συμβάλλουν στην αξιοπιστία του συστήματος, παρέχοντας έναν τρόπο να ελέγχει κάποιος τον σχεδιασμό του συστήματος και τη συνάφειά του με τις προδιαγραφές. Οι τυπικές οντολογίες, ανάλογα, μπορούν να κάνουν το ίδιο πράγμα αλλά με (ημι-) αυτοματοποιημένο τρόπο και όχι χειρωνακτικό, όπως πριν.

- *Επαναχρησιμοποίηση.* Για να είναι αποδοτικές οι οντολογίες πρέπει να υποστηρίζουν και την επαναχρησιμοποίηση, έτσι ώστε να υπάρχει η δυνατότητα εισαγωγής και εξαγωγής τμημάτων από ένα σύστημα λογισμικού σε ένα άλλο. Το πρόβλημα είναι ότι όταν ένα κομμάτι λογισμικού μεταφέρεται από ένα σύστημα σε ένα άλλο η συμπεριφορά του δεν είναι πάντα η αναμενόμενη, καθώς έχει σχεδιασθεί σε ένα διαφορετικό σύστημα. Οι οντολογίες, με τη δυνατότητα που έχουν να χαρακτηρίζουν τις κλάσεις των πεδίων και των εργασιών μεταξύ των πεδίων αυτών, παρέχουν ένα πλαίσιο για τον καθορισμό των κομματιών μιας οντολογίας, τα οποία να είναι επαναχρησιμοποιήσιμα.

1.3.2 Πρακτικές εφαρμογές των οντολογιών

1.3.2.1 Απλές οντολογίες

Αρχικά θα μελετήσουμε τις περιπτώσεις στις οποίες βρίσκουν εφαρμογή οι απλές οντολογίες και στη συνέχεια θα ασχοληθούμε με πιο σύνθετες οντολογίες. Εμπειρικά ορίζουμε τις απλές οντολογίες ως οντολογίες που περιέχουν μόνο ιεραρχίες κλάσεων και όρους. Σύνθετες ορίζουμε αυτές που περιλαμβάνουν επιπλέον ιδιότητες των κλάσεων και συσχετίσεις μεταξύ τους.

Οι απλές οντολογίες κατ' αρχάς παρέχουν ένα **ελεγχόμενο λεξιλόγιο**. Αυτό είναι ένα μεγάλο πλεονέκτημα καθώς οι χρήστες θα έχουν στη διάθεσή τους το ίδιο σύνολο όρων. Επίσης τα προγράμματα θα μπορούν να δημιουργούν διεπαφές που θα βοηθούν στη χρήση των συγκεκριμένων όρων.

Μια απλή οντολογία μπορεί να χρησιμοποιηθεί επίσης για **οργάνωση και πλοήγηση σε ιστοσελίδες**. Πολλές ιστοσελίδες σήμερα στο αριστερό μέρος εμφανίζουν τα πάνω επίπεδα μιας γενικής ιεραρχίας και επιλέγοντας κάποιο από τα θέματα εμφανίζονται οι υποκατηγορίες του (π.χ. υπολογιστές που αναλύονται σε software, hardware, περιφερειακά τα οποία αναλύονται περαιτέρω σε άλλες υποκατηγορίες).

Στον ίδιο χώρο οι οντολογίες μπορούν να συντελέσουν στον **ορισμό της προσδοκίας** του χρήστη. Δηλαδή εξερευνώντας τα πάνω μέρη της ιεραρχίας μιας ιστοσελίδας ο χρήστης ξέρει τι να περιμένει από αυτήν και αν το περιεχόμενό της τον ικανοποιεί.

Οι οντολογίες μπορούν επίσης να χρησιμεύσουν ως **δομές «ομπρέλα»** από τις οποίες να ανακτά κανείς περιεχόμενο. Υπάρχουν κάποιες οντολογίες που διανέμονται ελεύθερα και επιχειρούν να παρέχουν μια ταξονομική ιεραρχία υψηλού επιπέδου, την οποία μπορεί να χρησιμοποιήσει κάποιος άλλος για να εξάγει όρους. Αν και μια εφαρμογή θα χρειαστεί να επεκτείνει αυτού του είδους τις οντολογίες, η χρήση τους βοηθάει στην επικοινωνία, καθώς παρέχουν μια κοινή ιεραρχία υψηλού επιπέδου.

Στον τομέα των ιστοσελίδων μέσω των οντολογιών μπορεί να βελτιωθεί η **υποστήριξη browsing**. Το περιεχόμενο μιας ιστοσελίδας μπορεί να αντιστοιχιστεί με μετα-πληροφορίες οι οποίες στη συνέχεια μπορούν να χρησιμοποιηθούν από μηχανές αναζήτησης ώστε να βελτιωθεί η αναζήτηση.

Επίσης οι οντολογίες μπορούν να βοηθήσουν στην **υποστήριξη αναζήτησης**. Σε εφαρμογές που υποστηρίζουν αναζήτηση μέσω επερωτήσεων μια οντολογία μπορεί να ενσωματωθεί και να εμπλουτίζει κάθε φορά την επερώτηση του χρήστη με όρους από πιο συγκεκριμένες

κατηγορίες. Έτσι η διαδικασία της αναζήτησης μπορεί να βελτιωθεί σημαντικά.

Τέλος, οι απλές οντολογίες μπορούν να αποτελέσουν μέρος της **υποστήριξης αποσαφήνισης εννοιών** (sense disambiguation). Ένας όρος μπορεί να εμφανίζεται πολλές φορές σε μία ταξονομία, οπότε μπορούμε να ελέγξουμε το παραπάνω επίπεδο στην ιεραρχία και να βρούμε ποια έννοια είναι αυτή που αναζητείται. Για παράδειγμα αν ένας χρήστης ψάχνει για τη λέξη «ελιά» μπορεί να ψάχνει είτε για το δέντρο είτε για τον καρπό. Με μια αναζήτηση στο πάνω μέρος του κάθε όρου μπορεί να εμφανιστεί ένα μήνυμα στον χρήστη το οποίο να τον ρωτάει για τι ακριβώς ψάχνει (δέντρα ή καρπούς).

1.3.2.2 Σύνθετες οντολογίες

Όταν οι οντολογίες γίνουν πιο σύνθετες μπορούμε να χρησιμοποιήσουμε στις εφαρμογές πληροφορίες σχετικά με ιδιότητες των κλάσεων και γενικά έχουμε περισσότερες δυνατότητες.

Αρχικά οι σύνθετες οντολογίες μπορούν να χρησιμοποιηθούν για απλούς **ελέγχους συνέπειας**. Αν μια οντολογία περιέχει πληροφορία για ιδιότητες και για περιορισμούς τιμών πάνω σε αυτές τότε μπορούμε να κάνουμε κάποιους τυπικούς ελέγχους συνέπειας. Για παράδειγμα αν μία κλάση «Υπάλληλος» έχει μια ιδιότητα που λέγεται «Μισθός», η οποία έχει περιορισμό τιμής σε αριθμούς, θα μπορούμε να αναγνωρίσουμε ως λάθος την προσπάθεια εισαγωγής μιας τιμής που δεν είναι αριθμός. Με το ίδιο σκεπτικό αν μια ιδιότητα έχει περιορισμό τιμών από 1 έως 100 θα μπορούμε να απορρίψουμε μια τιμή που δεν βρίσκεται στο πεδίο αυτό.

Μια δεύτερη εφαρμογή είναι η παροχή **ολοκλήρωσης**. Δηλαδή από κάποιες γενικές πληροφορίες που μας παρέχει ο χρήστης να μπορούμε να εξάγουμε κάποια συμπεράσματα τα οποία θα βοηθήσουν στην περαιτέρω πλοήγηση του χρήστη στην εφαρμογή. Για παράδειγμα στην αναζήτηση για κάποια φωτογραφική μηχανή ο χρήστης μπορεί αρχικά να επιλέξει μεταξύ «Υψηλής ανάλυσης» και «Χαμηλής Ανάλυσης», τα οποία να αντιστοιχούν σε συγκεκριμένες τιμές που στη συνέχεια να παρουσιάζονται στον χρήστη.

Οι οντολογίες μπορούν επίσης να παρέχουν **υποστήριξη διαλειτουργικότητας**. Ακόμα και με τις απλές οντολογίες ισχύει αυτό καθώς όπως είπαμε οι εφαρμογές χρησιμοποιούν ένα κοινό σύνολο όρων. Με τις σύνθετες οντολογίες όμως είναι ακόμα περισσότερες οι δυνατότητες αφού μπορούμε να χρησιμοποιήσουμε αξιώματα και αντιστοιχίσεις μεταξύ όρων και ιδιοτήτων. Για παράδειγμα μπορούμε να εξάγουμε το συμπέρασμα ότι η ιδιότητα «ΥπάλληλοςΑΣΟΕΕ» αντιστοιχεί σε ένα «Άτομο» του οποίου η ιδιότητα «Εργοδότης» έχει την τιμή «ΑΣΟΕΕ». Έτσι μια εφαρμογή που δεν αναγνωρίζει τους όρους «ΥπάλληλοςΑΣΟΕΕ» ή «Υπάλληλος» (αλλά αναγνωρίζει τους όρους «Άτομο», «Εργοδότης», «ΑΣΟΕΕ») θα μπορεί να επικοινωνήσει με μια άλλη που τους αναγνωρίζει, μέσω της ανάλυσης αυτής.

Με τη βοήθεια μιας οντολογίας μπορούμε επίσης να παρέχουμε **υποστήριξη ελέγχου εγκυρότητας** δεδομένων και σχημάτων. Αν μια οντολογία περιέχει ορισμούς κλάσεων, αυτοί μπορούν να χρησιμοποιηθούν ως επερωτήσεις σε μια βάση δεδομένων ώστε να ελέγξουμε κατά πόσο η βάση αυτή είναι ολοκληρωμένη ή αν έχει ελλείψεις. Για παράδειγμα για να είναι κάποιος «ΥπάλληλοςΑΣΟΕΕ» πρέπει να έχει ακριβώς έναν εργοδότη, κάτι που μπορεί να ελεγχθεί με μία επερώτηση στη βάση δεδομένων.

Όμοια, αν μια οντολογία περιέχει πληροφορία markup μπορεί να χρησιμεύσει για τη **δημιουργία ολόκληρων εφαρμογών ελέγχου**. Για παράδειγμα μια οντολογία μπορεί να έχει έναν μεγάλο αριθμό από ορισμούς όρων και από ορισμούς στιγμιοτύπων των κλάσεων. Αν έχει και έναν ορισμό ο οποίος θεωρείται επερώτηση (π.χ. βρες όλα τα αντικείμενα που πληρούν τις παρακάτω προϋποθέσεις), τότε στον ορισμό αυτόν θα μπορούσε να ενσωματωθεί markup πληροφορία η οποία να υπαγορεύει τι είδους θα είναι η αναμενόμενη απάντηση. Έτσι με την επερώτηση αυτή μπορούμε να ελέγξουμε τα δεδομένα.

Πολύ σημαντική είναι η δυνατότητα που μας παρέχουν οι οντολογίες για **υποστήριξη σύνθεσης** (configuration). Για παράδειγμα σε μία ιστοσελίδα που ασχολείται με πωλήσεις ηλεκτρονικών υπολογιστών θα μπορούσε να ενσωματωθεί μία οντολογία η οποία να παρέχει τη δυνατότητα να καθορίζονται λεπτομέρειες όπως ποιοι επεξεργαστές

ταιριάζουν με ποιες μητρικές πλακέτες. Έτσι ο χρήστης επιλέγοντας έναν επεξεργαστή, αυτόματα θα εμφανίζονται οι μητρικές πλακέτες που είναι διαθέσιμες και συμβατές με τον επιλεγμένο επεξεργαστή.

Παρόμοια εφαρμογή με την προηγούμενη είναι η **υποστήριξη δομημένης, συγκριτικής και εξατομικευμένης αναζήτησης**. Για παράδειγμα ένας χρήστης μπορεί να εισάγει τις προδιαγραφές για μια τηλεόραση την οποία θα ήθελε να αγοράσει και το σύστημα να του εμφανίσει τα μοντέλα που είναι πιο κοντά στις απαιτήσεις του. Επίσης οι ιδιότητες αυτές των τηλεοράσεων που θα βρίσκονται στην οντολογία θα μπορούσαν να χρησιμοποιηθούν για να εμφανίζουν μια σύγκριση μεταξύ διαφόρων μοντέλων έτσι ώστε να είναι πιο αποδοτική η αναζήτηση του χρήστη.

Σχετικά με κάποια αναζήτηση μια οντολογία θα μπορούσε να χρησιμοποιηθεί για την **παροχή πληροφορίας γενίκευσης ή ειδίκευσης**. Για παράδειγμα αν σε μια εφαρμογή αναζήτησης η επερώτηση του χρήστη έχει πολλά αποτελέσματα μπορεί η εφαρμογή να χρησιμοποιήσει μια οντολογία για να δει αν η επερώτηση μπορεί να εξειδικευτεί. Έτσι τα αποτελέσματα μπορεί να γίνουν λιγότερα και πιο συγκεκριμένα ως προς τις απαιτήσεις του χρήστη. Για παράδειγμα αν κάποιος ψάχνει για συναυλίες, η εφαρμογή θα μπορούσε να τον ρωτήσει αν ψάχνει για rock, jazz ή άλλες συναυλίες οπότε τα αποτελέσματα θα είναι λιγότερα και πιο συγκεκριμένα.

Είδαμε κάποιες από τις πιο συνηθισμένες χρήσεις των απλών αλλά και των σύνθετων οντολογιών, χωρίς αυτό να σημαίνει ότι δεν μπορούν να χρησιμοποιηθούν σε άλλες εφαρμογές. Το φάσμα των χρήσεων των οντολογιών είναι ήδη πολύ μεγάλο και συνεχώς επεκτείνεται.

1.4 Μεθοδολογία για τη δημιουργία οντολογιών

Μια από τις πρώτες μεθοδολογίες που προτάθηκαν για τη δημιουργία οντολογιών είναι των Uschold και Gruninger και αποτελείται από τα παρακάτω βήματα:

- Καθορισμός του στόχου

- Δημιουργία της οντολογίας
    - Σύλληψη της οντολογίας
    - Κωδικοποίηση της οντολογίας
    - Ενσωμάτωση οντολογιών που υπάρχουν ήδη
- Αξιολόγηση
- Τεκμηρίωση
- Οδηγίες για κάθε φάση

Στη συνέχεια περιγράφουμε συνοπτικά καθεμία από τις φάσεις αυτές ώστε να έχουμε έναν σύντομο οδηγό για τη δημιουργία οντολογιών.

1.4.1 Καθορισμός του στόχου

Είναι πολύ σημαντικό να γνωρίζουμε για ποιον σκοπό θα δημιουργήσουμε την οντολογία. Μια καλή αρχή είναι οι ήδη υπάρχουσες εφαρμογές των οντολογιών, τις οποίες εξετάσαμε στην προηγούμενη παράγραφο. Εξίσου χρήσιμα συμπεράσματα ίσως μας προσφέρει ο καθορισμός της ομάδας χρηστών της οντολογίας.

1.4.2 Δημιουργία της οντολογίας

Αφού καθοριστεί ο σκοπός της οντολογίας έχουμε μια καλή βάση για να ξεκινήσουμε τη δημιουργία της. Τα στάδια που περιλαμβάνονται σε αυτή τη φάση είναι η σύλληψη, η κωδικοποίηση και η ενσωμάτωση έτοιμων οντολογιών.

1.4.2.1 Σύλληψη

Το στάδιο της σύλληψης της οντολογίας περιλαμβάνει τα εξής:

- τον καθορισμό των βασικών εννοιών και συσχετίσεων στο πεδίο ενδιαφέροντος,
- την παραγωγή ρητών ορισμών για τις έννοιες και τις συσχετίσεις αυτές,

- τον καθορισμό όρων που να αναφέρονται σε αυτές τις έννοιες και συσχετίσεις
- και τέλος τη συμφωνία των δημιουργών πάνω στα προηγούμενα.

1.4.2.2 Κωδικοποίηση

Με την έννοια της κωδικοποίησης εννοούμε την ακριβή αναπαράσταση της εννοιοποίησης που συλλάβαμε στο προηγούμενο στάδιο, σε μια επίσημη γλώσσα. Το στάδιο αυτό περιλαμβάνει τα εξής:

- τη συμφωνία σχετικά με τους όρους που θα αναπαραστήσουν την οντολογία (π.χ. κλάση, οντότητα, συσχέτιση). Αυτό συχνά ονομάζεται «μετα-οντολογία» καθώς στην ουσία είναι η οντολογία που βρίσκεται στο χαμηλότερο επίπεδο και αναπαριστά τον τρόπο έκφρασης της κύριας οντολογίας,
- επιλογή μιας γλώσσας αναπαράστασης, η οποία θα πρέπει να υποστηρίζει και την μετα-οντολογία,
- και τέλος τη συγγραφή του κώδικα.

Συχνά το στάδιο της σύλληψης και της κωδικοποίησης πραγματοποιούνται ταυτόχρονα σε ένα στάδιο. Μάλιστα κάποια εργαλεία δημιουργίας οντολογιών το θεωρούν αυτό δεδομένο. Το πώς θα εργαστεί κανείς αφήνεται στην κρίση του, συνήθως όμως είναι καλύτερα να διαχωριστούν τα δύο στάδια.

1.4.2.3 Ενσωμάτωση έτοιμων οντολογιών

Κατά τη διάρκεια των προηγούμενων σταδίων συχνά ανακύπτει το ερώτημα κατά πόσο μπορούμε να χρησιμοποιήσουμε οντολογίες που υπάρχουν ήδη, ώστε να βοηθήσουν στον σκοπό μας. Το πρόβλημα αυτό είναι πολύ δύσκολο και μέχρι σήμερα αρκετή έρευνα γίνεται στον τομέα αυτό.

Έχουν δημιουργηθεί αρκετές οντολογίες υψηλού επιπέδου που προσπαθούν να περιγράψουν κάποιες συγκεκριμένες έννοιες όπως χρόνος, χώρος, ύλη, δράση, γεγονός, οι οποίες να μην εξαρτώνται από κάποιο

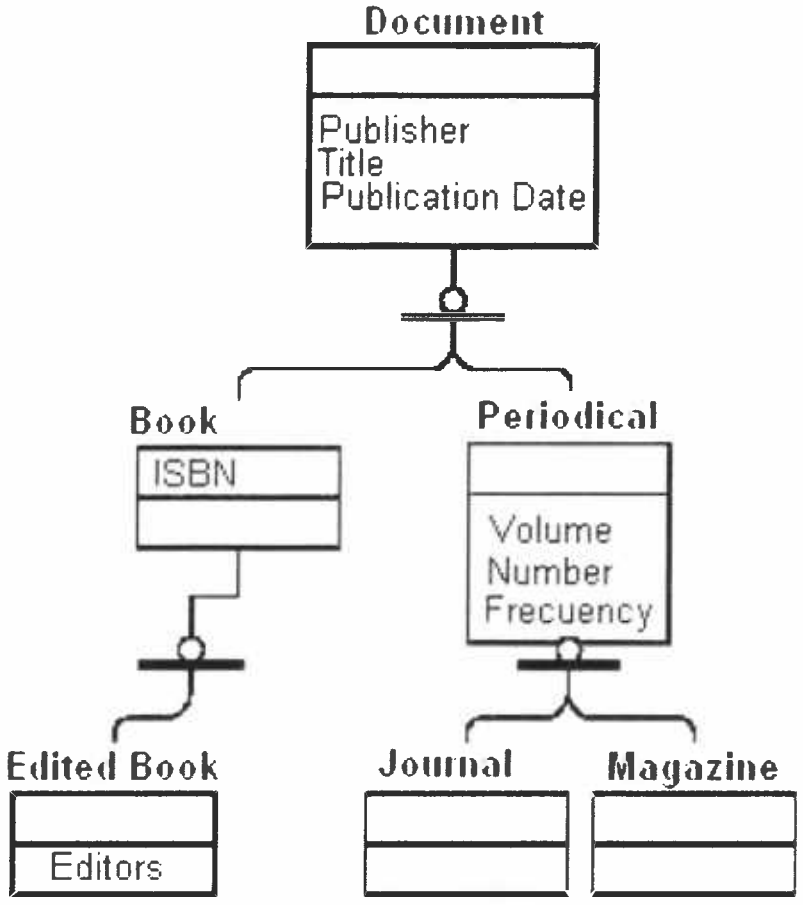
συγκεκριμένο πρόβλημα. Έτσι είναι εφικτή η χρήση τους σε ένα ευρύ πεδίο θεμάτων. Δυστυχώς όμως μέχρι σήμερα δεν έχει γίνει εφικτή η δημιουργία μιας υψηλού επιπέδου οντολογίας αρκετά γενικής ώστε να μπορεί να χρησιμοποιείται πάντα σαν αρχικό σημείο.

Υπάρχουν ωστόσο οντολογίες που περιγράφουν συγκεκριμένους τομείς (π.χ. βιολογία ή φαρμακευτική), οι οποίες είναι δυνατόν να ενσωματωθούν σε οντολογίες που ασχολούνται με τον συγκεκριμένο τομέα.

Η επιλογή αν και κατά πόσο θα χρησιμοποιηθεί μια έτοιμη οντολογία εναπόκειται στον δημιουργό και αποτελεί ένα σημαντικό θέμα προς συζήτηση.

1.5 Παράδειγμα οντολογίας

Για να κλείσουμε το τρέχον κεφάλαιο θα δούμε ένα παράδειγμα οντολογίας ώστε να έχουμε μια καλύτερη άποψη πριν συνεχίσουμε. Αρχικά θα δούμε μια οντολογία σε γραφική απεικόνιση. Η οντολογία αυτή περιγράφει έγγραφα (Εικόνα 3). Η κλάση των εγγράφων χωρίζεται σε βιβλία και περιοδικές εκδόσεις, οι οποίες με τη σειρά τους χωρίζονται σε επιστημονικές εφημερίδες και περιοδικά. Στην οντολογία επίσης βλέπουμε τις ιδιότητες κάθε κλάσης.



Εικόνα 3: Μια οντολογία για έγγραφα

Στη συνέχεια θα δούμε μια οντολογία όπως είναι γραμμένη στη γλώσσα SHOE (Simple HTML Ontology Extensions), η οποία είναι μια γλώσσα αναπαράστασης γνώσης για τον ιστό, συμβατή με XML (Εικόνα 4).

```
<ONTOLOGY ID="Department" VERSION ="1.1">
                                                    ONTOLOGY NAME
<DEF-CATEGORY NAME="Person" ISA = "base.SHOEEntity">
<DEF-CATEGORY NAME="Worker" ISA ="Person">
<DEF-CATEGORY NAME="Teacher" ISA ="Worker">
<DEF-CATEGORY NAME="Student" ISA ="Person">
                                                    CATEGORIES
<DEF-RELATION NAME="Tutor of">
<DEF-ARG POS="1" TYPE="Teacher">
<DEF-ARG POS="2" TYPE="Student">
</DEF-RELATION>
                                                    RELATIONSHIPS
```

Εικόνα 4: Μια οντολογία για τμήματα πανεπιστημίου γραμμένη σε SHOE

Στην οντολογία αυτή περιγράφεται ένα τμήμα πανεπιστημίου. Υπάρχει η γενική κατηγορία «Άτομο», στην οποία υπάγονται οι κατηγορίες «Εργαζόμενος» και «Φοιτητής». Επίσης υπάρχει η κλάση «Καθηγητής», η οποία είναι υποκλάση της κλάσης «Άτομο». Τέλος παρουσιάζεται και μία συσχέτιση μεταξύ των κλάσεων αυτών, η οποία είναι η συσχέτιση «διδάσκων του» με αριστερό όρισμα την κλάση «Καθηγητής» και με δεξί τη κλάση «Φοιτητής». Δηλαδή ο συγκεκριμένος «Καθηγητής» είναι «διδάσκων του» συγκεκριμένου «Φοιτητή».

Κεφάλαιο 2ο: Γλώσσες οντολογιών

2.1 Εισαγωγή

Για να δημιουργήσει κανείς μια οντολογία όπως είδαμε πρέπει να επιλέξει σε ποια γλώσσα θα γράψει την κωδικοποίησή της. Υπάρχουν αρκετές γλώσσες που υποστηρίζουν τη δημιουργία οντολογιών, όμως αυτή που τείνει να επικρατήσει είναι η γλώσσα OWL, καθώς έχει γίνει πλέον standard από το W3C (World Wide Web Consortium).

Θα δούμε συνοπτικά τις σημαντικότερες γλώσσες οντολογιών και στη συνέχεια θα περιγράψουμε αναλυτικά την OWL, η οποία θα είναι και η γλώσσα που θα επιλέξουμε για την εφαρμογή μας.

2.2 Σημαντικότερες γλώσσες

2.2.1 KIF (Knowledge Interchange Format)

Η γλώσσα KIF είναι από τις πρώτες, άρα και σημαντικότερες, επίσημες γλώσσες, δημιουργημένες για την αναπαράσταση και την ανταλλαγή γνώσης μεταξύ προγραμμάτων, αλλά και ανθρώπων. Τα βασικά χαρακτηριστικά της περιγράφονται παρακάτω:

- Η γλώσσα διαθέτει δηλωτική σημασιολογία. Δηλαδή είναι δυνατό να καταλάβει κανείς την έννοια μιας έκφρασης στη γλώσσα χωρίς τη βοήθεια κάποιου διερμηνέα που θα διαχειριζόταν την έκφραση αυτή.

- Η KIF είναι λογικά περιεκτική. Παρέχει την δυνατότητα για αυθαίρετες προτάσεις να εκφράζονται σε κατηγορηματικό λογισμό. Έτσι διαφέρει από γλώσσες που εκφράζουν σχεσιακές βάσεις κι από γλώσσες της μορφής της Prolog, που είναι περιορισμένες σε προτάσεις Horn.

- Παρέχει τη δυνατότητα αναπαράστασης της γνώσης σχετικά με την ίδια την αναπαράσταση της γνώσης. Έτσι όλες οι αποφάσεις σχετικά με την αναπαράσταση της γνώσης θα είναι ρητά καθορισμένες και μας δίνεται η δυνατότητα να δημιουργήσουμε νέες δομές αναπαράστασης γνώσης χωρίς να αλλοιώνεται η γλώσσα.

2.2.2 SHOE (Simple HTML Ontology Extensions)

Η γλώσσα SHOE είναι μία γλώσσα αναπαράστασης γνώσης για τον ιστό και είναι συμβατή με την XML. Παρέχει τη δυνατότητα στους δημιουργούς ιστοσελίδων να επισημειώνουν τα έγγραφα που διαθέτουν στον ιστό, αλλά πλέον δεν χρησιμοποιείται ιδιαίτερα.

2.2.3 RDF / RDFS (Resource Description Framework / Resource Description Framework Schema)

Η γλώσσα RDF είναι η πρώτη γλώσσα που προσπαθεί να αναπαραστήσει σημασιολογία. Για να το πετύχει αυτό χρησιμοποιεί συντακτικές συμβάσεις και απλά μοντέλα δεδομένων. Υποστηρίζει διαλειτουργικότητα με συσχετίσεις αντικειμένου – ιδιότητας – τιμής.

Η RDFS παρέχει κάποιες πρωταρχικές οδηγίες για να δημιουργήσει κανείς οντολογίες χρησιμοποιώντας την RDF.

2.2.4 OIL (Ontology Inference Layer / Ontology Interchange Language)

Η γλώσσα αυτή είναι από τις περισσότερο διαδεδομένες στον χώρο των οντολογιών και είναι συμβατή με την RDF. Παρέχει αρχές για τη δημιουργία και τη μοντελοποίηση οντολογιών, διαθέτει τυπική σημασιολογία και υποστηρίζει εξαγωγή συμπερασμάτων, βασισμένη σε περιγραφική λογική (descriptive logic).

