ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Απεικόνιση Οντολογιών Σε Σχήματα Σχεσιακών Βάσεων Δεδομένων Με Σκοπό Την Ανάκτηση Δεδομένων Σημασιολογικού Περιεχομένου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος - Εμμανουήλ Α. Σπανός

Μιχαήλ Ι. Χαλάς

Επιβλέπων : Νικόλαος Μήτρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2005

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Απεικόνιση Οντολογιών Σε Σχήματα Σχεσιακών Βάσεων Δεδομένων Με Σκοπό Την Ανάκτηση Δεδομένων Σημασιολογικού Περιεχομένου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δημήτριος - Εμμανουήλ Α. Σπανός

Μιχαήλ Ι. Χαλάς

Επιβλέπων : Νικόλαος Μήτρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 30η Σεπτεμβρίου 2005.

.....	.....	.....
Νικόλαος Μήτρου	Μιχαήλ Θεολόγου	Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Σεπτέμβριος 2005

.....
Δημήτριος – Εμμανουήλ Α. Σπανός

.....
Μιχαήλ Ι. Χαλάς

Διπλωματούχοι Ηλεκτρολόγοι Μηχανικοί και Μηχανικοί Υπολογιστών Ε.Μ.Π.

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής που επιτρέπει την αντιστοίχιση οντολογιών σε σχήματα σχεσιακών βάσεων δεδομένων και την ανάκτηση δεδομένων μέσω ενός σημασιολογικού ερωτήματος. Συγκεκριμένα, με τη βοήθεια γραφικού περιβάλλοντος, αναπαρίστανται η ιεραρχία και τα περιεχόμενα της οντολογίας καθώς και το σχήμα και τα περιεχόμενα της βάσης δεδομένων. Παράλληλα, καθίσταται δυνατή η επιλογή συγκεκριμένων δεδομένων της βάσης που θα αντιστοιχηθούν σε επιλεγμένο οντολογικό περιεχόμενο. Η αντιστοίχιση αυτή επιτυγχάνεται, με την εισαγωγή κατάλληλου ερωτήματος βάσης δεδομένων στην οντολογία. Συνεπώς, το σημασιολογικό ερώτημα μπορεί, μέσω της αντιστοίχισης, να επιστρέψει δεδομένα από τη βάση.

Η διπλωματική αυτή εργασία περιλαμβάνει μία αναφορά στις έννοιες του Σημασιολογικού Ιστού, των οντολογιών και των βάσεων δεδομένων που αποτελούν τις βασικές θεωρητικές έννοιες που σχετίζονται με την υλοποιηθείσα εφαρμογή. Στη συνέχεια, παρουσιάζονται οι τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής (Jena, Protégé, MySQL, PostgreSQL). Ακολουθεί αναλυτική περιγραφή της λειτουργίας και των περιπτώσεων χρήσης της εφαρμογής, καθώς και ανάλυση του κώδικα της εφαρμογής.

Λέξεις Κλειδιά

Σημασιολογικός Ιστός, Οντολογία, OWL, Jena, RDQL, Protégé, Βάση Δεδομένων, MySQL, PostgreSQL, Αντιστοίχιση, Χωρικά - Γεωγραφικά δεδομένα.

Abstract

The object of this thesis is the development of an application that enables ontologies to be mapped to relational databases' schemas, and data to be retrieved through the execution of a semantic query. In fact, the application, using a graphical interface, manages to represent the hierarchy and the contents of an ontology as well as the schema and the contents of a relational database. Furthermore, actual data from the database can be graphically selected and then mapped to a specific ontology class. This occurs through the inclusion of an SQL query into the ontology. Thus, the execution of the semantic query may, through the mapping process, return actual data from the database.

This thesis includes references to the concepts of Semantic Web, ontologies and databases, which represent the fundamental theoretical concepts related to the application developed. The technologies and tools used for the implementation of the application are also presented (Jena, Protégé, MySQL, PostgreSQL). An analytic description of the operation and the use cases of the application, as well as an analysis of the application code, follow.

Keywords

Semantic Web, Ontology, OWL, Jena, RDQL, Protégé, Database, MySQL, PostgreSQL, Mapping, Spatial - Geographic data

ΠΕΡΙΕΧΟΜΕΝΑ

1 ΕΙΣΑΓΩΓΗ

Η παρούσα διπλωματική εργασία έχει ως στόχο την υλοποίηση μιας εφαρμογής η οποία πραγματοποιεί **αντιστοίχιση** μεταξύ δεδομένων που είναι αποθηκευμένα σε κάποια **βάση δεδομένων** και κλάσεων μιας **οντολογίας**. Με αυτόν τον τρόπο επιτυγχάνεται η προσθήκη σημασιολογικής πληροφορίας στα περιεχόμενα μιας βάσης δεδομένων.

Η υλοποιηθείσα εφαρμογή, εκτός των άλλων, επιτρέπει την απεικόνιση του σχήματος μιας σχεσιακής βάσης δεδομένων, καθώς και προεπισκόπηση των δεδομένων που περιέχονται σε αυτή, την απεικόνιση μιας οντολογίας, την αντιστοίχιση δεδομένων της βάσης με μια κλάση της οντολογίας και τέλος, την **εκτέλεση ενός σημασιολογικού ερωτήματος** που συνοδεύεται από την επιστροφή **πραγματικών δεδομένων** που έχουν απεικονιστεί στις οντολογικές κλάσεις που απαντούν στο εν λόγω ερώτημα.

Η επιλογή των δεδομένων από τη βάση γίνεται με γραφικό τρόπο, ενώ παρασκηνιακά δημιουργείται ένα ερώτημα σε γλώσσα βάσης δεδομένων (SQL ερώτημα), το οποίο συσχετίζεται με μια κλάση της οντολογίας.

Η εφαρμογή ενσωματώνεται στο εργαλείο επεξεργασίας και ανάπτυξης οντολογιών **Protégé** και εκτελείται μέσω αυτού, υποστηρίζει οντολογίες που υπακούουν στις αρχές της γλώσσας **OWL** (Web Ontology Language), καθώς και τα συστήματα διαχείρισης βάσεων δεδομένων MySQL και PostgreSQL. Σημειώνεται ότι αμφότερα τα συστήματα διαχείρισης επιτρέπουν την αποθήκευση **χωρικών – γεωγραφικών** δεδομένων, στη χρήση των οποίων δίνεται έμφαση στην εργασία αυτή.

Στο **Κεφάλαιο 2** της εργασίας γίνεται μία θεωρητική αναφορά στις βασικές έννοιες που σχετίζονται με την εφαρμογή. Συγκεκριμένα, γίνεται αναφορά στο όραμα και τις βασικές αρχές του **Σημασιολογικού Ιστού**, αναλύεται η έννοια της **οντολογίας** και της σπουδαιότητάς της, παρουσιάζεται λεπτομερώς η οντολογική γλώσσα OWL, καθώς και άλλες οντολογικές γλώσσες και γλώσσες σημασιολογικών ερωτημάτων, ενώ παρατίθενται και μερικά από τα πιο γνωστά εργαλεία ανάπτυξης και επεξεργασίας οντολογιών. Ακόμα, εκτίθενται οι βασικές αρχές των βάσεων

δεδομένων, παρουσιάζεται η γλώσσα υποβολής ερωτημάτων SQL και γίνεται αναφορά στις βάσεις χωρικών δεδομένων.

Το **Κεφάλαιο 3** περιλαμβάνει τα εργαλεία και τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Εξετάζονται και παρουσιάζονται το εργαλείο για τη διαχείριση και επεξεργασία οντολογιών **Protégé**, η **Jena**, ένα Java API για τη σύνδεση με οντολογίες και την επεξεργασία τους, καθώς και τα δύο συστήματα διαχείρισης βάσεων δεδομένων που υποστηρίζονται από την εφαρμογή, **MySQL** και **PostgreSQL**.

Το **Κεφάλαιο 4** σκιαγραφεί τις σημαντικότερες προσπάθειες που έχουν γίνει μέχρι σήμερα στον τομέα της **αντιστοίχισης οντολογιών**. Περιλαμβάνει μια πληθώρα εργαλείων και προτάσεων που αναφέρονται σε αντιστοίχιση στοιχείων μιας οντολογίας με μία άλλη, σε τρόπους συγχώνευσης οντολογιών όπως και σε απεικόνιση στοιχείων μιας βάσης δεδομένων σε μια οντολογία.

Στο **Κεφάλαιο 5** δίνεται μια αναλυτική περιγραφή των δυνατοτήτων και του τρόπου λειτουργίας της υλοποιηθείσας **εφαρμογής**, ενώ αναφέρονται οι περιορισμοί της υλοποίησης καθώς και πιθανές μελλοντικές επεκτάσεις της εφαρμογής.

Το **Κεφάλαιο 6** εξετάζει με μεγαλύτερη λεπτομέρεια την υλοποίηση εστιάζοντας στον **κώδικα** και παραθέτοντας αρκετά μέρη του. Τέλος, στο Κεφάλαιο 7, παρατίθενται τα συμπεράσματα της παρούσας διπλωματικής εργασίας.

2 ΘΕΩΡΙΑ

2.1 ΣΗΜΑΣΙΟΛΟΓΙΚΟΣ ΙΣΤΟΣ

Ο **Παγκόσμιος Ιστός** (World Wide Web) σχεδιάστηκε όχι μόνο για να επικοινωνούν οι άνθρωποι μεταξύ τους, αλλά και για να μπορούν οι μηχανές να συμμετέχουν όσο το δυνατόν σε αυτήν την επικοινωνία και να την υποβοηθούν. Ένα από τα μεγαλύτερα εμπόδια για την πραγματοποίηση αυτής της προοπτικής είναι το γεγονός ότι το μεγαλύτερο μέρος της πληροφορίας που υπάρχει στον Ιστό προορίζεται για ανθρώπινη κατανάλωση, ενώ ακόμα και όταν αυτή η πληροφορία προέρχεται από μια βάση δεδομένων της οποίας οι στήλες έχουν σαφή σημασία, η δομή αυτών των δεδομένων δεν είναι φανερή για μια μηχανή [3].

Εξάλλου, όπως είναι διαμορφωμένος σήμερα ο Παγκόσμιος Ιστός, η γνώση των εγγράφων και των διαθέσιμων δυνατοτήτων βασίζεται στην **αναζήτηση με λέξεις – κλειδιά**, η οποία διευκολύνεται και από την έξυπνη χρήση της συνδετικότητας μεταξύ των διαφόρων εγγράφων. Όλη αυτή η μάζα των δεδομένων δεν μπορεί να διαχειριστεί χωρίς την υποστήριξη δυνατών υπολογιστικά εργαλείων. Σε μια προσπάθεια, λοιπόν, να χαρτογραφηθεί καλύτερα ο Ιστός, οι διάφοροι υπολογιστικοί πράκτορες χρειάζονται περιγραφές των περιεχομένων και των δυνατοτήτων των προσιτών (μέσω του Ιστού) πόρων, που να μπορούν να διαβαστούν από μηχανές. Αυτές οι περιγραφές πρέπει να προστεθούν στις αναγνώσιμες από τον άνθρωπο μορφές όλων αυτών των πληροφοριών [2].

Ο **Σημασιολογικός Ιστός** (Semantic Web) είναι μια προσπάθεια να γίνουν οι πόροι του Παγκόσμιου Ιστού προσιτοί σε αυτοματοποιημένες διεργασίες προσθέτοντας πληροφορίες που επεξηγούν ή ορίζουν το περιεχόμενό τους. Με αυτόν τον τρόπο, ο Σημασιολογικός Ιστός θα αποτελέσει ένα παγκόσμιο μέσο για ανταλλαγή πληροφοριών. Η λέξη «Σημασιολογία» έχει ρίζα τις ελληνικές λέξεις «σημάδι», «σημαίνω» και «σημαντικός» και σήμερα αναφέρεται στο νόημα συχνά σε επίπεδο γλώσσας. Πολλοί υποστηρίζουν ότι ο Σημασιολογικός Ιστός αποτελεί το μεγαλύτερο σε παγκόσμιο επίπεδο έργο έξυπνης ενσωμάτωσης συστημάτων ώστε να συνεργάζονται δια-λειτουργικά. Εμπνευστής και καθοδηγητής αυτού του έργου είναι ο δημιουργός του Παγκόσμιου Ιστού, Tim Berners-Lee της Κοινοπραξίας του

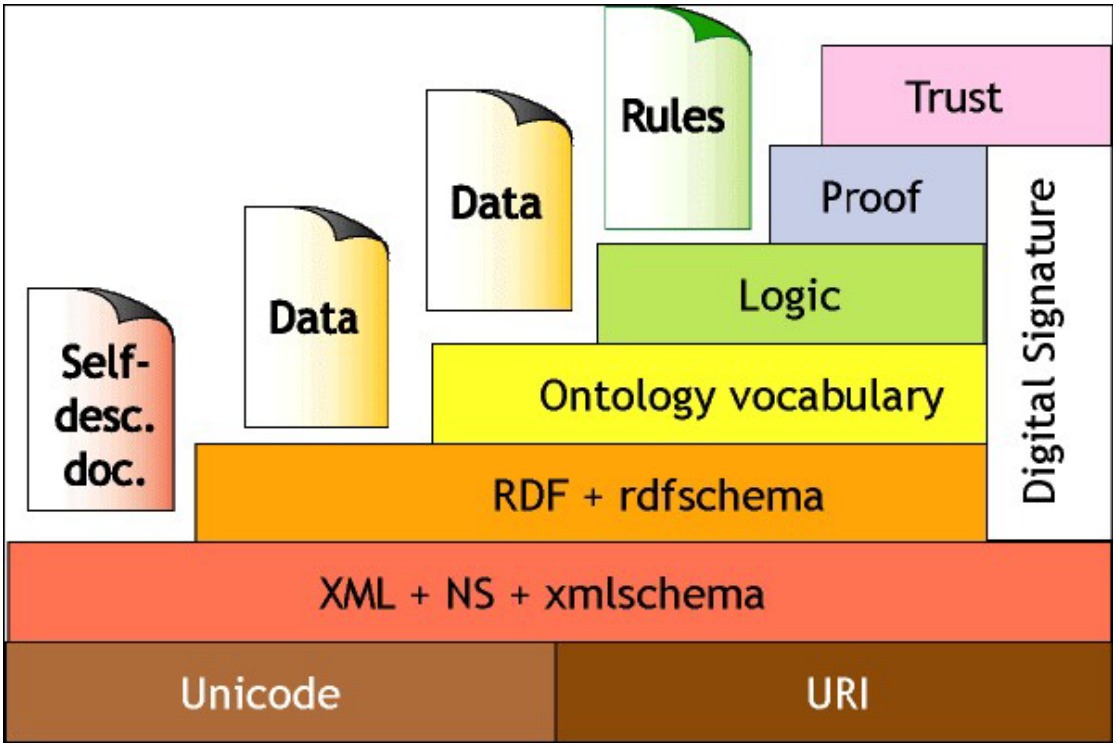
Παγκοσμίου Ιστού (World Wide Web Consortium, W3C), ο οποίος τονίζει ότι «ο Σημασιολογικός Ιστός δεν είναι ένας ξεχωριστός Ιστός, αλλά μια **επέκταση του Παγκόσμιου Ιστού** όπου η πληροφορία έχει καλά καθορισμένο νόημα, καθιστώντας τη συνεργασία μεταξύ ανθρώπων και υπολογιστών πιο αποτελεσματική» [5].

Σήμερα, ο Παγκόσμιος Ιστός είναι κατά μείζονα λόγο βασισμένος σε έγγραφα γραμμένα σε HTML, μια γλώσσα που είναι χρήσιμη για την περιγραφή δομημένου κειμένου διανθισμένου με πολυμεσικά αντικείμενα και η οποία δίνει ιδιαίτερη έμφαση στην οπτική απεικόνιση. Ακόμα, η HTML έχει μειωμένη δυνατότητα ταξινόμησης τμημάτων κειμένου σε μια σελίδα ανάλογα με τη σημασία τους. Αυτό το μειονέκτημα έρχεται να καλύψει ο Σημασιολογικός Ιστός, χρησιμοποιώντας τις περιγραφικές τεχνολογίες **RDF** (Resource Description Framework) και **OWL** (Web Ontology Language), καθώς και την προσαρμοζόμενη γλώσσα επισήμανσης (markup language) **XML** (eXtensible Markup Language). Αυτές οι τεχνολογίες συνδυάζονται προκειμένου να παρέχουν περιγραφές που συμπληρώνουν ή αντικαθιστούν το περιεχόμενο των εγγράφων του Ιστού. Επομένως, το περιεχόμενο μπορεί να φαίνεται ως περιγραφικά δεδομένα αποθηκευμένα σε προσιτές από τον Ιστό βάσεις δεδομένων ή ως επισήμανση μέσα σε έγγραφα. Αυτές οι περιγραφές που μπορούν να διαβαστούν και από μηχανές διευκολύνουν την αυτοματοποιημένη συλλογή πληροφοριών και την έρευνα από υπολογιστές [4].

Συνοπτικά, οι τεχνολογίες στις οποίες βασίζεται το όραμα του Σημασιολογικού Ιστού είναι:

- η XML, η οποία παρέχει μια επιφανειακή σύνταξη για δομημένα έγγραφα, αλλά δεν επιβάλλει σημασιολογικούς περιορισμούς στο νόημα αυτών των εγγράφων
- η XML Schema, η οποία είναι η γλώσσα με την οποία περιορίζεται η δομή των εγγράφων XML
- η RDF, η οποία αποτελεί ένα μοντέλο που περιλαμβάνει αντικείμενα (ή αλλιώς, πόρους) και σχέσεις μεταξύ τους. Ακόμα, η RDF παρέχει απλή σημασιολογία για αυτό το μοντέλο
- η RDF Schema, η οποία είναι ένα λεξιλόγιο (ή αλλιώς, οντολογία) για την περιγραφή ιδιοτήτων και κλάσεων (ομάδων) RDF αντικειμένων (ή πόρων) και διαθέτει σημασιολογία για γενίκευση ιεραρχιών τέτοιων ιδιοτήτων και κλάσεων
- η OWL, η οποία προσθέτει περισσότερο λεξιλόγιο για την περιγραφή ιδιοτήτων και κλάσεων.

Όλες οι παραπάνω τεχνολογίες ενσωματώνονται στο παρακάτω σχήμα, το οποίο απεικονίζει συνοπτικά το σχεδιασμό και το όραμα του Σημασιολογικού Ιστού, όπως τα συνέλαβε ο T. Berners-Lee:



Εικόνα 2.1 Διαστρωμάτωση Σημασιολογικού Ιστού

Είναι φανερό ότι η XML, τα URIs (Universal Resource Identifiers) με τα οποία προσδιορίζονται οι πόροι και οι χώροι ονομάτων (namespaces) αποτελούν τη βάση του οικοδομήματος του Σημασιολογικού Ιστού. Εκτός των όσων έχουν ήδη αναφερθεί, το Στρώμα Λογικής (Logic) χρησιμοποιείται για την επεξεργασία της πληροφορίας και των εξαγωγή συμπερασμάτων. Το Στρώμα Απόδειξης (Proof) συνδέει τη συμπερασματική διαδικασία, καθώς και την αναπαράσταση των αποδείξεων με την αξιοπιστία των αντίστοιχων αποδείξεων. Τέλος, το Στρώμα Αξιοπιστίας (Trust) θα αναδυθεί μέσω της χρήσης ψηφιακών υπογραφών και πληροφοριών άλλων ειδών, που θα βασίζονται σε υπηρεσίες ή πράκτορες πιστοποίησης [1].

Με αυτόν τον τρόπο, στόχος είναι η βελτίωση της χρησιμότητας του Παγκόσμιου Ιστού και των διασυνδεδεμένων πόρων του, μέσω της σηματοδότησης εγγράφων με σημασιολογική πληροφορία. Επίσης, η δημιουργία **κοινών λεξιλογίων μεταδεδομένων** (**οντολογιών**) καθώς και η **αντιστοίχιση** μεταξύ των διαφορετικών λεξιλογίων θα επιτρέψουν στους συγγραφείς εγγράφων να γνωρίζουν με ποιο τρόπο θα τα σηματοδοτήσουν, ώστε αυτοματοποιημένοι πράκτορες να χρησιμοποιήσουν την πληροφορία που κρύβεται στα μεταδεδομένα για να επιτελέσουν διάφορες εργασίες.

Εκτός των παραπάνω, ο Berners-Lee σε συνέντευξή του τον Ιούνιο του 2005 αναφέρει ότι ο σκοπός της πρωτοβουλίας του Σημασιολογικού Ιστού είναι η δημιουργία ενός **διεθνούς μέσου ανταλλαγής δεδομένων**, με τα δεδομένα να μπορούν να μοιραστούν και να επεξεργαστούν τόσο από αυτοματοποιημένα εργαλεία όσο και από ανθρώπους. Ο Σημασιολογικός Ιστός έχει σχεδιαστεί έτσι ώστε να αλληλοσυνδεθούν η διαχείριση της προσωπικής πληροφορίας, η ενσωμάτωση των εταιρικών εφαρμογών και η παγκόσμια διανομή εμπορικών, επιστημονικών και πολιτισμικών δεδομένων. Με τον όρο δεδομένα, εννοούνται κυρίως τα περιεχόμενα

σχεσιακών βάσεων δεδομένων και XML εγγράφων, και όχι μεμονωμένα ανθρώπινα έγγραφα. Συμπερασματικά λοιπόν, ο Σημασιολογικός Ιστός θα μπορεί να υποστηρίζει πράκτορες λογισμικού που όχι μόνο θα μπορούν να εντοπίζουν συγκεκριμένα δεδομένα, αλλά και να τα «καταλάβουν» με την έννοια ότι πλέον οι υπολογιστές θα μπορούν να εκτελέσουν σημαντικές εργασίες με δεδομένα αυτόματα και συνεχώς, ενώ σήμερα οι ίδιες εργασίες πρέπει να εκτελούνται χειρωνακτικά από τους χρήστες [7].

2.2 ΟΝΤΟΛΟΓΙΕΣ

2.2.1 Ορισμός

Ο όρος **Οντολογία** προέρχεται από τη φιλοσοφία και είναι το όνομα ενός πεδίου της φιλοσοφίας, συγκεκριμένα, της μελέτης της φύσης της ύπαρξης, ενός κλάδου της μεταφυσικής που ασχολείται με την αναγνώριση, σε γενικές γραμμές, των ειδών των πραγμάτων που υπάρχουν και τον τρόπο περιγραφής τους. Για παράδειγμα, η παρατήρηση ότι ο κόσμος αποτελείται από συγκεκριμένα αντικείμενα που μπορούν να ομαδοποιηθούν σε αφηρημένες κλάσεις, με βάση κοινές ιδιότητες είναι μια τυπική οντολογική δέσμευση [1].

Βέβαια, τα τελευταία χρόνια, η λέξη Οντολογία έγινε μία από τις πολλές λέξεις που χρησιμοποιήθηκαν από την επιστήμη των υπολογιστών και που πήραν ένα τελείως διαφορετικό νόημα σε σχέση με το αρχικό τους. Μια σύντομη ερμηνεία της σύγχρονης χρήσης της λέξης Οντολογία που έχει δοθεί από τον T.R. Gruber και βελτιωθεί από τον R. Studer είναι η εξής: «Μία Οντολογία είναι μια ρητή και μεθοδική προδιαγραφή μιας σημασιολογίας». Με τον όρο σημασιολογία, ο Gruber θεωρεί το σύνολο των **αντικειμένων**, **εννοιών** και άλλων **οντοτήτων** που υπάρχουν σε μια συγκεκριμένη περιοχή ενδιαφέροντος, καθώς και τις **σχέσεις με τις οποίες συνδέονται**. Μια σημασιολογία είναι μια αφηρημένη, απλοποιημένη όψη του κόσμου που χρειάζεται να αναπαρασταθεί για κάποιο λόγο. Κάθε βάση γνώσης ή σύστημα ή πράκτορας βασισμένος σε γνώση δεσμεύεται είτε άμεσα είτε έμμεσα από μια τέτοια σημασιολογία [8].

Πιο απλά, στις προδιαγραφές της γλώσσας OWL που έχουν προταθεί από την Κοινοπραξία του Παγκόσμιου Ιστού (W3C), αναφέρεται ότι μια οντολογία ορίζει τους όρους που χρειάζονται για να περιγραφεί και αναπαρασταθεί μια περιοχή γνώσης. Οι οντολογίες χρησιμοποιούνται από ανθρώπους, βάσεις δεδομένων και εφαρμογές που διαμοιράζουν πληροφορίες σχετικές με έναν τομέα (περιοχή γνώσης σχετική με ένα συγκεκριμένο αντικείμενο). Οι οντολογίες περιλαμβάνουν **ορισμούς βασικών εννοιών** αυτού του τομέα, που μπορούν να χρησιμοποιηθούν από τους υπολογιστές, καθώς και **σχέσεις μεταξύ αυτών των εννοιών**. Επίσης, κωδικοποιούν τη γνώση ενός τομέα καθώς και γνώση που εκτείνεται σε περισσότερους του ενός τομείς. Με αυτόν τον τρόπο, καθιστούν τη γνώση επαναχρησιμοποιήσιμη [9].

Η λέξη οντολογία έχει χρησιμοποιηθεί για την περιγραφή διαφόρων δομών που ποικίλλουν από απλές ταξινομήσεις (όπως η ιεραρχία του Yahoo), σχήματα μεταδεδομένων έως λογικές θεωρίες. Ο Σημασιολογικός Ιστός χρειάζεται οντολογίες

με μια **δομή** που να έχει κάποιο δεδομένο νόημα. Συγκεκριμένα, οι οντολογίες πρέπει να καθορίζουν περιγραφές για τις παρακάτω έννοιες:

- Κλάσεις (γενικά αντικείμενα) σε διάφορους τομείς ενδιαφέροντος
- Σχέσεις που μπορεί να υπάρχουν μεταξύ αντικειμένων
- Ιδιότητες που αυτά τα αντικείμενα μπορεί να έχουν

Γενικά, οι **κλάσεις** μιας οντολογίας δηλώνουν έννοιες ενός τομέα ενδιαφέροντος. Για παράδειγμα, στην περίπτωση ενός πανεπιστημίου, τα μέλη του προσωπικού, οι φοιτητές, τα μαθήματα και οι αίθουσες των μαθημάτων είναι ορισμένες σημαντικές έννοιες. Οι **σχέσεις** μεταξύ αντικειμένων τυπικά περιλαμβάνουν ιεραρχίες κλάσεων. Μια ιεραρχία ορίζει ότι μια κλάση C είναι υποκλάση μιας άλλης κλάσης C΄, αν κάθε αντικείμενο που ανήκει στη C, ανήκει και στη C΄. Εκτός όμως από τις σχέσεις υποκλάσεων, οι οντολογίες μπορούν να περιλαμβάνουν πληροφορίες όπως **ιδιότητες** (ο καθηγητής X διδάσκει το μάθημα Y), περιορισμούς τιμών (μόνο καθηγητές μπορούν να διδάξουν μαθήματα), αναφορές μη επικαλυπτόμενων κλάσεων (disjoint classes), καθώς και προδιαγραφές λογικών σχέσεων μεταξύ αντικειμένων (κάθε τμήμα οφείλει να περιλαμβάνει τουλάχιστον δέκα καθηγητές) [1].

Οι οντολογίες εκφράζονται συνήθως σε μια βασισμένη στη λογική γλώσσα, έτσι ώστε να μπορούν να γίνουν λεπτομερείς, ακριβείς, συνεπείς και έγκυρες διακρίσεις μεταξύ κλάσεων, ιδιοτήτων και σχέσεων. Κάποια εργαλεία, χρησιμοποιώντας τις οντολογίες μπορούν να εκτελούν αυτοματοποιημένες αιτιολογήσεις (reasoning), παρέχοντας με αυτόν τον τρόπο προχωρημένες υπηρεσίες σε έξυπνες εφαρμογές, όπως είναι η σημασιολογική αναζήτηση και εύρεση, οι πράκτορες λογισμικού, η υποστήριξη αποφάσεων, η κατανόηση ομιλίας και φυσικής γλώσσας, η διαχείριση γνώσης, οι έξυπνες βάσεις δεδομένων και το ηλεκτρονικό εμπόριο [9].

Στον αναπτυσσόμενο Σημασιολογικό Ιστό, οι οντολογίες εμφανίζονται καταφανώς ως ένας τρόπος για να αναπαρασταθεί η σημασιολογία των εγγράφων και να επιτραπεί σε αυτή να χρησιμοποιηθεί από δικτυακές εφαρμογές και έξυπνους πράκτορες. Οι οντολογίες μπορεί να αποδειχθούν για μια κοινότητα ως ένας τρόπος δόμησης και ορισμού του νοήματος κάποιων όρων μεταδεδομένων που, την τρέχουσα χρονική στιγμή, συλλέγονται και προτυποποιούνται. Με τη χρήση των οντολογιών, οι μελλοντικές εφαρμογές θα μπορούν να είναι «έξυπνες», με την έννοια ότι θα μπορούν να δουλεύουν σε ένα επίπεδο που θα πλησιάζει αρκετά το ανθρώπινο εννοιολογικό επίπεδο.

Οι οντολογίες είναι κρίσιμες για εφαρμογές που επιθυμούν να ψάξουν ή να συνενώσουν πληροφορίες από ποικίλες κοινότητες. Παρά το γεγονός ότι η XML και η XML Schema είναι επαρκείς για την ανταλλαγή δεδομένων μεταξύ μερών που έχουν συμφωνήσει σε κάποιους ορισμούς εκ των προτέρων, η έλλειψη σημασιολογίας εμποδίζει τις μηχανές από την αξιόπιστη εκτέλεση μιας συγκεκριμένης εργασίας με καινούρια XML λεξιλόγια. Ο ίδιος όρος μπορεί να χρησιμοποιείται με διαφορετικό νόημα σε διαφορετικές περιπτώσεις, ενώ διαφορετικοί όροι μπορεί να χρησιμοποιούνται για αντικείμενα που έχουν την ίδια σημασία. Η RDF και η RDF Schema αρχίζουν να προσεγγίζουν αυτό το πρόβλημα επιτρέποντας μια απλή σημασιολογία να συσχετίζεται με κάποια αναγνωριστικά στοιχεία, όπως είναι τα URIs. Με την RDF Schema, ο καθένας μπορεί να ορίσει κλάσεις που έχουν πολλαπλές υποκλάσεις και υπερκλάσεις, καθώς και να ορίσει ιδιότητες, που μπορεί να έχουν υπο-ιδιότητες, πεδία ορισμού και πεδία τιμών. Με αυτήν την έννοια, η RDF

Schema είναι μια απλή οντολογική γλώσσα. Όμως, προκειμένου να επιτευχθεί **διαλειτουργικότητα** μεταξύ πολυάριθμων, αυτόνομα αναπτυσσόμενων και διαχειριζόμενων σχημάτων, χρειάζεται μια **πλουσιότερη σημασιολογία**. Μια τέτοια σημασιολογία παρέχεται από τη γλώσσα **OWL** (Web Ontology Language), με τη χρήση της οποίας μπορεί να βρεθεί λύση στο παραπάνω πρόβλημα της ορολογίας. Για παράδειγμα, αν οι οντολογίες παρέχουν σχέσεις ισοδυναμίας, τότε θα υπάρχει κάπου η δυνατότητα αποθήκευσης (είτε σε μια ξεχωριστή οντολογία είτε ως απευθείας αντιστοίχιση μεταξύ δύο οντολογιών) της ισοδυναμίας του όρου Χ μιας οντολογίας με τον όρο Χ΄ μιας άλλης οντολογίας, οπότε δεν υπάρχει πρόβλημα διάκρισης των δύο όρων και διασφαλίζεται η σημασιολογική διαλειτουργικότητα των οντολογιών.

2.2.2 Σενάρια Χρήσης

Σήμερα, μία από τις τυπικές εργασίες των ανθρώπων που χρησιμοποιούν τον Παγκόσμιο Ιστό αφορά στην **αναζήτηση και χρήση πληροφορίας**. Οι **μηχανές αναζήτησης** που βασίζονται σε λέξεις – κλειδιά είναι τα κυριότερα εργαλεία για τη χρησιμοποίηση του σημερινού Ιστού. Είναι σαφές ότι ο Παγκόσμιος Ιστός δε θα είχε τόσο μεγάλη επιτυχία αν δεν υπήρχαν αυτές οι μηχανές αναζήτησης. Παρ' όλα αυτά, υπάρχουν σοβαρά προβλήματα που σχετίζονται με τη λειτουργία τους, όπως για παράδειγμα το γεγονός ότι οι αναζητήσεις είναι πολύ ευαίσθητες στο λεξιλόγιο. Ακόμα, είναι εξίσου πιθανό μια αναζήτηση να επιστρέψει πάρα πολλά ή και κανένα αποτέλεσμα, δηλαδή υπάρχει πρόβλημα χαμηλής ακρίβειας ή χαμηλής ανάκλησης αντίστοιχα. Επιπλέον, τα αποτελέσματα είναι μεμονωμένες σελίδες. Δηλαδή, αν κάποιος χρειάζεται πληροφορίες από διάφορα έγγραφα, τότε πρέπει να εκτελέσει πολλαπλές ερωτήσεις για να συλλέξει τα σχετικά έγγραφα και μετά, χειρωνακτικά, να εξάγει τις μερικές πληροφορίες και να τις συνθέσει.

Οι οντολογίες μπορούν να χρησιμοποιηθούν με έναν απλό τρόπο για τη **βελτίωση της ακρίβειας των αναζητήσεων του Ιστού**. Το πρόγραμμα αναζήτησης μπορεί να ψάχνει μόνο για αυτές τις σελίδες που αναφέρονται σε μια σαφή έννοια και όχι για αυτές που χρησιμοποιούν διφορούμενες λέξεις – κλειδιά. Με αυτόν τον τρόπο, μπορούν να ξεπεραστούν οι δυσκολίες που προκύπτουν από τη διαφορά ορολογίας μεταξύ των διαφόρων σελίδων και των ερωτήσεων του χρήστη. Ακόμα, οι αναζητήσεις στον Ιστό μπορούν να εκμεταλλευτούν και τις πληροφορίες γενίκευσης ή εξειδίκευσης. Αν για μια ερώτηση δε βρεθεί κάποιο σχετικό έγγραφο, η μηχανή αναζήτησης μπορεί να προτείνει στο χρήστη μια πιο γενική ερώτηση. Είναι μάλιστα δυνατό για τη μηχανή να εκτελεί τέτοια ερωτήματα από μόνη της προκειμένου να μειώσει το χρόνο αντίδρασης σε περίπτωση που ο χρήστης υιοθετήσει μια υπόδειξη. Επίσης, αν πολλές απαντήσεις έχουν βρεθεί, η μηχανή αναζήτησης μπορεί να υποδείξει στο χρήστη κάποιες εξειδικεύσεις [1]. Περισσότερο προχωρημένες εφαρμογές θα χρησιμοποιούν οντολογίες για να συσχετίζουν την πληροφορία σε μια σελίδα με την αντίστοιχη δομή γνώσης και με τους κανόνες για την εξαγωγή συμπερασμάτων. Γενικά, οι οντολογίες είναι πολύ χρήσιμες για την **οργάνωση και την πλοήγηση σε τοποθεσίες Ιστού** (web sites). Πολλές τέτοιες τοποθεσίες σήμερα δείχνουν στο αριστερό μέρος της σελίδας τα ανώτερα επίπεδα μιας εννοιολογικής ιεραρχίας όρων. Ο χρήστης μπορεί να πατήσει σε έναν από αυτούς για να εξαπλωθούν οι υποκατηγορίες. Αυτό συχνά ισχύει στην περίπτωση μεγάλων

οργανισμών ή εταιρειών που έχουν πολλές σελίδες σχετικές, για παράδειγμα, με συνεντεύξεις τύπου, περιγραφές προϊόντων, συγκρίσεις και προσφορές προϊόντων.

Για να πραγματοποιηθούν τα παραπάνω, τεχνικές **Αναπαράστασης Γνώσης** (Knowledge Representation) έρχονται στο επίκεντρο για τον Παγκόσμιο Ιστό, τεχνικές που έχουν μελετηθεί εδώ και καιρό στον τομέα της **Τεχνητής Νοημοσύνης** (Artificial Intelligence, AI). Οι σημασιολογικές μηχανές αναζήτησης θα γίνουν «ευφυείς» στο βαθμό που θα είναι εξοπλισμένες με μια εννοιολογική αναπαράσταση των σελίδων του Παγκόσμιου Ιστού. Αυτό θα βοηθήσει τους ανθρώπους ως άμεσους χρήστες, αλλά και τα πρακτορικά συστήματα της Τεχνητής Νοημοσύνης, που βασισμένα στην κεντρική τεχνολογία του Σημασιολογικού Ιστού προσπαθούν να προσφέρουν πιο προχωρημένες υπηρεσίες Ιστού, όπως σύγκριση πληροφοριών, ενσωμάτωση, αφαίρεση ή ανταλλαγή [10]. Για συστήματα Τεχνητής Νοημοσύνης, αυτό που «υπάρχει» είναι αυτό που μπορεί να αναπαρασταθεί. Επομένως, οι οντολογίες, καθώς αποτελούν ρητές προδιαγραφές όρων που ανήκουν σε έναν τομέα γνώσης, αποτελούν τη βάση γνώσης για τέτοια συστήματα.

Οι οντολογίες μπορούν, επίσης, να βοηθήσουν και στη **δημιουργία και συντήρηση** μιας πλήρους και ενημερωμένης **πύλης** στον Ιστό (web portal). Μια πύλη είναι απλώς μια τοποθεσία στον Ιστό που παρέχει πληροφορίες πάνω σε ένα κοινό θέμα, για παράδειγμα μια συγκεκριμένη πόλη ή κάποιον τομέα ενδιαφέροντος. Μια πύλη επιτρέπει σε άτομα που ενδιαφέρονται για αυτό το θέμα να επικοινωνήσουν μεταξύ τους, να δημιουργήσουν μια κοινότητα και να βρουν συνδέσμους σε άλλους πόρους του Ιστού κοινού ενδιαφέροντος. Για να είναι μια πύλη επιτυχημένη, πρέπει να αποτελεί μια αφετηρία για τον εντοπισμό ενδιαφέροντος υλικού. Αυτό το υλικό συνήθως παρέχεται από τα μέλη της κοινότητας, που συχνά το κατηγοριοποιούν κάτω από μια υποκατηγορία. Όμως, ένα απλό ευρετήριο των θεματικών περιοχών μπορεί να μην παρέχει στην κοινότητα τη δυνατότητα να ψάξει για το υλικό που τα μέλη της απαιτούν. Για αυτό το λόγο, οι πύλες μπορούν να ορίσουν μια οντολογία για την κοινότητα. Αυτή η οντολογία μπορεί να παρέχει μια ορολογία για την περιγραφή του υλικού καθώς και αξιώματα που ορίζουν όρους χρησιμοποιώντας άλλους όρους από την οντολογία. Όταν συνδυαστούν με γεγονότα, αυτοί οι ορισμοί επιτρέπουν άλλα γεγονότα να συναχθούν. Αυτά τα συμπεράσματα επιτρέπουν, με τη σειρά τους, στους χρήστες να ανακτήσουν αποτελέσματα αναζήτησης από την πύλη, τα οποία είναι αδύνατο να ανακτηθούν από συμβατικά συστήματα ανάκτησης.

Άλλος ένας τρόπος χρησιμοποίησης των οντολογιών είναι αυτός με τον οποίο μπορούν να παρέχουν **σημασιολογικά σχόλια** για συλλογές εικόνων, ήχων ή άλλων αντικειμένων που δεν περιέχουν κείμενο. Είναι πιο δύσκολο για τις μηχανές να εξάγουν κάποια σημασιολογία από **πολυμεσικά αντικείμενα** παρά από κείμενο φυσικής γλώσσας. Συνεπώς, αυτοί οι τύποι πόρων τυπικά δεικτοδοτούνται με κάποια μετα-δεδομένα. Δυστυχώς, καθώς διαφορετικοί άνθρωποι μπορούν να περιγράψουν αυτά τα πολυμεσικά αντικείμενα με διαφορετικούς τρόπους, είναι πολύ σημαντικό οι δυνατότητες της αναζήτησης να επεκτείνονται πέρα από ένα απλό ταίριασμα λέξης – κλειδιού. Ιδανικά, οι οντολογίες θα περιέχουν την επιπλέον πληροφορία, έτσι ώστε να **βελτιωθεί η ανάκτηση των αντικειμένων**. Οι οντολογίες που σχετίζονται με πολυμέσα μπορεί να είναι δύο ειδών: αυτές που περιέχουν πληροφορία για το μέσο και αυτές που περιέχουν πληροφορία για το περιεχόμενο. Οι σχετικές με το μέσο οντολογίες μπορούν να περιέχουν ταξινομήσεις διαφορετικών ειδών μέσων και να περιγράφουν ιδιότητες διαφορετικών μέσων. Για παράδειγμα, το μέσο του βίντεο

μπορεί να περιλαμβάνει ιδιότητες, όπως η διάρκεια της ταινίας και οι διάφορες σκηνές από τις οποίες αυτή απαρτίζεται. Οι σχετικές με το περιεχόμενο οντολογίες μπορούν να περιγράφουν το αντικείμενο του πόρου, όπως ο χρόνος, ο τόπος δράσης και οι συμμετέχοντες. Εφόσον αυτές οι οντολογίες δεν σχετίζονται με το μέσο, θα μπορούν να επαναχρησιμοποιηθούν, έτσι ώστε να βελτιωθεί η αναζήτηση για πληροφορίες πάνω σε ένα συγκεκριμένο θέμα, ανεξαρτήτως της μορφής του μέσου.

Ο Σημασιολογικός Ιστός μπορεί να παρέχει σε **πράκτορες** τη δυνατότητα να καταλάβουν και να ενσωματώσουν διάφορες πληροφορίες. Ένα συγκεκριμένο παράδειγμα είναι αυτό ενός συμβούλου κοινωνικών εκδηλώσεων, ο οποίος μπορεί να λάβει ως είσοδο τις προτιμήσεις του χρήστη και να του προτείνει κάποια διασκέδαση για το συγκεκριμένο βράδυ. Ένας πράκτορας αυτού του είδους απαιτεί οντολογίες που αναπαριστούν τους όρους για τα εστιατόρια, ξενοδοχεία κτλ, καθώς και οντολογίες υπηρεσιών που αναπαριστούν τους όρους που χρησιμοποιούνται στις σημερινές υπηρεσίες. Αυτές οι οντολογίες θα προσφέρουν τις απαραίτητες πληροφορίες, έτσι ώστε η αντίστοιχη εφαρμογή να επιτελέσει συγκρίσεις με τις προτιμήσεις του χρήστη.