2.2.5 DAML (DARPA Agent Markup Language)

Η DAML έχει δημιουργηθεί από την ένωση της DAML-ONT, η οποία είναι μια γλώσσα για οντολογίες, και της DAML-LOGIC, η οποία είναι μια γλώσσα που έχει τη δυνατότητα να αναπαριστά αξιώματα και κανόνες. Η DAML κληρονομεί πολλά χαρακτηριστικά από την OIL, παρόλο που δεν είναι τόσο συμβατή με την RDF, όσο η OIL.

Από τις γλώσσες DAML και OIL, δημιουργήθηκε η γλώσσα DAML+OIL, η οποία είναι πρόγονος της γλώσσας OWL, την οποία θα εξετάσουμε παρακάτω.

2.2.6 OWL (Web Ontology Language)

Πρωταρχικός σκοπός της OWL είναι να παρέχει τη δυνατότητα σε μηχανές να επεξεργαστούν την πληροφορία που περιέχεται σε ένα κείμενο. Μπορεί να χρησιμοποιηθεί για να δηλωθεί ρητά το νόημα των όρων σε ένα λεξιλόγιο και οι συσχετίσεις μεταξύ των όρων αυτών, δηλαδή να δημιουργήσει μια οντολογία. Η OWL έχει περισσότερες δυνατότητες για την αναπαράσταση αυτή, από τις γλώσσες XML, RDF, RDFS, οπότε τις ξεπερνάει στην ικανότητα αναπαράστασης περιεχομένου στον ιστό, το οποίο να διερμηνεύεται από υπολογιστή.

2.2.7 XSL (Extensible Stylesheet Language)

Η γλώσσα αυτή δεν είναι γλώσσα δημιουργίας οντολογιών, αλλά μία γλώσσα διαχείρισης οντολογιών. Παρέχει έναν μηχανισμό για την περιγραφή αντιστοιχήσεων μεταξύ διαφορετικών ορολογιών. Αποτελεί, δηλαδή, έναν τρόπο μετάφρασης μεταξύ XML εγγράφων και οντολογιών.

2.2.8 XOL (Ontology Exchange Language)

Το χαρακτηριστικό της γλώσσας αυτής είναι η απλότητα. Παρέχει μία γενική προσέγγιση για τη δημιουργία οντολογιών. Συντακτικά έχει δύο διαφορετικές υλοποιήσεις βασισμένες στην XML και στην RDFS.

2.3 Επιλογή γλώσσας

Για την επιλογή της γλώσσας που θα χρησιμοποιήσουμε στην εφαρμογή μας εστιάσαμε περισσότερο σε χαρακτηριστικά όπως η χρήση της γλώσσας σήμερα, η ευκολία στην υλοποίηση οντολογιών μέσω αυτής και η ευκολία στη διαχείρισή της, μέσω της ύπαρξης κατάλληλων εργαλείων.

Οι πιο διαδεδομένες γλώσσες οντολογιών σήμερα είναι η OIL και η OWL. Η πρωταρχική γλώσσα βέβαια είναι η RDF, αλλά οι γλώσσες αυτές επεκτείνουν σημαντικά την RDF και παρέχουν περισσότερες και σημαντικές λειτουργίες, όπως το μεγαλύτερο λεξιλόγιο, η τυπικά ορισμένη σημασιολογία, η περιγραφική λογική, η εξαγωγή συμπερασμάτων.

Η ευκολία στη δημιουργία οντολογιών φαίνεται από την ύπαρξη κατάλληλων εργαλείων για τη συγγραφή οντολογιών, με εύχρηστη διεπαφή χρήστη. Πράγματι, για τις γλώσσες αυτές υπάρχουν συγκεκριμένα εργαλεία που διαθέτουν πολλές λειτουργίες, πετυχαίνοντας τη δημιουργία απλών οντολογιών σε λίγα λεπτά, χωρίς να είναι απαραίτητο για κάποιον να θυμάται τη σύνταξη της εκάστοτε γλώσσας. Τα εργαλεία αυτά θα τα μελετήσουμε στο επόμενο κεφάλαιο.

Όσον αφορά τη διαχείριση των οντολογιών πρέπει να υπάρχει ένα εργαλείο, το οποίο να παρέχει την διεπαφή ώστε να μπορεί κάποιος να χρησιμοποιήσει προγραμματιστικά μια οντολογία. Πράγματι, υπάρχει το Jena API το οποίο μας δίνει τη δυνατότητα να διαχειριστούμε μια οντολογία μέσω της γλώσσας προγραμματισμού Java. Καθώς μάλιστα η εφαρμογή που θέλουμε να επεκτείνουμε είναι γραμμένη σε Java, το εργαλείο αυτό αποτελεί ιδανική επιλογή. Θα το εξετάσουμε εκτενέστερα σε επόμενο κεφάλαιο.

Η τελική επιλογή μας είναι η γλώσσα OWL. Δεδομένου ότι είχαμε καταλήξει σε αυτήν και στην OIL, η τελική απόφαση ήταν εύκολη, καθώς η OWL υπερτερεί στα εξής χαρακτηριστικά:

- Αποτελεί απόγονο της OIL, αφού είναι επέκταση της DAML+OIL, συνεπώς διαθέτει περισσότερες λειτουργίες.

- Διαθέτει τρεις διαφορετικές εκδόσεις, ανάλογα με την πολυπλοκότητα της οντολογίας που θέλουμε να δημιουργήσουμε, τις OWL Lite, OWL DL και OWL Full.

- Έχει προταθεί ως ένα από τα πρότυπα για τον σημασιολογικό ιστό από το W3C (World Wide Web Consortium).

Θα συνεχίσουμε με την λεπτομερέστερη περιγραφή της γλώσσας OWL, ώστε να γνωρίζουμε σε γενικές γραμμές τι μορφή έχει μια οντολογία γραμμένη σε OWL και τι πρέπει να έχουμε υπόψιν μας.

2.4 Η γλώσσα OWL (Web Ontology Language)

2.4.1 Εισαγωγή

2.4.1.1 Τι είναι η OWL

Η γλώσσα OWL δημιουργήθηκε με τον σκοπό να χρησιμοποιείται όταν η πληροφορία που περιέχεται σε έγγραφα πρέπει να επεξεργαστεί από εφαρμογές, σε αντίθεση με την περίπτωση που η πληροφορία πρέπει να παρουσιαστεί μόνο σε ανθρώπους. Η OWL μπορεί να χρησιμοποιηθεί για την ρητή αναπαράσταση της έννοιας οποιωνδήποτε όρων, καθώς και των συσχετίσεων μεταξύ των όρων αυτών. *Η OWL, λοιπόν, είναι μια γλώσσα αναπαράστασης οντολογιών.* Διαθέτει περισσότερα εργαλεία για την αναπαράσταση εννοιών και σημασιολογίας από ότι η XML, η RDF και η RDFS, συνεπώς υπερτερεί από αυτές στην αναπαράσταση περιεχομένου στον ιστό, ερμηνεύσιμο από μηχανές. Η OWL αποτελεί αναθεώρηση της γλώσσας οντολογιών DAML+OIL, περιλαμβάνοντας τεχνικές που ανακαλύφθηκαν κατά το σχεδιασμό και την εφαρμογή της DAML+OIL.

2.4.1.2 Η πορεία προς τον Σημασιολογικό Ιστό

Ο Σημασιολογικός Ιστός (Semantic Web) αποτελεί ένα όραμα για το μέλλον του Διαδικτύου, σύμφωνα με το οποίο, η πληροφορία επισημειώνεται με κάποια ρητή έννοια. Αυτό με τη σειρά του διευκολύνει τις εφαρμογές στον αυτοματισμό της επεξεργασίας και της ανταλλαγής της

πληροφορίας, που είναι ελεύθερα διαθέσιμη στον Ιστό. Οι βάσεις για την ανάπτυξη του σημασιολογικού ιστού είναι η XML, η οποία μπορεί να ορίζει επισημειώσεις (tags) για οποιοδήποτε περιεχόμενο, και η RDF, η οποία διαθέτει μια ευέλικτη προσέγγιση στην αναπαράσταση των δεδομένων. Το αμέσως επόμενο επίπεδο προς την κατεύθυνση αυτή είναι η ύπαρξη μιας γλώσσας οντολογιών ικανής να περιγράψει με τυπικό τρόπο την έννοια της ορολογίας που χρησιμοποιείται στα έγγραφα του ιστού. Αν όμως αναμένουμε από τους υπολογιστές να πραγματοποιούν χρήσιμες πράξεις συλλογιστικής και εξαγωγής συμπερασμάτων στα έγγραφα αυτά, θα πρέπει να έχουμε στη διάθεσή μας μια γλώσσα που να επεκτείνει τη βασική σημασιολογία της RDF.

Την ανάγκη αυτή έρχεται να καλύψει η OWL, η οποία αποτελεί μέρος των προτάσεων του W3C σχετικά με τον Σημασιολογικό Ιστό. Συνοπτικά οι δυνατότητες των γλωσσών αυτών παρουσιάζονται παρακάτω:

- *XML*: παρέχει συντακτικό για δομημένα έγγραφα, αλλά δεν επιβάλλει σημασιολογικούς περιορισμούς πάνω στις έννοιες που περιέχονται στα έγγραφα αυτά.
- *XML Schema*: χρησιμοποιείται για τον περιορισμό της δομής των εγγράφων XML και επίσης επεκτείνει την XML με τύπους δεδομένων.
- *RDF*: είναι ένα μοντέλο δεδομένων για αντικείμενα και συσχετίσεις μεταξύ τους. Παρέχει μια απλή σημασιολογία για το μοντέλο αυτό και η αναπαράσταση μπορεί να πραγματοποιηθεί σε σύνταξη τύπου XML.
- *RDF Schema*: είναι ένα λεξιλόγιο για την περιγραφή κλάσεων και ιδιοτήτων αντικειμένων RDF, μαζί με μία σημασιολογία για την αναπαράσταση ιεραρχίας των κλάσεων και των ιδιοτήτων αυτών.
- *OWL*: προσθέτει περισσότερο λεξιλόγιο για την περιγραφή κλάσεων και ιδιοτήτων. Διαθέτει, μεταξύ άλλων, συσχετίσεις μεταξύ κλάσεων (π.χ. disjointness), πολλαπλότητα (π.χ. «ακριβώς ένα», «ένα προς ένα», «ένα προς πολλά»), ισότητα,

πλουσιότερη σύνταξη ιδιοτήτων, χαρακτηριστικά ιδιοτήτων (π.χ. συμμετρία) και απαριθμήσιμες κλάσεις.

2.4.2 Οι τρεις υποκατηγορίες της OWL

Η OWL παρέχει τρεις εκδόσεις, η καθεμία από τις οποίες είναι υπερσύνολο της άλλης, όσον αφορά τις λειτουργίες που παρέχουν. Είναι σχεδιασμένες για συγκεκριμένους χρήστες και εφαρμογές.

- *OWL Lite*: υποστηρίζει τους χρήστες που κατά βάση χρειάζονται μια ιεραρχία κατηγοριοποίησης και απλούς περιορισμούς. Για παράδειγμα, αν και υποστηρίζει περιορισμούς πολλαπλότητας, επιτρέπει τιμές 0 και 1 μόνο. Η ανάπτυξη εφαρμογών για την OWL Lite είναι απλούστερη και η δημιουργία θησαυρών και άλλων ταξονομιών είναι πιο γρήγορη. Επίσης η πολυπλοκότητα της OWL Lite είναι μικρότερη από της OWL DL. Όπως φαίνεται και από το όνομά της είναι η «ελαφρύτερη» έκδοση της OWL.

- *OWL DL*: υποστηρίζει τους χρήστες εκείνους που επιζητούν τη μέγιστη δυνατή εκφραστικότητα, διατηρώντας ταυτόχρονα την υπολογιστική πληρότητα (όλα τα συμπεράσματα είναι εγγυημένο ότι είναι υπολογίσιμα) και την αποφασισιμότητα (όλοι οι υπολογισμοί θα τελειώσουν σε πεπερασμένο χρόνο). Η OWL DL περιλαμβάνει όλες τις δομές της OWL, αλλά μπορούν να χρησιμοποιηθούν υπό συγκεκριμένους περιορισμούς (για παράδειγμα, ενώ μια κλάση μπορεί να είναι υποκλάση πολλών κλάσεων, δεν μπορεί να είναι στιγμιότυπο μιας άλλης κλάσης). Το όνομα της OWL DL προέρχεται από την αντιστοιχία της με την Περιγραφική Λογική (Description Logics), έναν τομέα έρευνας που μελετά το πεδίο της Λογικής που ορίζει τα θεμέλια της OWL.

- *OWL Full*: υποστηρίζει τους χρήστες που επιζητούν τη μέγιστη δυνατή εκφραστικότητα και τη συντακτική ελευθερία της RDF, χωρίς καμία υπολογιστική εγγύηση. Για παράδειγμα μια κλάση μπορεί να αντιμετωπιστεί ως ένα σύνολο οντοτήτων (individuals) αλλά και ως οντότητα η ίδια. Η OWL Full επιτρέπει σε μια

οντολογία να προσαυξήσει το νόημα του καθορισμένου λεξιλογίου (RDF ή OWL). Είναι απίθανο οποιοδήποτε λογισμικό εξαγωγής συμπερασμάτων να καταφέρει να υποστηρίξει πλήρως όλες τις λειτουργίες της OWL Full.

Καθεμία από τις εκδόσεις αυτές αποτελεί επέκταση της απλούστερης προγόνου της, όσον αφορά το τι μπορεί να εκφραστεί νόμιμα αλλά και τι συμπέρασμα μπορεί να εξαχθεί έγκυρα. Ισχύουν οι παρακάτω προτάσεις, όχι όμως και οι αντίστροφές τους:

- Κάθε νόμιμη οντολογία OWL Lite είναι και νόμιμη οντολογία OWL DL.
- Κάθε νόμιμη οντολογία OWL DL είναι και νόμιμη οντολογία OWL Full.
- Κάθε έγκυρο συμπέρασμα OWL Lite είναι και έγκυρο συμπέρασμα OWL DL.
- Κάθε έγκυρο συμπέρασμα OWL DL είναι και έγκυρο συμπέρασμα OWL Full.

Οι σχεδιαστές οντολογιών που θέλουν να χρησιμοποιήσουν την OWL πρέπει να επιλέξουν την έκδοση εκείνη που καλύπτει περισσότερο τις ανάγκες τους. Η επιλογή μεταξύ OWL Lite και OWL DL εξαρτάται από τον βαθμό στον οποίο ο χρήστης θέλει να χρησιμοποιήσει τις πιο εκφραστικές δομές της OWL DL. Η επιλογή μεταξύ OWL DL και OWL Full εξαρτάται από τον βαθμό στον οποίο ο χρήστης επιθυμεί να διαθέτει τις λειτουργίες μετα-πληροφορίας της RDFS (π.χ. ορισμός κλάσης κλάσεων, επισύναψη ιδιοτήτων σε κλάσεις). Όταν όμως χρησιμοποιείται OWL Full η υποστήριξη εξαγωγής συμπερασμάτων είναι λιγότερο προβλέψιμη, καθώς δεν υπάρχουν πλήρεις υλοποιήσεις για την OWL Full.

Η OWL Full μπορεί να θεωρηθεί μια επέκταση της RDF, ενώ οι OWL Lite και OWL DL μπορούν να θεωρηθούν επεκτάσεις μιας περιορισμένης μορφής της RDF. Κάθε OWL έγγραφο (Lite, DL ή Full) είναι ένα RDF έγγραφο και κάθε RDF έγγραφο είναι ένα OWL Full έγγραφο, αλλά μόνο

ένα περιορισμένο υποσύνολο εγγράφων RDF είναι νόμιμα OWL Lite ή OWL DL έγγραφα. Συνεπώς χρειάζεται προσοχή όταν κάποιος θέλει να μετατρέψει ένα RDF έγγραφο σε OWL. Πρέπει να λάβουμε κάποια πράγματα υπόψιν μας ώστε το αρχικό RDF έγγραφο να είναι συμβατό με τους επιπλέον περιορισμούς της OWL Lite και της OWL DL. Ενδεικτικά, κάποιοι βασικοί κανόνες είναι οι εξής:

- Κάθε URI που χρησιμοποιείται σαν όνομα κλάσης πρέπει να δηλωθεί ότι είναι τύπου owl:Class (όμοια για τις ιδιότητες).
- Κάθε οντότητα πρέπει να δηλωθεί ότι ανήκει σε μία τουλάχιστον κλάση (ακόμα κι αν αυτή είναι η κορυφή της ιεραρχίας, η owl:Thing).
- Τα URI που χρησιμοποιούνται για κλάσεις, ιδιότητες, οντότητες πρέπει να είναι αμοιβαία διαχωρισμένα (mutually disjoint).

2.4.3 Περιγραφή της OWL Lite

Θα προχωρήσουμε σε μια συνοπτική περιγραφή της γλώσσας OWL Lite και των χαρακτηριστικών της. Η OWL Lite χρησιμοποιεί μόνο λίγα από τα χαρακτηριστικά της OWL και έχει περισσότερους περιορισμούς στη χρήση αυτών από τις OWL DL και OWL Full. Για παράδειγμα στην OWL Lite οι κλάσεις μπορούν να οριστούν μόνο σε σχέση με δηλωμένες (named) υπερκλάσεις (οι υπερκλάσεις δεν μπορούν να είναι αυθαίρετες εκφράσεις) και μόνο συγκεκριμένα είδη περιορισμών κλάσεων μπορούν να χρησιμοποιηθούν. Σχέσεις ισότητας μεταξύ κλάσεων και υποκλάσεων μπορούν να υπάρξουν μόνο μεταξύ δηλωμένων κλάσεων και όχι μεταξύ αυθαίρετων εκφράσεων κλάσεων. Όμοια, οι περιορισμοί στην OWL Lite χρησιμοποιούν μόνο δηλωμένες κλάσεις. Όσον αφορά την πολλαπλότητα και αυτή είναι περιορισμένη στην OWL Lite, καθώς επιτρέπονται μόνο οι τιμές 0 και 1.

2.4.3.1 Χαρακτηριστικά από την RDF Schema

Στην OWL Lite (συνεπώς στην OWL) περιλαμβάνονται τα παρακάτω χαρακτηριστικά δανεισμένα από την RDFS:

- *Class*: Μία κλάση ορίζει ένα σύνολο οντοτήτων που ανήκουν στην ίδια κατηγορία επειδή έχουν κάποιες κοινές ιδιότητες. Χρησιμοποιώντας την λειτουργία *subClassOf* μπορούμε να δημιουργούμε ιεραρχίες κλάσεων. Υπάρχει ενσωματωμένη στην OWL η πιο γενική κλάση που ονομάζεται *Thing*, η οποία είναι η κλάση όλων των οντοτήτων και είναι υπερκλάση όλων των κλάσεων της OWL. Υπάρχει επίσης η κλάση *Nothing*, η οποία είναι η κλάση που δεν έχει κανένα στιγμιότυπο και είναι υποκλάση όλων των κλάσεων της OWL.

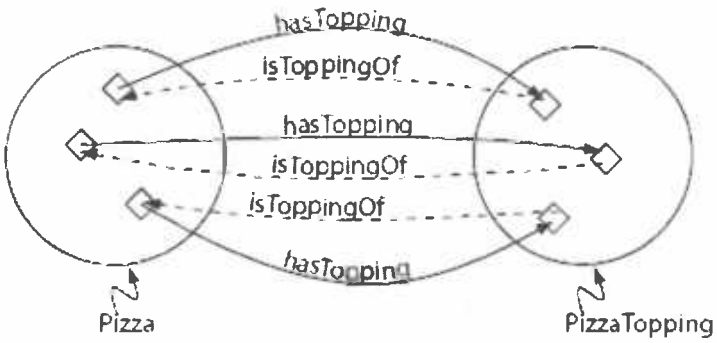
- *rdfs:subClassOf*: Χρησιμοποιώντας τη δήλωση πως μια κλάση είναι υποκλάση μιας άλλης μπορούμε να δημιουργήσουμε ιεραρχίες κλάσεων. Για παράδειγμα η κλάση «Άνθρωπος» είναι υποκλάση της κλάσης «Θηλαστικά». Έτσι εάν μια οντότητα ανήκει στην κλάση «Άνθρωπος», ένα πρόγραμμα εξαγωγής συμπερασμάτων θα κατέληγε στο συμπέρασμα πως η οντότητα αυτή ανήκει και στην κλάση «Θηλαστικά».

- *rdf:Property*: Οι ιδιότητες μπορούν να χρησιμοποιηθούν για να δηλώσουν συσχετίσεις μεταξύ οντοτήτων ή μεταξύ οντοτήτων και τιμών. Για παράδειγμα η ιδιότητα «έχειΠατέρα» είναι συσχέτιση μεταξύ δύο ατόμων, ενώ η ιδιότητα «έχειΗλικία» είναι συσχέτιση μεταξύ ενός ατόμου και ενός ακεραίου που υποδηλώνει την ηλικία του ατόμου.

- *rdf:subPropertyOf*: Με την δήλωση υποϊδιοτήτων μπορούμε να δημιουργήσουμε ιεραρχίες ιδιοτήτων. Για παράδειγμα η ιδιότητα «έχειΑδερφό» είναι υποϊδιότητα της «έχειΣυγγενή», άρα οι ιδιότητες που ανήκουν σε αυτήν ανήκουν και στην υπεριδιότητα αυτής.

- *rdfs:domain*: Με τη δήλωση αυτή περιορίζουμε το πεδίο ορισμού μιας ιδιότητας (όχι μόνο όταν η ιδιότητα συνδέεται με συγκεκριμένη κλάση αλλά πάντα – αυτό ονομάζεται περιορι μός ιδιότητας – property restriction).

- *rdfs:range*: Με τη δήλωση αυτή περιορίζουμε το πεδίο τιμών μιας ιδιότητας. Και αυτός ο περιορισμός είναι περιορισμός ιδιότητας. Παράδειγμα περιορισμών πεδίου ορισμού και πεδίου τιμών φαίνεται στην Εικόνα 5.



Εικόνα 5: Η κλάση Pizza είναι το πεδίο ορισμού της ιδιότητας hasTopping και η κλάση PizzaTopping είναι το πεδίο τιμών. Όμοια και για τις υπόλοιπες ιδιότητες.

- *Individual*: Οι οντότητες (individuals) είναι τα στιγμιότυπα των κλάσεων και μπορούμε να χρησιμοποιήσουμε ιδιότητες για να συσχετίσουμε μεταξύ τους κάποιες οντότητες. Για παράδειγμα η οντότητα «Παπαδόπουλος» μπορεί να είναι στιγμιότυπο της κλάσης «Άνθρωπος» και να σχετίζεται με την οντότητα «ΑΣΟΕΕ» μέσω της ιδιότητας «έχειΕργοδότη».

2.4.3.2 Ισότητα και ανισότητα

Παρακάτω εξετάζουμε τα χαρακτηριστικά της OWL Lite που σχετίζονται με ισότητα και ανισότητα:

- *equivalentClass*: Δύο κλάσεις μπορεί να δηλωθούν ως ισοδύναμες. Οι ισοδύναμες κλάσεις έχουν τις ίδιες οντότητες. Η

ισοδυναμία μπορεί να χρησιμοποιηθεί για να δημιουργήσουμε συνώνυμες κλάσεις. Π.χ. η κλάση «Αμάξι» μπορεί να δηλωθεί ισοδύναμη με την «Αυτοκίνητο».

- *equivalentProperty*: Όμοια με τις κλάσεις δηλώνουμε δύο ισοδύναμες ιδιότητες.

- *sameAs*: Δύο οντότητες μπορούν να δηλωθούν ως ίδιες. Η δομή αυτή χρησιμοποιείται για να δημιουργήσουμε ένα σύνολο από διαφορετικά ονόματα για την ίδια οντότητα. Είναι η δομή που χρησιμοποιούμε κατά κόρον στην εφαρμογή μας για να δηλώσουμε τα συνώνυμα των πανεπιστημίων.

- *differentFrom*: Δύο οντότητες είναι δυνατόν να δηλωθούν ότι είναι διαφορετικές. Αυτό χρησιμεύει στην εξαγωγή συμπερασμάτων για να λάβουμε σωστά αποτελέσματα.

- *AllDifferent*: Όμοια με πριν, όμως τώρα δηλώνονται όλες οι οντότητες ότι είναι διαφορετικές ανά ζεύγη. Συνεπώς η εντολή αυτή μας εξοικονομεί τον κόπο το να δηλώσουμε ρητά όσες οντότητες είναι διαφορετικές χρησιμοποιώντας την *differentFrom*.

2.4.3.3 Χαρακτηριστικά ιδιοτήτων

Υπάρχουν συγκεκριμένες δομές που χρησιμοποιούνται για να δώσουν πληροφορίες σχετικά με τις ιδιότητες και τις τιμές τους. Τις δομές αυτές εξετάζουμε στην παράγραφο αυτή.

- *ObjectProperty*: Η δήλωση αυτή ορίζει μία ιδιότητα μεταξύ οντοτήτων.

- *DatatypeProperty*: Με την *DatatypeProperty* ορίζουμε μία ιδιότητα μεταξύ μιας οντότητας και μιας τιμής. Και οι δύο δομές *owl:ObjectProperty* και *owl:DatatypeProperty* είναι υποκλάσεις της κλάσης της RDF *rdf:Property*.

- *inverseOf*: Μία ιδιότητα μπορεί να δηλωθεί ως η αντίστροφη μιας άλλης. Για παράδειγμα η «έχειΠαιδί» είναι ανάστροφη της

«έχειΓονέα». Έτσι αν η οντότητα «Γιώργος» σχετίζεται με την οντότητα «Κώστας» με την ιδιότητα «έχειΠαιδί» τότε και η οντότητα «Κώστας» σχετίζεται με την οντότητα «Γιώργος» με την ιδιότητα «έχειΓονέα». Στο παράδειγμα της Εικόνα 5 η ιδιότητα "hasTopping" είναι αντίστροφη της "isToppingOf".

- *TransitiveProperty*: Πολλές φορές κάποια ιδιότητα είναι μεταβατική ως προς κάποια άλλη. Για παράδειγμα η ιδιότητα «Πρόγονος». Αν ο «Γιώργος» είναι πρόγονος του «Κώστα» και ο «Κώστας» είναι πρόγονος της «Μαρίας» τότε ο «Γιώργος» είναι πρόγονος και της «Μαρίας». Η δήλωση αυτή μας επιτρέπει να χρησιμοποιούμε τέτοιες ιδιότητες.

- *SymmetricProperty*: Η δομή αυτή χρησιμοποιείται για να περιγράψει σχέσεις που είναι συμμετρικές. Για παράδειγμα η σχέση «Αδέρφια» είναι συμμετρική. Αν ο «Κώστας» συνδέεται με την «Ελένη» μέσω αυτής τότε και η «Ελένη» συνδέεται με τον «Κώστα» μέσω αυτής.

- *FunctionalProperty*: Για κάποιες ιδιότητες μπορεί να θέλουμε να δηλώσουμε ότι δέχονται μόνο μία τιμή. Οι ιδιότητες αυτές ονομάζονται λειτουργικές (functional) και στην ουσία έχουν ελάχιστη πολλαπλότητα 0 και μέγιστη 1. Για παράδειγμα η ιδιότητα «έχειΠατέρα» μπορεί να έχει μόνο μία τιμή, όχι όμως και η «έχειΠαιδί».

- *InverseFunctionalProperty*: Μπορούμε να δηλώσουμε ότι μια ιδιότητα είναι αντίστροφα λειτουργική, ότι δηλαδή η αντίστροφή της είναι λειτουργική και δέχεται μέχρι μία τιμή. Για παράδειγμα η ιδιότητα «έχειΠαιδί» είναι αντίστροφα λειτουργική αφού η αντίστροφή της, «έχειΠατέρα», είναι λειτουργική.

2.4.3.4 Περιορισμοί ιδιότητας

Η OWL Lite επιτρέπει να τοποθετηθούν κάποιοι περιορισμοί πάνω στον τρόπο που οι ιδιότητες χρησιμοποιούνται από τα στιγμιότυπα των κλάσεων. Η δομή που χρησιμοποιείται είναι η *owl:Restriction* και το

στοιχείο που καθορίζει την ιδιότητα πάνω στην οποία θα τοποθετηθεί ο περιορισμός είναι το *owl:onProperty*. Οι παρακάτω περιορισμοί περιορίζουν το ποιες τιμές θα χρησιμοποιηθούν ενώ στην επόμενη παράγραφο θα δούμε τους περιορισμούς που περιορίζουν το πόσες τιμές θα χρησιμοποιηθούν.

- *allValuesFrom*: Ο περιορισμός αυτός τοποθετείται σε μια ιδιότητα με αναφορά σε μια κλάση. Σημαίνει πως όλες οι οντότητες που είναι συνδεδεμένες μέσω της ιδιότητας αυτής με μία συγκεκριμένη κλάση, πρέπει να είναι οντότητες μιας συγκεκριμένης κλάσης. Για παράδειγμα αν η κλάση «Άνθρωπος» έχει μια ιδιότητα «έχειΚόρη», η οποία έχει περιορισμό *allValuesFrom* από την κλάση «Γυναίκα» τότε όλες οι τιμές που μπορεί να πάρει η ιδιότητα αυτή, θα πρέπει αναγκαστικά να είναι στιγμιότυπα της κλάσης «Γυναίκα». Ας σημειώσουμε εδώ πως δεν είναι υποχρεωτικό να υπάρχει κάποια τιμή, αλλά αν υπάρχει θα πρέπει να είναι από την κλάση που υποδεικνύει ο περιορισμός.

- *someValuesFrom*: Όμοια με πριν, τοποθετούμε τον περιορισμό αυτό σε μία ιδιότητα ως προς μια συγκεκριμένη κλάση. Η σημασία του είναι πως αν τοποθετηθεί σε μία ιδιότητα ως προς μία κλάση, τότε πρέπει να υπάρχει κάποια οντότητα που να είναι μέσα στις τιμές της ιδιότητας και να είναι στιγμιότυπο της συγκεκριμένης κλάσης.

2.4.3.5 Περιορισμοί πολλαπλότητας

Στην OWL Lite περιέχονται κάποιοι περιορισμοί πολλαπλότητας, οι οποίοι ονομάζονται και τοπικοί περιορισμοί επειδή αναφέρονται σε μία συγκεκριμένη κλάση, με την οποία συνδέεται η ιδιότητα που έχει τον περιορισμό. Οι περιορισμοί αυτοί στην OWL Lite είναι περιορισμένοι γιατί επιτρέπουν μόνο πολλαπλότητες με τιμές 0 και 1, σε αντίθεση με τις αυθαίρετες τιμές που μπορούν να δοθούν στις OWL DL και OWL Full.

- *minCardinality*: Όμοια με πριν και η πολλαπλότητα είναι περιορισμός που τίθεται με αναφορά σε μία κλάση. Η έννοια της ελάχιστης πολλαπλότητας για μια ιδιότητα είναι ότι πρέπει να

υπάρχουν τιμές για όλα τα στιγμιότυπα της κλάσης που έχει την ιδιότητα αυτή. Στην OWL Lite οι μόνες τιμές που μπορούμε να έχουμε είναι 0 και 1. Το 0 σε αυτή την περίπτωση σημαίνει πως η ιδιότητα είναι προαιρετική. Για παράδειγμα η ιδιότητα «έχειΠαιδί» έχει ελάχιστη πολλαπλότητα 0 για την κλάση «Άνθρωπος» αλλά έχει 1 για την κλάση «Γονέας», αφού ξέρουμε ότι κάποιος έχει τουλάχιστον ένα παιδί για να είναι γονέας.

- *maxCardinality*: Ο περιορισμός της μέγιστης πολλαπλότητας τίθεται και αυτός με αναφορά σε μία κλάση. Η έννοια της είναι να περιορίσει το μέγιστο αριθμό τιμών που μπορεί να έχει η ιδιότητα μιας κλάσης. Όταν ο περιορισμός είναι 1 η ιδιότητα λέγεται και λειτουργική (όπως είδαμε πριν). Ας σημειωθεί πως όταν η μέγιστη πολλαπλότητα είναι 1 δεν εννοείται πως είναι ακριβώς μία η τιμή της ιδιότητας, αλλά μπορεί και να μην υπάρχει καμία. Όταν ο περιορισμός είναι 0 σημαίνει πως η ιδιότητα δεν πρέπει να πάρει καμία τιμή. Για παράδειγμα στιγμιότυπα της κλάσης «Ανύπαντρος» δεν πρέπει να συνδεθούν με καμία οντότητα μέσω της ιδιότητας «έχειΣύζυγο».

- *cardinality*: Ο περιορισμός αυτός παρέχεται σαν διευκόλυνση όταν ο αριθμός των τιμών μίας ιδιότητας είναι ακριβώς 0 ή 1. Έτσι αντί να χρησιμοποιούσαμε *minCardinality* 0 και *maxCardinality* 0 (ή 1) χρησιμοποιούμε απλά *cardinality* 0 (ή 1).

2.4.3.6 Άλλα χαρακτηριστικά

- *Τομή κλάσεων*: Η OWL Lite χρησιμοποιεί τη δομή *intersectionOf* για να περιγράψει την τομή δύο κλάσεων. Για παράδειγμα η κλάση «ΠλούσιοιΆντρες» μπορεί να οριστεί ως η τομή της κλάσης «ΠλούσιοιΆνθρωποι» και «Άντρες».

- *Τύποι δεδομένων (datatypes)*: Η OWL χρησιμοποιεί τους μηχανισμούς της RDF για τιμές δεδομένων.

- *Πληροφορίες επικεφαλίδας*: Η OWL Lite υποστηρίζει την εισαγωγή οντολογιών και την επισύναψη πληροφορίας σε μία οντολογία.

- *Ιδιότητες επισημειώσεων*: Υποστηρίζεται η χρήση επισημειώσεων σε κλάσεις, ιδιότητες, οντότητες και επικεφαλίδες οντολογιών. Παρόλα αυτά η χρήση αυτή υπόκειται σε συγκεκριμένους περιορισμούς.

- *Υποστήριξη εκδόσεων*: Ήδη η RDF διαθέτει ένα μικρό λεξιλόγιο για την περιγραφή πληροφορίας διαφορετικών εκδόσεων (versioning). Η OWL επεκτείνει σημαντικά το λεξιλόγιο αυτό.

2.4.4 Περιγραφή των OWL DL και OWL Full

Η OWL DL και η OWL Full χρησιμοποιούν το ίδιο λεξιλόγιο, αν και η OWL DL υπόκειται σε κάποιους περιορισμούς. Σε γενικές γραμμές, η OWL DL απαιτεί διαχωρισμό των τύπων (δηλαδή μία κλάση δεν μπορεί να είναι ταυτόχρονα και οντότητα ή ιδιότητα). Αυτό σημαίνει ότι οι περιορισμοί δεν μπορούν να τοποθετηθούν πάνω σε στοιχεία της ίδιας της OWL (κάτι που επιτρέπεται στην OWL Full). Ως επί το πλείστον, η OWL DL απαιτεί οι ιδιότητες να είναι είτε ObjectProperties είτε DatatypeProperties (ιδιότητες μεταξύ οντοτήτων και ιδιότητες μεταξύ οντοτήτων και τιμών, αντίστοιχα).

Όπως έχουμε πει μέχρι τώρα η OWL DL αποτελεί επέκταση της OWL Lite και η OWL Full αποτελεί επέκταση της OWL DL. Συνεπώς τα χαρακτηριστικά της OWL Lite που μελετήσαμε αναλυτικά ισχύουν και για τις δύο άλλες εκδόσεις της OWL. Θα συνεχίσουμε με τα επιπλέον χαρακτηριστικά που παρέχουν οι εκδόσεις αυτές.

- *oneOf*: Η δομή αυτή χρησιμεύει για την απαρίθμηση των οντοτήτων που απαρτίζουν μια κλάση (enumerating). Για παράδειγμα η κλάση «ΜήνεςΤουΧρόνου» μπορεί να περιγραφεί απαριθμώντας τις οντότητες Ιανουάριος, Φεβρουάριος, Μάρτιος, Απρίλιος, Μάιος, Ιούνιος, Ιούλιος, Αύγουστος, Σεπτέμβριος, Οκτώβριος, Νοέμβριος, Δεκέμβριος. Από την κλάση αυτή μπορεί

να εξαχθεί το συμπέρασμα ότι η μέγιστη πολλαπλότητά της είναι 12.

- *hasValue*: Μια ιδιότητα μπορεί να απαιτηθεί να έχει μία συγκεκριμένη οντότητα ως τιμή. Για παράδειγμα τα στιγμιότυπα της κλάσης «ΥπάλληλοιΑΣΟΕΕ» μπορούν να χαρακτηριστούν ως οι άνθρωποι που έχουν την οντότητα «ΑΣΟΕΕ» της κλάσης «Εργοδότες», σαν τιμή της ιδιότητας «έχειΕργοδότη».

- *disjointWith*: Οι κλάσεις μπορούν να δηλωθούν ρητά ότι είναι διαφορετικές ανά δύο. Αν δε γίνει αυτό δεν υποδηλώνεται ότι είναι διαφορετικές οπότε δεν θα έχουμε σωστά συμπεράσματα κατά τη διαδικασία της εξαγωγής συμπερασμάτων. Για παράδειγμα οι κλάσεις «Άντρας» και «Γυναίκα» πρέπει να δηλωθούν ως disjoint κλάσεις. Αν μία οντότητα είναι στιγμιότυπο της μίας δεν θα είναι της άλλης και αντίστροφα.

- *unionOf*, *complementOf*, *intersectionOf*: Η OWL DL και η OWL Full επιτρέπουν αυθαίρετους Boolean συνδυασμούς κλάσεων και περιορισμών. Η δομή *unionOf* μας δίνει την ένωση, η *complementOf* μας δίνει το αντίθετο σύνολο και η *intersectionOf* μας δίνει την τομή.

- *minCardinality*, *maxCardinality*, *cardinality*: Αντίθετα με την OWL Lite έχουμε πλήρη πολλαπλότητα. Οι περιορισμοί μπορούν να πάρουν οποιεσδήποτε τιμές και όχι μόνο 0 ή 1.

- *complex classes*: Αυτό είναι ένα χαρακτηριστικό της OWL Full μόνο και αυτό που κάνει είναι να επεκτείνει την ιδιότητα της OWL Lite να περιορίζει την σύνταξη σε μοναδικά ονόματα κλάσεων (π.χ. στις δηλώσεις *subClassOf* και *equivalentClass*). Έτσι επιτρέπονται αυθαίρετα πολύπλοκες περιγραφές κλάσεων, που μπορεί να περιλαμβάνουν απαριθμήσιμες κλάσεις, περιορισμούς ιδιοτήτων και Boolean συνδυασμούς. Επίσης, η OWL Full, σε αντίθεση με τις OWL Lite και OWL DL, επιτρέπει στις κλάσεις να χρησιμοποιούνται ως στιγμιότυπα.

Κεφάλαιο 3ο: Εργαλεία οντολογιών

3.1 Εισαγωγή

Στο κεφάλαιο αυτό θα μελετήσουμε συνοπτικά τα σπουδαιότερα εργαλεία για την συγγραφή και τη διαχείριση οντολογιών. Τα εργαλεία για τη συγγραφή οντολογιών (ontology editors) μας παρέχουν μια εύχρηστη διεπαφή χρήστη, η οποία μας επιτρέπει να δημιουργούμε εύκολα και γρήγορα οντολογίες (απλές συνήθως αλλά και πιο πολύπλοκες), χωρίς να είμαστε υποχρεωμένοι να θυμόμαστε τη σύνταξη της γλώσσας οντολογιών που χρησιμοποιούμε. Επίσης μας παρέχουν ένα σύνολο από άλλες χρήσιμες λειτουργίες, όπως είναι η εξαγωγή συμπερασμάτων, ο καθορισμός της γλώσσας της οντολογίας που έχουμε δημιουργήσει μέχρι στιγμής (π.χ. OWL Lite, OWL DL ή OWL Full), η υποστήριξη διαφορετικών εκδόσεων της οντολογίας μας, ο καθορισμός και η τροποποίηση των εννοιών, των ιδιοτήτων, των περιορισμών.

Τα εργαλεία που έχουμε στη διάθεσή μας για τη διαχείριση των οντολογιών μας παρέχουν μια προγραμματιστική διεπαφή (API), ώστε να μπορούμε να μεταχειριστούμε την οντολογία που έχουμε δημιουργήσει προγραμματιστικά. Για παράδειγμα μπορούμε να διαβάζουμε μια οντολογία μέσω ενός προγράμματος Java, να την διατρέχουμε και να βλέπουμε ποιες είναι οι κλάσεις της, ποιες οι ιδιότητες, ποιες οι οντότητες και να επιτελέσουμε ό,τι λειτουργία θέλουμε πάνω σε αυτές.

3.2 Εργαλεία συγγραφής οντολογιών

Θα δούμε συνοπτικά τα σπουδαιότερα εργαλεία που ανακαλύψαμε κατά την έρευνά μας και θα εστιάσουμε σε εκείνο το οποίο θα αποτελέσει την τελική μας επιλογή, για τη δημιουργία της οντολογίας μας.

3.2.1 JOE (Java Ontology Editor)

Το JOE υποστηρίζει την δημιουργία οντοτήτων, ιδιοτήτων και συσχετίσεων για οντολογίες. Στο εργαλείο αυτό, μια οντολογία θεωρείται

μια συσχέτιση οντότητας – ιδιοτήτων ή αλλιώς ένα μοντέλο πλαισίων – σχισμών (frame – slot model). Ο σκοπός του είναι να παρέχει μια γραφική διεπαφή για την αναπαράσταση οντολογιών σε ανοικτά κατανεμημένα περιβάλλοντα. Το JOE ουσιαστικά είναι μια μικροεφαρμογή Java (applet), που αποτελείται από δύο κυρίως μέρη: ένα για τη συγγραφή οντολογιών και ένα για τη συγγραφή επερωτήσεων. Δεν υποστηρίζει όμως πιο περίπλοκες λειτουργίες όπως είναι η υποστήριξη διαφορετικών εκδόσεων οντολογιών.

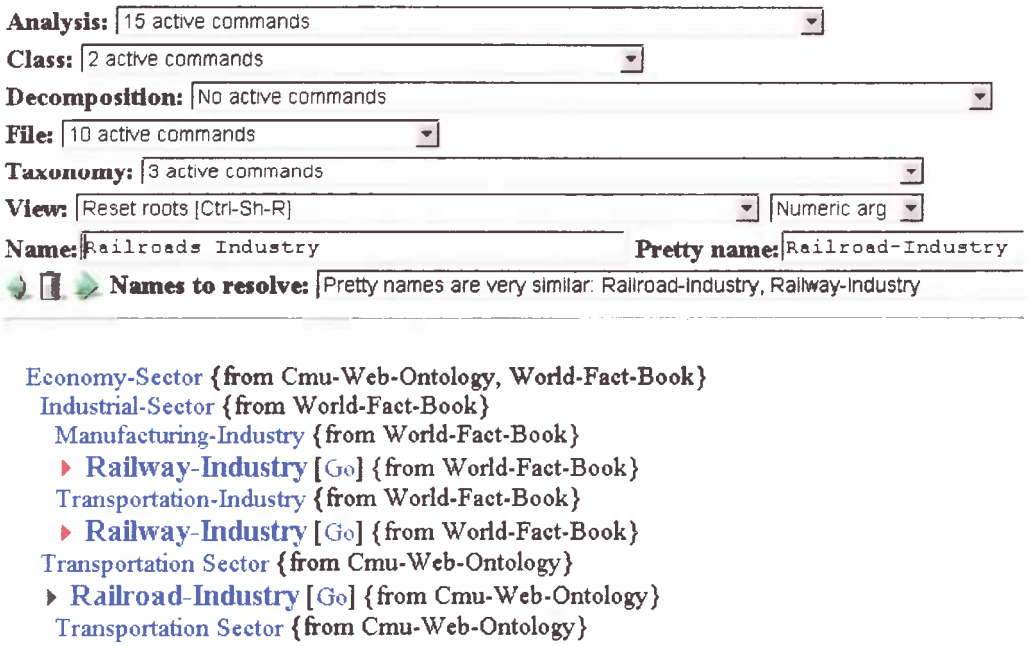
3.2.2 Ontolingua

Το εργαλείο αυτό αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης οντολογιών και καθιστά εφικτή την ανάγνωση, την δημιουργία, τη συγγραφή, την τροποποίηση και την χρήση οντολογιών. Περιλαμβάνει εργαλεία δημιουργίας οντολογιών μέσω της συγκέντρωσης και της επέκτασης οντολογιών, τις οποίες εξάγει από μια βιβλιοθήκη με επαναχρησιμοποιήσιμες οντολογίες. Συνδυάζει αξιώματα, ορισμούς και μη-λογικά σύμβολα τα οποία ονομάζει «λέξεις» από πολλαπλές οντολογίες.

3.2.3 Chimaera

Η Chimaera είναι ένα διαγνωστικό και συγχωνευτικό περιβάλλον οντολογιών, βασισμένο σε διεπαφή ιστοσελίδας. Με την έννοια συγχωνευτικό εννοούμε ότι δέχεται πολλές επιλογές ως είσοδο (μεταξύ άλλων δέχεται KIF, Ontolingua, Protégé και οποιαδήποτε άλλη γλώσσα μορφής OKBC – Open Knowledge Base Connectivity). Περιλαμβάνει δικό της περιβάλλον συγγραφής οντολογιών, αλλά είναι δυνατό για τους χρήστες να χρησιμοποιήσουν και το περιβάλλον της Ontolingua. Ένα δείγμα από την διεπαφή της Chimaera φαίνεται στην Εικόνα 6.

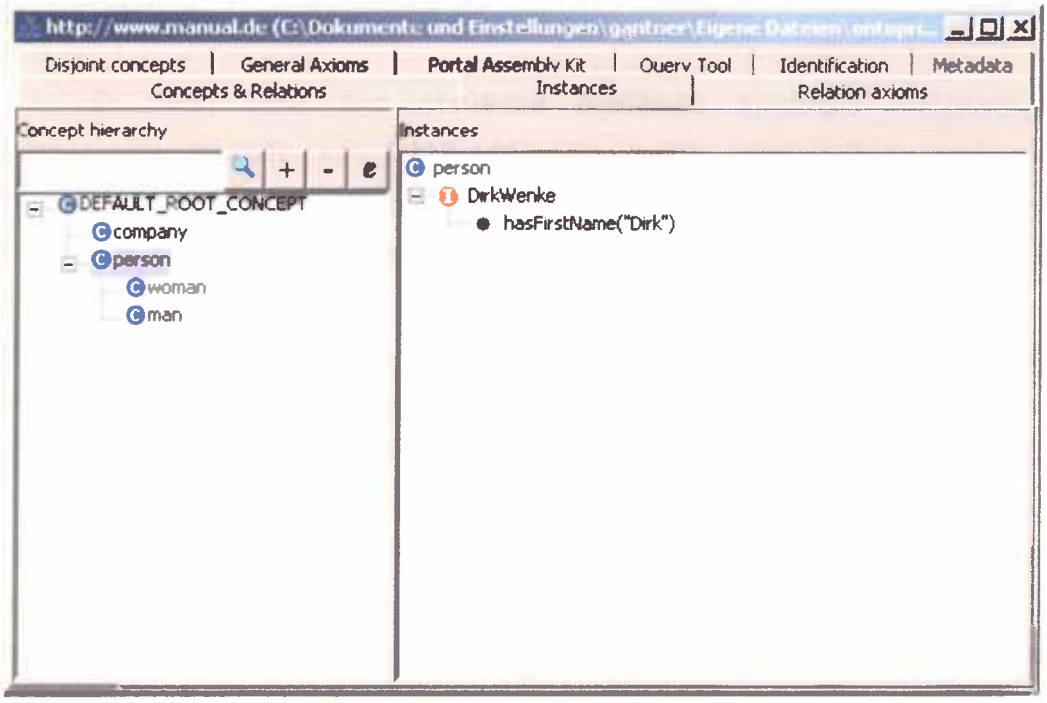
Η Chimaera έχει δυνατότητα ανάλυσης ώστε να πραγματοποιεί ελέγχους έλλειψης πληρότητας, σημασιολογικούς και συντακτικούς ελέγχους, καθώς και ταξονομική ανάλυση. Κάποιες από τις βασικότερες λειτουργίες της είναι η ανάγνωση βάσεων γνώσεων σε διαφορετικές μορφές (formats), οργάνωση ταξονομιών, επίλυση συγκρούσεων ονομάτων (name conflicts), ανάγνωση οντολογιών και σύνταξη όρων.



Εικόνα 6: Η γραφική διεπαφή της Chimaera

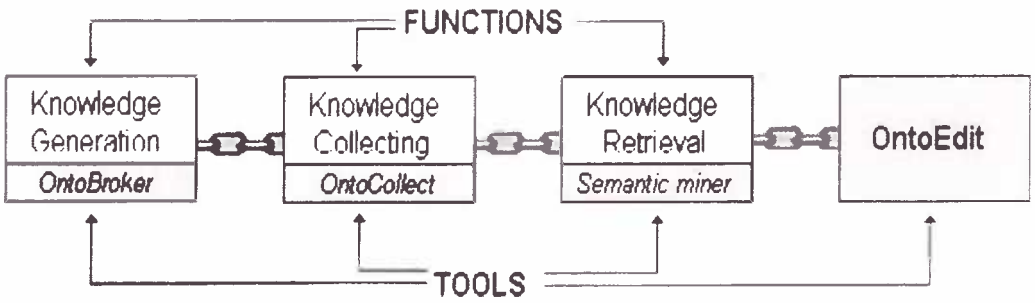
3.2.4 OntoEdit

Λόγω του μεγάλου όγκου και της μεγάλης πολυπλοκότητας των οντολογιών το εργαλείο αυτό υιοθετεί την αφομοίωση μετά από συνεργατική ανάπτυξη οντολογιών. Δίνει τη δυνατότητα να εισάγουμε γνωσιακά μοντέλα για συστήματα λογισμικού. Συνδυάζει την ανάπτυξη οντολογιών βασισμένη σε μεθοδολογία και την εξαγωγή συμπερασμάτων. Το OntoEdit επικεντρώνεται στα εξής σημεία: προδιαγραφές ή απαιτήσεις (η συλλογή των απαιτήσεων για την οντολογία), εκλέπτυνση (ημι-τυπική περιγραφή της οντολογίας χρησιμοποιώντας μια γλώσσα αναπαράστασης) και αξιολόγηση (η διαδικασία που ακολουθούν οι χρήστες και οι προγραμματιστές για να φανεί η χρησιμότητα της οντολογίας). Η γραφική διεπαφή του εργαλείου φαίνεται στην Εικόνα 7.



Εικόνα 7: Δείγμα από το γραφικό περιβάλλον του OntoEdit

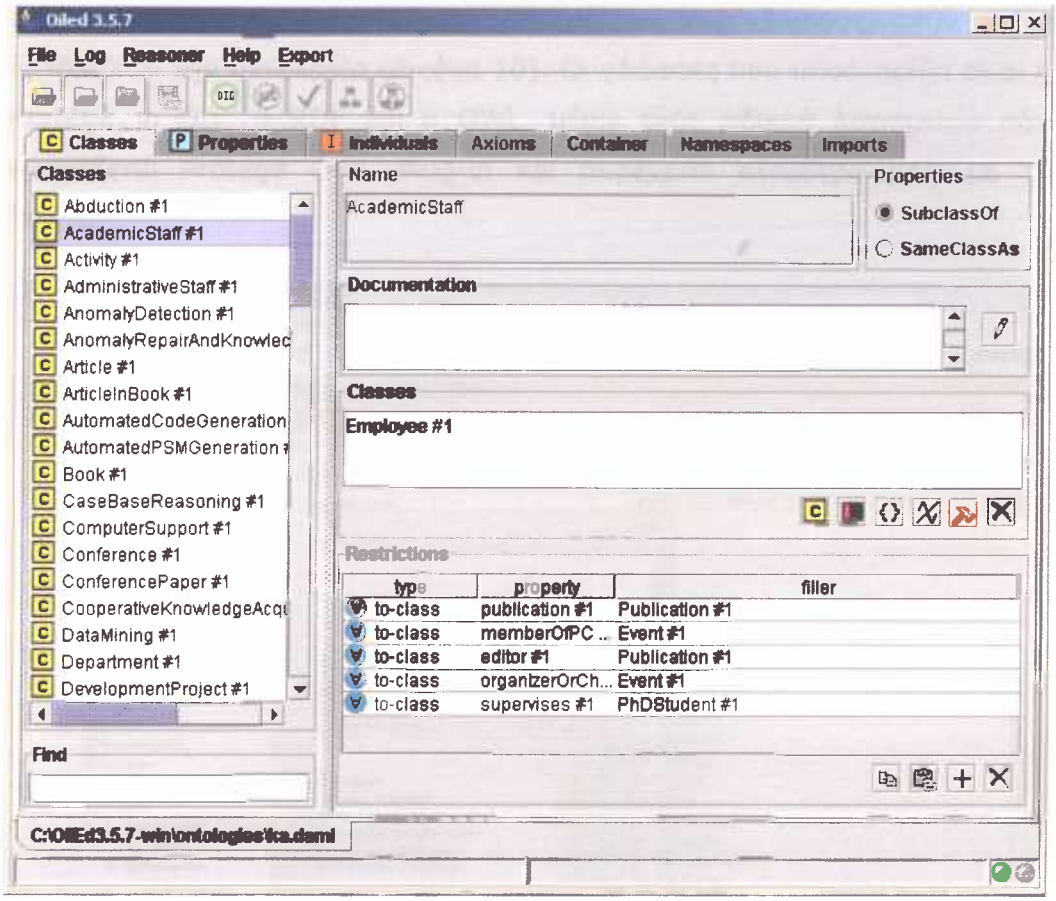
Το μοντέλο δεδομένων του OntoEdit είναι το OXML 2.0, το οποίο είναι βασισμένο σε πλαίσια και παρέχει μετα-κλάσεις, κατηγορήματα και αξιώματα. Το εργαλείο υποστηρίζει τις εξής γλώσσες: F-Logic (Fuzzy Logic), RDF Schema, OIL. Επίσης, αλληλεπιδρά με άλλα εργαλεία για να πετύχει μια δομή που ονομάζεται «Σημασιολογική Αλυσίδα Τιμών» (Semantic Value Chain) και παρουσιάζεται στην Εικόνα 8.