Τέλος, οι οντολογίες μπορούν να χρησιμοποιηθούν και στο **ηλεκτρονικό εμπόριο**, διευκολύνοντας σημαντικά το χρήστη – πελάτη. Ιδανικά, ένας χρήστης θα μάζευε πληροφορίες σχετικά με τιμές, όρους υπηρεσιών και κατάσταση (όπως η διαθεσιμότητα) από τα σημαντικότερα διαδικτυακά καταστήματα προκειμένου να διαλέξει την καλύτερη προσφορά. Όμως, η εκτέλεση αυτής της διαδικασίας από τον ίδιο το χρήστη είναι αρκετά χρονοβόρα. Προκειμένου λοιπόν να ανακουφιστεί ο χρήστης, υπάρχουν κάποιοι πράκτορες λογισμικού οι οποίοι επισκέπτονται αρκετά καταστήματα, εξάγουν πληροφορίες σχετικά με τα προϊόντα και τις τιμές τους και δημιουργούν ένα πανόραμα της αγοράς. Η εξαγωγή αυτών των χαρακτηριστικών γίνεται μέσω αναζητήσεων με λέξεις – κλειδιά και άλλων τεχνικών ανάλυσης κειμένου και συνεπώς, δεν είναι τόσο αξιόπιστες. Αντίθετα, με την εισαγωγή των οντολογιών, θα υπάρχουν πράκτορες λογισμικού, οι οποίοι θα ερμηνεύουν τις τιμές και τους όρους κάθε υπηρεσίας και θα εξάγουν σωστά τις πληροφορίες κάθε προϊόντος. Στη συνέχεια, όλες οι πληροφορίες θα συγκρίνονται με τις απαιτήσεις του χρήστη, ο οποίος από άλλες πηγές θα μπορεί να ανακτά πληροφορίες που αφορούν στη φήμη και την αξιοπιστία των διαδικτυακών καταστημάτων.

2.2.3 Οντολογικές Γλώσσες

Οι οντολογικές γλώσσες επιτρέπουν στο χρήστη να γράψει μια σαφή και επίσημη εννοιολογική θεώρηση για κάποιον τομέα γνώσης. Οι κυριότερες απαιτήσεις είναι μια **καλά ορισμένη σύνταξη**, μια **μεθοδική σημασιολογία**, **αποτελεσματική συλλογιστική υποστήριξη**, **επαρκής εκφραστική ισχύς** και **εκφραστική ευκολία**. Δύο από τις σημαντικότερες τεχνολογίες στις οποίες βασίζεται η ανάπτυξη του Σημασιολογικού Ιστού έχουν ήδη αναφερθεί και είναι η XML (eXtensible Markup Language) και η RDF (Resource Description Framework). Η XML επιτρέπει στον καθένα να δημιουργήσει τις δικές του ετικέτες – κρυμμένες ταμπέλες που σχολιάζουν σελίδες του Παγκόσμιου Ιστού ή κομμάτια κειμένου σε μια σελίδα. Προγράμματα μπορούν να χρησιμοποιήσουν αυτές τις ετικέτες με πολύπλοκους τρόπους, αλλά ο προγραμματιστής πρέπει να ξέρει για ποιο σκοπό χρησιμοποιεί ο συγγραφέας του

εγγράφου την κάθε ετικέτα. Με λίγα λόγια, η **XML** επιτρέπει στους χρήστες να προσθέσουν μια αυθαίρετη δομή στα έγγραφά τους, αλλά δεν προσφέρει καμιά πληροφορία σχετικά με τη σημασία αυτής της δομής.

Η σημασία εκφράζεται μέσω της **RDF**, η οποία την κωδικοποιεί σε σύνολα τριάδων, με κάθε τριάδα να μπορεί να παρομοιαστεί με το υποκείμενο, το ρήμα και το αντικείμενο μιας στοιχειώδους πρότασης. Η RDF ήταν η πρώτη γλώσσα που καθορίστηκε από το W3C για την αναπαράσταση σημασιολογικής πληροφορίας για αυθαίρετους πόρους. Οι τριάδες της RDF μπορούν να γραφτούν χρησιμοποιώντας ετικέτες της XML, επομένως μπορεί κανείς να πει ότι η RDF βασίζεται στην XML. Ένας από τους στόχους της RDF είναι να καθορίσει σημασιολογία για δεδομένα που βασίζονται σε XML με έναν πρότυπο, διαλειτουργικό τρόπο. Παρ' όλα αυτά, είναι σημαντικό το γεγονός ότι η σύνταξη XML είναι μόνο μια πιθανή σύνταξη για την RDF και ότι μπορεί να έρθουν στην επιφάνεια και εναλλακτικοί τρόποι αναπαράστασης του ίδιου RDF μοντέλου δεδομένων (μπορεί να αναφέρεται συνήθως ως γλώσσα, όμως η RDF είναι ουσιαστικά ένα μοντέλο δεδομένων). Ο ευρύτερος στόχος της RDF είναι να ορίσει ένα μηχανισμό για την περιγραφή πόρων που δεν κάνει κάποια υπόθεση για συγκεκριμένο τομέα εφαρμογής. Ο ορισμός του μηχανισμού πρέπει να είναι ανεξάρτητος από τομείς, αλλά ταυτόχρονα θα πρέπει να είναι σε θέση να περιγράψει πληροφορία σχετικά με οποιονδήποτε τομέα [11].

Οι βασικές έννοιες του μοντέλου δεδομένων RDF είναι οι **πόροι** (resources), οι **ιδιότητες** (properties) και οι **δηλώσεις** (statements). Ένας πόρος είναι ένα αντικείμενο για το οποίο θέλει κάποιος να μιλήσει. Μπορεί να είναι συγγραφείς, βιβλία, εκδότες, τοποθεσίες, άνθρωποι, ξενοδοχεία ή δωμάτια. Ένας πόρος μπορεί επίσης να είναι μια ολόκληρη σελίδα του Ιστού, δηλαδή ένα HTML έγγραφο για παράδειγμα ή ακόμα και μια συλλογή από σελίδες, όπως μια ολόκληρη τοποθεσία του Ιστού. Κάθε πόρος έχει και ένα **URI** (Universal Resource Identifier), δηλαδή έναν προσδιοριστή ο οποίος χαρακτηρίζει μοναδικά αυτόν τον πόρο. Ένα URI μπορεί να είναι ένα URL (Unified Resource Locator ή με άλλα λόγια, μια διεύθυνση στον Ιστό) ή κάποιο άλλο είδος προσδιοριστή. Σημειώνεται ότι αυτός ο προσδιοριστής δεν πρέπει απαραίτητα να επιτρέπει πρόσβαση στον πόρο που προσδιορίζει. Οι ιδιότητες είναι ένας ειδικός τύπος πόρων, καθώς περιγράφουν σχέσεις μεταξύ πόρων, όπως η ηλικία ενός συγγραφέα και ο τίτλος ενός μυθιστορήματος. Κάθε ιδιότητα έχει μια συγκεκριμένη σημασία, ορίζει τις επιτρεπόμενες τιμές της, τους τύπους των πόρων που μπορεί να περιγράψει, καθώς και τις σχέσεις της με άλλες ιδιότητες. Στην RDF, οι ιδιότητες χαρακτηρίζονται επίσης από URIs. Με αυτόν τον τρόπο, εξασφαλίζεται η δημιουργία ενός παγκόσμιου σχήματος, στο οποίο έχει μειωθεί το πρόβλημα της συνωνυμίας. Μια δήλωση είναι μια τριάδα που αποτελείται από έναν πόρο, μια ιδιότητα και μια τιμή. Η τιμή μπορεί να είναι είτε ένας πόρος είτε ένα κυριολεκτικό (literal). Τα τρία μέλη της τριάδας ονομάζονται υποκείμενο, κατηγόρημα και αντικείμενο αντίστοιχα.

Η RDF εμφανίζει αρκετά **πλεονεκτήματα** αλλά και **μειονεκτήματα**. Θετικό είναι το γεγονός ότι η RDF έχει επαρκή εκφραστική ισχύ, καθώς και η δυνατότητα της RDF να αντιστοιχεί την πληροφορία σε ένα μοντέλο αμφιμονοσήμαντα. Όμως, η RDF έχει και κάποια μειονεκτήματα, όπως η έλλειψη υποστήριξης μη δυαδικών ιδιοτήτων, ένας αρκετά σοβαρός περιορισμός καθώς συχνά χρησιμοποιούνται κατηγορήματα με περισσότερα των δύο ορισμάτων. Ακόμα, η RDF μεταχειρίζεται τις ιδιότητες ως πόρους, κάτι που μπορεί να αποδειχθεί αρκετά πολύπλοκο για μια γλώσσα

μοντελοποίησης, ενώ παράλληλα ο μηχανισμός αναπαράστασης των αφηρημένων εννοιών είναι αρκετά ισχυρός, κάτι που ξενίζει στην περίπτωση μιας απλής γλώσσας, όπως θα έπρεπε να είναι η RDF. Τέλος, η σύνταξή της που βασίζεται στην XML ταιριάζει στην περίπτωση επεξεργασίας από μηχανή, δεν είναι όμως ιδιαίτερα φιλική για τον άνθρωπο [1].

Η **RDF Schema** (RDFS) είναι μια επέκταση της RDF με σκοπό την περιγραφή λεξιλογίων RDF. Όπως αναφέρθηκε παραπάνω, η RDF δεν κάνει υποθέσεις σχετικά με κάποιον συγκεκριμένο τομέα εφαρμογής, ούτε καθορίζει τη σημασιολογία ενός τομέα. Αυτό μπορεί να το κάνει ο χρήστης χρησιμοποιώντας την RDF Schema. Η προσέγγιση της RDF Schema στην περιγραφή λεξιλογίων επιτρέπει στους σχεδιαστές λεξιλογίων να αναπαριστούν περιγραφές κλάσεων και ιδιοτήτων στον Παγκόσμιο Ιστό, περιγράφοντας για παράδειγμα τρόπους με τους οποίους συνδυασμοί κλάσεων, ιδιοτήτων και τιμών μπορούν να χρησιμοποιηθούν για να επιτελεστούν εργασίες που θα έχουν κάποιο νόημα. Το σύστημα κλάσεων και ιδιοτήτων της RDF Schema είναι παρόμοιο με το αντίστοιχο των αντικειμενοστραφών γλωσσών προγραμματισμού, όπως η Java, καθώς παρέχει ιεραρχίες κληρονομικότητας. Όμως, διαφέρει από τα παρόμοια συστήματα, καθώς αντί να ορίζει μια κλάση με βάση τις ιδιότητες που τα στιγμιότυπά της (instances) θα έχουν, ορίζει τις ιδιότητες με βάση τις κλάσεις των πόρων στους οποίους αυτές αναφέρονται. Αυτό επιτελείται με τα δομικά στοιχεία rdfs:domain και rdfs:range, με βάση τα οποία μπορεί να οριστεί το πεδίο ορισμού και το πεδίο τιμών αντίστοιχα μιας ιδιότητας. Βέβαια, υπάρχουν και άλλοι μηχανισμοί, οι οποίοι ορίζονται στο πεδίο ονομάτων rdfs και το οποίο προσδιορίζεται από την αναφορά URI `http://www.w3.org/2000/01/rdf-schema#`. Συνεπώς, γίνεται φανερό ότι η RDF Schema είναι μια στοιχειώδης οντολογική γλώσσα η οποία όμως αποτελεί τη βάση για άλλες πιο πολύπλοκες [12].

Παράλληλα με την ανάπτυξη των RDF και RDF Schema, έγιναν και κάποιες άλλες προσπάθειες ανάπτυξης οντολογικών γλωσσών για το Σημασιολογικό Ιστό, όπως για παράδειγμα η MCF (Meta Content Framework), η SHOE (Simple HTML Ontology Extensions, University of Maryland), η XOL (Ontology Exchange Language) η αμερικάνικη DAML-ONT (DARPA Agent Markup Language), η ευρωπαϊκή OIL (Ontology Interchange Language) και κυρίως, ο συνδυασμός των δύο τελευταίων, DAML+OIL. Η DAML+OIL με τη σειρά της θεωρήθηκε ως αφετηρία για το W3C προκειμένου να οριστεί η OWL, η γλώσσα η οποία σκοπεύει να προτυποποιηθεί και να γίνει ευρέως αποδεκτή ως οντολογική γλώσσα του Σημασιολογικού Ιστού.

2.2.4 OWL

Η **Γλώσσα Οντολογιών Ιστού OWL** (Web Ontology Language) [14] είναι μια γλώσσα για τον ορισμό οντολογιών του Ιστού και τη δημιουργία στιγμιοτύπων τους. Μια οντολογία της OWL μπορεί να περιλαμβάνει περιγραφές κλάσεων, ιδιοτήτων και τα στιγμιότυπά τους. Δεδομένης μιας τέτοιας οντολογίας, η μεθοδική σημασιολογία της OWL καθορίζει πώς εξάγονται οι λογικές συνέπειες, δηλαδή γεγονότα που δεν αναφέρονται ρητά μέσα στην οντολογία, αλλά συνεπάγονται από τη σημασιολογία. Αυτές οι συνεπαγωγές μπορούν να βασίζονται σε ένα μεμονωμένο έγγραφο ή σε πολλαπλά διανεμημένα έγγραφα που έχουν συνδυαστεί χρησιμοποιώντας κάποιους

από τους μηχανισμούς της OWL, οι οποίοι επιτρέπουν τη ρητή εισαγωγή πληροφορίας από άλλες οντολογίες [2].

Η OWL συγκεντρώνει όλα τα πλεονεκτήματα και τις απαιτήσεις που χρειάζεται να έχει μια οντολογική γλώσσα, προκειμένου να χρησιμοποιηθεί ευρέως ως πρότυπο για το Σημασιολογικό Ιστό. Υπερέχει σαφώς της RDF Schema, εισάγοντας νέες δυνατότητες, όπως η δυνατότητα ορισμού δύο ή περισσότερων **κλάσεων** ως **ξένων** μεταξύ τους, η δυνατότητα **λογικού συνδυασμού** κάποιων **κλάσεων** χρησιμοποιώντας λογικές πράξεις, όπως η ένωση, η τομή και το συμπλήρωμα. Ακόμα, σε αντίθεση με την RDF Schema, η OWL παρέχει τη δυνατότητα εισαγωγής **περιορισμών** του πεδίου τιμών μιας ιδιότητας μόνο για ορισμένες κλάσεις και όχι για όλες, επιτρέπει την τοποθέτηση περιορισμών σχετικά με τον αριθμό των διαφορετικών τιμών που μια ιδιότητα μπορεί ή πρέπει να πάρει (περιορισμοί πληθικότητας) και τέλος, υποστηρίζει ειδικά χαρακτηριστικά για τις ιδιότητες, όπως η μεταβατικότητα, η μοναδικότητα και η αντιστρεπτότητα.

Μεταξύ των απαιτήσεων που πρέπει να πληροί μια οντολογική γλώσσα, περιλαμβάνονται η εκφραστική ισχύς, καθώς και η επαρκής συλλογιστική υποστήριξη, δύο απαιτήσεις οι οποίες δεν μπορούν να εκπληρωθούν ταυτόχρονα. Γενικά ισχύει ότι όσο πιο πλούσια εκφραστικά είναι μια γλώσσα, τόσο πιο αναποτελεσματική γίνεται η συλλογιστική υποστήριξη, με αποτέλεσμα μετά από κάποιο σημείο, να ξεπερνά τα όρια της υπολογισιμότητας. Αυτός είναι και ο λόγος που παρακίνησε την Ομάδα Εργασίας για Οντολογίες Ιστού (Web Ontology Working Group) του W3C να ορίσει την OWL ως **τρεις διαφορετικές υπογλώσσες αυξανόμενης εκφραστικότητας**, η κάθε μία ρυθμισμένη ώστε να εκπληρώνει διαφορετικές απαιτήσεις.

OWL Full

Η OWL Full προορίζεται για τους χρήστες που επιθυμούν τη **μέγιστη εκφραστικότητα** και τη **συντακτική ελευθερία της RDF χωρίς υπολογιστικές εγγυήσεις**. Για παράδειγμα, στην OWL Full μια κλάση μπορεί να αντιμετωπιστεί ταυτόχρονα ως μια συλλογή αντικειμένων, αλλά και ως ξεχωριστό αντικείμενο η ίδια. Ουσιαστικά, η OWL Full δεν είναι υπογλώσσα, αλλά η πλήρης γλώσσα που περιέχει όλα τα δομικά στοιχεία της OWL. Επιτρέπει επίσης τον αυθαίρετο συνδυασμό αυτών των δομικών στοιχείων με την RDF και την RDF Schema. Με αυτόν τον τρόπο, γίνεται πιθανή και η αλλαγή των προκαθορισμένων αρχών της RDF εφαρμόζοντας τα δομικά στοιχεία της γλώσσας τη μια πάνω στην άλλη. Για παράδειγμα, θα μπορούσαμε να περιορίσουμε τον αριθμό των κλάσεων που μπορούν να περιγραφούν σε μια οντολογία, θέτοντας έναν περιορισμό πληθικότητας (cardinality constraint) στην κλάση «όλων των κλάσεων».

Στην OWL Full, επιτρέπεται η χρήση όλων των δομικών στοιχείων της RDF χωρίς κάποιον περιορισμό. Για παράδειγμα, το owl:Class (που περιλαμβάνει όλες τις κλάσεις) είναι ισοδύναμο με το rdfs:Class. Ακόμα, όλες οι τιμές των δεδομένων θεωρούνται μέλη του τομέα των ατόμων (individuals), έτσι ώστε το σύνολο των ατόμων στην OWL Full να περιλαμβάνει όλους τους πόρους και συνεπώς, το owl:Thing (που περιλαμβάνει όλα τα στοιχεία μιας οντολογίας) να είναι ισοδύναμο με το rdfs:Resource. Άρα, οι object και οι datatype properties δεν είναι σύνολα ξένα

μεταξύ τους (οι object properties παίρνουν ως τιμή ένα αντικείμενο γενικά, ενώ οι datatype έναν απλό τύπο δεδομένων, όπως κάποιο αλφαριθμητικό ή έναν ακέραιο). Στην OWL Full, το owl:ObjectProperty είναι ισοδύναμο με το rdf:Property.

Το πλεονέκτημα της OWL Full είναι η πλήρης προς τα πάνω συμβατότητα με την RDF, τόσο συντακτικά όσο και σημασιολογικά. Έτσι, οποιοδήποτε σύμφωνο με την RDF έγγραφο, είναι και ένα σύμφωνο με την OWL Full έγγραφο, ενώ ένα έγκυρο συμπέρασμα της RDF ή της RDF Schema είναι επίσης έγκυρο συμπέρασμα και για την OWL Full. Το μειονέκτημα της OWL Full είναι ότι η γλώσσα έχει γίνει τόσο ισχυρή, ώστε να υπάρχουν λίγες ελπίδες για πλήρη (ή έστω αποτελεσματική) συλλογιστική υποστήριξη. Συνεπώς, η OWL Full είναι χρήσιμη για τους ανθρώπους που θέλουν να συνδυάσουν την εκφραστικότητα της OWL με την ευκαμψία της RDF και οι οποίοι θυσιάζουν κάποιες εγγυήσεις που προσφέρουν οι δύο άλλες υπογλώσσες όσον αφορά σε συστήματα αιτιολόγησης.

OWL DL

Η OWL DL (Description Logic) είναι μια υπογλώσσα της OWL που θέτει έναν αριθμό από περιορισμούς στη χρήση των μηχανισμών της OWL. Η OWL DL προορίζεται για αυτούς τους χρήστες που επιθυμούν **μέγιστη εκφραστικότητα** αλλά και να διατηρήσουν την **υπολογιστική και αποφασιστική πληρότητα** (δηλαδή, εξασφαλίζεται ότι όλα τα συμπεράσματα θα μπορούν να υπολογιστούν και όλοι οι υπολογισμοί θα ολοκληρωθούν σε πεπερασμένο χρονικό διάστημα). Συγκεκριμένα, στην OWL DL τα σύνολα των object και datatype properties είναι ξένα μεταξύ τους, δεν επιτρέπονται περιορισμοί πληθικότητας (cardinality constraints) σε μεταβατικές ιδιότητες, το μεγαλύτερο μέρος του λεξιλογίου της RDF ή της RDF Schema δεν μπορεί να χρησιμοποιηθεί και τέλος, επιβάλλεται ο ρητός ορισμός όλων των κλάσεων και ιδιοτήτων που αναφέρονται στην οντολογία. Αυτοί οι περιορισμοί έχουν ως στόχο να παρέχουν στο δημιουργό ή χρήστη της οντολογίας συλλογιστική υποστήριξη. Το μειονέκτημα είναι ότι χάνεται η πλήρης συμβατότητα με την RDF, καθώς ένα RDF έγγραφο πρέπει να τροποποιηθεί με κάποιους τρόπους ώστε να γίνει ένα έγγραφο που υπόκειται στους κανόνες της OWL DL. Αντίστροφα, κάθε έγγραφο που είναι σύμφωνο με τους κανόνες της OWL DL είναι σύμφωνο και με τους κανόνες της RDF.

OWL Lite

Η OWL Lite είναι μια υπογλώσσα της OWL που προορίζεται για τους χρήστες που χρειάζονται κυρίως μια **ιεραρχία ταξινόμησης και απλούς περιορισμούς**. Στην OWL Lite, ισχύουν όλοι οι περιορισμοί που ισχύουν και στην OWL DL, χωρίς όμως να επιτρέπεται η χρήση των owl:oneOf, owl:UnionOf, owl:complementOf, owl:hasValue, owl:disjointWith και owl:DataRange. Επίσης, στην OWL Lite δεν επιτρέπεται η χρήση ανώνυμων κλάσεων στα δομικά στοιχεία owl:equivalentClass, rdfs:subClassOf, owl:intersectionOf, owl:allValuesFrom, rdf:type, rdfs:domain και rdfs:range, ενώ παράλληλα στους περιορισμούς πληθικότητας (cardinality) δεν επιτρέπει άλλες τιμές εκτός από 0 ή 1. Συνεπώς, η OWL Lite έχει σαφώς πιο εύκολη σύνταξη για το χρήστη και για τον κατασκευαστή μιας οντολογίας, αλλά έχει και μειωμένη εκφραστικότητα [13].