Εικόνα 8: Μια σημασιολογική αλυσίδα τιμών με αναφορά στο OntoEdit

3.2.5 OilEd

Ο OilEd είναι ένα ελεύθερα διαθέσιμο εργαλείο συγγραφής οντολογιών σε OIL και DAML+OIL, το οποίο αναπτύχθηκε από το Πανεπιστήμιο του Manchester και χρησιμοποιεί εξαγωγή συμπερασμάτων για να υποστηρίξει το σχεδιασμό και την συντήρηση οντολογιών. Το μεγαλύτερο πλεονέκτημά του είναι ότι διαθέτει μια επέκταση ενός προτύπου εργαλείου συγγραφής ώστε να διαχειριστεί μία γλώσσα αναπαράστασης και μία μηχανή εξαγωγής συμπερασμάτων βασισμένη σε περιγραφική λογική, η οποία χρησιμοποιείται για να ελέγξει τη συνέπεια των κλάσεων και να εξάγει συμπεράσματα από τις συσχετίσεις. Παρέχει ορισμούς κλάσεων, σχισμές (slots – ιδιότητες) και αξιώματα. Σε αντίθεση με άλλα συστήματα παρέχει στους χρήστες τη δυνατότητα χρήσης αυθαίρετων Boolean συνδυασμών κλάσεων, συνδεδεμένων με *και*, *ή* και *όχι*. Παρόλα αυτά δεν υποστηρίζει διαφορετικές εκδόσεις ούτε πολλαπλές οντολογίες. Ένα δείγμα από την γραφική διεπαφή του OilEd φαίνεται στην Εικόνα 9. Το λογισμικό εξαγωγής συμπερασμάτων (Reasoner) ονομάζεται FaCT και είναι εύκολα προσβάσιμο από το κεντρικό μενού.

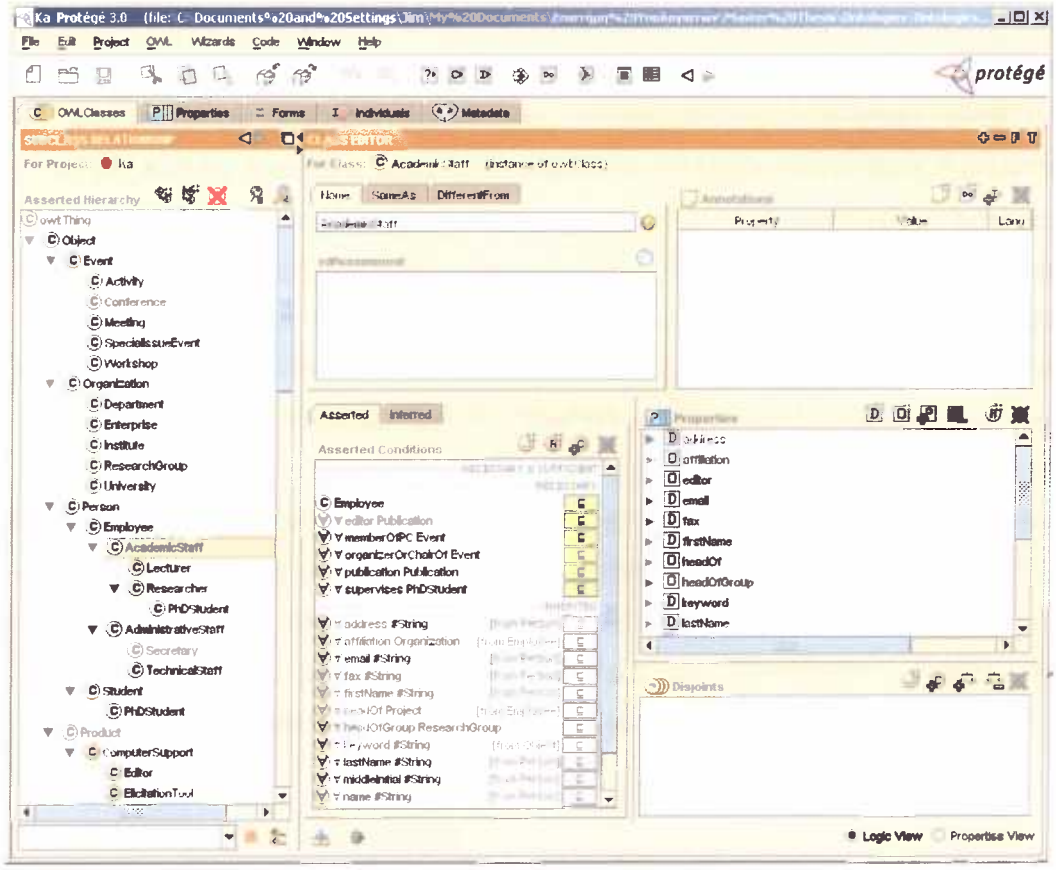


Εικόνα 9: Το γραφικό περιβάλλον του OilEd. Έχουμε φορτώσει μια έτοιμη οντολογία που περιγράφει τον ακαδημαϊκό χώρο, γραμμένη σε DAML+OIL.

3.2.6 Protégé

Το Protégé είναι ένα εργαλείο ανοιχτού κώδικα, γραμμένο σε Java. Επιτρέπει στους χρήστες να δημιουργήσουν μια οντολογία που να καλύπτει ένα γνωσιακό πεδίο, να παραμετροποιήσουν φόρμες ανάκτησης γνώσης και να εισάγουν πληροφορία σχετική με το γνωσιακό πεδίο. Έχει την ικανότητα να λειτουργεί ως πλατφόρμα που θα παρέχει πρόσβαση σε άλλα γνωσιακά συστήματα ή εμφωλευμένες εφαρμογές, είτε ως μια βιβλιοθήκη εύκολα προσβάσιμη από άλλες εφαρμογές με σκοπό την πρόσβαση και την οπτικοποίηση βάσεων γνώσης. Το Protégé διαθέτει ισχυρό βαθμό παραμετροποίησης και επιτρέπει την ανάκτηση γνώσης. Επίσης, διαθέτει μια πολύ εύχρηστη διεπαφή χρήστη που επιτρέπει στους σχεδιαστές οντολογιών να επικεντρωθούν στην μοντελοποίηση των εννοιών, χωρίς να

ασχολούνται με τις συντακτικές λεπτομέρειες της γλώσσας, στην οποία σχεδιάζουν την οντολογία (Εικόνα 10). Οι γλώσσες που υποστηρίζει είναι η Protégé, η OIL, η RDF και η OWL, μέσω ενός ειδικού λογισμικού που ονομάζεται Protégé OWL Plug-in και παρέχεται ελεύθερα μαζί με το Protégé.



Εικόνα 10: Δείγμα της διεπαφής του Protégé με χρήση του OWL Plug-in. Έχουμε φορτώσει την οντολογία που περιγράφει τον ακαδημαϊκό χώρο, γραμμένη σε OWL αυτή τη φορά.

3.3 Επιλογή εργαλείου συγγραφής

Η επιλογή του εργαλείου που θα χρησιμοποιήσουμε για την οντολογία μας ήταν κυρίως επιλογή μεταξύ δύο εργαλείων. Η Chimaera και το Ontolingua δεν διατίθενται δωρεάν για εγκατάσταση οπότε η χρήση αυτών απορρίπτεται. Κάτι παρόμοιο συμβαίνει και για τον JOE καθώς είναι

εφαρμογή applet και για να τη χρησιμοποιήσει κανείς πρέπει να επισκεφτεί την σελίδα της στο διαδίκτυο, πράγμα μη βολικό. Το OntoEdit απορρίπτεται λόγω της μη εύχρηστης διεπαφής χρήστη αλλά και της δυσκολίας την οποία αντιμετωπίσαμε στην εύρεση του πακέτου εγκατάστασης.

Συνεπώς μένουμε με την επιλογή του OilEd και του Protégé, καθώς μάλιστα αυτά τα εργαλεία είναι που παρέχουν καλύτερη υποστήριξη της OWL, η οποία όπως έχουμε ήδη αποφασίσει θα είναι η γλώσσα στην οποία θα αναπτύξουμε την οντολογία μας. Η τελική μας απόφαση είναι το Protégé, καθώς είναι ένα έργο που υπόκειται συνεχώς βελτιώσεις (μάλιστα μέσα σε λίγους μήνες διατέθηκε καινούρια έκδοση του εργαλείου, η οποία διαθέτει σημαντικές βελτιώσεις στο γραφικό περιβάλλον, και όχι μόνο, σε σχέση με την προκάτοχό της). Ειδικά όσον αφορά το γραφικό περιβάλλον το Protégé είναι κατά πολύ ανώτερο του OilEd, κάτι που φαίνεται και από τις εικόνες Εικόνα 9 και Εικόνα 10. Ένα άλλο σημαντικό πλεονέκτημά του που μας οδήγησε στην επιλογή του, είναι η πολύ καλή τεκμηρίωση που διαθέτει (documentation). Στην ιστοσελίδα του εργαλείου μπορεί να βρει κανείς αναλυτικούς οδηγούς για την δημιουργία οντολογιών (σε Protégé αλλά και σε OWL), έτοιμες οντολογίες, απαντήσεις σε προβλήματα και μια ενεργή κοινότητα χρηστών που μπορούν να βοηθήσουν άμεσα σε οποιοδήποτε πρόβλημα συναντήσει κανείς κατά την ενασχόλησή του με το εργαλείο.

3.4 Εργαλεία διαχείρισης οντολογιών

Στην σύγκριση που πραγματοποιήσαμε για τα εργαλεία αυτά συμπεριλάβαμε αυτά, τα οποία παρείχαν διαχείριση οντολογιών προγραμματιστικά, δηλαδή μία πλήρη προγραμματιστική διεπαφή (API). Αυτά τα οποία απλά παρείχαν μόνο αποθήκευση οντολογιών ή μόνο εξαγωγή συμπερασμάτων, δεν τα αναφέρουμε.

3.4.1 EOR (Extensible Open RDF)

Το EOR αποτελεί ένα έργο ανοικτού κώδικα και είναι αποτέλεσμα της συνεργασίας του OCLC Office of Research και του Dublin Core Metadata

Initiative. Ο σκοπός του είναι να αναδείξει την ταχύτατη ανάπτυξη των εφαρμογών RDF, παρέχοντας δυνατότητες σχεδιασμού διεπαφής εστιάζοντας περισσότερο στην ανακάλυψη, τη διαχείριση, την ενσωμάτωση και την πλοήγηση σε μετα-δεδομένα. Αποτελείται από μια συλλογή επεκτάσιμων κλάσεων Java και υπηρεσιών, οι οποίες λειτουργούν ως μια έτοιμη βιβλιοθήκη κώδικα, η οποία μπορεί να χρησιμοποιηθεί για την ανάπτυξη μιας πλειάδας εφαρμογών, όπως είναι η συλλογή μετα-δεδομένων, οι μηχανές αναζήτησης κλπ. Έτσι, η βασική λειτουργικότητα που παρέχει το εργαλείο αυτό περιλαμβάνει τη δημιουργία, τη διαγραφή και τη διαχείριση RDF βάσεων δεδομένων.

3.4.2 Redland

Το εργαλείο αυτό έχει αναπτυχθεί στο Πανεπιστήμιο του Bristol και είναι μια βιβλιοθήκη που παρέχει μία υψηλού επιπέδου διεπαφή για την αποθήκευση, την πραγματοποίηση επερωτήσεων και τη διαχείριση RDF μοντέλων. Το Redland υλοποιεί καθένα από τα στοιχεία της RDF σε δική του κλάση, καθιστώντας το έτσι ένα αντικειμενοστραφές API, βασισμένο σε αυτά.

3.4.3 Jena

Το Jena είναι ένα περιβάλλον λογισμικού που αναπτύχθηκε από την εταιρεία Hewlett-Packard και είναι μια συλλογή εργαλείων RDF γραμμένα σε Java. Περιλαμβάνονται ένα Java API, ένας parser RDF, ένα σύστημα επερωτήσεων σε οντολογίες, κλάσεις για υποστήριξη οντολογιών DAML+OIL και OWL και αποθήκευση οντολογιών.

3.5 Επιλογή εργαλείου διαχείρισης

Η επιλογή μας βασίστηκε στην καλύτερη και πληρέστερη υποστήριξη της γλώσσας OWL. Ένα άλλο σημαντικό πλεονέκτημα είναι η ύπαρξη επαρκούς documentation του εργαλείου. Το Jena υπερείχε σε αυτές τις κατηγορίες και ως εκ τούτου είναι το εργαλείο της επιλογής μας. Παρέχει εκτός του RDF API, ένα ξεχωριστό OWL API και διαθέτει για αυτό πλήρη

τεκμηρίωση με έγγραφα που περιγράφουν τις βασικές λειτουργίες του, έγγραφα εκμάθησης (tutorials) και πλήρες JavaDoc για όλες τις κλάσεις του.

Κεφάλαιο 4ο: Η εφαρμογή

4.1 Εισαγωγή

Η εργασία αυτή βασίζεται στην Διπλωματική εργασία της Γιγουρτσή Ευθυμίας, με τίτλο «Ανάκτηση Πληροφορίας κατά Τμήματα: Μία Προσέγγιση για Συστήματα Ερωταποκρίσεων». Στην εργασία αυτή έχει υλοποιηθεί ένα σύστημα ερωταποκρίσεων, δηλαδή ένα σύστημα ανάκτησης πληροφορίας, το οποίο έχει ως βάση δεδομένων ένα σύνολο κειμένων που ανακτήθηκε από τις ιστοσελίδες ελληνικών Πανεπιστημίων. Σκοπός της παρούσης εργασίας είναι η βελτίωση του συστήματος αυτού σε δύο ξεχωριστά μεταξύ τους στάδια.

Στο πρώτο στάδιο επιχειρούμε τη βελτίωση της βάσης δεδομένων, χρησιμοποιώντας κάποιο πρόγραμμα crawler, το οποίο να είναι ελεύθερα διαθέσιμο στο διαδίκτυο. Το προηγούμενο πρόγραμμα Crawler, αν και δούλευε σωστά είχε αρκετά προβλήματα. Έτσι εστιάσαμε στην ανεύρεση ενός καλύτερου έτοιμου προγράμματος, το οποίο όμως να διαθέτει τον κώδικά του, ώστε να μπορούμε να το ενσωματώσουμε στο σύστημα.

Στο δεύτερο στάδιο επιχειρούμε να πραγματοποιήσουμε την ανάκτηση πληροφορίας χρησιμοποιώντας οντολογίες. Οι τρόποι που μπορεί να γίνει αυτό είναι αρκετοί και περιγράφονται στο επόμενο κεφάλαιο. Αυτό που υλοποιήσαμε εμείς ήταν η ένταξη στο σύστημα μιας απλής οντολογίας, η οποία περιείχε τα συνώνυμα των πανεπιστημίων, τα οποία θα μπορούσε να συμπεριλάβει ο χρήστης στις επερωτήσεις του. Η διαδικασία αυτή γινόταν πριν με έναν πίνακα στον οποίο ήταν αποθηκευμένα τα συνώνυμα και η χρήση της οντολογίας κάνει πιο εύχρηστη τη διαχείριση του συνόλου αυτού των συνωνύμων. Στη συνέχεια βελτιώσαμε τα αποτελέσματα που επιστρέφει η μηχανή αναζήτησης εμπλουτίζοντας το ερώτημα του χρήστη με λέξεις – κλειδιά τις οποίες ανακτάμε από την οντολογία. Για το σκοπό αυτό επεκτείναμε την οντολογία μας, που αρχικά περιείχε μόνο τα συνώνυμα, με μερικές κλάσεις, υποκλάσεις και ιδιότητες. Την οντολογία αυτή θα παρουσιάσουμε στη συνέχεια.

4.2 Δημιουργία βάσης δεδομένων

4.2.1 Τι είναι οι Crawlers

Αρχικά πραγματοποιήσαμε μια μελέτη των υπαρχόντων Web Crawlers, ώστε να επιλεγεί ένας από αυτούς για χρήση του στο σύστημα ερωταποκρίσεων. Ο Web Crawler που θα επιλεγεί πρέπει να πληροί κάποια χαρακτηριστικά. Κύρια πηγή εύρεσης Crawlers ήταν η ιστοσελίδα των Web Robots (http://www.robotstxt.org/wc/robots.html). Αφού η μελέτη αυτή πραγματοποιήθηκε, καταλήξαμε σε δύο-τρεις επικρατέστερους Crawlers και επιλέξαμε έναν ώστε να ενσωματώσουμε στην εφαρμογή του συστήματος ανάκτησης. Στη συνέχεια θα περιγράψουμε τους Web Crawlers καθώς και την εφαρμογή που αναπτύξαμε.

Το Διαδίκτυο έχει αναπτυχθεί σημαντικά στις μέρες μας. Από την αρχή του, καθώς μεγάλωνε σε βαθμό που δεν ήταν πλέον εφικτό να το διασχίσει κανείς χειροκίνητα, φάνηκε η ανάγκη για πιο αυτοματοποιημένες διαδικασίες που θα βοηθούσαν στην ανακάλυψη πληροφοριών.

Έτσι αναπτύχθηκαν οι εφαρμογές Web Crawlers. Ένας Web Crawler (εναλλακτικά ονομάζεται robot ή spider) είναι ένα πρόγραμμα που διασχίζει αυτόματα το διαδίκτυο σύμφωνα με τη δομή του. Ανακτά ένα αρχείο υπερκειμένου ως αφετηρία και αναδρομικά ανακτά όλα τα κείμενα που αναφέρονται σε αυτό.

Για την δημιουργία της βάσης δεδομένων πρέπει να χρησιμοποιήσουμε ένα τέτοιο πρόγραμμα Crawler, το οποίο συλλέγει τις επιλεγμένες ιστοσελίδες από το διαδίκτυο και τις αποθηκεύει στον σκληρό δίσκο του χρήστη. Οι ιστοσελίδες αυτές πρέπει στη συνέχεια να μετατραπούν σε απλά αρχεία κειμένου, ώστε να δημιουργηθεί η βάση με τα κείμενα, η οποία δίνεται ως είσοδος στο πρόγραμμα Lucene. Το πρόγραμμα αυτό πραγματοποιεί τη δεικτοδότηση των κειμένων, ώστε να εμφανίζονται τα αποτελέσματα που ζητάμε μετά από κάποια αναζήτηση στο σύστημα ερωταποκρίσεων.

Οι Crawlers μπορούν να χρησιμοποιηθούν για πολλές χρήσιμες λειτουργίες:

- *Στατιστική ανάλυση*: Για παράδειγμα θα μπορούσαμε να αναλύσουμε στατιστικά όπως ο αριθμός των υπαρχόντων Web Servers, ο μέσος αριθμός κειμένων ανά server, η αναλογία συγκεκριμένων τύπων αρχείων, το μέσο μέγεθος μίας ιστοσελίδας κλπ.

- *Συντήρηση (maintenance)*: Ένας crawler μπορεί να βοηθήσει στη διατήρηση της σωστής δομής του διαδικτύου καθώς μπορεί να ανακαλύψει τους σπασμένους συνδέσμους (broken links – σύνδεσμοι που αναφέρονται σε περιεχόμενο το οποίο δεν υπάρχει πια). Σε διαφορετική περίπτωση είναι πολύ δύσκολη η εύρεσή τους, καθώς θα πρέπει είτε ο ιδιοκτήτης μιας ιστοσελίδας να τους ανακαλύψει μόνος του, είτε να τους ανακαλύψει κάποιος χρήστης και να το αναφέρει στον ιδιοκτήτη μέσω email, πράγμα πολύ σπάνιο.

- *Mirroring*: Το mirroring είναι μια διαδεδομένη τεχνική για την διατήρηση αρχείων FTP. Ένα πρόγραμμα mirror αντιγράφει αναδρομικά ένα ολόκληρο δέντρο καταλόγων και στη συνέχεια ανακτά σε τακτά χρονικά διαστήματα τα έγγραφα που έχουν αλλάξει. Υπάρχει το πρόβλημα των αναφορών όμως σε αυτά τα έγγραφα, καθώς θα πρέπει να αλλάξουν για να δείχνουν στις σελίδες που έχουν γίνει mirrored (αν έχουν γίνει). Γενικά αν θέλουμε να αντιγράψουμε το σημαντικότερο περιεχόμενο μιας ιστοσελίδας στον δίσκο μας, ένας crawler μπορεί να το πετύχει σε πολύ καλό βαθμό.

- *Ανακάλυψη πληροφοριών*: Οι crawlers μπορούν να βοηθήσουν σημαντικά στην ανακάλυψη πληροφοριών, καθώς είναι πολύ πιο εύκολο να αφήσει κανείς έναν υπολογιστή να κάνει αυτή τη λειτουργία παρά να αντιμετωπίσει μόνος του τον τεράστιο όγκο πληροφοριών που είναι διαθέσιμος. Υπάρχουν αρκετοί crawlers που δεικτοδοτούν μεγάλα μέρη του web και παρέχουν πρόσβαση σε μία βάση δεδομένων με τα αποτελέσματα αυτά μέσω μιας μηχανής αναζήτησης. Έτσι ακόμα και αν η βάση δεδομένων δεν περιέχει την πληροφορία που αναζητάει ένας χρήστης το

πιθανότερο είναι ότι θα περιέχει τουλάχιστον αναφορές σε σχετικές σελίδες που θα περιέχουν αυτό που ψάχνει.

- *Συνδυασμένες χρήσεις*: Κάποιοι crawlers (ελάχιστοι δυστυχώς) έχουν τη δυνατότητα να πραγματοποιούν περισσότερες από μία από τις παραπάνω λειτουργίες. Για παράδειγμα στατιστική ανάλυση αλλά και ανακάλυψη πληροφοριών (RBSE Spider).

4.2.2 Αξιολόγηση Web Crawlers

Μετά από διεξοδική έρευνα στην ιστοσελίδα των Web Crawlers αλλά και γενικά στο διαδίκτυο καταλήξαμε σε 15 περίπου προγράμματα crawlers κάποια από τα οποία επιτελούν τη λειτουργία που επιθυμούμε, δηλαδή κατέβασμα μίας ιστοσελίδας σε ένα τοπικό directory στον σκληρό δίσκο. Τα προγράμματα αυτά και κάποια αξιολόγηση των χαρακτηριστικών τους φαίνονται στον παρακάτω πίνακα:

Crawlers	Γλώσσα Προγραμματισμού	Ανοιχτού Κώδικα	Χαρακτηριστικά	Αρνητικά
arale	Java	√	Απορρίπτει εξωτερικούς συνδέσμους, μετονομάζει τα URL, μετατρέπει δυναμικές ιστοσελίδες σε στατικά έγγραφα, επιλογή των τύπων αρχείων που θα κατέβουν	Δεν κατεβάζει σωστά όλες τις σελίδες από το ίδιο domain.
acme.spider	Java	√	BFS αναζήτηση, αναφέρει σπασμένους συνδέσμους	Επισκέπτεται εξωτερικούς συνδέσμους

agent	Java	√		Δεν λειτούργησε
bloodhound	Perl	X		
checkbot	Perl	√	Ελέγχει την εγκυρότητα των συνδέσμων	
collective	Perl	X		
CyberSpyder 21		X	Ελέγχει την εγκυρότητα των συνδέσμων	Δεν λειτούργησε
Heritrix	Java	√	Λειτουργεί σε Linux	
JoBo	Java	√	Απορρίπτει εξωτερικούς συνδέσμους, κάθε σελίδα κατεβαίνει μια φορά, αναφέρει σπασμένους συνδέσμους, κατεβάζει ολόκληρο το domain, επιλογή των τύπων αρχείων που θα κατέβουν	Κολλάει σε κάποιες σελίδες
mnogosearch	C	√	Πραγματοποιεί δεικτοδότηση και αναζήτηση στον Ιστό	
ookce	Java	√		Γαλλική γλώσσα

				προγράμματος και τεκμηρίωσης, δεν λειτούργησε
SiteSearcher		X		Δοκιμαστική έκδοση
WebMirror		X		Δοκιμαστική έκδοση
WebReaper		X	Δωρεάν χρήση	
websphinx	Java	√	Βιβλιοθήκη κλάσεων Java	Δεν λειτούργησε

Συμπερασματικά καταλήγουμε σε τρία από αυτά τα προγράμματα, το *Arale*, το *Acme.spider* και το *JoBo*. Θεωρητικά χρήσιμα είναι και τα *Agent* και *Websphinx*, όμως δεν καταφέραμε να τα χρησιμοποιήσουμε. Παρόλα αυτά διαθέτουν ελεύθερα τον κώδικά τους και θα μπορούσαμε να βασιστούμε σε αυτόν μελλοντικά για την ανάπτυξη του δικού μας crawler.

Τα *bloodhound*, *checkbot*, *collective* απορρίπτονται γιατί είναι γραμμένα σε Perl. Το *checkbot* εξάλλου απλά ελέγχει για την εγκυρότητα των συνδέσμων, δεν κατεβάζει ιστοσελίδες. Για αυτόν τον λόγο απορρίπτουμε και το *CyberSpyder21*. Το *Heritrix* παρουσιάζει πολύ καλή εικόνα με όλον τον κώδικά του γραμμένο σε Java και είναι ελεύθερα διαθέσιμο, αλλά προορίζεται για χρήση κάτω από το λειτουργικό σύστημα Linux για αυτό και το απορρίπτουμε. Το *mnogosearch* είναι γραμμένο σε C και το *ookce*, ενώ είναι γραμμένο σε Java, η γλώσσα που είναι γραμμένο το documentation είναι η γαλλική και δεν καταφέραμε τελικά να το χρησιμοποιήσουμε. Τέλος τα προγράμματα *SiteSearcher*, *WebMirror* και *WebReaper* δεν παρέχουν τον κώδικά τους αλλά είναι εφαρμογές windows, είτε demo είτε freeware με δικό τους interface μη παραμετροποιήσιμο, για αυτό και δεν αποτελούν πιθανή επιλογή για εμάς.

Καταλήξαμε λοιπόν στα προγράμματα *Acme.spider*, *Arale* και *JoBo*. Το *Acme.spider* με τη χρήση της κλάσης WebCopy.java η οποία παρέχεται, θα πρέπει να πραγματοποιεί το «κατέβασμα» μιας ιστοσελίδας. Αυτό συμβαίνει αλλά ακόμα κι αν δώσουμε την παράμετρο "-d 1" που σημαίνει να περιοριστεί σε βάθος 1 κατεβάζει πολλά sites τα οποία είναι εξωτερικά links του site του οποίου μας ενδιαφέρει. Έτσι έχουμε πολλή άχρηστη πληροφορία και σπατάλη bandwidth. Η κλάση όμως αυτή παρέχεται σαν δοκιμαστική έκδοση και δεν αποτελεί μέρος του προγράμματος. Έτσι η χρήση του ενδείκνυται περισσότερο για όποιον θέλει να δημιουργήσει τον δικό του Crawler και να χρησιμοποιήσει το *Acme.spider* σαν βιβλιοθήκη κλάσεων.

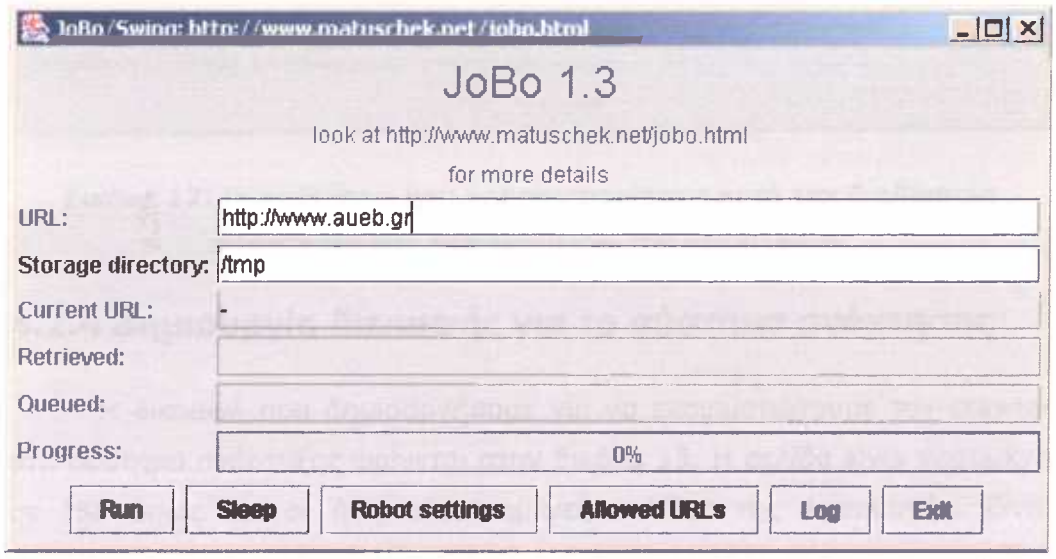
Το *Arale* το οποίο έχει πολλές επιλογές και εκτελείται από τη γραμμή εντολών. Είναι μέλος του Jakarta project. Κάποιες από τις επιλογές είναι επιλογή βάθους, επιλογή links σε διαφορετικό domain, επιλογή αριθμού ταυτόχρονων συνδέσεων για μεγαλύτερη ταχύτητα (αλλά και περισσότερη χρήση των διαθέσιμων πόρων), επιλογή μετατροπής των δυναμικών ιστοσελίδων όπως jsp, asp, php σε html, ούτως ώστε να μη χάνουμε καμία πληροφορία. Επίσης επιτρέπει τον περιορισμό των αρχείων που θα κατέβουν, π.χ. αποκλείουμε αν θέλουμε τα pdf και τα jpg. Από την άλλη δυσκολία αντιμετωπίσαμε στο κατέβασμα των διευθύνσεων που βρίσκονται στο ίδιο domain, κάτι που δε συνέβη με το *JoBo*.

Το πρόγραμμα *JoBo* είναι εξαιρετικό καθώς διαθέτει πολλές επιλογές όπως το βάθος αναζήτησης, την αναζήτηση σε site του ιδίου domain, να περιορίζει το bandwidth το οποίο χρησιμοποιεί, το κατέβασμα των δυναμικών σελίδων και διαθέτει και πολύ εύχρηστη γραφική διεπαφή εκτός από τη διεπαφή γραμμής εντολών. Έχει επίσης την σημαντική επιλογή του τύπου των αρχείων που επιθυμούμε. Για παράδειγμα στην εφαρμογή μας θέλουμε μόνο τα αρχεία html και όχι τα αρχεία ήχου, εικόνας κλπ. Έτσι θα είχαμε κάποια σπατάλη bandwidth, αν δεν μπορούσαμε να απορρίψουμε τα αρχεία αυτά. Τέλος, το JoBo είναι λογισμικό ανοικτού κώδικα, συνεπώς με κάποιες επεμβάσεις σε αυτόν θα μπορούσαμε να φέρουμε το πρόγραμμα στα μέτρα μας, αν κάτι δεν μας ικανοποιεί σε αυτό.

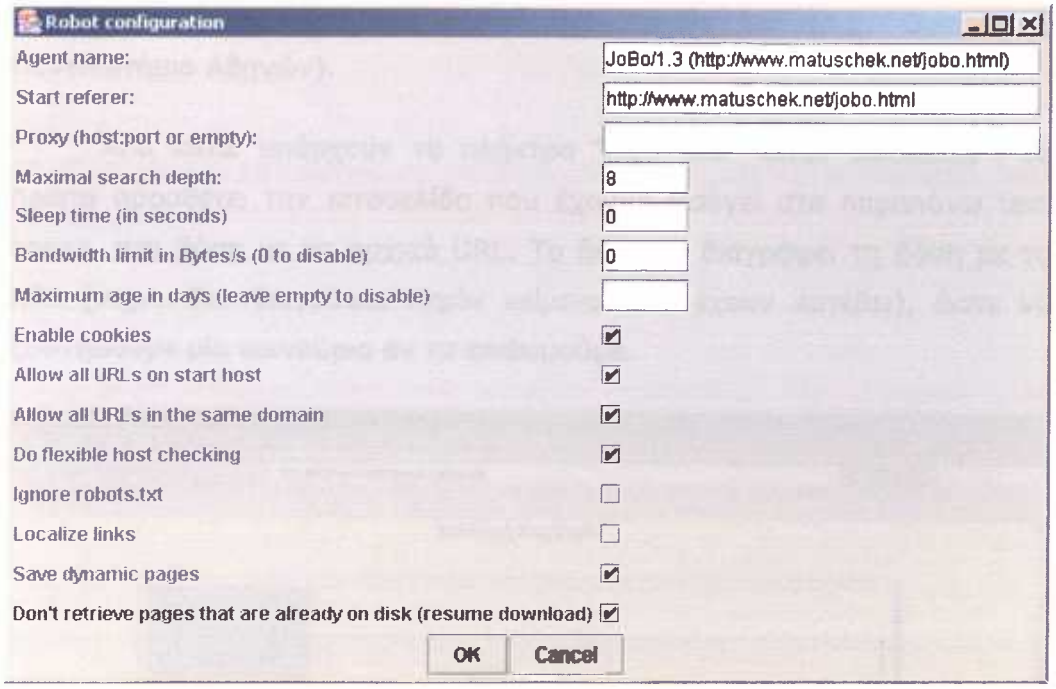
4.2.3 Επιλογή Crawler

Καταλήγουμε λοιπόν στην επιλογή του JoBo. Το πρόγραμμα αυτό με ρύθμιση βάθους 8, αν και έκανε αρκετή ώρα με χρήση ταχύτατης σύνδεσης (περίπου 3 ώρες με χρήση καλωδιακής σύνδεσης) κατέβασε στην εντέλεια όλες τις ιστοσελίδες που περιλαμβάνει ο server του Οικονομικού Πανεπιστημίου ξεκινώντας μόνο από τη διεύθυνση http://www.aueb.gr.

Στις παρακάτω εικόνες φαίνεται η γραφική διεπαφή του JoBo (Εικόνα 11) και οι ρυθμίσεις που χρησιμοποιήσαμε (Εικόνα 12) για να ελέγξουμε την αποτελεσματικότητά του.



Εικόνα 11: Η γραφική διεπαφή του JoBo



Εικόνα 12: Οι ρυθμίσεις που χρησιμοποιήσαμε κατά την διαδικασία ανάκτησης του περιεχομένου της ιστοσελίδας

4.2.4 Δημιουργία διεπαφής για το σύστημα ανάκτησης

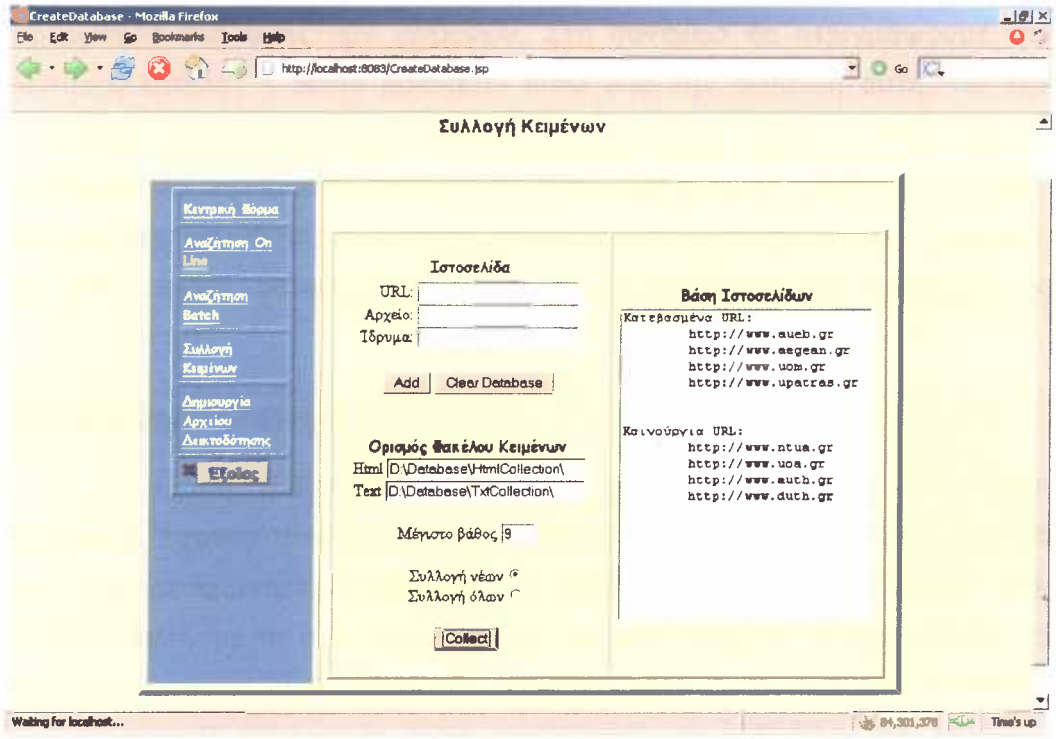
Η διεπαφή που δημιουργήσαμε για να ενσωματώσουμε τον crawler στο σύστημα ανάπτυξης φαίνεται στην Εικόνα 13. Η σελίδα είναι γραμμένη σε JSP όπως και οι ήδη υλοποιημένες σελίδες της εφαρμογής. Είναι χωρισμένη σε δύο μέρη, από τα οποία το πρώτο χωρίζεται και αυτό σε δύο μέρη.

Στο αριστερά μέρος έχουμε την λειτουργικότητα και στο δεξιά βλέπουμε την βάση με τα αρχικά URL, τα οποία θέλουμε να κατεβάσουμε στον σκληρό δίσκο. Αυτά χωρίζονται σε κατεβασμένα URL και σε νέα URL. Όταν προσθέτουμε ένα URL, αυτό προφανώς πηγαίνει στα νέα URL στην βάση δεδομένων.

Αυτό γίνεται με την επιλογή προσθήκης νέου URL, η οποία βρίσκεται στο αριστερά πάνω μέρος. Εκεί δίνουμε το URL (π.χ. http://www.aueb.gr), το όνομα που θα έχουν τα αρχεία html και κατ' επέκταση και τα txt, στον σκληρό δίσκο (π.χ. AUEB, οπότε τα αρχεία θα είναι της μορφής AUEB1.html, AUEB2.html, κ.ο.κ.) και τέλος το όνομα του ιδρύματος

αναλυτικά, ώστε να ενσωματωθεί στα αρχεία txt (π.χ. Οικονομικό Πανεπιστήμιο Αθηνών).

Από κάτω υπάρχουν τα πλήκτρα "Add" και "Clear Database". Το πρώτο προσθέτει την ιστοσελίδα που έχουμε εισάγει στα παραπάνω text boxes, στη βάση με τα αρχικά URL. Το δεύτερο διαγράφει τη βάση με τα URL (σημ.: δεν διαγράφει τυχόν κείμενα που έχουν κατέβει), ώστε να ξεκινήσουμε μία καινούρια αν το επιθυμούμε.



Εικόνα 13: Διεπαφή της εφαρμογής συλλογής των κειμένων

Κάτω από την προσθήκη ιστοσελίδας έχουμε την επιλογή να καθορίσουμε σε ποιον κατάλογο στον σκληρό δίσκο (του εξυπηρετητή στον οποίο τρέχει η εφαρμογή) θα αποθηκευτούν τα αρχεία. Αρχικά επιλέγουμε τον κατάλογο για τα html αρχεία και στη συνέχεια για τα txt. Αν αφήσουμε κάποιο πεδίο κενό η εφαρμογή δεν θα ξεκινήσει.

Στη συνέχεια επιλέγουμε το βάθος αναζήτησης στο οποίο θα φτάσει ο crawler. Όσο πιο μεγάλο είναι αυτό τόσο περισσότερα αποτελέσματα θα έχουμε αλλά και τόσο περισσότερος χρόνος θα χρειαστεί. Εμπειρικά μία καλή τιμή για το βάθος είναι 8 με 9.

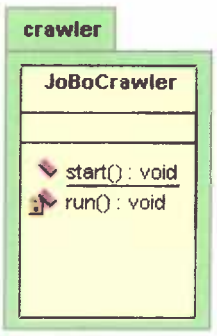
Τέλος έχουμε την επιλογή να κατεβάσουμε μόνο τις καινούριες ιστοσελίδες που έχουν προστεθεί στη βάση, είτε όλες που υπάρχουν σε αυτή (αφού έχουμε διαγράψει τα προηγούμενα αρχεία, για αποφυγή προβλημάτων). Πατώντας το πλήκτρο "Collect" ξεκινά η συλλογή των αρχείων.

4.2.5 Η εφαρμογή συλλογής κειμένων

Στην προηγούμενη παράγραφο εξετάσαμε την διεπαφή της εφαρμογής συλλογής των κειμένων. Στη συνέχεια θα περιγράψουμε την λειτουργικότητα, η οποία βρίσκεται πίσω από την διεπαφή αυτή. Για την εφαρμογή αυτή αναπτύξαμε μια σειρά κλάσεων σε Java, οι οποίες έχουν ομαδοποιηθεί στο πακέτο crawler, το οποίο έχει ενσωματωθεί στο σύστημα. Έχουμε δημιουργήσει την σελίδα *CreateDatabase.jsp* και τέσσερις κλάσεις, τις *JoBoCrawler*, *DataAccess*, *Dircrawler* και *HtmlToTxt*, τις οποίες και θα περιγράψουμε.

4.2.5.1 JoBoCrawler

Η κλάση *JoBoCrawler* παρέχει τη λειτουργικότητα του Jobo (crawler). Έχουμε υλοποιήσει δύο μεθόδους, τις *start()* και *run()* (Εικόνα 14). Για να ξεκινήσει ο crawler την αναζήτηση και το κατέβασμα των αρχείων καλούμε τη συνάρτηση *start()*, με παραμέτρους το αρχικό URL, τη διαδρομή στον σκληρό όπου θα αποθηκευτούν τα αρχεία και το βάθος αναζήτησης. Η συνάρτηση αυτή αρχικοποιεί τις ρυθμίσεις του crawler, δημιουργεί ένα καινούριο στιγμιότυπο της κλάσης JoBoCrawler (έναν καινούριο crawler δηλαδή) και καλεί τη συνάρτηση *run()*, η οποία είναι αυτή που εκκινεί ουσιαστικά τον crawler.



Εικόνα 14: Το διάγραμμα της κλάσης JoBoCrawler

Η μέθοδος *run()* δημιουργεί αρχικά ένα καινούριο WebRobot, ένα καινούριο σύνολο κανόνων (DownloadRuleSet) και έναν καινούριο διαχειριστή αρχείων (HttpDocToFile). Το WebRobot είναι η κλάση του JoBo που πραγματοποιεί την αναζήτηση στο δίκτυο ξεκινώντας από ένα συγκεκριμένο URL και προχωράει σύμφωνα με κάποιες ρυθμίσεις που του δίνουμε στη συνέχεια. Αυτές είναι το αρχικό URL, το βάθος αναζήτησης, η επιλογή όλων των ιστοσελίδων που βρίσκονται στον ίδιο διακομιστή, ακόμα κι αν το URL τους ξεκινάει από άλλο πρόθεμα (π.χ. http://dmst.aueb.gr) και η επιλογή να μην ξανακατεβάζουμε σελίδες που έχουμε ήδη κατεβάσει.

Το σύνολο κανόνων που θέτουμε στο WebRobot περιλαμβάνει κάποιες οδηγίες σχετικά με τα είδη των αρχείων που θέλουμε να κατέβουν. Για την εφαρμογή μας έχουμε απορρίψει τα αρχεία ήχου, βίντεο, εικόνων και εφαρμογών, καθώς χρειαζόμαστε μόνο τα αρχεία κειμένου.

Τέλος ο διαχειριστής αρχείων είναι μία κλάση του JoBo που μας επιτρέπει να ορίσουμε τον κατάλογο στον σκληρό μας δίσκο, όπου θα αποθηκεύονται τα αρχεία. Αν δεν ορίσουμε έναν τέτοιο διαχειριστή αρχείων το WebRobot απλά θα επισκέπτεται τις ιστοσελίδες χωρίς να τις κατεβάζει. Η τελευταία εντολή της μεθόδου *run()* που υλοποιήσαμε είναι η *robby.run()* (όπου *robby* είναι το στιγμιότυπο της κλάσης *WebRobot* που δημιουργήσαμε), η οποία καλεί τη μέθοδο *run()* της κλάσης *WebRobot* του JoBo, εκκινώντας πλέον τον crawler με όλες τις ρυθμίσεις που έχουμε ορίσει.

4.2.5.2 DataAccess

Για την εφαρμογή μας έχει δημιουργηθεί μια βάση δεδομένων σε MS Access, στην οποία φυλάγονται τα αρχικά URL των ιστοσελίδων που θα κατεβάσουμε καθώς και τα URL κάθε ξεχωριστού αρχείου το οποίο κατεβαίνει στο σκληρό (θα δούμε αργότερα που χρησιμεύει αυτό). Για την επεξεργασία της βάσης αυτής έχουμε δημιουργήσει την κλάση DataAccess την οποία περιγράφουμε στη συνέχεια.

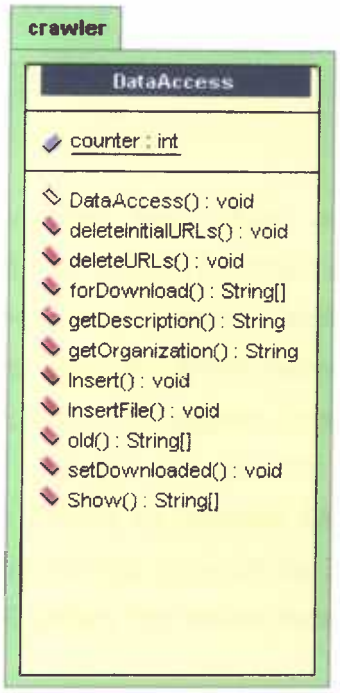
Αρχικά όμως πρέπει να δούμε το σχεδιασμό της βάσης. Η βάση αποτελείται από δύο πίνακες, τον InitialURLs, που περιέχει τα αρχικά URLs και τον URLs, που περιέχει κάθε αρχείο που κατεβάζουμε. Ο πίνακας **InitialURLs** αποτελείται από τα πεδία:

- *initialURLS_ID:* αύξων αριθμός των αρχικών URLs
- *initialURLS_BaseURL:* το αρχικό URL (π.χ. http://www.aueb.gr/)
- *initialURLS_Description:* μία περιγραφή για το πανεπιστήμιο η οποία θα χρησιμοποιηθεί και ως όνομα για τα αρχεία (π.χ. AUEB, οπότε τα αρχεία θα ονομαστούν AUEB1, AUEB2, κλπ.)
- *initialURLS_Organization:* το όνομα του ιδρύματος (π.χ. Οικονομικό Πανεπιστήμιο Αθηνών)
- *Downloaded:* ένα πεδίο η τιμή του οποίου μπορεί να είναι αληθής ή ψευδής και δείχνει αν το εκάστοτε URL έχει κατεβεί ή είναι καινούργιο.

Ο πίνακας **URLs** αποτελείται από τα πεδία:

- *ID:* αύξων αριθμός των αρχείων που κατεβαίνουν
- *ADDRESS:* το URL του εκάστοτε αρχείου (π.χ. http://www.aueb.gr/index.html)
- *LOCAL_PATH:* το αρχείο που κατέβηκε μετονομάζεται και εδώ αποθηκεύεται το καινούριο όνομα, ώστε να ξέρουμε σε ποιο URL και ποιο ίδρυμα αντιστοιχεί κάθε αρχείο που έχει κατέβει

- *DESCRIPTION:* η περιγραφή που βρίσκεται και στον προηγούμενο πίνακα (χρησιμεύει ώστε να χαρακτηρίζεται κάθε αρχείο με το ίδρυμα στο οποίο ανήκει)
- *ORGANIZATION:* το ίδρυμα από το οποίο κατέβηκε το αρχείο.



Εικόνα 15: Το διάγραμμα της κλάσης DataAccess

Η μέθοδος *Show()* μας επιστρέφει σε έναν πίνακα (array) τα στοιχεία του πίνακα (table) initialURLs της βάσης. Δηλαδή είναι ένας τρόπος για να πάρουμε σε πίνακα τα αρχικά URLs και να τα χρησιμοποιήσουμε. Η μέθοδος *forDownload()* μας επιστρέφει τα URLs του ίδιου πίνακα, τα οποία όμως έχουν χαρακτηριστεί ως καινούργια. Αντίστοιχα η μέθοδος *old()* μας επιστρέφει τα υπόλοιπα, δηλαδή αυτά που έχουν κατεβαστεί.

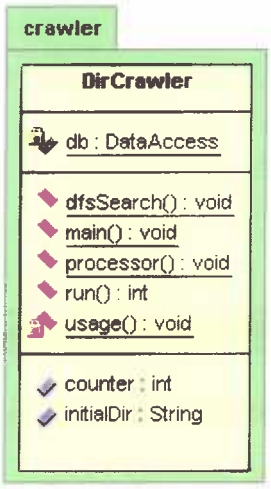
Η μέθοδος *Insert()* εισάγει μια νέα εγγραφή στα αρχικά URLs, ενώ η μέθοδος *InsertFile()* εισάγει ένα καινούριο αρχείο στον πίνακα URLs. Οι μέθοδοι *deleteInitialURLs()* και *deleteURLs()* διαγράφουν όλες τις εγγραφές του πίνακα initialURLs και URLs αντίστοιχα. Η μέθοδος

setDownloaded(String url) δέχεται σαν παράμετρο ένα αρχικό URL και το θέτει κατεβασμένο στον πίνακα initialURLs.

Οι μέθοδοι *getDescription(String url)* και *getOrganization(String url)* δέχονται σαν παράμετρο ένα URL και επιστρέφουν την περιγραφή και το ίδρυμα αντίστοιχα. Τέλος υπάρχει το attribute counter, το οποίο μετράει τα αρχεία, ώστε όταν κατεβάζουμε ένα καινούργιο ίδρυμα η αρίθμηση να συνεχίζει από εκεί που είχε μείνει.

4.2.5.3 Dircrawler

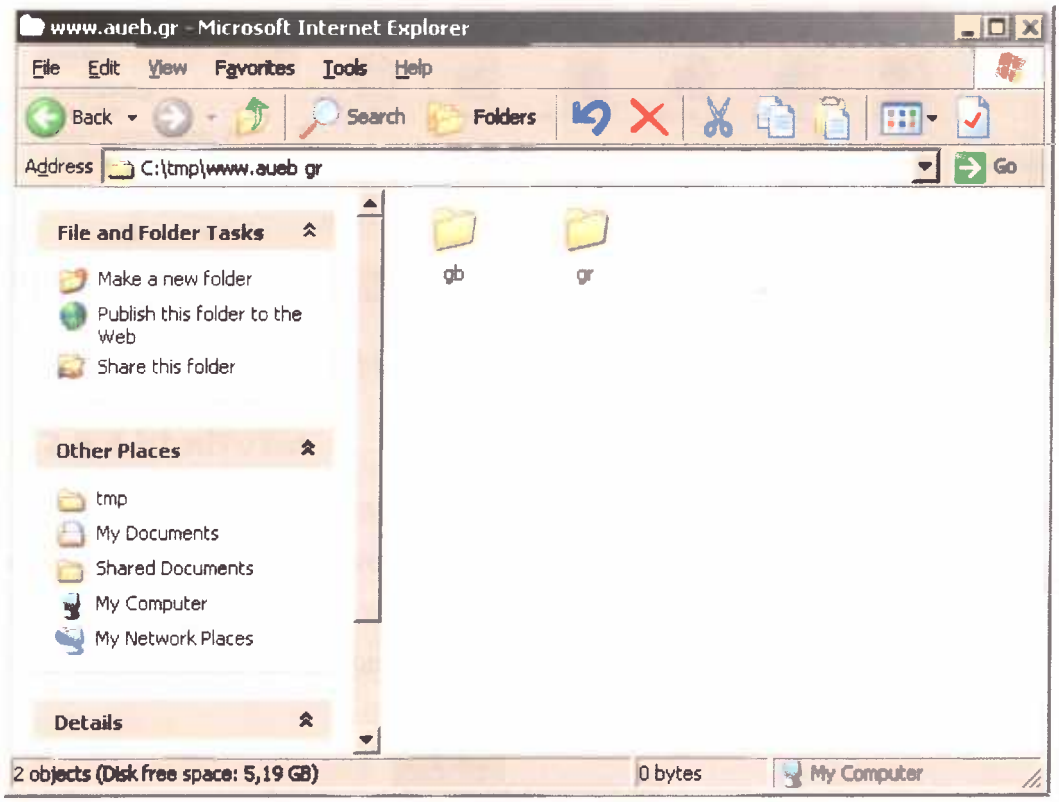
Τα προγράμματα crawlers που είδαμε κατεβάζουν ένα site στον σκληρό δίσκο με τη δομή που βρίσκεται στον server. Δηλαδή με όλους τους φακέλους και υποφακέλους στους οποίους βρίσκονται όλα τα απαραίτητα αρχεία του site, όπως έγγραφα, φωτογραφίες, αρχεία ήχου και video. Εμείς θέλουμε μόνο τα αρχεία κειμένου καθώς αυτά θα δημιουργήσουν τη βάση δεδομένων που θα χρησιμοποιηθεί από το σύστημα ερωταποκρίσεων. Επίσης τα θέλουμε όλα σε έναν φάκελο και όχι στην δενδρική δομή που βρίσκονται στον server. Αυτή ακριβώς είναι και η λειτουργία της κλάσης DirCrawler, την οποία αναπτύξαμε (Εικόνα 16).