Μεταξύ των τριών αυτών γλωσσών, υπάρχουν αυστηρές έννοιες προς τα πάνω συμβατότητας και ισχύουν τα παρακάτω:

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

Δομικά Στοιχεία OWL Lite

owl:Class: Μια κλάση ορίζει μια ομάδα ατόμων (στιγμιοτύπων) που έχουν κοινές ορισμένες ιδιότητες. Υπάρχει μια προκατασκευασμένη γενική κλάση που ονομάζεται Thing και η οποία είναι η κλάση όλων των ατόμων και υπερκλάση όλων των OWL κλάσεων. Επίσης, υπάρχει και η κλάση Nothing, η οποία δεν περιέχει κανένα στιγμιότυπο και είναι υποκλάση όλων των OWL κλάσεων.

rdfs:subClassOf: Ιεραρχίες κλάσεων μπορούν να δημιουργηθούν πραγματοποιώντας μία ή περισσότερες δηλώσεις ότι μια κλάση είναι υποκλάση μιας άλλης.

rdf:Property: Οι ιδιότητες χρησιμοποιούνται για να δηλώσουν ιδιότητες μεταξύ ατόμων (στιγμιοτύπων) ή μεταξύ ατόμων και τιμών. Ιδιότητες που συνδέουν άτομα μεταξύ τους λέγονται object properties, ενώ ιδιότητες που συνδέουν άτομα με τιμές λέγονται datatype properties. Τόσο η owl:ObjectProperty όσο και η owl:DatatypeProperty είναι υποκλάσεις της rdf:Property.

rdfs:subPropertyOf: Ιεραρχίες ιδιοτήτων μπορούν να δημιουργηθούν κάνοντας μία ή περισσότερες δηλώσεις ότι μια ιδιότητα είναι υπο-ιδιότητα μιας ή περισσοτέρων ιδιοτήτων. Για παράδειγμα, η ιδιότητα «έχωΑδελφό» είναι υπο-ιδιότητα της ιδιότητας «έχωΣυγγενή».

rdfs:domain: Το πεδίο ορισμού μιας ιδιότητας περιορίζει τα στιγμιότυπα στα οποία μπορεί αυτή να εφαρμοστεί. Αν μια ιδιότητα συσχετίζει ένα στιγμιότυπο με ένα άλλο και η ιδιότητα έχει ως πεδίο ορισμού μια κλάση, τότε το πρώτο στιγμιότυπο πρέπει να ανήκει σε αυτήν την κλάση. Η rdfs:domain είναι ένας γενικός περιορισμός, καθώς χαρακτηρίζει γενικά μια ιδιότητα, και όχι ειδικά μια ιδιότητα όταν συσχετιστεί με μια συγκεκριμένη κλάση.

rdfs:range: Το πεδίο τιμών μιας ιδιότητας περιορίζει τα στιγμιότυπα που αυτή μπορεί να έχει ως τιμή. Αν μια ιδιότητα συσχετίζει ένα στιγμιότυπο με ένα άλλο και η ιδιότητα έχει ως πεδίο τιμών μια κλάση, τότε το δεύτερο στιγμιότυπο πρέπει να ανήκει σε αυτήν την κλάση. Η rdfs:range είναι επίσης ένας γενικός περιορισμός.

owl:Individual: Τα άτομα είναι στιγμιότυπα κλάσεων και οι ιδιότητες μπορούν να συσχετίσουν ένα άτομο με ένα άλλο.

owl:equivalentClass: Δύο κλάσεις μπορούν να δηλωθούν ότι είναι ισοδύναμες. Ισοδύναμες κλάσεις έχουν τα ίδια στιγμιότυπα. Η ισοδυναμία μπορεί να χρησιμοποιηθεί για να δημιουργηθούν συνώνυμες κλάσεις.

owl:equivalentProperty: Δύο ιδιότητες μπορούν να δηλωθούν ότι είναι ισοδύναμες. Ισοδύναμες ιδιότητες συνδέουν ένα άτομο με το ίδιο σύνολο ατόμων. Η ισοδυναμία μπορεί να χρησιμοποιηθεί για να δημιουργηθούν συνώνυμες ιδιότητες.

owl:sameAs: Δύο άτομα μπορούν να δηλωθούν ότι είναι τα ίδια. Αυτό το δομικό στοιχείο μπορεί να χρησιμοποιηθεί για να δημιουργηθεί ένας αριθμός διαφορετικών ονομάτων για το ίδιο άτομο.

owl:differentFrom: Ένα άτομο μπορεί να δηλωθεί ότι είναι διαφορετικό από ένα άλλο. Η ρητή δήλωση ότι δύο άτομα είναι διαφορετικά μπορεί να είναι πολύ σημαντική σε γλώσσες όπως η OWL και η RDF, οι οποίες δεν υποθέτουν ότι τα άτομα έχουν ένα και μοναδικό όνομα.

owl:AllDifferent: Ένας αριθμός ατόμων μπορούν να δηλωθούν ως διαφορετικά μεταξύ τους με τη χρήση μιας owl:AllDifferent δήλωσης. Το δομικό στοιχείο owl:AllDifferent χρησιμοποιείται όταν υπάρχουν σύνολα διαφορετικών αντικειμένων και οι κατασκευαστές μιας οντολογίας επιθυμούν να ενισχύσουν την υπόθεση μοναδικών ονομάτων σε αυτά τα σύνολα.

owl:InverseOf: Μια ιδιότητα μπορεί να δηλωθεί ως αντίστροφη μιας άλλης. Αν η ιδιότητα P1 είναι η αντίστροφη της P2, τότε αν το X συσχετίζεται με το Y μέσω της P1, τότε το Y συσχετίζεται με το X μέσω της P2.

owl:TransitiveProperty: Μια ιδιότητα μπορεί να δηλωθεί ως μεταβατική. Αν μια ιδιότητα είναι μεταβατική, τότε αν το ζεύγος (x,y) είναι στιγμιότυπο της μεταβατικής ιδιότητας P και το ζεύγος (y,z) είναι στιγμιότυπο της P, ισχύει ότι το ζεύγος (x,z) είναι επίσης στιγμιότυπο της P. Η OWL Lite επιβάλλει τη συνθήκη ότι οι μεταβατικές ιδιότητες και οι υπερ-ιδιότητές τους δεν μπορούν να έχουν έναν περιορισμό μέγιστης πληθικότητας (cardinality) 1. Χωρίς αυτή τη συνθήκη, δε θα μπορούσε να εξασφαλιστεί ότι όλοι οι υπολογισμοί θα ολοκληρωθούν σε πεπερασμένο χρονικό διάστημα.

owl:SymmetricProperty: Μια ιδιότητα μπορεί να δηλωθεί ως συμμετρική. Αν μια ιδιότητα είναι συμμετρική, τότε αν το ζεύγος (x,y) είναι στιγμιότυπο της συμμετρικής ιδιότητας P, ισχύει ότι και το ζεύγος (y,x) είναι στιγμιότυπο της P.

owl:FunctionalProperty: Μια ιδιότητα μπορεί να δηλωθεί ότι έχει μοναδική τιμή. Αν μια ιδιότητα είναι μονότιμη, τότε δεν έχει παραπάνω από μία τιμή για κάθε άτομο (μπορεί βέβαια, να μην έχει τιμή για κάποιο άτομο).

owl:InverseFunctionalProperty: Μια ιδιότητα μπορεί να δηλωθεί ως αντιστρόφως μονότιμη. Αν μια ιδιότητα είναι αντιστρόφως μονότιμη, αυτό σημαίνει ότι η αντίστροφή της είναι μονότιμη. Επομένως, η αντίστροφη αυτής της ιδιότητας έχει το πολύ μία τιμή για κάθε άτομο.

Επίσης, η OWL Lite επιτρέπει κάποιους περιορισμούς που μπορούν να καθορίσουν πώς οι ιδιότητες μπορούν να χρησιμοποιηθούν από στιγμιότυπα μιας κλάσης. Οι περιορισμοί αυτοί είναι:

owl:allValuesFrom: Ο περιορισμός allValuesFrom τίθεται σε μια ιδιότητα και αναφέρεται σε μία κλάση. Σημαίνει ότι η ιδιότητα στη συγκεκριμένη αυτή κλάση περιορίζεται από έναν περιορισμό πεδίου τιμών. Επομένως, αν ένα στιγμιότυπο της κλάσης σχετίζεται μέσω αυτής της ιδιότητας με ένα δεύτερο άτομο, τότε μπορεί να συναχθεί ότι το άτομο αυτό είναι στιγμιότυπο της κλάσης που αποτελεί τον περιορισμό του πεδίου τιμών. Για παράδειγμα, η κλάση Πρόσωπο μπορεί να έχει μια ιδιότητα που ονομάζεται «έχειΚόρη», η οποία περιορίζεται να έχει όλες τις τιμές της από την κλάση Γυναίκα.

owl:someValuesFrom: Ο περιορισμός someValuesFrom τίθεται σε μια ιδιότητα και αναφέρεται σε μία κλάση. Μια συγκεκριμένη κλάση μπορεί να έχει έναν περιορισμό που να δηλώνει ότι μια τουλάχιστον τιμή για μία ιδιότητά της πρέπει να είναι συγκεκριμένου τύπου. Σε αντίθεση με την allValuesFrom, η someValuesFrom δεν περιορίζει όλες τις τιμές της ιδιότητας να είναι στιγμιότυπα της ίδιας κλάσης.

owl:minCardinality: Ο περιορισμός ελάχιστης πληθικότητας τίθεται σε μια ιδιότητα και αναφέρεται σε μία κλάση. Αν τεθεί σε μια ιδιότητα ελάχιστη πληθικότητα 1, αυτό σημαίνει ότι κάθε στιγμιότυπο της κλάσης στην οποία έχει τεθεί αυτός ο περιορισμός θα πρέπει να έχει τουλάχιστον μία τιμή για αυτήν την ιδιότητα. Για παράδειγμα, η κλάση Πατέρας θα πρέπει να έχει ελάχιστη πολλαπλότητα τιμής 1 για την ιδιότητα «έχειΑπόγονο».

owl:maxCardinality: Ομοίως με παραπάνω, αυτός ο περιορισμός θέτει τη μέγιστη πληθικότητα για μια ιδιότητα σε μια συγκεκριμένη κλάση.

owl:cardinality: Αυτός ο περιορισμός θέτει ακριβώς την πληθικότητα που πρέπει να έχει μια ιδιότητα σε μια συγκεκριμένη κλάση.

Τέλος, η OWL Lite επιτρέπει και τη λογική πράξη της τομής (**owl:intersectionOf**), αλλά περιορίζει τη χρήση της, επιτρέποντας την τομή μόνο επώνυμων κλάσεων και περιορισμών.

Επιπλέον Δομικά Στοιχεία OWL DL και OWL Full

Τόσο η OWL DL όσο και η OWL Full χρησιμοποιούν το ίδιο λεξιλόγιο παρόλο που η OWL DL υπόκειται σε ορισμένους περιορισμούς. Χονδρικά, η OWL DL επιβάλλει χωρισμό τύπων (μια κλάση δεν μπορεί να είναι επίσης άτομο ή ιδιότητα και μια ιδιότητα δεν μπορεί επίσης να είναι άτομο ή κλάση). Οι OWL DL και OWL Full περιέχουν το λεξιλόγιο που έχει ήδη περιγραφεί στην OWL Lite, καθώς και τα παρακάτω δομικά στοιχεία:

owl:oneOf: Οι κλάσεις μπορούν να περιγραφούν ως απαρίθμηση των ατόμων που αποτελούν την κλάση. Τα μέλη της κλάσης είναι ακριβώς το σύνολο των απαριθμημένων ατόμων.

owl:hasValue: Μια ιδιότητα μπορεί να χρειάζεται να έχει ως τιμή ένα συγκεκριμένο άτομο.

owl:disjointWith: Δύο ή περισσότερες κλάσεις μπορούν να δηλωθούν ότι είναι ξένες μεταξύ τους, ότι δεν έχουν δηλαδή κοινά στοιχεία.

owl:unionOf, owl:complementOf, owl:intersectionOf: Οι OWL DL και OWL Full επιτρέπουν αυθαίρετους λογικούς συνδυασμούς κλάσεων και περιορισμών, με τη χρησιμοποίηση των μηχανισμών της ένωσης, του συμπληρώματος και της τομής.

owl:minCardinality, owl:maxCardinality, owl:cardinality: Ενώ στην OWL Lite, η πληθικότητα μιας ιδιότητας μπορούσε να τεθεί μόνο 0 ή 1, στις OWL DL και OWL Full επιτρέπονται δηλώσεις πληθικότητας με αυθαίρετους μη αρνητικούς ακεραίους.

Βέβαια, η OWL δεν αποτελεί την τελευταία λέξη στις οντολογικές γλώσσες του Σημασιολογικού Ιστού. Υπάρχει ήδη ένας σημαντικός αριθμός στοιχείων που έχουν αναγνωριστεί και των οποίων η προσθήκη βρίσκεται υπό συζήτηση από το W3C. Ένα από αυτά αφορά στην εισαγωγή οντολογιών σε άλλες, μια διαδικασία που θα είναι ο κανόνας στο Σημασιολογικό Ιστό. Όμως, η διαδικασία εισαγωγής που διαθέτει η OWL είναι πολύ απλοϊκή, καθώς επιτρέπει μόνο την εισαγωγή ολόκληρης οντολογίας. Ακόμα και αν κάποιος ήθελε να εισάγει μόνο ένα μικρό κομμάτι μιας οντολογίας, θα ήταν αναγκασμένος να την εισάγει ολόκληρη.

Άλλο ένα πρόβλημα της OWL είναι η υπόθεση μοναδικών ονομάτων ή μάλλον η έλλειψή της. Οι τυπικές εφαρμογές βάσεων δεδομένων υποθέτουν ότι άτομα με διαφορετικά ονόματα είναι διαφορετικά άτομα. Η OWL ακολουθεί το δρόμο της λογικής όπου κάτι τέτοιο δεν ισχύει, εφόσον μπορεί να εξαχθεί μέσω λογικών συνεπαγωγών ότι δύο άτομα με διαφορετικά ονόματα μπορεί να περιγράφουν ουσιαστικά το ίδιο άτομο. Η υπόθεση μη-μοναδικών ονομάτων, όπως είναι γνωστή αυτή η προσέγγιση, μπορεί να είναι αρκετά εύλογη στην περίπτωση του Παγκοσμίου Ιστού, όμως υπάρχουν δυστυχώς και περιπτώσεις όπου η υπόθεση μοναδικών ονομάτων μπορεί να αποδειχθεί χρήσιμη. Πιο συγκεκριμένα, μπορεί κάποιος να θέλει να δηλώσει μέρη της οντολογίας στα οποία ισχύει αυτή η υπόθεση και άλλα στα οποία δεν ισχύει.

Επίσης, μια κοινή αρχή στην Αναπαράσταση Γνώσης είναι ο ορισμός ενός όρου όχι μέσω ρητών δηλώσεων όπως στην OWL, αλλά με την προσκόλληση ενός κομματιού κώδικα ο οποίος θα πρέπει να εκτελεστεί για να υπολογιστεί το νόημα του όρου. Αυτή η αρχή δεν έχει συμπεριληφθεί στην OWL, όπως άλλωστε και η δυνατότητα σύνθεσης των ιδιοτήτων που σε πολλές εφαρμογές είναι μια χρήσιμη διαδικασία.

2.2.5 Γλώσσες Σημασιολογικών Ερωτημάτων

Μετά την ανάπτυξη και την καθιέρωση της RDF ως προτύπου, αναπτύχθηκαν αρκετές γλώσσες για την εκτέλεση ερωτημάτων σε ένα μοντέλο RDF.

RQL (RDF Query Language)

Η RQL [17] είναι μια γλώσσα ερωτημάτων για μοντέλα RDF. Είναι μια γλώσσα που ακολουθεί μια λειτουργική προσέγγιση και η οποία καθορίζεται από ομάδες βασικών ερωτημάτων και επαναλήψεων. Αυτά τα βασικά ερωτήματα είναι και τα δομικά στοιχεία αυτής της γλώσσας ερωτημάτων, τα οποία χρησιμοποιούνται προκειμένου να δημιουργηθούν περισσότερο πολύπλοκα ερωτήματα μέσω λειτουργικής σύνθεσης, διατηρώντας τους περιορισμούς ακεραιότητας τύπων που σχετίζονται με κάθε διαδικασία, επιτρέποντας με αυτόν τον τρόπο αυθαίρετη εμφώλευση σε ένα ερώτημα. Στη συνέχεια, παρατίθενται τα βασικά στοιχεία της γλώσσας RQL.

Το απλό ερώτημα Class επιστρέφει όλες τις κλάσεις του μοντέλου και αντίστοιχα, το ερώτημα Property επιστρέφει όλες τις ιδιότητες του μοντέλου. Για να επιστραφούν όλα τα στιγμιότυπα μιας κλάσης, έστω της κλάσης course, αρκεί το όνομα της κλάσης, δηλαδή να υποβληθεί το ερώτημα course. Αυτό το ερώτημα θα επιστρέψει και τα στιγμιότυπα των υποκλάσεων της course. Αν δε χρειάζεται να επιστραφούν τα κληρονομημένα στιγμιότυπα, τότε υποβάλλεται το ερώτημα ^course. Επίσης, όλοι οι πόροι και οι τιμές των τριάδων που σχετίζονται με μια συγκεκριμένη ιδιότητα μπορούν να βρεθούν με την πληκτρολόγηση του ονόματος της ιδιότητας, όπως και στην περίπτωση της κλάσης.

Όπως και στη γλώσσα υποβολής ερωτημάτων σε βάση δεδομένων SQL, η λέξη select καθορίζει τον αριθμό και τη σειρά των ανακτημένων δεδομένων, η λέξη from χρησιμοποιείται για την πλοήγηση μέσα στο μοντέλο και η λέξη where επιβάλλει περιορισμούς στα πιθανά αποτελέσματα. Για παράδειγμα, για να ανακτηθούν όλοι οι τηλεφωνικοί αριθμοί των μελών του πανεπιστημίου, θα υποβληθεί το ερώτημα:

```
select X, Y
from {X}phone{Y}
```

Εδώ τα X, Y είναι μεταβλητές και η έκφραση {X}phone{Y} παριστάνει μια τριάδα πόρου – ιδιότητας – τιμής. Για να ανακτηθούν όλοι οι τηλεφωνικοί αριθμοί των λεκτόρων του πανεπιστημίου, πρέπει να υποβληθεί το ερώτημα:

```
select X, Y
from lecturer{X}.phone{Y}
```

Εδώ, η έκφραση lecturer{X} συλλέγει όλα τα στιγμιότυπα της κλάσης lecturer και συνδέει το αποτέλεσμα με τη μεταβλητή X. Το δεύτερο μέρος συλλέγει όλες τις τριάδες με κατηγόρημα την ιδιότητα phone. Έτσι, υπάρχει εδώ μια συνεπαγόμενη ένωση (join), καθώς περιορίζεται το δεύτερο ερώτημα μόνο στις τριάδες των οποίων ο πόρος είναι η μεταβλητή X. Η τελεία είναι αυτή που δηλώνει τη συνεπαγόμενη ένωση.

Επίσης, η RQL επιτρέπει την ανάκτηση πληροφοριών σχετικά με το σχήμα του μοντέλου RDF. Οι μεταβλητές του σχήματος έχουν ένα όνομα με πρόθεμα $ για τις κλάσεις και πρόθεμα @ για τις ιδιότητες. Για παράδειγμα, το ερώτημα

```
select X, $X, Y, $Y
from {X:$X}phone{Y:$Y}
```

επιστρέφει όλους τους πόρους και τιμές των τριάδων που έχουν ως ιδιότητα (κατηγόρημα) τη phone (ή κάποια από τις υπο-ιδιότητές της), καθώς και τις κλάσεις στις οποίες αυτές ανήκουν. Το πεδίο ορισμού (domain) και το πεδίο τιμών (range) μιας ιδιότητας μπορούν να ανακτηθούν ως ακολούθως:

```
select domain(@P),range(@P)
from @P
where @P=phone
```

RDQL

Η RDQL [15] είναι ακόμα μία **γλώσσα υποβολής ερωτημάτων** για την RDF σε μοντέλα της Jena (API της Java για επεξεργασία μοντέλων RDF). Η ιδέα πίσω από την RDQL είναι η παροχή ενός προσανατολισμένου σε δεδομένα μοντέλου ερωτημάτων, έτσι ώστε να υπάρχει μια πιο δηλωτική προσέγγιση που θα συμπληρώσει τη Jena. Είναι προσανατολισμένη στα δεδομένα με την έννοια ότι χρησιμοποιεί μόνο την πληροφορία που αναφέρεται στα μοντέλα, χωρίς να κάνει καμία συνεπαγωγή. Το σύστημα της RDQL μπορεί να φαίνεται «έξυπνο», όμως στην πραγματικότητα το μόνο που κάνει είναι να δέχεται ως είσοδο το σημασιολογικό ερώτημα σε μια καθορισμένη μορφή και να επιστρέφει την αντίστοιχη πληροφορία, στη μορφή ενός συνόλου τριάδων.