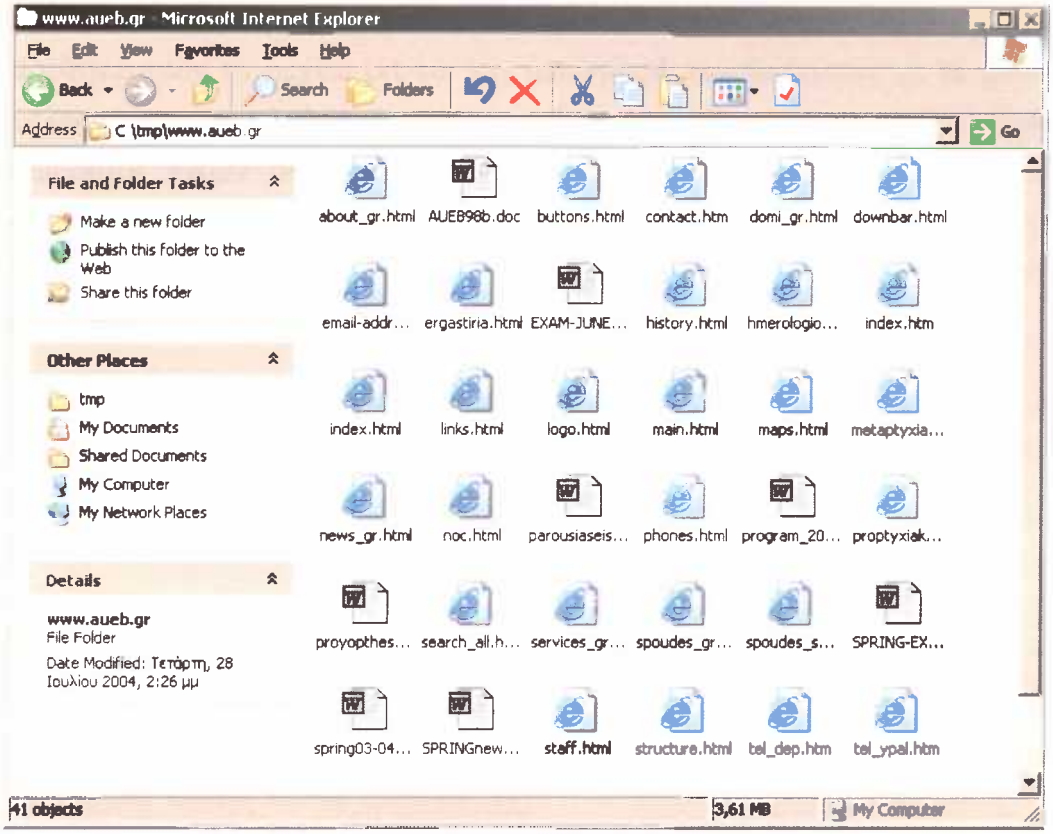
Εικόνα 16: Μέθοδοι της κλάσης DirCrawler

Αρχικά ξεκινάμε καλώντας τη συνάρτηση *run()*. Το attribute initialDir παίρνει σαν τιμή την συμβολοσειρά του αρχικού directory. Η εφαρμογή ξεκινάει από εκεί την αναζήτησή της και διατρέχει έναν προς

έναν τους φακέλους αναδρομικά. Αυτό πραγματοποιείται μέσω της συνάρτησης *dfsSearch()*. Για κάθε αρχείο που συναντά η συνάρτηση αυτή καλεί την συνάρτηση *processor()*, η οποία διαγράφει το αρχείο αν δεν είναι τύπου txt, html, htm, asp, php και jsp. Αν είναι κάποιου τέτοιου τύπου, το αντιγράφει στο αρχικό directory, εκτός αν βρισκόμαστε ήδη σε αυτό. Στη συνέχεια η *dfsSearch()* διαγράφει το directory το οποίο μόλις επεξεργαστήκαμε. Τελικά το αρχικό directory περιλαμβάνει μόνο τα χρήσιμα αρχεία τα οποία έχουν μεταφερθεί σε αυτό από τους υποκαταλόγους στους οποίους βρίσκονταν. Αυτό φαίνεται στο παράδειγμα των εικόνων (Εικόνα 17 και Εικόνα 18)που ακολουθούν:



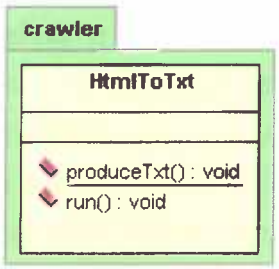
Εικόνα 17: Το αρχικό directory πριν την εκτέλεση της εφαρμογής



Εικόνα 18: Το αρχικό directory μετά την εκτέλεση της εφαρμογής

4.2.5.4 HtmlToTxt

Η κλάση αυτή υπήρχε ήδη στο σύστημα και την τροποποιήσαμε ελαφρά ώστε να ενσωματωθεί στις καινούργιες κλάσεις που δημιουργήσαμε. Η λειτουργία της είναι αυτή που φαίνεται και από το όνομά της, δηλαδή μετατροπή των αρχείων html σε κείμενα (txt) (Εικόνα 19).



Εικόνα 19: Το διάγραμμα της κλάσης HtmlToTxt

Η μέθοδος *run(String directory, String txtPath, String description)* παίρνει σαν παραμέτρους τον φάκελο στον σκληρό δίσκο, στον οποίο

βρίσκονται τα αρχεία html, τον φάκελο στον οποίο θέλουμε να αποθηκευτούν τα αρχεία txt και την περιγραφή του ιδρύματος με το οποίο θέλουμε να ασχοληθούμε. Να σημειώσουμε εδώ πως τα αρχεία κάθε ιδρύματος έχουν αποθηκευτεί σε διαφορετικό φάκελο, που ονομάζεται σύμφωνα με την περιγραφή που έχουμε δώσει (description). Έτσι με την περιγραφή αυτή ψάχνουμε σε συγκεκριμένο φάκελο ώστε να μετατρέψουμε τα αρχεία.

Η μέθοδος *run(...)* παίρνει από την βάση δεδομένων (από τον πίνακα URLs) τα αρχεία που ανήκουν στο ίδρυμα που δίνεται από την παράμετρο description και καλεί για το καθένα την μέθοδο *produceTxt()*, η οποία παράγει και το τελικό κείμενο.

4.2.5.5 CreateDatabase

Όλες οι κλάσεις που περιγράψαμε μέχρι τώρα χρησιμοποιούνται από την σελίδα CreateDatabase.jsp η οποία παρέχει τη λειτουργικότητα της εφαρμογής καθώς και τη διεπαφή της.

Αρχικά εμφανίζουμε σε μια περιοχή κειμένου τα κατεβασμένα και τα νέα URLs, από πληροφορίες που παίρνουμε από τη βάση δεδομένων, χρησιμοποιώντας την κλάση DataAccess.

Επίσης, καλώντας τη μέθοδο Insert της ίδιας κλάσης εισάγουμε ένα καινούριο URL στα αρχικά. Ο χρήστης πρέπει να έχει εισάγει πληροφορίες στα πεδία «Αρχείο» (description) και «Ίδρυμα» εκτός από το URL, αλλιώς η εισαγωγή δεν θα γίνει.

Η τρίτη λειτουργία είναι η συλλογή των κειμένων. Εδώ αρχικά διαγράφεται ο πίνακας URLs, της βάσης ώστε να ξεκινήσει μια καινούρια διαδικασία αναζήτησης για τα URLs που έχουν επιλεχθεί (νέα ή όλα). Στη συνέχεια παίρνουμε τα URLs ένα ένα και α) καλούμε τον crawler με όρισμα το URL αυτό, β) καλούμε τον DirCrawler ώστε να μετατραπεί ο φάκελος στην μορφή που τον θέλουμε και γ) καλούμε το HtmlToTxt ώστε να γίνει και η μετατροπή των αρχείων σε κείμενα.

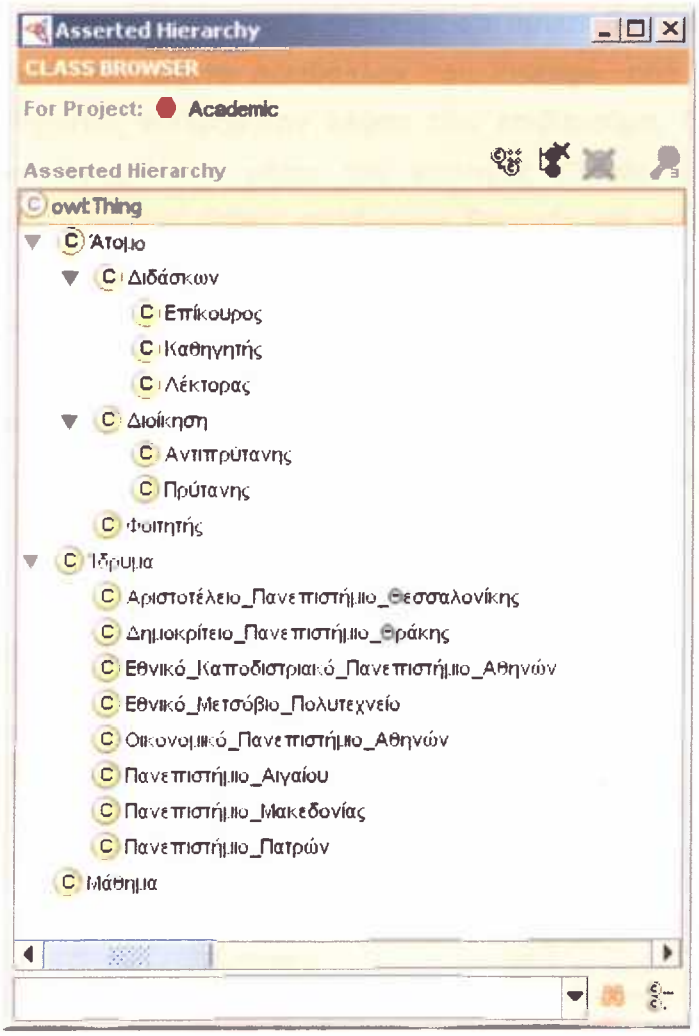
4.3 Η χρήση της οντολογίας με το σύστημα ανάκτησης

Η οντολογία που δημιουργήσαμε για να χρησιμοποιήσουμε με το σύστημα ανάκτησης είναι μια απλή και μικρή οντολογία, αλλά μπορεί να εύκολα να επεκταθεί στο μέλλον ώστε να υποστηρίζονται και άλλες λειτουργίες. Στην παράγραφο αυτή θα περιγράψουμε τον τρόπο δημιουργίας της οντολογίας με το Protégé (και το OWL Plug-in) και στη συνέχεια τον τρόπο ενσωμάτωσής της στο σύστημα με τη βοήθεια του Jena API.

4.3.1 Δημιουργία της οντολογίας

Η οντολογία που θέλαμε να δημιουργήσουμε θα περιγράφει τον ακαδημαϊκό χώρο. Αρχικά ξεκινήσαμε με την εισαγωγή σε αυτήν τα Ιδρύματα τα οποία θα υποστηρίζονται από την εφαρμογή, δηλαδή τα Πανεπιστήμια, Πολυτεχνεία κλπ. που οι ιστοσελίδες τους έχουν εισαχθεί στην βάση δεδομένων, οπότε και μπορεί ο χρήστης να κάνει ερωτήματα που τα αφορούν. Επειδή όμως συνήθως αναφερόμαστε στα Ιδρύματα αυτά με διαφορετικά ονόματα, είναι απαραίτητο να υπάρχει μια βάση με τα συνώνυμα αυτά, ώστε όταν κάνουμε μια ερώτηση να πάρουμε το σωστό αποτέλεσμα. Για παράδειγμα, όταν κάνουμε μια ερώτηση σχετικά με το Οικονομικό Πανεπιστήμιο, η απάντηση μπορεί να βρίσκεται σε ένα κείμενο όπου το Ίδρυμα αυτό αναφέρεται με την ονομασία ΑΣΟΕΕ ή ΟΠΑ. Αν δεν εμπλουτίσουμε την αναζήτησή μας και με τα συνώνυμα αυτά, το σωστό κείμενο δεν πρόκειται να μας επιστραφεί.

Η οντολογία που δημιουργήσαμε φαίνεται στην Εικόνα 20. Στο ανώτερο επίπεδο (κάτω από το owl:Thing φυσικά, που είναι το ανώτερο επίπεδο όλων) βρίσκεται η κλάση *Ίδρυμα*, που περιλαμβάνει τα Πανεπιστημιακά Ιδρύματα. Έχουμε συμπεριλάβει οκτώ ιδρύματα αλλά πολύ εύκολα μπορούμε να προσθέσουμε όσα επιθυμούμε, επιλέγοντας δημιουργία υποκλάσης (*Create Subclass*) στο Protégé, έχοντας επιλεγμένη την κλάση Organization.

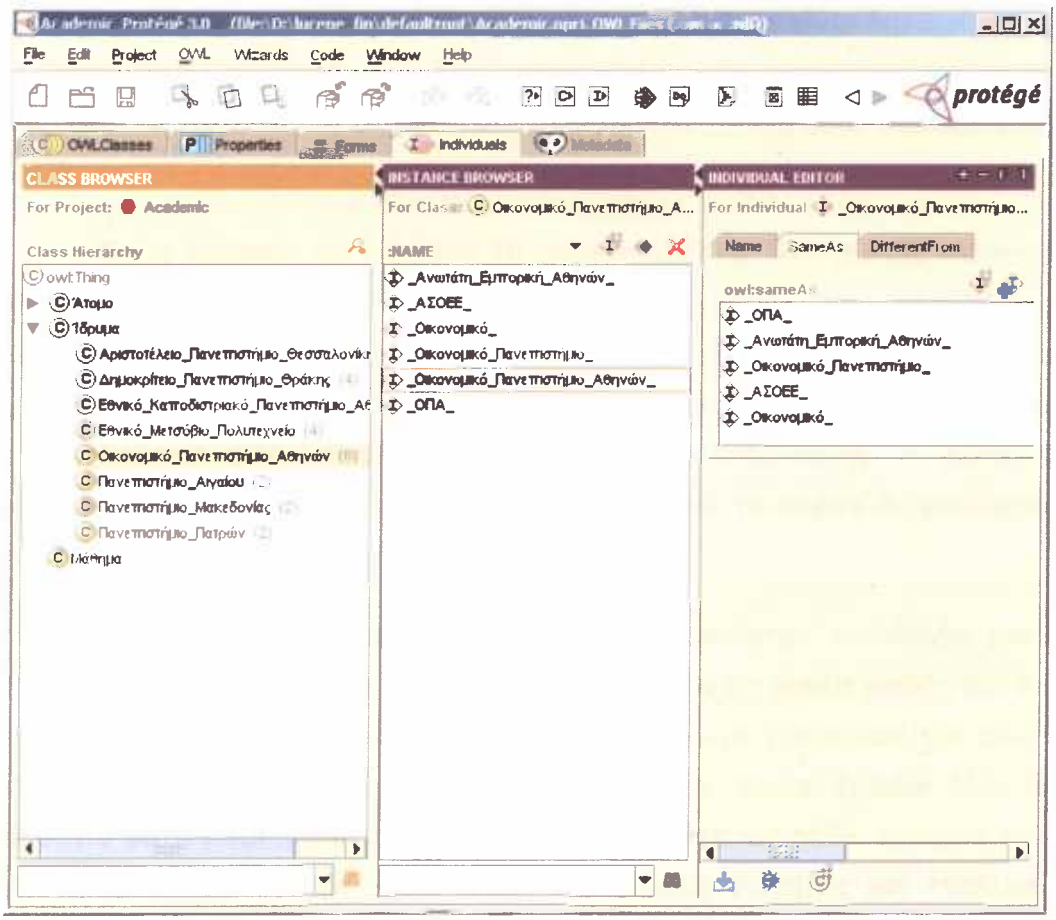


Εικόνα 20: Οι κλάσεις της οντολογίας μας

Έτσι τα συνώνυμα, στα οποία αναφερθήκαμε, ενσωματώνονται ως διαφορετικές οντότητες της κλάσης του κάθε Πανεπιστημίου[1]. Για παράδειγμα η κλάση «Οικονομικό Πανεπιστήμιο Αθηνών» έχει έξι στιγμιότυπα τα οποία έχουν δηλωθεί ως ίδια, με την δήλωση της OWL,

[1] Να σημειώσουμε εδώ μια παραδοχή που έχουμε κάνει στο σχεδιασμό της οντολογίας. Επειδή τα συνώνυμα ήταν πολλά σε αριθμό έχουμε οργανώσει κάθε ίδρυμα σαν μία κλάση τα στιγμιότυπα της οποίας είναι τα συνώνυμα του αντίστοιχου ιδρύματος. Αυτό εννοιολογικά είναι λάθος, καθώς θα έπρεπε τα ιδρύματα να είναι στιγμιότυπα της κλάσης Ίδρυμα και όχι να υπάρχει ξεχωριστή υποκλάση για κάθε ίδρυμα. Τότε όμως η λίστα των συνωνύμων θα ήταν πολύ μεγάλη και πολύ δύσκολο να τη διαχειριστούμε. Σε κάθε περίπτωση είναι εύκολη η μετακίνηση στον «σωστό» τρόπο σχεδιασμού. Το μόνο που πρέπει να γίνει είναι να διαγραφούν όλες οι υποκλάσεις της κλάσης Ίδρυμα και να αντιστοιχιστούν τα συνώνυμα σε αυτήν.

owl:sameAs. Στα συνώνυμα αυτά μπορούν να προστεθούν κι άλλα πολύ εύκολα χρησιμοποιώντας το περιβάλλον του Protégé, από την καρτέλα *Individuals*. Έχοντας επιλέξει την κλάση που επιθυμούμε, δημιουργούμε ένα καινούριο στιγμιότυπο μέσω της επιλογής *Create Instance*. Στη συνέχεια στον Individual Editor επιλέγουμε *SameAs* και προσθέτουμε το κεντρικό στιγμιότυπο, δηλαδή το στιγμιότυπο στο οποίο έχουν αντιστοιχιστεί όλα τα συνώνυμα και το οποίο έχει το ίδιο όνομα με την κλάση. Έτσι δεν είναι απαραίτητο να ενημερώνουμε όλα τα στιγμιότυπα κάθε φορά που προσθέτουμε ένα καινούριο. Έχοντας τα συνώνυμα μόνο σε ένα στιγμιότυπο τα αποτελέσματα είναι πάλι σωστά αφού κατά την αναζήτηση στην οντολογία την διατρέχουμε διεξοδικά και προσθέτουμε όλα τα συνώνυμα του οργανισμού τον οποίο αναζητούμε.



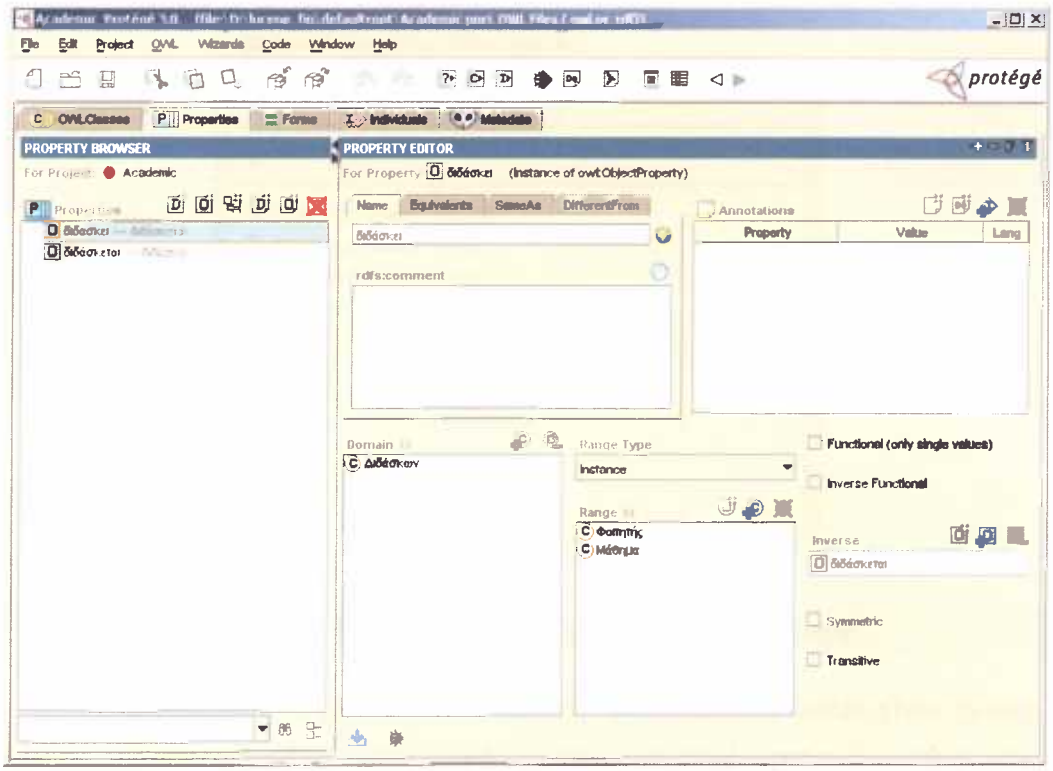
Εικόνα 21: Το τμήμα που φαίνονται οι οντότητες των κλάσεων. Στο δεξί μέρος βλέπουμε τις πληροφορίες για κάθε οντότητα (στιγμιότυπο) και έχουμε τοποθετήσει εκεί τα συνώνυμά της

Το παράδειγμα του Οικονομικού Πανεπιστημίου φαίνεται στην Εικόνα 21.

Έχουμε τοποθετήσει το σύμβολο '_' στις οντότητες και τις κλάσεις για δύο λόγους. Πρώτα επειδή η OWL και γενικά οι οντολογίες δεν επιτρέπουν κενά στα ονόματα. Ο λόγος για αυτό είναι ότι κάθε οντότητα και κλάση σε μία οντολογία δηλώνεται με ένα URI της μορφής http://www.owl-ontologies.com/unnamed.owl# και στη συνέχεια το όνομα της οντότητας. Σε ένα URI όπως είναι φυσικό δεν είναι δυνατόν να υπάρχουν κενά για αυτό και ο περιορισμός αυτός. Στα στιγμιότυπα όμως έχουμε τοποθετήσει πρόσθετα '_' στην αρχή και το τέλος της καθεμίας. Αυτό έγινε για λόγους συμβατότητας με το σύστημα ερωταποκρίσεων καθώς είναι υλοποιημένο να διαβάζει τις οντότητες με κενά στην αρχή και στο τέλος. Για παράδειγμα το Οικονομικό Πανεπιστήμιο είναι δηλωμένο ως « Οικονομικό Πανεπιστήμιο ». Στην εφαρμογή μας πριν επεξεργαστούμε τα αντικείμενα της οντολογίας μετατρέπουμε όλα τα '_' της συμβολοσειράς σε κενά.

Όπως φαίνεται στην Εικόνα 20 υπάρχουν άλλες δύο γενικές κλάσεις. Η πρώτη είναι η κλάση *Άτομο*, η οποία χωρίζεται στις υποκλάσεις Διδάσκων, *Διοίκηση* και *Φοιτητής*. Η πρώτη από τις υποκλάσεις αυτές χωρίζεται περαιτέρω στις υποκλάσεις *Επίκουρος*, *Καθηγητής* και *Λέκτορας*, και η δεύτερη στις υποκλάσεις *Πρύτανης* και *Αντιπρύτανης*. Η δεύτερη γενική κλάση είναι η κλάση μάθημα, η οποία προς το παρόν δε χωρίζεται περαιτέρω.

Επίσης έχουμε προσθέσει κάποιες ιδιότητες στην οντολογία μας. Πηγαίνοντας στην καρτέλα *Properties* μπορούμε να τις δούμε καθώς και να προσθέσουμε άλλες (Εικόνα 22). Ουσιαστικά έχουμε προσθέσει μία μόνο ιδιότητα αλλά και την αντίστροφή της (inverse), οπότε έχουμε δύο. Η ιδιότητα αυτή είναι η ιδιότητα *διδάσκει*, η οποία έχει ως πεδίο ορισμού την κλάση *Διδάσκων* και ως πεδίο τιμών τις κλάσεις *Φοιτητής* και *Μάθημα*. Δηλώνει δηλαδή την πρόταση «ο Καθηγητής Χ διδάσκει το μάθημα Υ», κάτι που θα χρησιμοποιήσουμε αργότερα για να απαντήσουμε σε ερωτήματα του τύπου «Ποιος διδάσκει το Υ μάθημα;».

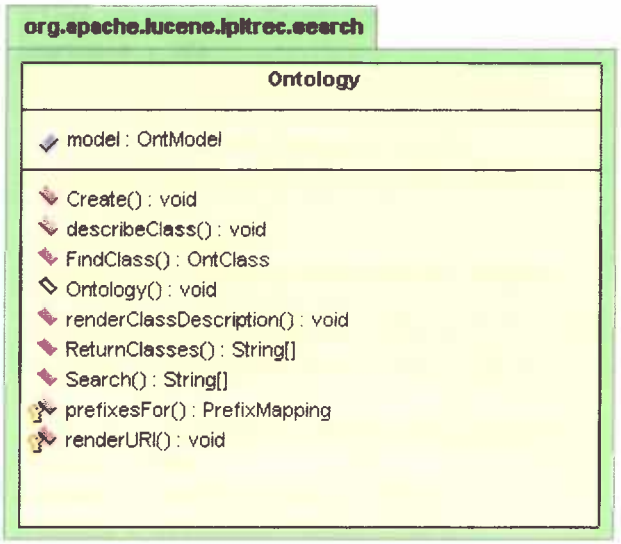


Εικόνα 22: Η καρτέλα με τις ιδιότητες του Protégé

4.3.2 Η εφαρμογή

Για να ενσωματώσουμε την οντολογία στο σύστημα ερωταποκρίσεων έπρεπε να κάνουμε κάποιες απαραίτητες αλλαγές στον κώδικά του. Τις αλλαγές αυτές καθώς και όλη τη διαδικασία χρήσης της οντολογίας μέσα από το πρόγραμμα παρουσιάζουμε στη συνέχεια.

Αρχικά έπρεπε να δημιουργήσουμε μία κλάση, η οποία να περιέχει τις λειτουργίες τις οντολογίας. Η κλάση αυτή είναι η κλάση *Ontology* και το διάγραμμά της φαίνεται στην Εικόνα 23.



Εικόνα 23: UML διάγραμμα της κλάσης Ontology

Η κλάση αυτή διαθέτει τη μεταβλητή *OntModel*, η οποία είναι τύπου model. Δηλαδή είναι ένα οντολογικό μοντέλο, το οποίο χρησιμοποιείται σαν ο χώρος στη μνήμη στον οποίο θα φυλάξουμε την οντολογία, αφού τη διαβάσουμε από το αρχείο που είναι γραμμένη. Αφού γίνει αυτό μπορούμε να τη διαχειριστούμε χρησιμοποιώντας τις εντολές που μας παρέχει το Jena API.

Η μέθοδος της κλάσης που χρησιμοποιούμε αρχικά είναι η μέθοδος *Create()*, με την οποία δημιουργούμε ένα οντολογικό μοντέλο και φορτώνουμε σε αυτό μια οντολογία από ένα αρχείο.

Οι μέθοδοι *describeClass*, *renderClassDescription* και *renderURI* υπάρχουν μόνο για λόγους ελέγχου. Διατρέχουν την κλάση τις οντολογίας που δέχονται σαν παράμετρο και εμφανίζουν στην βασική έξοδο το περιεχόμενό της. Αυτό γίνεται ουσιαστικά από την *renderClassDescription* η οποία καλεί την *renderURI*, για να εμφανίσει μόνο τα ονόματα των κλάσεων και των οντοτήτων και όχι ολόκληρο το URI τους. Παράδειγμα εκτέλεσης των κλάσεων αυτών με την οντολογία μας φαίνεται στην Εικόνα 24.

```
Class :Οικονομικό Πανεπιστήμιο Αθηνών
Instance : ΟΠΑ
Synonym : Οικονομικό Πανεπιστήμιο Αθηνών

Instance : Ανωτάτη Εμπορική Αθηνών
Synonym : Οικονομικό Πανεπιστήμιο Αθηνών

Instance : ΑΣΟΕΕ
Synonym : Οικονομικό Πανεπιστήμιο Αθηνών

Instance : Οικονομικό
Synonym : Οικονομικό Πανεπιστήμιο Αθηνών

Instance : Οικονομικό Πανεπιστήμιο Αθηνών
Synonym : ΟΠΑ
Synonym : Ανωτάτη Εμπορική Αθηνών
Synonym : Οικονομικό Πανεπιστήμιο
Synonym : ΑΣΟΕΕ
Synonym : Οικονομικό

Instance : Οικονομικό Πανεπιστήμιο
Synonym : Οικονομικό Πανεπιστήμιο Αθηνών


Class :Εθνικό Μετσόβιο Πολυτεχνείο
Instance : Εθνικό Μετσόβιο Πολυτεχνείο
Synonym : Μετσόβιο
Synonym : Μετσόβιο Πολυτεχνείο
Synonym : ΕΜΠ

Instance : ΕΜΠ
Synonym : Εθνικό Μετσόβιο Πολυτεχνείο
```

Εικόνα 24: Δείγμα εκτέλεσης της μεθόδου describeClass() με την οντολογία μας

Η μέθοδος *prefixesFor* δέχεται ένα αντικείμενο οντολογίας (κλάση, ιδιότητα, οντότητα) και κρατάει μόνο το όνομά της. Δηλαδή αφαιρεί τα περιττά URI. Η διαφορά της με την *renderURI* είναι ότι αυτή δεν εμφανίζει κάτι στην οθόνη αλλά κρατάει το χρήσιμο κομμάτι του αντικειμένου, ώστε να το χρησιμοποιήσουμε αργότερα. Πράγματι, στις μεθόδους *FindClass* και *Search* χρειαζόμαστε μόνο το όνομα της οντολογίας και το παίρνουμε μέσω της μεθόδου αυτής.

Η μέθοδος *FindClass* πραγματοποιεί μία αναζήτηση σε όλες τις κλάσεις τις οντολογίας, ψάχνοντας για εκείνη που είναι ίδια με ένα συγκεκριμένο κλειδί. Το κλειδί αυτό είναι η μεταβλητή *organization*, η

οποία περιέχει το ίδρυμα του οποίου τα συνώνυμα αναζητούμε. Έτσι αφού βρούμε την κλάση που ψάχνουμε στην οντολογία είμαστε έτοιμοι να βρούμε και να επιστρέψουμε τα συνώνυμα του ιδρύματος.

Αυτό γίνεται στη μέθοδο *Search* η οποία δέχεται σαν είσοδο το κλειδί με το οποίο γίνεται η σύγκριση και την κλάση που βρήκαμε πριν. Τώρα το κλειδί συγκρίνεται με όλα τα στιγμιότυπα, ώστε να βρούμε αυτό που είναι το βασικό (αυτό δηλαδή στο οποίο έχουν αντιστοιχιστεί όλα τα συνώνυμα). Όταν βρεθεί αυτό λαμβάνουμε όλα τα συνώνυμα χρησιμοποιώντας την εντολή του Jena API, *listSameAs*. Αυτά τοποθετούνται σε έναν πίνακα συμβολοσειρών και επιστρέφονται από τη μέθοδο.

Τα αποτελέσματα αυτά επιστρέφονται στη μέθοδο *FindSynonyms* που προσθέσαμε στην κλάση *SearchDocuments* του συστήματος ανάκτησης. Η μέθοδος αυτή παίρνει σαν παραμέτρους το ίδρυμα του οποίου τα συνώνυμα αναζητούμε και μία συμβολοσειρά, η οποία υποδηλώνει τη διαδρομή στον σκληρό δίσκο στην οποία βρίσκεται το αρχείο της οντολογίας. Αρχικά δημιουργεί μια καινούρια οντολογία χρησιμοποιώντας τη διαδρομή αυτή και τη μέθοδο *Create* της κλάσης *Ontology*. Στη συνέχεια αναζητά την κλάση της οντολογίας που περιέχει το ίδρυμα που αναζητούμε, με τη βοήθεια της μεθόδου *FindClass* και τέλος, αποθηκεύει σε έναν πίνακα συμβολοσειρών τα συνώνυμα του ιδρύματος, τα οποία επιστρέφει η μέθοδος *Search*.

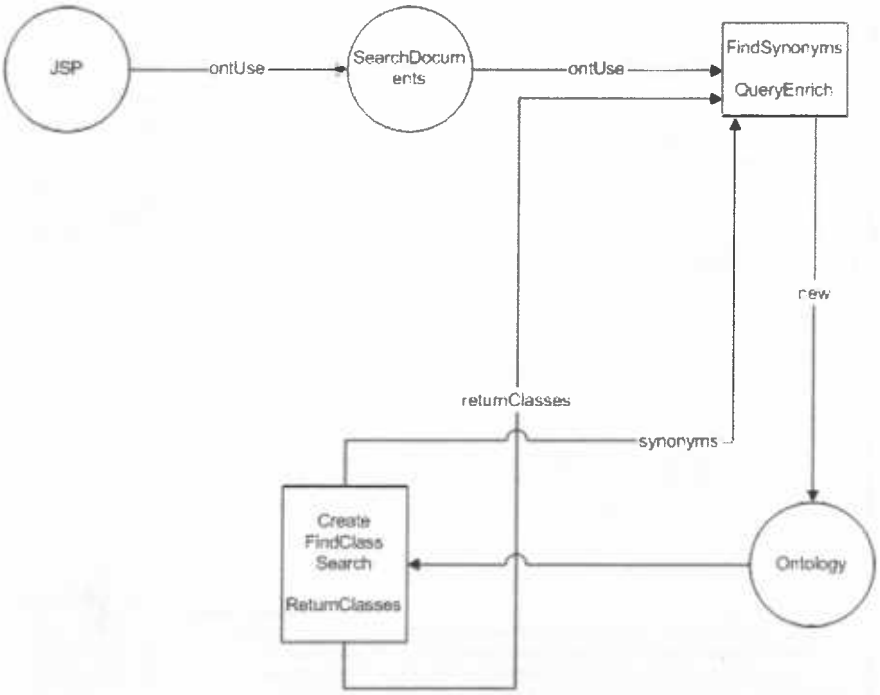
Τέλος, στην κλάση *Ontology* υπάρχει η μέθοδος *ReturnClasses(String property, OntModel m)*, η οποία δέχεται σαν παράμετρο μία ιδιότητα οντολογίας και ένα οντολογικό μοντέλο και επιστρέφει σε έναν πίνακα τις κλάσεις που ανήκουν στο πεδίο ορισμού τις ιδιότητας αυτής. Αυτό γίνεται μόνο στην περίπτωση που η ιδιότητα αυτή υπάρχει στο οντολογικό μοντέλο.

Η μέθοδος που περιγράψαμε μόλις, χρησιμοποιείται από την μέθοδο *QueryEnrich(String query, String ontPath)* που επίσης προσθέσαμε στην κλάση *SearchDocuments* του Lucene. Η μέθοδος αυτή δέχεται σαν παραμέτρους ένα ερώτημα και τη διαδρομή μιας οντολογίας στον δίσκο και

επιστρέφει το ίδιο ερώτημα εμπλουτισμένο ώστε η αναζήτηση να έχει καλύτερα αποτελέσματα. Αυτό που κάνει είναι να ψάχνει στο ερώτημα για μία λέξη που να είναι ίδια με κάποια από τις ιδιότητες της δοσμένης οντολογίας. Αν βρει μια τέτοια λέξη καλεί τη μέθοδο *ReturnClasses()* της κλάσης *Ontology* και προσθέτει στο ερώτημα τις κλάσεις που επιστρέφει η μέθοδος αυτή. Τέλος επιστρέφει το εμπλουτισμένο ερώτημα.

Η εφαρμογή του συστήματος ερωταποκρίσεων διαθέτει γραφική διεπαφή, γραμμένη σε JSP σελίδες. Η σελίδα JSP που πραγματοποιεί την αναζήτηση ήταν αυτή που έπρεπε να τροποποιήσουμε ώστε να καλείται η δική μας μέθοδος σε περίπτωση που ο χρήστης θέλει να χρησιμοποιήσει την οντολογία αντί για τον πίνακα συνωνύμων που ήταν ενσωματωμένος στην εφαρμογή. Σε αυτή τη σελίδα λοιπόν, προσθέτουμε μία παράμετρο *ontUse* τύπου Boolean, η οποία θα καθορίζει αν θα χρησιμοποιείται η οντολογία ή όχι. Μία ίδια παράμετρο έχουμε προσθέσει στην κλάση *SearchDocuments* του συστήματος ανάκτησης, η οποία καθορίζει ποια από τις δύο συναρτήσεις *FindSynonyms* θα κληθεί και ποιο ερώτημα θα χρησιμοποιηθεί (το απλό ή το εμπλουτισμένο). Η τιμή της παραμέτρου στην JSP σελίδα λαμβάνεται από ένα Checkbox το οποίο ενεργοποιεί ο χρήστης. Έτσι, όταν δημιουργείται ένα καινούριο στιγμιότυπο της κλάσης *SearchDocuments* η παράμετρος *ontUse* παίρνει τιμή αληθή ή ψευδή ανάλογα με την επιλογή του χρήστη οπότε και καλείται η αντίστοιχη μέθοδος.

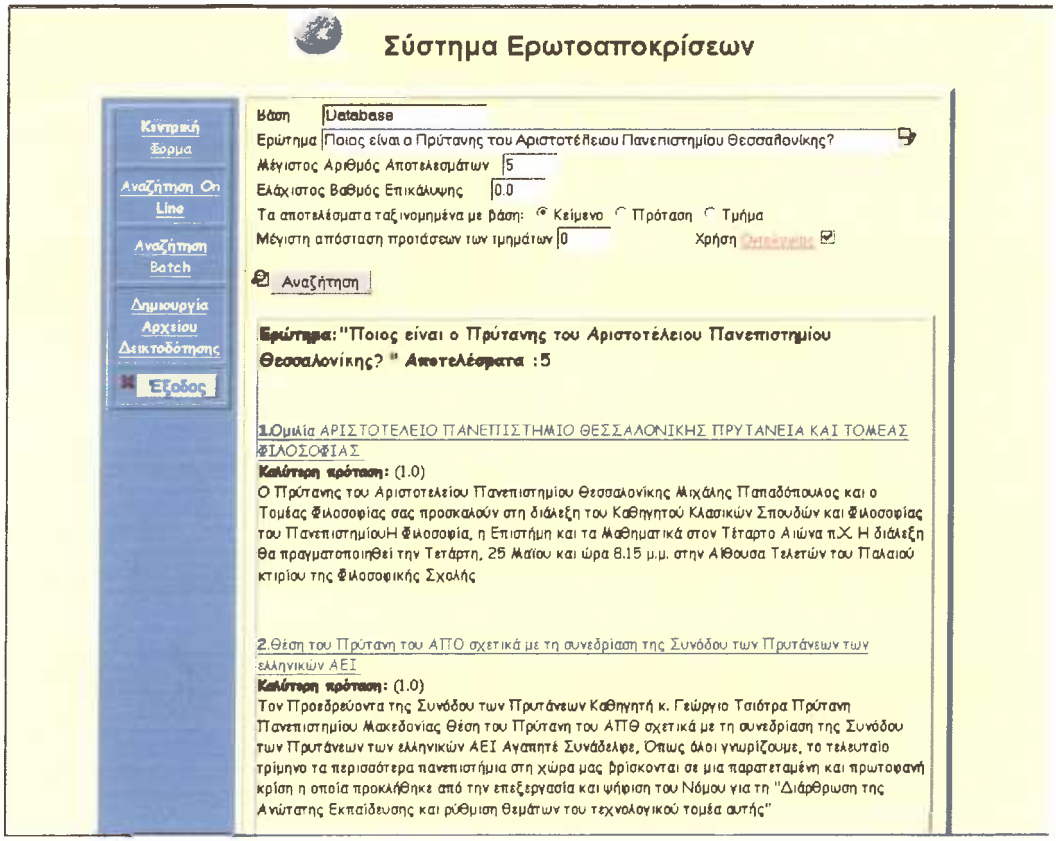
Στην Εικόνα 25 φαίνεται γραφικά η λειτουργία της εφαρμογής από την πλευρά του χρήστη που εκκινεί μια διαδικασία αναζήτησης, υποβάλλοντας ένα ερώτημα στο σύστημα.



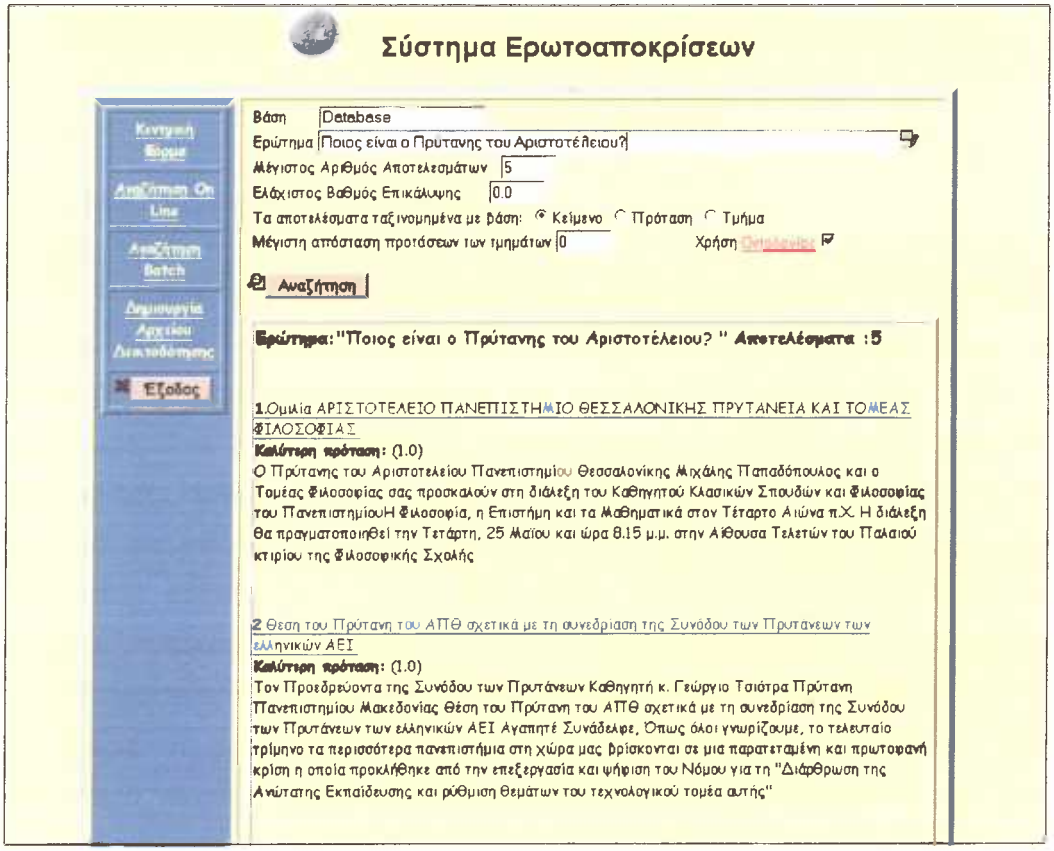
Εικόνα 25: Η λειτουργία της εφαρμογής σχηματικά

4.3.3 Παράδειγμα εκτέλεσης της εφαρμογής

Θα παρουσιάσουμε και ένα παράδειγμα υποβολής μιας επερώτησης στο σύστημα κατά την οποία γίνεται χρήση της οντολογίας. Η ερώτηση μας είναι η εξής: «Ποιος είναι ο Πρύτανης του Αριστοτέλειου Πανεπιστήμιου Θεσσαλονίκης?». Έχουμε ορίσει μέγιστο αριθμό αποτελεσμάτων 5 και πράγματι μας επιστρέφονται πέντε αποτελέσματα. Το σωστό όμως αποτέλεσμα βρίσκεται στην πρώτη θέση. Το ίδιο θα συνέβαινε αν είχαμε χρησιμοποιήσει στο ερώτημα κάποιο από τα συνώνυμα του ιδρύματος. Τα παραδείγματα αυτά φαίνονται στις παρακάτω εικόνες Εικόνα 26 και Εικόνα 27.



Εικόνα 26: Παράδειγμα χρήσης του συστήματος



Εικόνα 27: Το ίδιο παράδειγμα χρησιμοποιώντας ένα συνώνυμο αυτή τη φορά. Τα αποτελέσματα δεν επηρεάζονται.

Κεφάλαιο 5ο: Μελλοντικές επεκτάσεις και συμπεράσματα

5.1 Μελλοντικές επεκτάσεις

Η εφαρμογή που αναπτύξαμε έχει περιθώρια βελτίωσης σε πολλούς τομείς. Αρχικά μπορούμε να βελτιώσουμε το σύστημα που δημιουργεί τη βάση δεδομένων χρησιμοποιώντας τον crawler που ανακαλύψαμε (JoBo). Αυτό μπορεί να γίνει επιλέγοντας ένα καλύτερο HtmlToTxt πρόγραμμα και επιλύοντας κάποια από τα προβλήματα του JoBo, το οποίο δείχνει να κολλάει σε κάποιες σελίδες. Είναι φανερό από τις πρώτες δοκιμές του εργαλείου αυτού ότι η βάση δεδομένων που θα δημιουργεί θα είναι υπερπλήρης, καθώς ουσιαστικά κατεβάζει στον σκληρό δίσκο όλα τα αρχεία ενός διαδικτυακού τόπου, χωρίς να κατεβάζει το ίδιο αρχείο πολλές φορές και χωρίς να «ξεφεύγει» σε εξωτερικές ιστοσελίδες. Έχει μάλιστα την ικανότητα να ανακαλύπτει και να κατεβάζει σελίδες που βρίσκονται στο ίδιο domain ενός πανεπιστημίου, παρόλο που η διεύθυνσή τους δεν ξεκινάει με το ίδιο αναγνωριστικό. Για παράδειγμα η σελίδα του Οικονομικού Πανεπιστημίου έχει το URL www.aueb.gr, αλλά η σελίδα του τμήματος Πληροφορικής του Πανεπιστημίου αυτού βρίσκεται στην διεύθυνση www.cs.aueb.gr και όχι www.aueb.gr/... (δηλαδή σε έναν υποκατάλογο της αρχικής διεύθυνσης), όπως θα ήταν πιο εύκολο για έναν crawler. Το πρόγραμμα αυτό διαθέτει τη δική του διεπαφή οπότε μπορεί να χρησιμοποιηθεί σαν ξεχωριστό πρόγραμμα, αλλά είναι επίσης ανοικτού κώδικα, πράγμα που σημαίνει πως είναι δυνατή η ενσωμάτωσή του στο σύστημα και η κλήση του μέσα από μεθόδους που θα δημιουργήσουμε εμείς.

Όσον αφορά τη βελτίωση του συστήματος ανάκτησης οι βελτιώσεις που επιδέχεται έχουν να κάνουν με την εκτενέστερη χρήση της οντολογίας. Η πρώτη λύση έχει να κάνει με την επέκταση αρχικά της οντολογίας. Θα πρέπει να προστεθούν κι άλλες κλάσεις και ιδιότητες, όπως η ιδιότητα «παρακολουθεί», η οποία θα συνδέει τον «Φοιτητή» και το «Μάθημα». Όταν τώρα υποβάλλουμε ερωτήματα στο σύστημα η διαδικασία επιστροφής αποτελεσμάτων μπορεί να βελτιωθεί φιλτράροντας τα

καλύτερα αποτελέσματα σύμφωνα με τις κλάσεις και τις ιδιότητες που προσθέσαμε στην οντολογία.

Ένας άλλος τρόπος χρήσης της οντολογίας αυτής, είναι ο εμπλουτισμός του ερωτήματος με συνώνυμα των λέξεων που βρίσκονται σε αυτό, είτε με λέξεις που ιεραρχικά βρίσκονται πιο πάνω ή πιο κάτω (γενίκευση / ειδίκευση). Έτσι τα αποτελέσματα θα είναι περισσότερα και πιθανότατα να μας επιστραφούν και τα κείμενα εκείνα που περιέχουν την απάντηση στο ερώτημά μας.

Η δεύτερη λύση, η οποία έχει και διαφορετική φιλοσοφία από την τωρινή υλοποίηση του συστήματος ανάκτησης, έχει να κάνει με την χρήση της οντολογίας σαν βάσης δεδομένων. Θα πρέπει να δημιουργήσουμε μια πολύ μεγάλη οντολογία που να καλύπτει επαρκώς τον ακαδημαϊκό χώρο και να την «γεμίσουμε» με πληροφορία, την οποία θα συλλέξουμε από τα κείμενα που θα έχουμε συγκεντρώσει από τους διαδικτυακούς τόπους των Πανεπιστημίων. Έτσι η οντολογία θα περιέχει όλη την απαραίτητη πληροφορία (καθηγητές, μαθήματα κλπ.) και η διαδικασία αναζήτησης θα είναι πολύ γρήγορη αφού η απάντηση θα αναζητείται στην οντολογία και όχι σε ένα σύνολο κειμένων. Εξάλλου το αποτέλεσμα θα είναι ποιοτικότερο αφού θα εμφανίζεται αμέσως με τη μορφή απάντησης και όχι κειμένου. Αυτό σημαίνει όμως ότι πρέπει να προβλεφθούν όλες οι πιθανές ερωτήσεις και να δημιουργηθούν οι κατάλληλες κλάσεις και ιδιότητες στην οντολογία για αυτές. Αν ωστόσο κάποια ερώτηση δεν έχει απάντηση μέσα στην οντολογία, το σύστημα μπορεί να εκτελέσει την αναζήτηση και με τον προηγούμενο τρόπο.

5.2 Συμπεράσματα

Στην εργασία αυτή ασχοληθήκαμε με τους τρόπους βελτίωσης ενός υπάρχοντος συστήματος ανάκτησης πληροφορίας. Μελετήσαμε τις οντολογίες, τα εργαλεία συγγραφής και διαχείρισής τους, ώστε να δημιουργήσουμε και να ενσωματώσουμε στο σύστημα μια οντολογία που να έχει ως θέμα τον ακαδημαϊκό χώρο. Τελικά καταφέραμε να

δημιουργήσουμε μια απλή οντολογία και να την χρησιμοποιήσουμε επιτυχώς με το σύστημα. Η οντολογία αυτή περιέχει τα συνώνυμα των Ιδρυμάτων που μπορεί να αποτελέσουν στόχο του ερωτήματός μας και μέσα από την εφαρμογή την διαχειριζόμαστε.

Όσον αφορά τη βελτίωση που πραγματοποιήσαμε εμπλουτίζοντας το ερώτημα παρατηρήσαμε ότι έχει καλά αποτελέσματα, όταν η υπάρχει η σελίδα που μας δίνει την απάντηση. Όταν δεν υπάρχει εμφανίζονται πολλές άσχετες σελίδες και για αυτό ευθύνονται οι λέξεις που έχουν μπει στο ερώτημα. Έτσι πρέπει να δίνουμε στις λέξεις αυτές μικρότερη βαρύτητα.

Έγινε επίσης προσπάθεια βελτίωσης του συστήματος στο αρχικό κομμάτι του, δηλαδή στη δημιουργία της βάσης δεδομένων. Αναζητήσαμε στο διαδίκτυο ένα καλύτερο πρόγραμμα crawler και από τα πολλά που ανακαλύψαμε, καταλήξαμε στο πιο κατάλληλο για να χρησιμοποιηθεί με την εφαρμογή μας.

Παράρτημα Α: Κώδικας της εφαρμογής

Α.1 Κώδικας της κλάσης *JoBoCrawler*

```
public class JoBoCrawler {

  private void run(String url, String path, int depth) throws Exception {
    WebRobot robby = new WebRobot();
    robby.setStartURL(new URL(url));
    robby.setMaxDepth(depth);
    robby.setSleepTime(0);

    HttpDocToFile doc = new HttpDocToFile(path);
    robby.setDocManager(doc);
    robby.run();
  }

  public static void start(String url, String path, int depth) throws
Exception {
    BasicConfigurator.configure();
    JoBoCrawler crawler = new JoBoCrawler();
    System.out.println("Crawler: Depth = " + depth);
    crawler.run(url, path, depth);
  }
}
```

Α.2 Κώδικας της κλάσης *DataAccess*

```
public class DataAccess {

  public static int counter = 0;

  public DataAccess() {
  }

  public String[] Show() {

    String driver, dburl, user, passwd;
```

```
driver = "sun.jdbc.odbc.JdbcOdbcDriver";
dburl = "jdbc:odbc:crawler_database";
user = "";
passwd = "";

String[] urls = new String[100];

try {
  Class.forName(driver);
}
catch (java.lang.ClassNotFoundException e) {
  System.err.print("ClassNotFoundException: ");
  System.err.println(e.getMessage());
}
Connection conn = null;
Statement s;

try {
  conn = DriverManager.getConnection(dburl, "", "");
  s = conn.createStatement();
  System.out.println("Pragmatopoihthike h syndesi");

  s.execute("SELECT * FROM initialURLS");

  ResultSet rs = s.getResultSet();
  String baseURL = new String(""); ;

  int urlCounter = 0;
  while (rs.next()) {
      Object value = rs.getObject(2);
      baseURL = value.toString();
      urls[urlCounter] = baseURL;
      urlCounter++;
  }
  s.close();
  conn.close();
}
catch (java.sql.SQLException e) {
```

```
    System.err.println(e);
  }
  return urls;
}

public void Insert(String url, String description, String organization) {

  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.out.println(e);
  }

  Connection conn = null;
  Statement s;

  try {
    conn = DriverManager.getConnection(dburl, user, passwd);
    s = conn.createStatement();
    s.execute("SELECT * FROM initialURLS");
    ResultSet rs = s.getResultSet();
    while (rs.next()) {
       counter++;
    }
    counter++;
    s.executeUpdate("INSERT INTO initialURLS VALUES (
       " + counter + ", '" + url + "', '" + description + "', '" +
       organization + "', False ) ");

    s.close();
    conn.close();
```

```
 }
 catch (java.sql.SQLException e) {
  System.out.print(e);
 }

}

public void InsertFile(int id, String address, String path,
                       String description, String organization) {

 String driver, dburl, user, passwd;
 driver = "sun.jdbc.odbc.JdbcOdbcDriver";
 dburl = "jdbc:odbc:crawler_database";
 user = "";
 passwd = "";

 try {
  Class.forName(driver);
 }
 catch (java.lang.ClassNotFoundException e) {
  System.out.println(e);
 }

 Connection conn = null;
 Statement s;

 address = address.replace('\\', '/');

 try {
  conn = DriverManager.getConnection(dburl, user, passwd);
  s = conn.createStatement();
  s.executeUpdate("INSERT INTO URLS VALUES (
      " + id + ", '" + address + "', '"+ path + "', '" + description +
      "', '" + organization + "' ) ");

  s.close();
  conn.close();
 }
```

```
 catch (java.sql.SQLException e) {
  System.out.print(e);
 }

}

public String[] forDownload() {
 String driver, dburl, user, passwd;
 driver = "sun.jdbc.odbc.JdbcOdbcDriver";
 dburl = "jdbc:odbc:crawler_database";
 user = "";
 passwd = "";

 String[] urlsForDownload = new String[100];

 try {
  Class.forName(driver);
 }
 catch (java.lang.ClassNotFoundException e) {
  System.err.print("ClassNotFoundException: ");
  System.err.println(e.getMessage());
 }
 Connection conn = null;
 Statement s;

 try {
  conn = DriverManager.getConnection(dburl, "", "");
  s = conn.createStatement();
  System.out.println("Pragmatopoihthike h syndesi");

  s.execute("SELECT * FROM initialURLS WHERE Downloaded = False");

  ResultSet rs = s.getResultSet();
  String baseURL = new String(""); ;

  int urlCounter = 0;
  while (rs.next()) {
     Object value = rs.getObject(2);
```

```
        baseURL = value.toString();
        urlsForDownload[urlCounter] = baseURL;
        urlCounter++;
      }
      s.close();
      conn.close();
    }
    catch (java.sql.SQLException e) {
      System.err.println(e);
    }
    return urlsForDownload;
  }

  public String[] old() {
    String driver, dburl, user, passwd;
    driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    dburl = "jdbc:odbc:crawler_database";
    user = "";
    passwd = "";

    String[] urlsOld = new String[100];

    try {
      Class.forName(driver);
    }
    catch (java.lang.ClassNotFoundException e) {
      System.err.print("ClassNotFoundException: ");
      System.err.println(e.getMessage());
    }
    Connection conn = null;
    Statement s;

    try {
      conn = DriverManager.getConnection(dburl, "", "");
      s = conn.createStatement();
      System.out.println("Pragmatopoihthike h syndesi");

      s.execute("SELECT * FROM initialURLS WHERE Downloaded = True");
```

```
    ResultSet rs = s.getResultSet();
    String baseURL = new String(""); ;

    int urlCounter = 0;
    while (rs.next()) {
        Object value = rs.getObject(2);
        baseURL = value.toString();
        urlsOld[urlCounter] = baseURL;
        urlCounter++;
    }
    s.close();
    conn.close();
  }
  catch (java.sql.SQLException e) {
    System.err.println(e);
  }
  return urlsOld;
}

public void setDownloaded(String url) {
  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
  }
  Connection conn = null;
  Statement s;

  try {
```

```
    conn = DriverManager.getConnection(dburl, "", "");
    s = conn.createStatement();
    System.out.println("Pragmatopoihthike h syndesi");

    s.executeUpdate("UPDATE initialURLS SET Downloaded = True " +
                    "WHERE initialURLS_BaseURL = '" + url + "'");

    s.close();
    conn.close();
   }
   catch (java.sql.SQLException e) {
    System.err.println(e);
   }
}

public String getDescription(String url) {
  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";
  String description = new String("");

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
  }
  Connection conn = null;
  Statement s;

  try {
    conn = DriverManager.getConnection(dburl, "", "");
    s = conn.createStatement();
    System.out.println("Pragmatopoihthike h syndesi");
```

```
    s.execute("SELECT initialURLS_Description FROM initialURLS " +
              "WHERE initialURLS_BaseURL = '" + url + "'");

    ResultSet rs = s.getResultSet();

    while (rs.next()) {
        Object value = rs.getObject("initialURLS_Description");
        description = value.toString();
    }

    s.close();
    conn.close();
  }
  catch (java.sql.SQLException e) {
    System.err.println(e);
  }
  return description;
}

public String getOrganization(String url) {
  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";
  String organization = new String("");

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
  }
  Connecti  n conn = null;
  Statement s;

  try {
```

```
    conn = DriverManager.getConnection(dburl, "", "");
    s = conn.createStatement();
    System.out.println("Pragmatopoihthike h syndesi");

    s.execute("SELECT initialURLS_Organization FROM initialURLS " +
              "WHERE initialURLS_BaseURL = '" + url + "'");

    ResultSet rs = s.getResultSet();

    while (rs.next()) {
       Object value = rs.getObject("initialURLS_Organization");
       organization = value.toString();
    }

    s.close();
    conn.close();
  }
  catch (java.sql.SQLException e) {
    System.err.println(e);
  }
  return organization;
}

public void deleteURLs() {
  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
  }
  Connection conn = null;
```

```
  Statement s;

  try {
    conn = DriverManager.getConnection(dburl, "", "");
    s = conn.createStatement();
    System.out.println("Pragmatopoihthike h syndesi");

    s.execute("DELETE * FROM URLS");

    s.close();
    conn.close();
  }
  catch (java.sql.SQLException e) {
    System.err.println(e);
  }
}

public void deleteInitialURLs() {
  String driver, dburl, user, passwd;
  driver = "sun.jdbc.odbc.JdbcOdbcDriver";
  dburl = "jdbc:odbc:crawler_database";
  user = "";
  passwd = "";

  try {
    Class.forName(driver);
  }
  catch (java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
  }
  Connection conn = null;
  Statement s;

  try {
    conn = DriverManager.getConnection(dburl, "", "");
    s = conn.createStatement();
    System.out.println("Pragmatopoihthike h syndesi");
```

```
      s.execute("DELETE * FROM InitialURLS");

      s.close();
      conn.close();
    }
    catch (java.sql.SQLException e) {
      System.err.println(e);
    }
  }
}
```