Η RDQL είναι μια υλοποίηση της γλώσσας ερωτημάτων RDF SquishQL, η οποία με τη σειρά της προέρχεται από την rdfDB. Αυτή η ομάδα γλωσσών ερωτημάτων αντιμετωπίζει την RDF ως τριάδες δεδομένων, χωρίς πληροφορία για το σχήμα ή την οντολογία, εκτός και αν αυτά έχουν ρητά αναφερθεί στο μοντέλο RDF. Η RDF παρέχει ένα γράφο με κόμβους που είναι είτε πόροι είτε κυριολεκτικά. Η RDQL παρέχει έναν τρόπο καθορισμού ενός υποδείγματος του γράφου που συγκρίνεται με το γράφο για να προκύψει ένα σύνολο ταιριασμάτων.

Η σύνταξη της RDQL είναι παρόμοια με αυτή της SQL και συνεπώς, και με την αντίστοιχη της RQL, που παρουσιάστηκε παραπάνω. Έτσι, μετά τη λέξη select ακολουθούν οι μεταβλητές που θα επιστραφούν ως αποτέλεσμα μετά το ερώτημα, μετά τη λέξη from αναφέρεται το URI του RDF μοντέλου στο οποίο θα γίνει το ερώτημα, ενώ μετά τη λέξη where ακολουθεί ένα υπόδειγμα του γράφου σε μορφή τριάδων. Επίσης, υπάρχει η λέξη using η οποία χρησιμοποιείται προκειμένου να μικρύνει το μέγεθος των URIs και πρόκειται ουσιαστικά για έναν μηχανισμό συντομογραφίας που καθορίζει ένα πρόθεμα για τα μακροσκελή URIs. Ένα απλό παράδειγμα το οποίο βρίσκει τους πόρους που έχουν το κυριολεκτικό “John Smith” ως τιμή της ιδιότητας FN:

```
SELECT ?x
WHERE (?x  <http://www.w3.org/2001/vcard-rdf/3.0#FN>  "John Smith")
```

Βέβαια, ξεκινώντας από τα απλά ερωτήματα, μπορεί κανείς να υποβάλλει πιο πολύπλοκα ερωτήματα, όπως:

```
SELECT ?givenName
```

```
WHERE (?y  <http://www.w3.org/2001/vcard-rdf/3.0#Family>  "Smith") ,
    (?y  <http://www.w3.org/2001/vcard-rdf/3.0#Given>  ?givenName)
```