Α.3 Κώδικας της κλάσης *DirCrawler*

```
public class DirCrawler {

 //Το αρχικό directory από το οποίο ξεκινά η επεξεργασία.
 private static String initialDir;
 private static int counter;
 private static DataAccess db = new DataAccess();


 //Συνάρτηση που πραγματοποιεί την επεξεργασία ενός αρχείου.
       //Το διαγράφει αν
       //δεν έχει κατάληξη html, htm, asp, jsp, php, doc ή pdf
       //και το μετακινεί στο αρχικό
       //directory.
 public static void processor(String name, String path,
       String description, String organization, int initial) {
  File f = new File(name);
 // path = path.concat("\\" + f.getName());
  if (f.isFile()) {
    if (f.getName().indexOf(".txt") == -1 &&
         f.getName().indexOf(".html") == -1 &&
         f.getName().indexOf(".htm") == -1 &&
         f.getName().indexOf(".asp") == -1 &&
         f.getName().indexOf(".jsp") == -1 &&
         f.getName().indexOf(".php") == -1)
```

```
        // f.getName().indexOf(".doc") == -1 &&
        //  f.getName().indexOf(".pdf") == -1)
        f.delete();
      else {
     // System.out.println(f.getParent());
        File rname = new File(f.getParent() + "\\" +
                description + counter + ".html");

        int index = getInitialDir().length();
        String url = new String("http://");

        if (index != -1)
          url = url + f.getAbsolutePath().substring(index + 1);

        db.InsertFile(counter, url, rname.getName(),
                description, organization);
        counter++;

        f.renameTo(rname);
        path = path.concat("\\" + rname.getName());
        f.delete();


        //Η αντιγραφή θα γίνει μόνο αν βρισκόμαστε σε διαφορετικό
        //directory από
        //το αρχικό. Χωρίς αυτόν τον έλεγχο αν εκτελέσουμε το πρόγραμμα
        // δεύτερη
        //συνεχόμενη φορά, όπου θα υπάρχουν μόνο τα χρήσιμα αρχεία,
        //τα αρχεία αυτά
        //θα σβηστούν.
        if (initial != 0) {

        try {

//δημιουργούμε δύο κανάλια αρχείων ένα για είσοδο και ένα για έξοδο
            FileChannel in = new FileInputStream(rname.getPath()).
                getChannel();
            FileChannel out = new FileOutputStream(path).getChannel();
```

```
            //αντιγραφή του αρχείου από το directory που βρίσκεται (name)
            //στο initialDir (path)
            in.transferTo(0, in.size(), out);
            in.close();
            out.close();
            rname.delete();
          }
          catch (IOException ex) {
            ex.printStackTrace();
          }
        }
    }
  }
}

//Συνάρτηση που πραγματοποιεί την αναζήτηση κατά βάθος. Αν
      //συναντήσει directory
      //καλεί τον εαυτό της αναδρομικά και αν συναντήσει αρχείο καλεί
      //την συνάρτηση
      //processor για να το επεξεργαστεί.
public static void dfsSearch(String name, String description,
      String organization) {
  File f = new File(name);
  if (f.isDirectory()) {
    String[] list;
    list = f.list();
    String path;// = new String(name + "\\");
    File temp;
    int initial = name.compareTo(initialDir);

    //παίρνουμε ένα ένα τα περιεχόμενα του αρχικού directory
    for (int i = 0; i < list.length; i++) {
       path = new String(name+"\\");
       path = path.concat(list[i]);
       temp = new File(path);
```

```
        //αν πρόκειται για αρχείο καλούμε τη συνάρτηση processor
        if (temp.isFile())
          processor(path, initialDir, description, organization, initial);

        //αν πρόκειται για directory καλούμε αναδρομικά τη συνάρτηση
        //dfsSearch και αφού τελειώσουμε με αυτό το διαγράφουμε
        else if (temp.isDirectory()) {
          dfsSearch(path, description, organization);
        //  System.out.println("Sbino to path " + temp.getAbsolutePath());
          temp.delete();
        }
      }
    }
  }

//Η συνάρτηση που εκτυπώνει τη χρήση της εφαρμογής.
private static void usage() {
  System.err.println("Usage: java DirCrawler absolute_path\n");
  System.exit(1);
}

public static String getInitialDir() {
  return initialDir;
}

public void setInitialDir(String dir) {
  initialDir = dir;
}

public int getCounter() {
  return counter;
}

public void setCounter(int counter) {
  this.counter = counter;
}
```

```
  public int run(String dir, String description,
        String organization, int counter) {
    setInitialDir(dir);
    setCounter(counter);
    File f = new File(initialDir);
    dfsSearch(initialDir, description, organization);
    return getCounter();
   }
}
```

Α.4 Κώδικας της σελίδας *CreateDatabase*

```
<%@ page contentType="text/html; charset=windows-1253"%>
<%@ page import = "org.apache.lucene.analysis.StopAnalyzer,
java.awt.*,
org.apache.lucene.index.IndexWriter,org.apache.lucene.demo.FileDocument,org.
apache.lucene.demo.SentenceDocument,org.apache.lucene.ipltrec.FileResolver,or
g.apache.lucene.ipltrec.ExpressFileResolver,org.apache.lucene.ipltrec.FtFileResolv
er,java.io.File,java.util.Date"%>
<%@ page import = "crawler.*" %>

<html>
<head>
<title>CreateDatabase</title>
</head>
<body bgcolor="#FFFFCC">

<h1 align = "center"> <img src="Aueb_logo.bmp">
 <font face="Arial" size="5" >Σύστημα Ερωταποκρίσεων
 </font>
  <br> <font face="Arial" size="4">Συλλογή Κειμένων</font></p>

<%
request.setCharacterEncoding("windows-1253");

String path = request.getParameter("database");
String txtPath = request.getParameter("txtCollection");
```

```
String finalPath = new String();
String url = request.getParameter("url");
String description = request.getParameter("description");

String organization = request.getParameter("organization");
int depth;

if (request.getParameter("depth") == null)
        depth = 2;
else
        depth = Integer.parseInt(request.getParameter("depth"));

if (request.getParameter("organization") == null)
organization = "";
else
organization  = request.getParameter("organization");

int fileCounter = 1;
int temp;

JoBoCrawler crawler = new JoBoCrawler();
DataAccess db = new DataAccess();
DirCrawler dirCrawler = new DirCrawler();
HtmlToTxt h2t = new HtmlToTxt();

String[] urls = new String[100];
urls = db.Show();
String[] urlsOld = new String[100];
urlsOld = db.old();
String[] urlsForDownload = new String[100];
urlsForDownload = db.forDownload();

%>



</h1>
```

```
        <form name="CreateDatabase" action="CreateDatabase.jsp"
method="post" >

        <table width="750" border="5" height="250" align="center"
cellpadding="5" cellspacing="5" bordercolor="#6699CC" bgcolor="#FFFFCC">
        <td rowspan="2" bordercolor="#FFFFCC99" bgcolor="#6699CC"
valign="top">
         <table width="80%" border="2" cellpadding="4" cellspacing="4"
align="center">
         <tr>

          <td><b><a href="JspMain.jsp"><font face="Comic Sans MS" size="2"
color="#FFFFCC">Κεντρική Φόρμα</font></a></b></td>
         </tr>

          <tr>
          <td><b><a href="Jsp2.jsp"><font face="Comic Sans MS" size="2"
color="#FFFFCC">Αναζήτηση On Line</font></a></b></td>
         </tr>

           <tr>
          <td><b><a href="SearchBatch.jsp"><font face="Comic Sans MS"
size="2" color="#FFFFCC">Αναζήτηση Batch</font></a></b></td>
         </tr>

         <tr>
          <td><b><a href="CreateDatabase.jsp"><font face="Comic Sans MS"
size="2" color="#FFFFCC">Συλλογή Κειμένων</font></a></b></td>
         </tr>

          <tr>
          <td><b><a href="JSPIndexing.jsp"><font face="Comic Sans MS"
size="2" color="#FFFFCC">Δημιουργία Αρχείου
Δεικτοδότησης</font></a></b></td>
         </tr>
          <tr>

         <td>
```

```
<img src="cancel.gif"><b><font face="Arial" size="2"
color="#FFFFCC"><input type = "submit" value = "Εξοδος"  onClick =
"window.close();"> </font></b></td>
</tr>
</table>
</td>
<td>
<table width="550" border="3" height="320" align="center">


<td width="268" align="center">
<br>
<font face="Comic Sans MS">Ιστοσελίδα</font><br>
<table>
<td align="right">
URL: <input name="url" size="23" value=""/><br>
Αρχείο: <input name="description" size="23" value="" /><br>
Ίδρυμα: <input name="organization" size="23" value="" />
</td>
</table>
<br>
<input type="submit" name="submit" value="Add"/> <input
type="submit" name="submit" value="Clear Database"/>
<br>
<br>
<br>
<font face="Comic Sans MS">Ορισμός Φακέλου Κειμένων</font>
<br>
Html <input name="database" size="29 " value=""/><br>
 Text <input name="txtCollection" size="29 " value=""/>
<br><br>
Μέγιστο βάθος <input name="depth" size="2" value="2" />
<br><br>
Συλλογή νέων<input name="update" type="radio" value="1"
onClick="agree=1" checked>
<br>
Συλλογή όλων<input name="update" type="radio" value="2"
onClick="agree=2" >
```

```
<br>
<br>
<input type="submit" name="submit" value="Collect"/>
<input type="hidden" value="1" name="sylogi"/>
<br>
<br>


<td width="262" align="center">
<p align="center">
<font face="Comic Sans MS"> Βάση Ιστοσελίδων</font><br>
<textarea rows="18" cols="28" name="urls" readonly>
Κατεβασμένα URL:
<%
int i = 0;
while (urlsOld[i] != null) {
%>    <%=urlsOld[i]%>
<%
        i++;
        }
%>

Καινούργια URL:
<%
int j = 0;
while (urlsForDownload[j] != null) {
%>    <%=urlsForDownload[j]%>
<%
        j++;
        }
%></textarea>



<%
//System.out.println("Sykk:" + request.getParameter("sylogi"));
//System.out.println("Sykk1:" + request.getParameter("update"));
```

```
        //System.out.println("Sub:$$$" + request.getParameter("submit") +
"$$$");

        if ((request.getParameter("submit") != null) &&
(request.getParameter("submit").compareTo("Add") == 0)) {
                if (((url != null) && (url.compareTo("") != 0))
                && ((description != null) && (description.compareTo("") != 0))
                && ((organization!= null) && (organization.compareTo("") != 0))) {
                        db.Insert(url, description, organization);
                        out.println("Προστέθηκε το: \n" + url);
                }
        }
        else
                if ((request.getParameter("submit") != null) &&
(request.getParameter("submit").compareTo("Collect") == 0)) {
                        if ((path != null) && (path.compareTo("") != 0)
                        && (txtPath != null) && (txtPath.compareTo("") != 0)) {
                                db.deleteURLs();
                                if (request.getParameter("update").
                                        compareTo("1") == 0) {
                                        i = 0;
                                        //System.out.println("urls for download:");
                                        while (urlsForDownload[i] != null) {
                                        //  System.out.println(urlsForDownload[i]);
                                         finalPath = path +
                                         db.getDescription(urlsForDownload[i]);

                                          crawler.start(urlsForDownload[i],
                                                finalPath, depth);
                                          db.setDownloaded(urlsForDownload[i]);
                                          temp = dirCrawler.run(finalPath,
                                         db.getDescription(urlsForDownload[i]),
                                        db.getOrganization(urlsForDownload[i]),
                                        fileCounter);
                                          fileCounter = temp;
                                          h2t.run(finalPath, txtPath,
                                         db.getDescription(urlsForDownload[i]));
                                          i++;
```

```
                        }
                    }
                    else
                        if
(request.getParameter("update").compareTo("2") == 0) {
                            i = 0;
                            while (urls[i] != null) {
                              finalPath = path +
                                    db.getDescription(urls[i]);
                              crawler.start(urls[i],
                                    finalPath, depth);
                              db.setDownloaded(urls[i]);
                              temp = dirCrawler.run(finalPath,
                             db.getDescription(urls[i]),
                             db.getOrganization(urls[i]),
                            fileCounter);
                              fileCounter = temp;
                              h2t.run(finalPath, txtPath,
                             db.getDescription(urls[i]));
                              i++;
                            }
                        }
                }

            }
            else
                if ((request.getParameter("submit") != null)
                && (request.getParameter("submit").
                        compareTo("Clear Database") == 0)) {
                        db.deleteInitialURLs();
                }

        %>
        </font>
        </td>
        <br>
        <br>
        </tr>
```

```
</table>
</table>
<p align = "right"> <font face= "Arial" size=2>Οικονομικό Πανεπιστήμιο
Αθηνών</font>
</form>
</body>
</html>
```

A.5 Κώδικας της κλάσης *Ontology*

Η κλάση αυτή είναι υπεύθυνη για τη δημιουργία μιας οντολογίας, για την εμφάνισή της στην οθόνη, καθώς και για τις λειτουργίες που χρειαζόμαστε για την εφαρμογή μας. Δηλαδή για την αναζήτηση σε μία οντολογία για μία οντότητα και την επιστροφή των συνωνύμων αυτής.

```
public class Ontology {

 public OntModel model;

 public Ontology() {
 }

 public void Create(String ontPath) {

  try {
   model = ModelFactory.createOntologyModel(
              OntModelSpec.OWL_MEM, null);
   model.read("file:" + ontPath);
  }
  catch (Exception ex) {
   ex.printStackTrace();
  }

 }

 public OntClass FindClass(String key, OntModel m) {

  OntClass searchCls, found = null;
```

```
  for (Iterator i = m.listClasses(); i.hasNext(); ) {
    searchCls = (OntClass) i.next();
    for (Iterator j = searchCls.listInstances(); j.hasNext(); ) {

        Individual ind = (Individual) j.next();
        String search = prefixesFor(ind).usePrefix(ind.getURI()).
                replace('_', ' ').substring(1);

        if (key.compareTo(search) == 0) {
          found = searchCls;
        }

    }
  }
  return found;
}

public String[] Search(String key, OntClass c) {

  String[] synonym = new String[1000];

  for (Iterator i = c.listInstances(); i.hasNext(); ) {

    Individual ind = (Individual) i.next();
    String search = prefixesFor(ind).usePrefix(ind.getURI()).
                replace('_', ' ').substring(1);

    if (key.compareTo(search) == 0) {
       int k = 0;
       for (Iterator j = ind.listSameAs(); j.hasNext(); ) {
         OntResource same = (OntResource) j.next();
         synonym[k] = prefixesFor(same).usePrefix(same.getURI()).
                replace('_', ' ').substring(1);
         k++;
       }
    }
  }
```

```
  return synonym;
}


public String[] ReturnClasses(String property, OntModel m) {

  OntProperty searchProperty, found = null;
  String[] returnClasses = new String[20];
  int counter = 0;

  for (Iterator i = m.listObjectProperties(); i.hasNext(); ) {

    searchProperty = (OntProperty) i.next();
    String search = prefixesFor(searchProperty).
        usePrefix(searchProperty.getURI()).replace('_', ' ').substring(1);

    if (search.compareTo(property) == 0) {

       for (Iterator j = searchProperty.listDomain(); j.hasNext(); ) {

         OntResource returnClass = (OntResource) j.next();
         String cls = prefixesFor(returnClass).
               usePrefix(returnClass.getURI()).
               replace('_', ' ').substring(1);
         returnClasses[counter] = cls;
         counter++;

         OntClass rClass = returnClass.asClass();

         for (Iterator k = rClass.listSubClasses(); k.hasNext(); ) {

           OntClass subClass = (OntClass) k.next();
           String subCls = prefixesFor(subClass).
                usePrefix(subClass.getURI()).replace('_', ' ').substring(1);
           returnClasses[counter] = subCls;
           counter++;
         }
       }
```

```
        }
      }
      return returnClasses;
    }


    public void describeClass(PrintStream out, OntClass cls) {
      renderClassDescription(out, cls);
      out.println();
    }

    public void renderClassDescription(PrintStream out, OntClass c) {
      for (Iterator i = c.listInstances(); i.hasNext(); ) {
        out.print("Instance ");
        Individual ind = (Individual) i.next();
        renderURI(out, prefixesFor(ind), ind.getURI());
        out.println();
        for (Iterator j = ind.listSameAs(); j.hasNext(); ) {
            OntResource same = (OntResource) j.next();
            out.print("Synonym ");
            renderURI(out, prefixesFor(same), same.getURI());
            out.println();
        }
        out.println();
      }

    }

  protected void renderURI(PrintStream out, PrefixMapping prefixes,
          String uri) {
      out.print(prefixes.usePrefix(uri).replace('_', ' ').substring(1));
    }

    protected PrefixMapping prefixesFor(Resource n) {
      return n.getModel().getGraph().getPrefixMapping();
    }

}
```

Α.6 Κώδικας της μεθόδου *FindSynonyms*

Η μέθοδος αυτή είναι εκείνη που δημιουργεί την οντολογία καλώντας τις μεθόδους της κλάσης *Ontology* και στη συνέχεια επιστρέφει τα συνώνυμα του δεδομένου κάθε φορά ιδρύματος σε έναν πίνακα από συμβολοσειρές.

```
public static String[] FindSynonyms(String organization, String ontPath) {
   Ontology ontology = new Ontology();
   ontology.Create(ontPath);
   OntClass found;
   String[] synonyms = new String[1000];

   found = ontology.FindClass(organization, ontology.model);
   if (found != null)
        synonyms = ontology.Search(organization, found);

    return synonyms;
 } //FindSynonyms
```

Στην κλάση *SearchDocuments* όπου καλείται η *FindSynonyms* έχουμε προσθέσει την κλήση της διαδικασίας με τον εξής τρόπο:

```
if (ontUse)
        synonyms = FindSynonyms(sub_result[s][1],
        "D:/lucene_fin/defaultroot/Academic.owl");
    else
        synonyms = FindSynonyms(sub_result[s][1]);
```

Οπότε έτσι ελέγχουμε τον τρόπο που βρίσκονται τα συνώνυμα. Υπάρχει η επιλογή να γίνονται και με τον προηγούμενο τρόπο, όπου είναι αποθηκευμένα σε έναν πίνακα.

Α.7 Κώδικας της μεθόδου *QueryEnrich*

Η μέθοδος αυτή εμπλουτίζει το ερώτημα που θέτει ο χρήστης, ώστε να έχουμε καλύτερα αποτελέσματα στην αναζήτηση.

```
public String QueryEnrich(String query, String ontPath) {

  Ontology ontology = new Ontology();

  int temp_keno = query.indexOf(" ", 0);
  int keno_start = 0;

  String enrichedQuery = new String("");
  String[] enrich = new String[20];
  String[] tempReturn = new String[20];
  String[] word = new String[25];
  int index1 = 0;

  try {
   while (temp_keno != -1) {
      word[index1] = query.substring(keno_start, temp_keno);
      keno_start = temp_keno + 2;
      temp_keno = query.indexOf(" ", temp_keno + 2);
      index1++;
   }
   index1--;
  }
  catch (Exception e) {
      e.printStackTrace();
   }

  ontology.Create(ontPath);

  for (int i = 0; i < index1; i++) {
   tempReturn = ontology.ReturnClasses(word[i], ontology.model);
   if (tempReturn[0] != null)
      for (int k = 0; k < 20; k++)
        enrich[k] = tempReturn[k];
```

```
    }


    int index2 = 0;
    while (enrich[index2] != null) {
  //    System.out.println("Enrich[" + index2 + "]: " + enrich[index2]);
      index2++;
    }

    int i;
    for (i = index1 + index2; i - index2 >= 1; i--) {
      word[i] = word[i - index2];
    }

    for (i = 1; i <= index2; i++) {
      word[i] = enrich[i-1];
    }

    index2 = 0;
    while (word[index2] != null) {
      enrichedQuery += word[index2] + "  ";
      index2++;
    }

    return enrichedQuery;
  }
```

Η μέθοδος αυτή καλείται από τη σελίδα JSP, έχοντας προσθέσει σε αυτή τον παρακάτω κώδικα:

```
if (ontUse) {
  String enrichedQuery = s.QueryEnrich(main_query,
        "D:/lucene_fin/defaultroot/Academic.owl");
  query.setTitle(enrichedQuery);
  System.out.println("enriched query: " + enrichedQuery);
}
else
  query.setTitle(main_query);
```

Αναφορές

1. Γιγουρτσή Ευθυμία Μαρία, *Ανάκτηση Πληροφορίας κατά Τμήματα: Μία Προσέγγιση για Συστήματα Ερωταποκρίσεων*, Αθήνα, Φεβρουάριος 2004

2. María Auxilio Medina Nieto, *An Overview of Ontologies*, March 2003

3. Natalya F. Noy and Deborah L. McGuinness, *Ontology Development 101: A Guide to Creating Your First Ontology*

4. Andreas Hotho, Steffen Staab, Gerd Stumme, *Ontologies Improve Text Document Clustering*

5. Tom Gruber, *What is an Ontology?*

6. Rifat Ozcan, Y. Alp Aslandogan, *Concept Based Information Access Using Ontologies and Latent Semantic Analysis*, 2004

7. Ali Shiri, *Schemas and Ontologies: Building a Semantic Infrastructure for the Grid and Digital Libraries*, May 2003

8. Holger Knublauch, *Three Patterns for the Implementation of Ontologies in Java*, 1999

9. Michael R. Genesereth, Richard E. Fikes, *Knowledge Interchange Format, Version 3.0, Reference Manual*, 1992

10. B. Chandrasekaran, Jorn R. Josephson, V. and Richard Benjamins, *What Are Ontologies, and Why Do We Need Them?*, 2003

11. Deborah L. McGuinness, *Ontologies Come of Age*, 2001

12. Asunción Gómez Pérez, *A survey on ontology tools*, 2002

13. Vladan Devedzic, *ONTOLOGIES: Borrowing From Software Patterns*, 1999

14. Michael Gruninger and Jintae Lee, *Ontology Applications and Design*, 2002

15. Clyde W. Holsapple and K.D. Joshi, *A Collaborative Approach to Ontology Design*, 2002

16. Henry Kim, *Predicting How Ontologies for the Semantic Web Will Evolve*, 2002

17. John O. Everett, Daniel G. Bobrow, Reinhard Stolle, Richard Crouch, Valeria de Paiva, Cleo Condoravdi, Martin van den Berg, and Livia Polanyi, *Making Ontologies Work for Resolving Redundancies Across Documents*, 2002

18. Gloria L. Zuniga, *Ontology: Its Transformation From Philosophy to Information Systems*, 2001

19. B. J.Wielinga, A. Th. Schreiber, J.Wielemaker, J. A. C. Sandberg, *From Thesaurus to Ontology*, October 2001

20. Antonio M. Lopez, Jr., *Ontology Development As Undergraduate Research*, 2002

21. AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy, *Learning to Map between Ontologies on the Semantic Web*, 2002

22. Barry Smith, Christopher Welty, *Ontology: Towards a New Synthesis*, 2001

23. Mike Uschold, Michael Gruninger, Ontologies: Principles, Methods and Applications, February 1996

24. Holger Knublauch and Mark A. Musen, *Editing Description Logic Ontologies with the Protégé OWL Plugin*, 2004

25. Holger Knublauch, Ray W. Fergerson, Natalya F. Noy and Mark A. Musen, *The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications*, 2004

26. Deborah L. McGuinness, Frank van Harmelen, *OWL Web Ontology Language Overview*, 2004

27. Michael K. Smith, Chris Welty, Deborah L. McGuinness, *OWL Web Ontology Language Guide*, 2004

28. Jeff Heflin, *OWL Web Ontology Language Use Cases and Requirements*, 2004

29. Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, Lynn Andrea Stein, *OWL Web Ontology Language Reference*, 2004

30. Matthew Horridge, Holger Knublauch, Alan Rector, Robert Stevens, Chris Wroe, *A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools*, August 2004

31. Volker Haarslev, Ralf Moller, Michael Wessel, RACER User's Guide and Reference Manual, April 2004

32. Protégé Documentation, *Tables: A Guide for the Perplexed*

33. Protégé Documentation, *User Guide*

34. Jena Documentation, *The Jena API*

35. Jena Documentation, *The Jena Ontology API*

36. Deborah L. McGuinness, *Description Logics Emerge from Ivory Towers*, 2001

37. Sean Bechhofer, *OilEd 3.4 Manual*

38. ontoprise GmbH, *How to Work with OntoEdit*, 2003

39. James Rice, Adam Farquhar, Philippe Piernot, and Thomas Gruber, *Using the Web Instead of a Window System*, Ontolingua, 1996

40. JOE: http://www.cse.sc.edu/research/cit/demos/java/joe/

41. Ontolingua: http://www.ksl.stanford.edu/software/ontolingua/

42. Chimaera: http://www.ksl.Stanford.EDU/software/chimaera

43. OilED: http://oiled.man.ac.uk/

44. Protégé: http://protege.stanford.edu/

45. Ontolingua: http://ontolingua.standord.edu

46. Jena: http://jena.sourceforge.net/