Εδώ χρησιμοποιούνται περισσότερες της μίας μεταβλητής προκειμένου να δημιουργηθεί ένας συσχετισμός με αποτέλεσμα το ερώτημα αυτό να επιστρέφει όλα τα μικρά ονόματα (given names) όσων ανήκουν στη Family Smith. Το πρώτο ερώτημα γίνεται πιο ευανάγνωστο με τη χρήση του using ως εξής:

```
SELECT ?x
WHERE (?x  vCard:FN  "John Smith")
USING vCard FOR <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Για ευκολία σημειώνεται ότι δε χρειάζεται να χρησιμοποιηθεί ο μηχανισμός using για τους χώρους ονομάτων rdf, rdfs, owl και xsd, τα οποία αναγνωρίζονται άμεσα από την RDQL.

OWL-QL

Η OWL-QL [16] είναι μια **γλώσσα υποβολής ερωτημάτων** για το Σημασιολογικό Ιστό (αποτελεί εξέλιξη της DQL (DAML Query Language)) και φιλοδοξεί να γίνει πρότυπη γλώσσα ερωτημάτων που θα χρησιμοποιείται σε διαλόγους ερωταποκρίσεων μεταξύ υπολογιστικών πρακτόρων που θα χρησιμοποιούν γνώση που θα αναπαρίσταται σε OWL οντολογίες. Η OWL-QL καθορίζει τις σημασιολογικές σχέσεις σε ένα ερώτημα, μια απάντηση στο ερώτημα καθώς και τις βάσεις γνώσης που χρησιμοποιήθηκαν για να δοθεί η απάντηση. Αντίθετα με τις συνηθισμένες γλώσσες σημασιολογικών ερωτημάτων, η OWL-QL υποστηρίζει διαλόγους ερωταποκρίσεων στους οποίους ο πράκτορας που απαντά μπορεί να χρησιμοποιήσει κάποιες αυτοματοποιημένες συλλογιστικές μεθόδους για να ανακτήσει απαντήσεις, καθώς και διαλόγους στους οποίους η γνώση που χρησιμοποιείται μπορεί να βρίσκεται σε πολλαπλές βάσεις γνώσης στο Σημασιολογικό Ιστό που δεν καθορίζονται στο ερώτημα. Σε μια τέτοια περίπτωση, το σύνολο των απαντήσεων θα έχει απροσδιόριστο μέγεθος και μπορεί να απαιτεί απροσδιόριστο χρόνο για να υπολογιστεί.

Μια βάση γνώσης της OWL είναι ένα σύνολο λογικών προτάσεων που αναπαριστούν μια λογική θεωρία. Είναι φυσικό λοιπόν, να θεωρηθεί το σημασιολογικό ερώτημα ως μια αναζήτηση των προτάσεων εκείνων που ανήκουν στη βάση γνώσης και «ικανοποιούν» ένα «σχήμα πρότασης», όπως και ο καθορισμός απαντήσεων σε αυτό το ερώτημα μπορεί να θεωρηθεί ως η σύνδεση κάποιων μεταβλητών με τις τιμές τους σε αυτήν την πρόταση. Ένα OWL-QL ερώτημα είναι ένα αντικείμενο που περιέχει απαραιτήτως ένα πρότυπο ερωτήματος που καθορίζει μια συλλογή OWL προτάσεων, στις οποίες κάποιες αναφορές URI θεωρούνται μεταβλητές. Για παράδειγμα, το ερώτημα “Who owns a red car?” θα είχε το παρακάτω πρότυπο ερωτήματος: {(owns ?p ?c) (type ?c Car) (has-color ?c Red)}, εφόσον στην OWL-QL οι τριάδες γράφονται στη μορφή (<ιδιότητα> <υποκείμενο> <αντικείμενο>) και κάθε μεταβλητή ξεκινά με «?». Συνοπτικά, η OWL-QL έχει σχεδιαστεί ώστε να απαντά σε ερωτήματα της μορφής «Ποιες αναφορές URI ή κυριολεκτικά υπάρχουν στη βάση γνώσης που δηλώνουν αντικείμενα που καθιστούν το πρότυπο του ερωτήματος αληθές;». Οι μεταβλητές που περιέχονται σε ένα ερώτημα διακρίνονται σε τρία είδη: αυτές που

πρέπει να συνδεθούν, αυτές που μπορούν να συνδεθούν και αυτές που δε θα συνδεθούν. Για παράδειγμα, έστω το παρακάτω ερώτημα: "{(hasFather ?p ?f)}" που σημαίνει "?p has father ?f". Αν η ?f είναι μια μεταβλητή που δε θα συνδεθεί, τότε η απάντηση στο παραπάνω ερώτημα αναφέρει όλα τα πρόσωπα, αλλά δεν αναφέρει τους πατέρες των προσώπων. Αντίθετα, αν η ?f είναι μια μεταβλητή που πρέπει να συνδεθεί, τότε η απάντηση περιέχει τόσο τα πρόσωπα όσο και τους πατέρες των προσώπων. Τέλος, αν η ?f είναι μεταβλητή που μπορεί να συνδεθεί, τότε η απάντηση περιέχει όλα τα πρόσωπα, καθώς και τους πατέρες των προσώπων, μόνο στην περίπτωση όμως που αυτοί είναι γνωστοί.

2.2.6 Εργαλεία Ανάπτυξης Οντολογιών (Ontology Development Tools)

Στην κατηγορία αυτή συμπεριλαμβάνονται εργαλεία, περιβάλλοντα και «σουίτες» που χρησιμοποιούνται για την ανάπτυξη μιας οντολογίας από την αρχή ή για την επαναχρησιμοποίηση κάποιας υπάρχουσας οντολογίας. Εκτός από την κλασική δυνατότητα αυτών των εργαλείων για σύνταξη (editing) και διάβασμα (browsing) οποιασδήποτε οντολογίας, τα εργαλεία αυτά προσφέρουν τεκμηρίωση (documentation) οντολογίας, εξαγωγή (exportation) και εισαγωγή (importation) από διάφορες τυποποιήσεις, γραφική αναπαράσταση της οντολογίας, βιβλιοθήκες οντολογιών κ.ά.

Παρακάτω ακολουθεί η σύνοψη κάποιων διαθέσιμων εργαλείων για την ανάπτυξη οντολογιών. Δίνεται μια σύντομη περιγραφή του κάθε εργαλείου, της ομάδας που το αναπτύσσει, καθώς και τα χαρακτηριστικά και οι βασικές λειτουργίες του καθενός [22].

2.2.6.1 Protégé

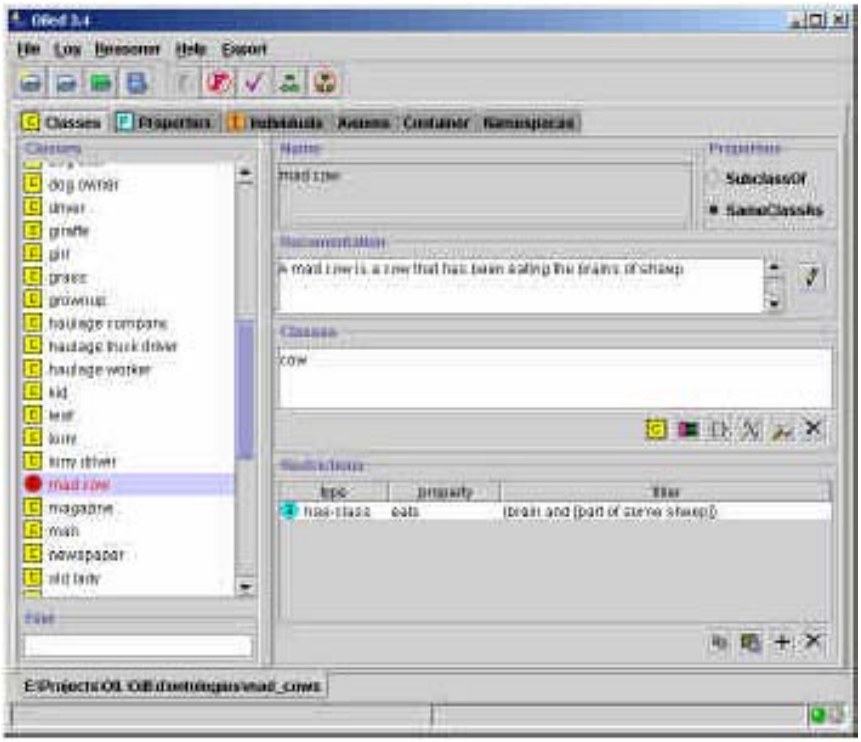
(Για αναλυτική παρουσίαση βλ. Κεφ 3.1)

2.2.6.2 OILEd

Το OILEd [23] είναι ένα γραφικό εργαλείο σύνταξης οντολογιών που αναπτύχθηκε στο πανεπιστήμιο του Manchester και επιτρέπει στους χρήστες να δημιουργήσουν οντολογίες χρησιμοποιώντας τη γλώσσα **DAML+OIL**.

Το μοντέλο γνώσης του OILEd βασίζεται σε αυτό της DAML+OIL και υποστηρίζει την πλούσια εκφραστικότητα του τελευταίου. Οι κλάσεις ορίζονται με βάση τις υπερκλάσεις τους και τους περιορισμούς ιδιοτήτων (property restrictions), έχοντας πρόσθετα αξιώματα όπως η μη επικάλυψη μεταξύ τους (disjointness). Αυτό το εκφραστικό μοντέλο γνώσης επιτρέπει τη χρήση πολύπλοκων σύνθετων περιγραφών για την πλήρωση ενδιάμεσων κενών (role fillers), σε αντίθεση με πολλά υπάρχοντα εργαλεία σύνταξης βασιζόμενα σε πλαίσια (frame-based editors), στα οποία τέτοια ανώνυμα πλαίσια πρέπει να ονοματιστούν πριν χρησιμοποιηθούν ως μοντέλα.

Το κύριο σημείο στο οποίο στοχεύει το OILEd είναι αυτό της σύνταξης οντολογιών ή σχημάτων, σε αντίθεση με την απόκτηση γνώσης ή την κατασκευή μεγάλων βάσεων γνώσης στιγμιότυπων.



Εικόνα 2.2 Στιγμιότυπο του OILEd

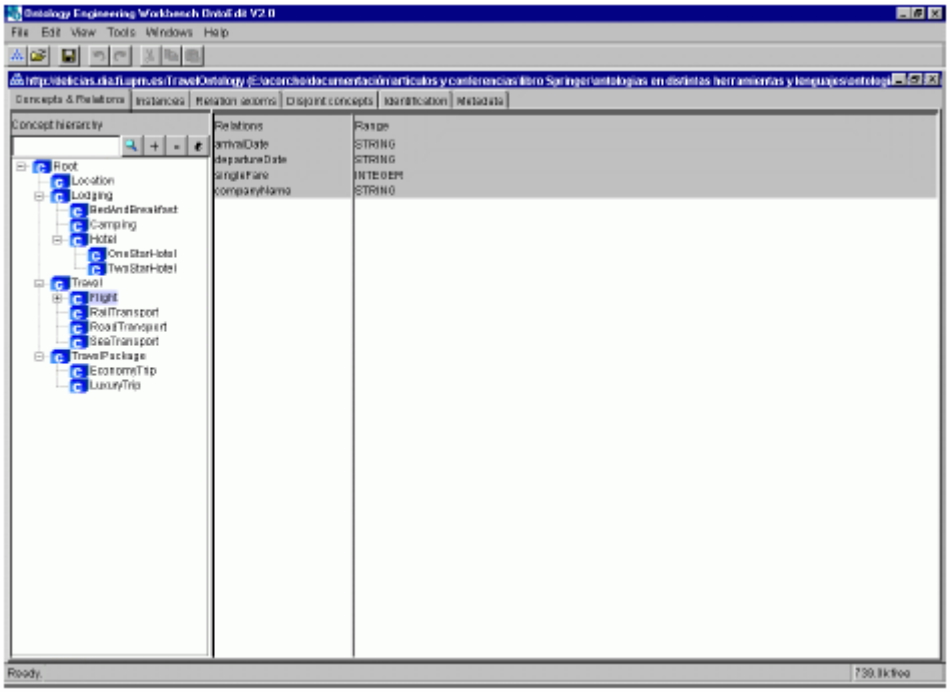
Το σημείο κλειδί στη συμπεριφορά του OILEd είναι η χρήση του FaCT reasoner [25] για την ταξινόμηση των οντολογιών και για τον έλεγχο συνέπειας μέσω μετάφρασης από DAML+OIL σε περιγραφική λογική SHIQ. Αυτό επιτρέπει στο χρήστη να περιγράψει τις κλάσεις της οντολογίας και να αφήσει τον reasoner να αποφασίσει την κατάλληλη θέση για τον ορισμό στην ιεραρχία [24].

Η τελευταία έκδοση του OILEd υλοποιείται σε Java και διατίθεται στην ιστοσελίδα ανάπτυξης του εργαλείου.

2.2.6.3 OntoStudio

Το OntoStudio [26] είναι ένα περιβάλλον διαχείρισης οντολογιών (Ontology Engineering Environment) το οποίο υποστηρίζει την ανάπτυξη και συντήρηση οντολογιών με γραφικό τρόπο και είναι χτισμένο πάνω σε ένα ισχυρό οντολογικό μοντέλο. Οι διάφορες γραφικές όψεις των δομών που περιέχονται στην οντολογία υποστηρίζουν τη μοντελοποίηση διαφορετικών φάσεων του κύκλου της οντολογικής διαχείρισης. Το εργαλείο επιτρέπει στο χρήστη να επέμβει στην ιεραρχία των εννοιών (concepts) ή κλάσεων (classes) όπως φαίνεται στο σχήμα που ακολουθεί. Οι έννοιες αυτές μπορεί να είναι αφηρημένες ή πραγματικές, κάτι που υποδεικνύει αν επιτρέπεται ή όχι να κατασκευαστούν απευθείας στιγμιότυπα της έννοιας. Μία έννοια (concept) μπορεί να έχει διάφορα ονόματα κάτι που ουσιαστικά αποτελεί έναν τρόπο για τον ορισμό συνωνυμιών για αυτή την έννοια. Για την αναδιοργάνωση των

εννοιών στην ιεραρχία το εργαλείο παρέχει επίσης τη δυνατότητα «αντιγραφής – επικόλλησης» (copy-paste).

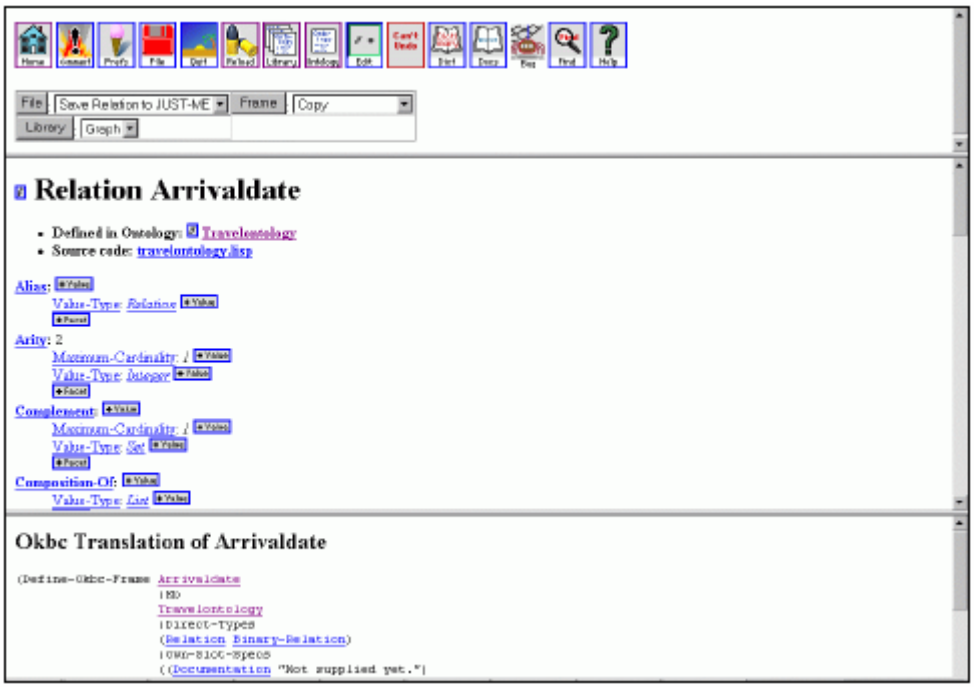


Εικόνα 2.3 Στιγμιότυπο του OntoEdit

Το OntoStudio είναι ο διάδοχος του πετυχημένου εργαλείου OntoEdit και βασίζεται στο αναπτυξιακό περιβάλλον Eclipse. Το εργαλείο OntoEdit βασίζεται σε ένα ευέλικτο πλαίσιο εργασίας με επεκτάσεις (plugins), κάτι που επιτρέπει την επέκταση της λειτουργικότητάς του από τρίτους. Έτσι, αποτελούμενο από ένα ευρύ σύνολο διαθέσιμων επεκτάσεων, το OntoEdit μπορεί, όντας φιλικό προς το χρήστη, να υιοθετείται για πολλά σενάρια χρήσης [27][28][29].

2.2.6.4 Ontolingua Server

Το Ontolingua Server [30] είναι ένα σύνολο εργαλείων και υπηρεσιών που υποστηρίζουν την ανάπτυξη οντολογιών μεταξύ κατανεμημένων ομάδων και αναπτύσσεται από το εργαστήριο συστημάτων γνώσης (Knowledge Systems Laboratory) στο πανεπιστήμιο του Stanford. Η αρχιτεκτονική του οντολογικού εξυπηρετητή παρέχει πρόσβαση σε μια βιβλιοθήκη οντολογιών, μεταφραστών σε γλώσσες (Prolog, CORBA IDL, CLIPS, Loom κ.ά.) και ένα εργαλείο σύνταξης (editor) για τη δημιουργία και τη διαχείριση οντολογιών. Οι απομακρυσμένοι συντάκτες (remote editors) μπορούν να περιηγηθούν και να επεξεργαστούν οντολογίες, ενώ οι απομακρυσμένες ή οι τοπικές εφαρμογές μπορούν να έχουν πρόσβαση σε οποιαδήποτε από τις οντολογίες της βιβλιοθήκης με τη χρήση του πρωτοκόλλου OKBC (Open Knowledge Based Connectivity) [31].

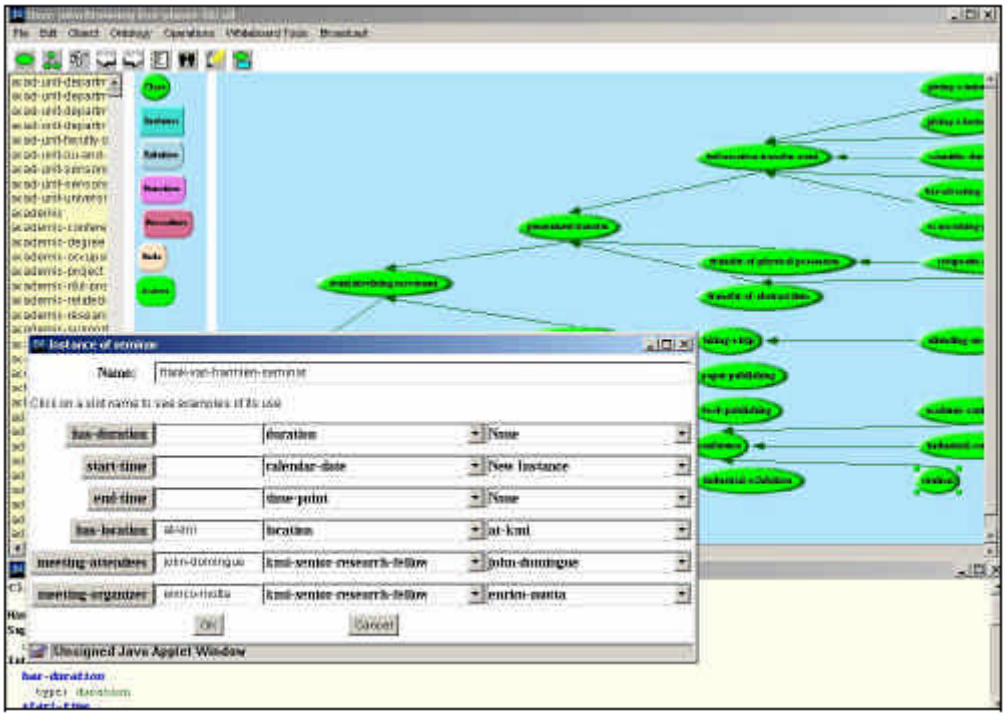


Εικόνα 2.4 Στιγμιότυπο του Ontolingua Server

2.2.6.5 WebOnto

Το WebOnto [32] είναι ένα εργαλείο που αναπτύσσεται από το Knowledge Media Institute (KMI) του Open University (England). Υποστηρίζει τη συνεργατική περιήγηση (browsing), δημιουργία και επεξεργασία οντολογιών οι οποίες αναπαρίστανται στην γλώσσα μοντελοποίησης γνώσης OCML. Τα κύρια χαρακτηριστικά του εργαλείου WebOnto είναι η διαχείριση οντολογιών μέσω γραφικού περιβάλλοντος, η αυτόματη γέννηση στιγμιότυπων από τον ορισμό των κλάσεων, η επιθεώρηση (inspection) των στοιχείων λαμβάνοντας υπόψη την κληρονομικότητα των ιδιοτήτων και τον έλεγχο συνέπειας και η διεπαφή για την υποστήριξη συνεργατικής εργασίας (broadcast/receive).

Το WebOnto είναι μια υπηρεσία που διατίθεται στην κοινότητα ενώ υπάρχει και μια βιβλιοθήκη με πάνω από 100 οντολογίες που είναι προσβάσιμες μέσω αυτού.



Εικόνα 2.5 Στιγμιότυπο του WebOnto

2.3 ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ

2.3.1 Εισαγωγή

Οι βάσεις δεδομένων [33] και η τεχνολογία βάσεων δεδομένων εξασκούν σημαντική επίδραση στην αυξανόμενη χρήση των υπολογιστών και είναι εύλογο να ειπωθεί ότι θα διαδραματίσουν κρίσιμο ρόλο σε όλες τις περιοχές όπου χρησιμοποιούνται υπολογιστές.

Μια βάση δεδομένων έχει τις ακόλουθες ιδιότητες:

- Αναπαριστά κάποια άποψη του πραγματικού κόσμου η οποία μερικές φορές λέγεται μικρόκοσμος (miniworld) ή πεδίο αναφοράς (Universe of Discourse, UoD). Οι αλλαγές στον μικρόκοσμο αντανακλώνται στη βάση δεδομένων.
- Είναι μια λογικά συνεκτική συλλογή δεδομένων που έχει κάποια εγγενή σημασία. Μια τυχαία διευθέτηση δεδομένων δεν είναι σωστό να αναφέρεται ως βάση δεδομένων.
- Σχεδιάζεται, χτίζεται και γεμίζει με δεδομένα για κάποιο συγκεκριμένο σκοπό. Προορίζεται για μια συγκεκριμένη ομάδα χρηστών και για κάποιες προκαθορισμένες εφαρμογές για τις οποίες οι χρήστες αυτοί ενδιαφέρονται.

Με άλλα λόγια, μια βάση δεδομένων έχει κάποια πηγή από την οποία παράγονται τα δεδομένα, αλληλεπιδρά σε κάποιο βαθμό με τα γεγονότα του πραγματικού κόσμου και απευθύνεται σε χρήστες που ενδιαφέρονται ενεργά για το περιεχόμενό της. Μπορεί να έχει οποιοδήποτε μέγεθος και κυμαινόμενη πολυπλοκότητα.

Η αλματώδης ανάπτυξη της επιστήμης της πληροφορικής και των επικοινωνιών τα τελευταία χρόνια έχει καταστήσει την πληροφορία ως ένα από τα βασικότερα και πολυτιμότερα αγαθά. Είναι κοινός τόπος σήμερα η εκτίμηση ότι το αγαθό της πληροφορίας είναι επιθυμητό από όλους τους εργαζόμενους αλλά και τους εκπαιδευόμενους, ώστε να είναι πιο αποδοτικοί, ανταγωνιστικοί αλλά και παραγωγικοί στην εργασία τους.

Τα συστήματα βάσεων δεδομένων τα χρησιμοποιούμε για να μπορούμε να αποθηκεύσουμε, να επεξεργαστούμε αλλά και να εκμεταλλευτούμε αποδοτικά αυτόν τον τεράστιο όγκο των πληροφοριών που αυξάνονται με αλματώδεις ρυθμούς καθημερινά.

2.3.2 Βασικές Γνώσεις

Τα Δεδομένα και οι Πληροφορίες

Με τον όρο **πληροφορία** αναφερόμαστε συνήθως σε ειδήσεις, γεγονότα και έννοιες που αποκτάμε από την καθημερινή μας επικοινωνία και τα θεωρούμε ως αποκτηθείσα γνώση, ενώ τα δεδομένα μπορούν να είναι μη κατάλληλα επεξεργασμένα και μη ταξινομημένα σύνολα πληροφοριών. Ένας αυστηρός ορισμός για το τι είναι δεδομένα και τι είναι πληροφορία, σύμφωνα με την επιτροπή ANSI των ΗΠΑ, είναι ο εξής :

- **Δεδομένα (data)** είναι μια παράσταση, όπως γράμματα, αριθμοί, σύμβολα κ.ά. στα οποία μπορούμε να δώσουμε κάποια σημασία (έννοια).
- **Πληροφορία (information)** είναι η σημασία που δίνουμε σ' ένα σύνολο από δεδομένα, τα οποία μπορούμε να επεξεργαστούμε βάσει προκαθορισμένων κανόνων και να βγάλουμε έτσι κάποια χρήσιμα συμπεράσματα. Με τις πληροφορίες περιορίζεται η αβεβαιότητα που έχουμε για διάφορα πράγματα και βοηθούμαστε έτσι στο να λάβουμε σωστές αποφάσεις.

Τα δεδομένα μπορούν να θεωρηθούν ως τρόποι αναπαράστασης εννοιών και γεγονότων που μπορούν να υποστούν διαχείριση και επεξεργασία. Η συλλογή και αποθήκευση ενός τεράστιου όγκου δεδομένων όπως απαιτούν οι κοινωνικές συνθήκες σήμερα, δεν λύνει τελείως το πρόβλημα της σωστής οργάνωσης και ταξινόμησης των δεδομένων. Τα δεδομένα θα πρέπει να οργανωθούν με τέτοιο τρόπο έτσι ώστε να μπορούμε να τα εντοπίζουμε και να τα αξιοποιούμε εύκολα και γρήγορα και τη στιγμή που τα χρειαζόμαστε.

Η Οργάνωση Αρχείων

Ο πιο γνωστός τρόπος οργάνωσης δεδομένων με τη χρήση ηλεκτρονικών υπολογιστών είναι σε αρχεία εγγραφών. Ένα **αρχείο (file)** μπορούμε να το χαρακτηρίσουμε σαν ένα σύνολο που αποτελείται από οργανωμένα ομοειδή στοιχεία. Τα στοιχεία ενός αρχείου μπορούμε να τα οργανώσουμε σε λογικές ενότητες και το σύνολο των στοιχείων που περιέχει μια λογική ενότητα καλείται **εγγραφή (record)**. Το κάθε στοιχείο της εγγραφής καλείται **πεδίο (field)**. Το πεδίο αποτελεί και τη μικρότερη δυνατή υποδιαίρεση των στοιχείων ενός αρχείου.

Στα αρχικά στάδια της οργάνωσης αρχείων, ήταν πολύ συνηθισμένη πρακτική η δημιουργία ξεχωριστών εφαρμογών (προγραμμάτων) και ξεχωριστών αρχείων, όπως για παράδειγμα η δημιουργία ενός αρχείου πελατών και ενός άλλου ανεξάρτητου αρχείου για τις παραγγελίες των πελατών. Τα προβλήματα που προέκυψαν από την πρακτική αυτή είναι τα εξής :

- **Πλεονασμός των δεδομένων (data redundancy)**. Υπάρχει η περίπτωση να έχουμε επανάληψη των ίδιων δεδομένων σε αρχεία διαφορετικών εφαρμογών.
- **Ασυνέπεια των δεδομένων (data inconsistency)**. Αυτό μπορεί να συμβεί όταν υπάρχουν τα ίδια στοιχεία (πλεονασμός) σε δύο αρχεία, οπότε είναι πολύ πιθανό να γίνει μια διόρθωση μόνο στο ένα αρχείο και όχι στο άλλο.
- **Αδυναμία μερισμού δεδομένων (data sharing).** Μερισμός δεδομένων σημαίνει δυνατότητα για κοινή χρήση των στοιχείων κάποιων αρχείων. Η αδυναμία μερισμού δεδομένων δημιουργεί καθυστέρηση στη λήψη αποφάσεων και στην εξυπηρέτηση των χρηστών.
- **Αδυναμία προτυποποίησης**. Έχει να κάνει με την ανομοιομορφία και με τη διαφορετική αναπαράσταση και οργάνωση των δεδομένων στα αρχεία των εφαρμογών. Η αδυναμία αυτή δημιουργεί προβλήματα προσαρμογής των χρηστών καθώς και προβλήματα στην ανταλλαγή δεδομένων μεταξύ διαφορετικών συστημάτων.

Οι Βάσεις Δεδομένων και τα ΣΔΒΔ (DBMS)

Για να δοθεί μια λύση σε όλα τα παραπάνω προβλήματα, και με βάση το γεγονός ότι η χρήση των ηλεκτρονικών υπολογιστών και συνεπώς η ηλεκτρονική καταχώρηση και επεξεργασία δεδομένων αυξήθηκε κατακόρυφα ήδη από τη δεκαετία του ’70 στις μεγάλες επιχειρήσεις και άρα είχαμε πάρα πολλές εφαρμογές να επεξεργάζονται δεδομένα σε πάρα πολλά αρχεία ταυτόχρονα, προτάθηκε η συνένωση όλων των αρχείων μιας εφαρμογής. Εκτός, όμως, από τη συνένωση των αρχείων, απαιτείτο και μια σωστή οργάνωσή τους. Δημιουργήθηκαν έτσι οι Τράπεζες Πληροφοριών ή Βάσεις Δεδομένων (DataBases).

Μια **Βάση Δεδομένων (ΒΔ)** είναι ένα σύνολο αρχείων με υψηλό βαθμό οργάνωσης τα οποία είναι συνδεδεμένα μεταξύ τους με λογικές σχέσεις, έτσι ώστε να μπορούν να χρησιμοποιούνται από πολλές εφαρμογές και από πολλούς χρήστες ταυτόχρονα. Υπάρχει ένα ειδικό λογισμικό το οποίο μεσολαβεί ανάμεσα στις αρχεία δεδομένων και τις εφαρμογές που χρησιμοποιούν οι χρήστες και αποκαλείται **Σύστημα Διαχείρισης Βάσης Δεδομένων (ΣΔΒΔ)** ή **DBMS (Data Base Management System)**. Το ΣΔΒΔ είναι στην ουσία ένα σύνολο από προγράμματα και υπορουτίνες

που έχουν να κάνουν με τον χειρισμό της βάσης δεδομένων, όσον αφορά στη δημιουργία, τροποποίηση, διαγραφή στοιχείων, με ελέγχους ασφαλείας κ.ά.

Οι χρήστες των εφαρμογών αντλούν τα στοιχεία που τους ενδιαφέρουν από τη βάση δεδομένων χωρίς να είναι σε θέση να γνωρίζουν με ποιον τρόπο είναι οργανωμένα τα δεδομένα σε αυτήν. Το ΣΔΒΔ παίζει τον ρόλο του μεσάζοντα ανάμεσα στο χρήστη και τη βάση δεδομένων και μόνο μέσω του ΣΔΒΔ μπορεί ο χρήστης να αντλήσει πληροφορίες από τη βάση δεδομένων. Ένα ΣΔΒΔ μπορεί να είναι εγκατεστημένο σε ένα μόνο υπολογιστή ή και σε ένα δίκτυο υπολογιστών και μπορεί να χρησιμοποιείται από έναν χρήστη ή και από πολλούς χρήστες.

Ένα **Σύστημα Βάσης Δεδομένων (ΣΒΔ)** ή **DBS (Data Base System)** αποτελείται από το υλικό, το λογισμικό, τη βάση δεδομένων και τους χρήστες. Είναι δηλαδή ένα σύστημα με το οποίο μπορούμε να αποθηκεύσουμε και να αξιοποιήσουμε δεδομένα με τη βοήθεια ηλεκτρονικού υπολογιστή.

Οι Οντότητες (Entities)

Με τον όρο **οντότητα (entity)** εννοούμε ένα αντικείμενο, ένα πρόσωπο, μια κατάσταση και γενικά οτιδήποτε μπορεί να προσδιορισθεί σαν ανεξάρτητη ύπαρξη (αυτόνομη μονάδα του φυσικού κόσμου).

Το **Μοντέλο Οντοτήτων Συσχετίσεων (Entity Relationship Model, ER Model)** είναι μια διαγραμματική αναπαράσταση της δομής μιας βάσης δεδομένων και χρησιμοποιείται κατά τη φάση του λογικού σχεδιασμού της βάσης. Δηλαδή, δεν ασχολείται με τον τρόπο που αποθηκεύονται τα δεδομένα της βάσης, αλλά με την ταυτοποίηση των δεδομένων και με τον τρόπο με τον οποίο αυτά συσχετίζονται μεταξύ τους.

Οι Ιδιότητες (Attributes)

Με τον όρο **ιδιότητα** ή **χαρακτηριστικό** ή και **πεδίο (attribute)** μιας οντότητας, αναφερόμαστε σε ένα από τα συστατικά της στοιχεία που την περιγράφουν και την κάνουν να ξεχωρίζει από τα άλλα στοιχεία της ίδιας οντότητας.

Το Πρωτεύον Κλειδί (Primary Key)

Πρωτεύον κλειδί (primary key) μιας οντότητας καλείται εκείνη η ιδιότητα (ή ο συνδυασμός ιδιοτήτων) που έχει μοναδική τιμή για όλα τα στιγμιότυπα της οντότητας.

Υπάρχουν περιπτώσεις όπου το πεδίο κλειδί ενός τύπου οντότητας μπορεί να μην είναι απλό αλλά σύνθετο, να αποτελείται δηλαδή από πολλά απλά πεδία και τότε η συνθήκη της μοναδικότητας για την τιμή του κλειδιού δεν εφαρμόζεται σε κάθε πεδίο του σύνθετου κλειδιού αλλά στο σύνολο του συνδυασμού αυτών των πεδίων.

Οι Συσχετίσεις (Relationships)

Με τον όρο **συσχέτιση (relationship)** αναφερόμαστε στον τρόπο σύνδεσης (επικοινωνίας) δύο ξεχωριστών οντοτήτων, ώστε να μπορούμε να αντλούμε στοιχεία (πληροφορίες) από τον συνδυασμό τους.

Το Ιεραρχικό Μοντέλο Βάσεων Δεδομένων

Υπάρχουν τρία βασικά μοντέλα που έχουν επικρατήσει στις βάσεις δεδομένων, το ιεραρχικό, το δικτυωτό και το σχεσιακό, και τα οποία αναπτύχθηκαν με βάση αντίστοιχες δομές. Το ιεραρχικό μοντέλο (hierarchical) έχει μια ιεραρχική δομή που θυμίζει δένδρο. Οι οντότητες μοιάζουν με απολήξεις από κλαδιά δένδρων και τοποθετούνται σε επίπεδα ιεραρχίας. Τα κλαδιά παριστάνουν τις συσχετίσεις ανάμεσα στις οντότητες.

Από μια οντότητα που βρίσκεται σε ένα ανώτερο επίπεδο εκκινούν πολλά κλαδιά, καθένα από τα οποία καταλήγει σε μια οντότητα που βρίσκεται σε ένα χαμηλότερο επίπεδο. Αλλά, σε κάθε οντότητα που βρίσκεται σε ένα χαμηλότερο επίπεδο αντιστοιχεί μία και μόνο μία οντότητα που βρίσκεται σε ένα ανώτερο επίπεδο. Το μοντέλο αυτό ήταν το πρώτο που εμφανίσθηκε αλλά σήμερα θεωρείται δύσχρηστο και ξεπερασμένο.

Το Δικτυωτό Μοντέλο Βάσεων Δεδομένων

Στο δικτυωτό (network) μοντέλο, τα στοιχεία τοποθετούνται σε ένα επίπεδο ιεραρχίας, αλλά κάθε στοιχείο μπορεί να συσχετισθεί με πολλά στοιχεία είτε σε ένα κατώτερο ή σε ένα ανώτερο επίπεδο.

Το Σχεσιακό Μοντέλο Βάσεων Δεδομένων

Το σχεσιακό (relational) μοντέλο έχει επικρατήσει σήμερα στην αναπαράσταση των δεδομένων καθώς διαθέτει σημαντικά πλεονεκτήματα ως προς τα άλλα δύο και οι βάσεις δεδομένων που σχεδιάζονται σύμφωνα με αυτό αποκαλούνται σχεσιακές (relational databases). Με τις σχεσιακές βάσεις δεδομένων διαθέτουμε ένα σαφή, απλό και εύκολα κατανοητό τρόπο για να μπορέσουμε να αναπαραστήσουμε και να διαχειριστούμε τα δεδομένα μας. Υστερούν μόνο σε ταχύτητα υπολογισμών και σε χώρο αποθήκευσης, αλλά μόνο όταν έχουμε να κάνουμε με πολύ μεγάλες βάσεις δεδομένων.

Στο μοντέλο αυτό οι βάσεις δεδομένων περιγράφονται με αυστηρές μαθηματικές έννοιες και ο χρήστης βλέπει τις οντότητες και τις συσχετίσεις με τη μορφή πινάκων (tables) και σχέσεων (relations) αντίστοιχα.

Ένας **πίνακας (table)** αποτελείται από γραμμές (rows) και στήλες (columns), όπου τοποθετούμε τα στοιχεία σε οριζόντια και κάθετη μορφή. Η κάθε στήλη του πίνακα χαρακτηρίζει κάποια ιδιότητα της οντότητας και αποκαλείται **χαρακτηριστικό (attribute)** ή **πεδίο (field),** ενώ η κάθε γραμμή του πίνακα περιέχει όλες τις

πληροφορίες (στήλες) που αφορούν ένα στοιχείο της οντότητας και αποκαλείται **πλειάδα (tuple)** ή **εγγραφή (record).**

Κάθε πεδίο του πίνακα μπορεί να πάρει ορισμένες μόνο τιμές, οι οποίες μπορεί να καθορίζονται από τον τύπο δεδομένων της ιδιότητας, όπως ονόματα ή αριθμοί για παράδειγμα, ή και από αυτό που εκφράζει, όπως το ότι δεν μπορούμε να έχουμε αρνητικό βάρος ή αρνητικό ΑΦΜ, για παράδειγμα. Το σύνολο των αποδεκτών τιμών μιας οντότητας αποκαλείται **πεδίο ορισμού (domain)**.

Η βασικότερη εργασία που έχουμε να κάνουμε κατά τον σχεδιασμό μιας σχεσιακής βάσης δεδομένων είναι να ορίσουμε τους πίνακες που θα χρησιμοποιήσουμε καθώς και τα πεδία που θα περιέχει ο καθένας απ' αυτούς. Η διαδικασία αυτή αποκαλείται κατασκευή του **σχήματος (schema)** μιας βάσης δεδομένων.

Οι κανόνες που πρέπει να ακολουθούνται πιστά κατά τον σχεδιασμό μιας σχεσιακής βάσης δεδομένων είναι οι εξής :

- Η κάθε οντότητα πρέπει να παριστάνεται ως ένας ξεχωριστός πίνακας.
- Η κάθε στήλη του πίνακα αντιστοιχεί σε μια ιδιότητα της οντότητας.
- Η κάθε γραμμή του πίνακα αντιστοιχεί σε μια εμφάνιση της οντότητας.
- Η κάθε γραμμή πρέπει να είναι μοναδική, δηλαδή αποκλείεται να υπάρχουν δύο ή και περισσότερες γραμμές που να περιέχουν τα ίδια ακριβώς στοιχεία.
- Η σειρά εμφάνισης των γραμμών δεν έχει καμία σημασία.
- Η κάθε στήλη έχει μια δική της μοναδική ονομασία.
- Οι τιμές που ανήκουν στην ίδια στήλη πρέπει να είναι του ιδίου τύπου.
- Η στήλη που αποτελεί το πρωτεύον κλειδί (primary key) μιας οντότητας, δεν πρέπει να είναι ποτέ κενή (null).
- Αποκλείεται να υπάρχουν δύο ή και περισσότερες γραμμές που να περιέχουν την ίδια τιμή στο πρωτεύον κλειδί.
- Το πρωτεύον κλειδί μιας οντότητας αποκαλείται ξένο κλειδί (foreign key) σε μια άλλη οντότητα, με την οποία υπάρχει συσχετισμός.
- Μπορεί να υπάρχουν πολλές γραμμές που να έχουν την ίδια τιμή στο ξένο κλειδί.

Τα Σχεσιακά ΣΔΒΔ (RDBMS)

Τα Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΣΔΒΔ) ή RDBMS (Relational DataBase Management Systems) αναπτύχθηκαν με βάση το σχεσιακό μοντέλο και έχουν επικρατήσει πλήρως στον χώρο. Κατά το σχεδιασμό και τη δημιουργία μιας σχεσιακής βάσης δεδομένων, οι πίνακες αποτελούν το μοναδικό δομικό και απαραίτητο στοιχείο για μπορέσουν να αναπαρασταθούν οι πληροφορίες που περιέχονται στη βάση δεδομένων.

Για να μπορέσουμε να προσθέσουμε, διαγράψουμε ή τροποποιήσουμε τα στοιχεία που περιέχονται σε μια βάση δεδομένων, χρησιμοποιούμε ειδικές γλώσσες προγραμματισμού που αποκαλούνται **γλώσσες ερωταπαντήσεων (query**

languages). Η γλώσσα που αποτελεί σήμερα ένα διεθνές πρότυπο για την επικοινωνία των χρηστών με τα Σχεσιακά ΣΔΒΔ είναι η **SQL (Structured Query Language)** ή **Δομημένη Γλώσσα Ερωτημάτων**. Μπορεί να λειτουργήσει αυτόνομα αλλά και σε συνεργασία με άλλες γλώσσες προγραμματισμού.

Τα Σχεσιακά ΣΔΒΔ τα διακρίνουμε στα μεγάλα, τα οποία αφορούν κυρίως μεγάλους οργανισμούς και επιχειρήσεις, έχουν τεράστιο όγκο δεδομένων και πολλούς χρήστες ταυτόχρονα, και τέτοια συστήματα είναι τα Oracle, Ingres, Informix, SQL Server κ.ά. και τα μικρά, τα οποία αφορούν κυρίως απλούς χρήστες, όπως είναι η Microsoft Access, η Paradox, η FoxPro κ.ά.

Τα Κλειδιά

Όπως είδαμε και νωρίτερα, με τον όρο **κλειδί (key)** ή πιο σωστά **πρωτεύον κλειδί (primary key)** αναφερόμαστε σε μια ιδιότητα (πεδίο), ή σπανιότερα σ' ένα σύνολο ιδιοτήτων (πεδίων), η τιμή της οποίας είναι μοναδική σ' ολόκληρη την οντότητα (πίνακας). Στην πράξη, το πρωτεύον κλειδί έχει διαφορετική τιμή για κάθε εμφάνιση της οντότητας ή για κάθε γραμμή (εγγραφή) του πίνακα και ποτέ δεν μπορεί να έχει μηδενική (κενή) τιμή (null).

Ο συνδυασμός δύο ή και περισσοτέρων ιδιοτήτων (πεδίων) για τη δημιουργία ενός πρωτεύοντος κλειδιού αποκαλείται **σύνθετο κλειδί**.

Ξένο κλειδί αποκαλείται μια ιδιότητα (πεδίο) που είναι πρωτεύον κλειδί σε μια οντότητα (πίνακας) αλλά που υπάρχει και σε μια άλλη οντότητα (πίνακας) σαν απλή ιδιότητα. Τα ξένα κλειδιά είναι απαραίτητα για να μπορέσουμε να κάνουμε τις συσχετίσεις (συνδέσεις, επικοινωνίες) ανάμεσα στις οντότητες (πίνακες).

2.3.3 Η Γλώσσα SQL

2.3.3.1 Εισαγωγή

Η γλώσσα **SQL (Structured Query Language)** είναι η πιο διαδεδομένη διαλογική γλώσσα ερωταπαντήσεων που χρησιμοποιείται για την επικοινωνία του χρήστη με σχεσιακές ΒΔ. Πρόκειται για μία μη-διαδικαστική γλώσσα τέταρτης γενιάς, στην οποία ο χρήστης διατυπώνει διάφορα αιτήματα και το ΣΔΒΔ αναλαμβάνει να τα ικανοποιήσει. Η SQL δίνει τη δυνατότητα στο χρήστη να δημιουργήσει, να τροποποιήσει και να ενημερώσει τους πίνακες της βάσης, καθώς και να αναζητήσει πληροφορίες από τη βάση εφαρμόζοντας σύνθετα κριτήρια αναζήτησης.

Η SQL αποτελείται από δύο υποσύνολα, τη DDL και τη DML. Η **DDL (Data Definition Language)** είναι μια γλώσσα ορισμού δεδομένων και αποτελείται από τις εντολές με τις οποίες καθορίζουμε τη λογική οργάνωση των δεδομένων της βάσης, δηλαδή δημιουργούμε τους πίνακες και τις μεταξύ τους σχέσεις. Η **DML (Data Manipulation Language)** είναι μια γλώσσα χειρισμού δεδομένων και αποτελείται

από τις εντολές με τις οποίες ενημερώνουμε τα δεδομένα της βάσης και δημιουργούμε ερωτήματα για ανάκληση πληροφοριών από τη βάση.

2.3.3.2 Διαχείριση της Δομής της Βάσης Δεδομένων

Με τις εντολές που ακολουθούν διαχειριζόμαστε τη δομή της ΒΔ, δηλαδή δημιουργούμε, τροποποιούμε και διαγράφουμε τους πίνακες, τα πεδία και τις μεταξύ τους συσχετίσεις. Είναι προτιμότερο, τέτοιου είδους εντολές να χρησιμοποιούνται μόνο κατά τον αρχικό σχεδιασμό, όταν η βάση είναι κενή δεδομένων. Αν είναι απαραίτητο να γίνουν αλλαγές στη δομή κατά τη διάρκεια της λειτουργίας της βάσης, αυτές πρέπει να γίνουν με ιδιαίτερη προσοχή.

Δημιουργία πίνακα

Για να δημιουργήσουμε τη δομή ενός καινούργιου πίνακα, χρησιμοποιούμε την εντολή:

CREATE TABLE Πίνακας*1* (Πεδίο*1* Τύπος*1,* Πεδίο*2* Τύπος*2, ...,* ΠεδίοΝ ΤύποςΝ,
[**CONSTRAINT** Περιορισμός*1* **PRIMARY KEY** (Πεδίο*i* , Πεδίο*j* ,...)]**,**
[**CONSTRAINT** Περιορισμός*2* **NOT NULL** (Πεδίο*i* , Πεδίο*j* ,...)]**,**
[**CONSTRAINT** Περιορισμός*3* **UNIQUE** (Πεδίο*i* , Πεδίο*j* ,...)]**,**
[**CONSTRAINT** Περιορισμός*4* **FOREIGN KEY** (Πεδίο*i*) **REFERENCES** Πίνακας*2* (Πεδίο*k*)]);

• Στις παραμέτρους Πίνακας*,* Πεδίο*,* Περιορισμός δίνουμε κάποιο όνομα. Τα ονόματα μπορούν να αποτελούνται και από ελληνικούς χαρακτήρες, καλύτερα κεφαλαίους για να αποφεύγονται οι τόνοι. Δεν πρέπει να περιέχουν κενά.
• Στην παράμετρο Τύπος δηλώνουμε έναν τύπο.
• Οι περιορισμοί (**CONSTRAINTS**) είναι προαιρετικοί.
• Ο περιορισμός **PRIMARY KEY** δηλώνει το πρωτεύον κλειδί, ο **NOT NULL** δηλώνει τα πεδία που δεν επιτρέπεται να μένουν κενά, ενώ ο **UNIQUE** δηλώνει τα πεδία που δεν επιτρέπεται να έχουν δύο ίδιες τιμές.
• Ο περιορισμός **FOREIGN KEY** δηλώνει ότι το Πεδίο*i* είναι ξένο κλειδί και συγκεκριμένα είναι το πρωτεύον κλειδί Πεδίο*k* του πίνακα Πίνακας*2*, όπως δηλώνει η δεσμευμένη λέξη **REFERENCES.** Με τον τρόπο αυτό καθορίζουμε τη συσχέτιση που υπάρχει ανάμεσα στους πίνακες Πίνακας*1* και Πίνακας*2*.

Τροποποίηση της δομής πίνακα

Με τις παρακάτω εντολές η SQL μας δίνει τη δυνατότητα προσθήκης ενός πεδίου στα δεξιά του πίνακα, αφαίρεσης κάποιου πεδίου, αλλαγής του τύπου του πεδίου, προσθήκης ή αφαίρεσης περιορισμού αντίστοιχα.

ALTER TABLE Πίνακας*1*
    ADD COLUMN Πεδίο*1* Τύπος*1***;**
ALTER TABLE Πίνακας*1*
    DROP COLUMN Πεδίο*1***;**
ALTER TABLE Πίνακας*1*
    ALTER COLUMN Πεδίο*1* Τύπος*2***;**
ALTER TABLE Πίνακας*1*
    ADD CONSTRAINT Περιορισμός*1* ….
ALTER TABLE Πίνακας*1*
    DROP CONSTRAINT Περιορισμός*1***;**

Διαγραφή πίνακα
Η εντολή αυτή θέλει επίσης ιδιαίτερη προσοχή γιατί διαγράφει ολόκληρο τον πίνακα με τα δεδομένα του.
DROP TABLE Πίνακας*1* **;**

2.3.3.3 Διαχείριση των Δεδομένων της Βάσης Δεδομένων

Με τις επόμενες εντολές διαχειριζόμαστε τα δεδομένα της βάσης, εφόσον έχει ήδη καθοριστεί η δομή της.

Εισαγωγή δεδομένων

INSERT INTO Πίνακας*1* **(** Πεδίο*i* , Πεδίο*j* , ... **) VALUES (** Τιμή*i* , Τιμή*j* , ... **);**

• Μία εντολή INSERT μπορεί να προσθέσει μία μόνο γραμμή στο τέλος του πίνακα.
• Η INSERT μας επιτρέπει να βάλουμε επιλεκτικά τιμές μόνο σε ορισμένα από τα πεδία της τελευταίας γραμμής. Τα υπόλοιπα μπορούν να μείνουν κενά.
• Για πεδία χαρακτήρων οι τιμές δίνονται μέσα σε απλά εισαγωγικά (π.χ. 'Κώστας') ενώ για πεδία ημερομηνίας δίνονται μέσα σε # (π.χ. #25/3/2000#).

Τροποποίηση δεδομένων

UPDATE Πίνακας*1* **SET** Πεδίο*i* =Τιμή*i* , Πεδίο*j* =Τιμή*j* , ... **WHERE** Συνθήκη**;**

• Ως Τιμή μπορεί να μπει και μία αριθμητική παράσταση (π.χ. Πεδίο*i* = Πεδίο*k +100*)
• Η Συνθήκη μπορεί να περιέχει τελεστές σύγκρισης (>, <, =, κλπ) και λογικούς τελεστές (**NOT, AND, OR**). Επίσης μπορεί να περιέχει τους τελεστές **IS**, **IN, BETWEEN … AND** και **LIKE**.
• Μ' αυτό τον τρόπο μπορούμε να μεταβάλλουμε επιλεκτικά τα δεδομένα ενός πίνακα.

Διαγραφή γραμμών

DELETE FROM Πίνακας*1* [**WHERE** Συνθήκη]**;**

• Αν δεν υπάρχει η επιλογή WHERE διαγράφονται όλες οι γραμμές του πίνακα.
• Δεν μπορούμε να διαγράφουμε επιλεκτικά τιμές από ορισμένα πεδία του πίνακα

2.3.3.4 Ερωτήματα Επιλογής Δεδομένων

Με την εντολή **SELECT** που ακολουθεί, δε μεταβάλλουμε τα δεδομένα της βάσης, αλλά επιλέγουμε να εμφανίσουμε κάποια από αυτά με βάση κάποια κριτήρια. Κατά την επιλογή μπορούν να γίνουν πράξεις πάνω στα δεδομένα, ή να γίνει ομαδοποίηση των δεδομένων και να εξαχθούν κάποια συγκεντρωτικά στοιχεία.

Η γενική της σύνταξη είναι η ακόλουθη:

SELECT [**DISTINCT**] [**TOP** *n*]
***** ή Πίνακας*1*.Πεδίο*i* [**AS** Όνομα*i*], Πίνακας*1*.Πεδίο*j* [**AS** Όνομα*j*], ...,
Πίνακας*2*.Πεδίο*k* [**AS** Όνομα*k*], ..., ή Παράσταση*i* με κάποια πεδία [**AS** Όνομα*i*]
FROM Πίνακας*1* , Πίνακας*2*, ...
[**WHERE** Συνθήκη*1*]
[**GROUP BY** Πίνακας*1*.Πεδίο*j*, Πίνακας*2*.Πεδίο*k* ,...]
[**HAVING** Συνθήκη*2*]
[**ORDER BY** Πίνακας*1*.Πεδίο*i* [**DESC**], Πίνακας*2*.Πεδίο*k* [**DESC**], ...]**;**

Κύριο τμήμα της **SELECT**

Είναι το απαραίτητο κομμάτι της SELECT που περιλαμβάνει τις δεσμευμένες λέξεις **SELECT** και **FROM**.

• Η επιλογή **DISTINCT** δηλώνει ότι δε θέλουμε στο αποτέλεσμα της SELECT να εμφανιστούν δύο ίδιες γραμμές.
• Η επιλογή **TOP** *n* δηλώνει ότι θέλουμε το αποτέλεσμα της SELECT να περιέχει μόνο τις πρώτες *n* γραμμές που προκύπτουν από την αναζήτηση που κάναμε (συνήθως συνδυάζεται με κάποιου είδους ταξινόμηση).
• Το αποτέλεσμα της SELECT θα έχει τόσες στήλες όσα και τα πεδία, εκφράσεις κλπ που δηλώνουμε στο κύριο τμήμα της SELECT.
• Ο χαρακτήρας ***** αντί των διαφόρων πεδίων, δηλώνει ότι στο αποτέλεσμα της SELECT θα εμφανιστούν όλα τα πεδία των πινάκων που δηλώνουμε στη **FROM.**
• Με τη δεσμευμένη λέξη **FROM** δηλώνουμε τους πίνακες που θα χρησιμοποιηθούν για τη λήψη των δεδομένων. Όταν δηλώνουμε περισσότερους από έναν πίνακες, είναι απαραίτητο, μπροστά από τα ονόματα των πεδίων που χρησιμοποιούμε σε οποιοδήποτε τμήμα της SELECT, να γράφουμε και το όνομα του πίνακα στον οποίο ανήκει το κάθε πεδίο, ακολουθούμενο από τελεία π.χ. Πίνακας*1*.Πεδίο*j*, Πίνακας*2*.Πεδίο*k*.
• Αν όμως χρησιμοποιούμε στοιχεία από ένα μόνο πίνακα, μπορούμε να αναφερόμαστε στα πεδία του μόνο με το όνομά τους.
• Η επιλογή **AS** Όνομα*i* χρησιμοποιείται μόνο αν θέλουμε στο αποτέλεσμα της SELECT, το συγκεκριμένο πεδίο ή παράσταση να εμφανιστεί με άλλο όνομα από αυτό που έχει στον πίνακα. Το Όνομα*i* ονομάζεται ψευδώνυμο **(alias)** του πεδίου ή της παράστασης. Η **AS** είναι πολύ χρήσιμη όταν πρόκειται για μία παράσταση πεδίων, γιατί σ' αυτήν την περίπτωση το όνομα που δίνει η SQL στη στήλη που προκύπτει είναι συνήθως δυσνόητο. Με την επιλογή **AS** μπορούμε να δώσουμε στην παράσταση ό,τι όνομα επιθυμούμε.
• Μία παράσταση πεδίων μπορεί να περιέχει, εκτός από αριθμητικές πράξεις (αν τα πεδία είναι αριθμητικά), συναρτήσεις που αφορούν μία ή περισσότερες τιμές ενός πεδίου. Παραδείγματα τέτοιων συναρτήσεων είναι οι **ABS** και **YEAR**, που εφαρμόζονται σε κάθε μία από τις τιμές ενός αριθμητικού πεδίου ή πεδίου ημερομηνίας/ώρας αντίστοιχα, ενώ υπάρχουν και οι συναρτήσεις ομαδοποίησης δεδομένων.

Ταξινόμηση

Μπορούμε να ταξινομήσουμε τα αποτελέσματα μιας SELECT ως προς ένα ή περισσότερα πεδία, χρησιμοποιώντας τη δεσμευμένη λέξη **ORDER BY**.
• Η **ORDER BY** ταξινομεί τα αποτελέσματα της αναζήτησης σε αύξουσα σειρά, με βάση τη σειρά των πεδίων που δηλώνονται σ' αυτήν. Αν θέλουμε φθίνουσα σειρά στην ταξινόμηση με βάση ένα πεδίο, τότε δεξιά του ονόματός του προσθέτουμε τη δεσμευμένη λέξη **DESC** (είναι το αντίστοιχο του **ASC** (αύξουσα σειρά), που συνήθως παραλείπεται).

Συνθήκες επιλογής γραμμών
Ενώ στο κύριο τμήμα της SELECT επιλέγουμε τις στήλες των πινάκων που μας ενδιαφέρουν, χρησιμοποιώντας την επιλογή **WHERE** ή **HAVING** επιλέγουμε ορισμένες μόνο γραμμές από τις παραπάνω στήλες. Οι γραμμές αυτές θα πρέπει να πληρούν τα κριτήρια (συνθήκες) που θέτουμε στη WHERE ή τη HAVING. Η **HAVING** χρησιμοποιείται μόνο σε συνδυασμό με συναρτήσεις ομαδοποίησης και την επιλογή **GROUP BY**, γι' αυτό θα αναφερθούμε σ' αυτήν παρακάτω.
• Η Συνθήκη*1* της **WHERE** είναι λογική συνθήκη που μπορεί να περιέχει οποιαδήποτε από τα πεδία των πινάκων που εμφανίζονται στη **FROM**, τελεστές πράξεων (+,-,*,/), τελεστές σύγκρισης (>, <, =, κλπ) αλλά και λογικούς τελεστές (NOT, AND, OR).
• Όταν η σύγκριση γίνεται με την κενή τιμή (**NULL**), δε χρησιμοποιείται ο τελεστής "=" αλλά ο τελεστής **IS**.
• Μία συνθήκη μπορεί επίσης να περιέχει τους τελεστές **IN, BETWEEN … AND** και **LIKE**, που χρησιμοποιούνται ως εξής:

Πεδίο*i* **IN** (Τιμή*1,*Τιμή*2,…,*Τιμή*N*)
Πεδίο*i* **BETWEEN** Τιμή*1* **AND** Τιμή*2*
Πεδίο*i* **LIKE** Αλφαριθμητικό

όπου το Αλφαριθμητικό περιέχει τους χαρακτήρες *"wildcards" (*μπαλαντέρ*)* ***** ή **?**. Ο χαρακτήρας ***** αντικαθιστά οποιοδήποτε αριθμό χαρακτήρων (ή και 0 χαρακτήρες), ενώ ο χαρακτήρας **?** αντικαθιστά ακριβώς 1 χαρακτήρα.

• Αν δεν υπάρχει η **WHERE** τότε εμφανίζονται όλες οι τιμές των στηλών που έχουμε δηλώσει στο κύριο τμήμα της SELECT. Χρειάζεται όμως προσοχή στην περίπτωση που στη **FROM** έχουμε δηλώσει περισσότερους του ενός πίνακες: συνήθως οι πίνακες αυτοί έχουν κάποιο κοινό πεδίο που δηλώνει τη μεταξύ τους συσχέτιση. Η ταύτιση των δύο πεδίων πρέπει να δηλωθεί στη WHERE, αλλιώς το αποτέλεσμα θα είναι το καρτεσιανό γινόμενο των αντίστοιχων τμημάτων των δύο πινάκων.

Ομαδοποίηση γραμμών

Συχνά είναι απαραίτητο να εξάγουμε κάποια συγκεντρωτικά στοιχεία από τους πίνακες της βάσης μας, όπως αθροίσματα, μέσους όρους, μέγιστες ή ελάχιστες τιμές, πλήθος δεδομένων κλπ. Για το σκοπό αυτό η SQL διαθέτει τις συναρτήσεις ομαδοποίησης **SUM, AVG, MIN, MAX** και **COUNT**. Οι συναρτήσεις αυτές δεν εφαρμόζονται μεμονωμένα σε μία τιμή, αλλά σε ομάδες τιμών ενός πεδίου του πίνακα. Για κάθε ομάδα τιμών, η συνάρτηση ομαδοποίησης επιστρέφει μία μόνο τιμή

(το άθροισμα των τιμών της ομάδας, το μέσο όρο των τιμών, την ελάχιστη τιμή, τη μέγιστη τιμή, ή το πλήθος των τιμών της ομάδας αντίστοιχα).
• Από τις παραπάνω συναρτήσεις διαφοροποιείται λίγο η συνάρτηση **COUNT**, καθώς δεν αναφέρεται σε συγκεκριμένο πεδίο αλλά στην ουσία μετράει το πλήθος των γραμμών της κάθε ομάδας. Γι' αυτό συνήθως χρησιμοποιείται χωρίς να περιέχει για όρισμα ένα συγκεκριμένο πεδίο, αλλά ως **COUNT(*)**. Ο καθορισμός των ομάδων των γραμμών πάνω στις οποίες θα εφαρμοστεί μία συνάρτηση ομαδοποίησης γίνεται με την επιλογή **GROUP BY**.
• Αν δεν υπάρχει η **GROUP BY,** τότε ολόκληρη η στήλη στην οποία εφαρμόζεται η συνάρτηση ομαδοποίησης λαμβάνεται σαν μία ομάδα.
• Μπορούμε να δηλώσουμε περισσότερα από ένα πεδία μέσα στη **GROUP BY**. Αυτό σημαίνει ότι μία ομάδα γραμμών, αποτελείται από εκείνες τις γραμμές του πίνακα που έχουν τον ίδιο συνδυασμό τιμών σε όλα τα πεδία που δηλώνονται στη **GROUP BY**. Έτσι, όσο πιο πολλά πεδία δηλώνουμε στη **GROUP BY**, τόσο περισσότερες (και μικρότερες) ομάδες δημιουργούμε.
• Αν χρησιμοποιήσουμε τη **GROUP BY** χωρίς να έχουμε συνάρτηση ομαδοποίησης, τότε απλά ταξινομούμε σε αύξουσα σειρά τον πίνακα, με βάση τα πεδία που είναι δηλωμένα στη **GROUP BY** (δηλ. η **GROUP BY** αντικαθιστά την **ORDER BY**).
• Η εντολή **HAVING** χρησιμοποιείται μόνο εφόσον έχει προηγηθεί η **GROUP BY** και επιλέγει κάποιες από τις ομάδες γραμμών που έχουν δημιουργηθεί με τη GROUP BY, ανάλογα με τη Συνθήκη*2*. Στη Συνθήκη*2* περιέχονται συναρτήσεις ομαδοποίησης. Ο λόγος που η Συνθήκη*2* δεν μπορεί να ενσωματωθεί σε μία **WHERE**, είναι γιατί η **WHERE** εκτελείται πριν από τη **GROUP BY**. Έτσι δεν είναι γνωστό στη **WHERE** αν υπάρχουν ομάδες γραμμών και ποιες είναι.

Σειρά εκτέλεσης των ενεργειών επιλογής

Συνολικά η σειρά με την οποία γίνονται ενέργειες επιλογής στα δεδομένα με βάση τα διάφορα τμήματα της εντολής SELECT είναι η ακόλουθη:

FROM (επιλέγουμε τους σχετικούς πίνακες)
WHERE (επιλέγουμε ορισμένες γραμμές)
GROUP BY (ομαδοποιούμε αυτές τις γραμμές)
HAVING (επιλέγουμε κάποιες από τις παραπάνω ομάδες γραμμών)
SELECT (επιλέγουμε μόνο τις στήλες που μας ενδιαφέρουν)
ORDER BY (ταξινομούμε τις παραπάνω γραμμές)
DISTINCT (αφαιρούμε τα διπλότυπα)
TOP *n* (από τις γραμμές που προκύπτουν, κρατάμε τις πρώτες *n*)

2.3.3.5 Σύνθετα Ερωτήματα Επιλογής Δεδομένων

Συχνά οι ανάγκες ενός ερωτήματος είναι τέτοιες ώστε χρειάζεται να χρησιμοποιηθούν περισσότερες από μία εντολές SELECT, φωλιασμένες η μία μέσα στην άλλη. Η δεύτερη SELECT ενσωματώνεται μέσα στη συνθήκη του τμήματος WHERE (ή HAVING) της πρώτης SELECT. Η δεύτερη SELECT μπαίνει σε παρένθεση και ακολουθεί έναν από τους τελεστές >, <, =, … , **IN**. Π.χ:
SELECT Πεδίο*1,* Πεδίο*2 ,…***FROM** Πίνακας*1*
WHERE Πεδίο*i* **IN** (**SELECT** Πεδίο*j* **FROM** Πίνακας*2* **WHERE** Συνθήκη)

Απαραίτητη προϋπόθεση είναι τα πεδία Πεδίο*i* και Πεδίο*j* να είναι του ίδιου τύπου.

• Σημειώνουμε ότι οι τελεστές σύγκρισης (<, >, = κλπ) μπορούν να χρησιμοποιηθούν μόνο αν η δεύτερη **SELECT** επιστρέφει μία μοναδική τιμή (π.χ. αν είναι **SELECT MAX**(Πεδίο*j*) **FROM**… ή **SELECT TOP** 1 Πεδίο*j* **FROM**…). Συνήθως όμως, περιμένουμε να πάρουμε από τη δεύτερη **SELECT** ένα πλήθος τιμών. Σ' αυτές τις περιπτώσεις χρησιμοποιούνται οι τελεστές σύγκρισης σε συνδυασμό με τους τελεστές **ALL** και **SOME** (ή **ANY**). Π.χ:
SELECT Πεδίο*1,* Πεδίο*2 ,*...**FROM** Πίνακας*1*
WHERE Πεδίο*i* **> ALL** (**SELECT** Πεδίο*j* **FROM** Πίνακας*2* **WHERE** Συνθήκη)
δηλαδή το Πεδίο*i* είναι μεγαλύτερο από όλα τα αποτελέσματα της δεύτερης **SELECT**. Αν αντί για **ALL** γράφαμε **SOME** (ή **ANY**), τότε για να αληθεύει η συνθήκη της πρώτης **WHERE**, θα έπρεπε το Πεδίο*i* να είναι μεγαλύτερο από τουλάχιστον ένα από τα αποτελέσματα της δεύτερης **SELECT**.

Τέλος, αναφέρουμε ότι δύο SELECT μπορούν να χρησιμοποιηθούν ταυτόχρονα και χωρίς να είναι φωλιασμένες, αλλά με την πράξη της ένωσης, υπό την προϋπόθεση ότι τα αντίστοιχα πεδία των δύο SELECT είναι του ίδιου τύπου:

(SELECT Πεδίο*i*, Πεδίο*j*, … **FROM** Πίνακας*1* **WHERE** Συνθήκη*1***)**
UNION [ALL]
(SELECT Πεδίο*i*, Πεδίο*j*, … **FROM** Πίνακας*2* **WHERE** Συνθήκη*2***)**

• Τα αποτελέσματα της δεύτερης SELECT μπαίνουν μετά από αυτά της πρώτης SELECT, γι' αυτό πρέπει τα αντίστοιχα πεδία των δύο SELECT να είναι συμβατά.
• Η επιλογή **ALL** δηλώνει ότι συμπεριλαμβάνονται και διπλότυπες γραμμές που μπορεί να προκύψουν από τις δύο **SELECT**. Χωρίς την **ALL**, τα διπλότυπα αφαιρούνται.

2.3.4 Βάσεις Χωρικών Δεδομένων

2.3.4.1 Εισαγωγή

Οι βάσεις χωρικών δεδομένων [34] αποτελούν σημείο αιχμής της έρευνας στον τομέα των βάσεων δεδομένων τα τελευταία 15 χρόνια. Η βασική εφαρμογή τους είναι η υποστήριξη των Γεωγραφικών Πληροφοριακών Συστημάτων (Geographical Information Systems). Τα γεωγραφικά συστήματα πληροφοριών είναι ιδιαίτερα χρήσιμα στη χαρτογράφηση τόσο περιοχών όσο και διαφόρων δικτύων όπως οδικών, τηλεφωνικών, υπολογιστικών. Χωρικά δεδομένα, όμως, συναντούνται και σε άλλες εφαρμογές, όπως βάσεις δεδομένων εικόνων και πολυμέσων.

2.3.4.2 Ορισμός

Μια βάση χωρικών δεδομένων παρέχει όλα όσα παρέχουν και οι «απλές» βάσεις. Επιπλέον, προσφέρει τη δυνατότητα να παρασταθούν και να αποθηκευτούν **τύποι χωρικών δεδομένων** όπως ένα σημείο στο χώρο, μια ευθεία ή και ένα πολύπλοκο γεωμετρικό σχήμα. Επίσης, υποστηρίζονται οι **σχέσεις** μεταξύ τους (π.χ. αν ένα ευθύγραμμο τμήμα τέμνεται με ένα άλλο), οι **ιδιότητές** τους (π.χ. αν ένα τετράγωνο έχει εμβαδό μεγαλύτερο από κάποια τιμή), καθώς και διάφορες **πράξεις** τους (π.χ. η τομή δύο παραλληλογράμμων). Τέλος, υπάρχει αποδοτικός τρόπος αναζήτησης και προσπέλασης των τύπων χωρικών δεδομένων όπως ακριβώς και στις «απλές» βάσεις δεδομένων, με τη διαφορά ότι οι μέθοδοι αναζήτησης και προσπέλασης είναι εκ των πραγμάτων διαφορετικές και πιο ισχυρές (π.χ. τεχνικές κατακερματισμού και δενδρικές μορφές) σε σχέση με τις αντίστοιχες των «απλών» βάσεων.

2.3.4.3 Τύποι Χωρικών Δεδομένων

Στις βάσεις χωρικών δεδομένων χρειάζεται να μοντελοποιηθεί ο **χώρος** και διάφορα **αντικείμενα** στο χώρο (όπως πόλεις, ποτάμια, δρόμοι καθώς και πιθανές συνδέσεις ανάμεσά τους).

Για τη μοντελοποίηση αντικειμένων στο χώρο χρειάζονται σημεία, γραμμές και περιοχές (σχήματα με εμβαδό). Τα σημεία αντιπροσωπεύουν την περίπτωση όπου μόνο η θέση στο χώρο παίζει ρόλο και όχι η έκταση. Οι γραμμές αποτελούν τον τρόπο σύνδεσης αντικειμένων και επομένως είναι κατάλληλες για την αναπαράσταση δρόμων, ποταμών ή άλλων σχετικών αντικειμένων. Τέλος, οι περιοχές αναπαριστούν αντικείμενα που έχει σημασία και η έκτασή τους στο χώρο εκτός της θέσης τους. Όσον αφορά τη μοντελοποίηση του ίδιου του χώρου χρειάζεται ένας τρόπος να χωριστεί σε τεμάχια και να δημιουργηθούν δίκτυα μέσα του.

Από τα παραπάνω είναι φανερό πως τα χωρικά δεδομένα είναι πολύπλοκές δομές που αποτελούνται είτε από ένα απλό σημείο είτε από πάρα πολλά σημεία αυθαίρετα τοποθετημένα στο χώρο. Κατά συνέπεια, είναι αδύνατο να αποθηκευτούν τέτοια δεδομένα στους κλασικούς σχεσιακούς πίνακες. Επίσης, τα δεδομένα (άρα και οι αντίστοιχες βάσεις δεδομένων) τείνουν να καταλαμβάνουν πολύ μεγαλύτερο χώρο στο δίσκο. Ακόμα, δεν υπάρχει κοινώς αποδεκτή (πρότυπη) άλγεβρα για τα χωρικά δεδομένα, οπότε η άλγεβρα που χρησιμοποιείται κάθε φορά εξαρτάται από την εκάστοτε εφαρμογή.

3 ΕΡΓΑΛΕΙΑ ΑΝΑΠΤΥΞΗΣ

3.1 PROTÉGÉ

3.1.1 Εισαγωγή

Το Protégé [20] είναι μια εφαρμογή ανεπτυγμένη για το σκοπό της **ανάπτυξης και αναπαράστασης οντολογιών και βάσεων γνώσης**. Αποτελεί επίσης και ένα αξιόπιστο, «ανοιχτού κώδικα» εργαλείο (open source) βασισμένο σε Java, το οποίο παρέχει μια ευρεία αρχιτεκτονική για την κατασκευή άλλων εργαλείων βάσεων γνώσης.

Το πρόγραμμα πρωτοεμφανίστηκε το 1988 και αποτελούσε απλώς ένα μέσο για τη δημιουργία εργαλείων ανάκτησης γνώσης για έμπειρα συστήματα. Σήμερα, μετά από 17 χρόνια, με τη δουλειά του τμήματος ιατρικών πληροφοριών του πανεπιστημίου του Stanford (Stanford Medical Informatics – SMI) το Protégé έχει εξελιχθεί σε ένα σύγχρονο εργαλείο μοντελοποίησης γνώσης. Αυτή τη στιγμή απασχολεί ένα σημαντικό κομμάτι των ερευνητών στον τομέα της οντολογίας, έχοντας σχεδόν 4.500 χρήστες σε όλο τον κόσμο, ενώ και η ίδια η εφαρμογή αναπτύσσεται με ταχύτατους ρυθμούς με την τελευταία της έκδοση να είναι η 3.1, η οποία κυκλοφόρησε τον Ιούλιο του 2005.

Οι χρήστες του Protégé προσφέρουν τις γνώσεις τους σε μια on-line κοινότητα που έχει δημιουργηθεί για αυτόν ακριβώς το λόγο, την εξάπλωση της γνώσης και την επίλυση προβλημάτων που οι χρήστες αντιμετωπίζουν, γεγονός το οποίο εκτός των άλλων έχει συντελέσει σημαντικά και στην περαιτέρω ανάπτυξη της εφαρμογής από την προγραμματιστική ομάδα.

Το Protégé είναι ένα ολοκληρωμένο εργαλείο που χρησιμοποιείται για την ανάπτυξη συστημάτων βασισμένων σε γνώση (knowledge-based systems). Οι εφαρμογές που αναπτύσσονται μέσω του Protégé χρησιμοποιούνται στη λύση προβλημάτων

(problem-solving) και στη λήψη αποφάσεων (decision-making) γύρω από έναν τομέα γνώσης. Γενικά, η χρησιμότητα του Protégé έγκειται στο γεγονός ότι τα βασισμένα σε γνώση συστήματα απαιτούν πολλούς πόρους για την κατασκευή και τη συντήρησή τους. Η ανάπτυξη ενός τέτοιου συστήματος συνήθως απασχολεί τόσο προγραμματιστές όσο και ειδικούς του συγκεκριμένου τομέα γνώσης, οι οποίοι μπορεί να έχουν μικρή εξοικείωση με το λογισμικό εν γένει. Το Protégé έχει σχεδιαστεί έτσι ώστε να καθοδηγήσει σωστά την ομάδα των προγραμματιστών και των ειδικών του τομέα στην ανάπτυξη του knowledge-based συστήματος. Επίσης, επιτρέπει στους προγραμματιστές να επαναχρησιμοποιούν οντολογίες σχετικές με τον τομέα καθώς και μεθόδους επίλυσης προβλημάτων, με αποτέλεσμα να μειώνεται ο χρόνος ανάπτυξης και συντήρησης.

Με το Protégé, καθίσταται δυνατή η ταυτόχρονη εργασία με κλάσεις οντολογιών και στιγμιότυπα (instances). Συνεπώς, ένα μεμονωμένο στιγμιότυπο μπορεί να χρησιμοποιηθεί στον ορισμό κάποιας κλάσης, ενώ μια κλάση μπορεί να αποθηκευθεί και ως στιγμιότυπο. Ομοίως, οι ιδιότητες (slots) βρίσκονται πλέον στο ίδιο επίπεδο με τις κλάσεις στο μοντέλο γνώσης του Protégé. Με αυτόν τον τρόπο επιτυγχάνεται και η συμμόρφωση προς το πρωτόκολλο Open Knowledge Base Connectivity (OKBC) για πρόσβαση βάσεων γνώσεων σε συστήματα αναπαράστασης γνώσης. Ακόμα, εφαρμογές που έχουν αναπτυχθεί πάνω στα συστατικά του Protégé μπορούν να εκτελεστούν μέσω του περιβάλλοντός του.

3.1.2 Περιγραφή

Το Protégé παρέχει πρόσβαση σε όλα τα παραπάνω χαρακτηριστικά μέσω μιας ενιαίας Γραφικής Διασύνδεσης Χρήστη (Graphical User Interface, GUI), το ανώτερο επίπεδο της οποίας περιλαμβάνει ένα σύνολο από καρτέλες (tabs) για συμπαγή αναπαράσταση των τμημάτων της εφαρμογής και για βολική επεξεργασία μιας οντολογίας. Αυτός ο σχεδιασμός που είναι βασισμένος σε tabs επιτρέπει:

- Τη μοντελοποίηση μιας οντολογίας κλάσεων που περιγράφουν ένα συγκεκριμένο αντικείμενο
- Τη δημιουργία ενός εργαλείου ανάκτησης γνώσης για τη συλλογή πληροφοριών
- Την εισαγωγή συγκεκριμένων στιγμιοτύπων δεδομένων και τη δημιουργία μιας βάσης γνώσης
- Την εκτέλεση εφαρμογών

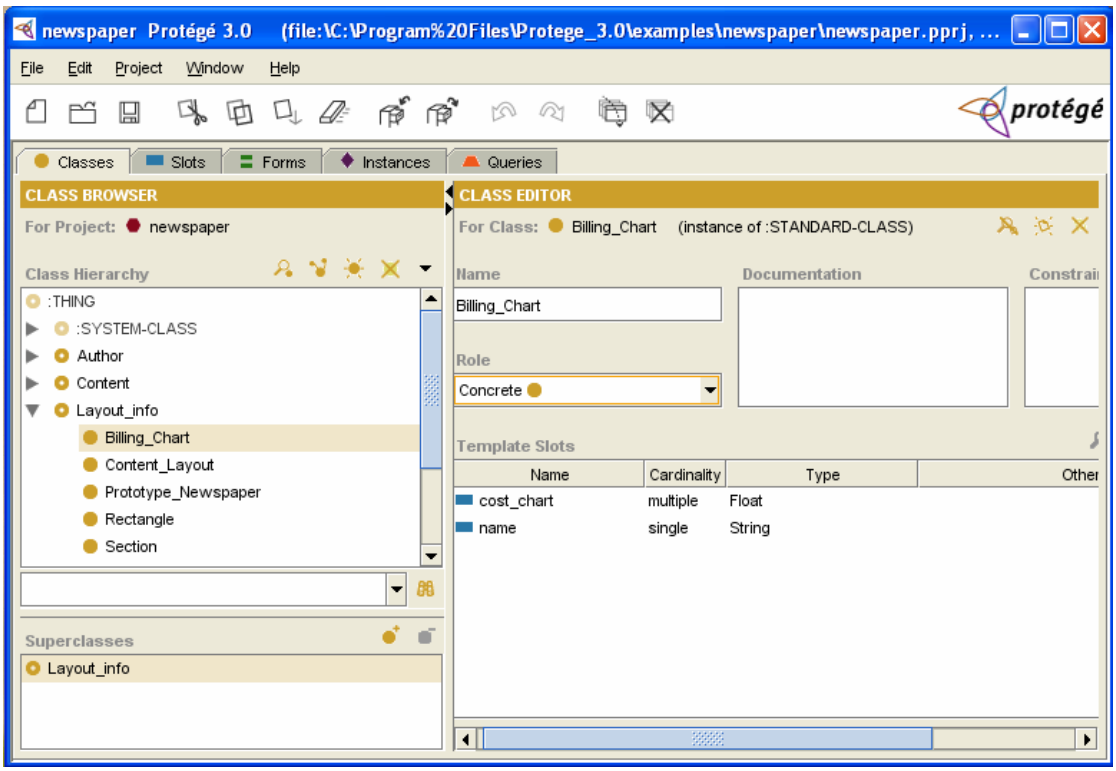
Μια οντολογία ορίζει ένα σύνολο εννοιών και τις σχέσεις που υπάρχουν μεταξύ τους. Το Protégé, ως **εργαλείο ανάκτησης γνώσης** είναι σχεδιασμένο με τέτοιο τρόπο ώστε ο τομέας γνώσης στον οποίο αναφέρεται μια οντολογία να είναι πλήρως καθορισμένος και οι ειδικοί αυτού του τομέα να μπορούν εύκολα να εισάγουν τις γνώσεις τους για αυτόν τον τομέα. Η προκύπτουσα βάση γνώσης μπορεί τότε να χρησιμοποιηθεί σε μια μέθοδο επίλυσης προβλημάτων προς απάντηση ερωτήσεων και επίλυση προβλημάτων σχετικών με τον τομέα. Τέλος, μια εφαρμογή είναι το τελικό προϊόν που δημιουργείται όταν η βάση γνώσης χρησιμοποιείται για την επίλυση ενός προβλήματος του χρήστη, μέσω της εφαρμογής κατάλληλων μεθόδων επίλυσης προβλημάτων, έμπειρων συστημάτων ή υποστήριξης αποφάσεων.

Οι βασικές καρτέλες του Protégé είναι οι εξής:

Classes Tab

Η καρτέλα των κλάσεων παρέχει ένα απλό παράθυρο στο οποίο ο χρήστης μπορεί να δει, να δημιουργήσει και να επεξεργαστεί **κλάσεις** οντολογίας, που αναπαριστούν έννοιες σε κάποιο πεδίο γνώσης. Το παράθυρο αυτό αποτελείται από τρεις τομείς:

- Τον τομέα Class Hierarchy στο πάνω αριστερό μέρος, στον οποίο απεικονίζονται οι κλάσεις σε μια ιεραρχία και δίνεται η δυνατότητα στο χρήστη να επεξεργαστεί, να δημιουργήσει και να διαγράψει κλάσεις της οντολογίας. Επίσης, είναι δυνατή η αναδιάταξη της ιεραρχίας σέρνοντας (dragging) μια κλάση στη θέση μιας άλλης κλάσης, κάτω από την οποία θα τοποθετηθεί.
- Τον τομέα Superclasses στο κάτω αριστερό μέρος, ο οποίος δείχνει τις υπερκλάσεις της επιλεγμένης κάθε φοράς κλάσης και προσφέρει τη δυνατότητα πρόσθεσης ή αφαίρεσης κάποιας υπερκλάσης.
- Τον τομέα Class Editor, στον οποίο, όταν είναι επιλεγμένη κάποια κλάση εμφανίζεται μια φόρμα για αυτήν την κλάση με τη βοήθεια της οποίας ο χρήστης μπορεί να ονομάσει την κλάση, να επιλέξει το ρόλο της, να ορίσει περιορισμούς, να της προσθέσει κάποιες επεξηγηματικές σημειώσεις και κυρίως, να ορίσει και να επεξεργαστεί τις ιδιότητές της (slots).



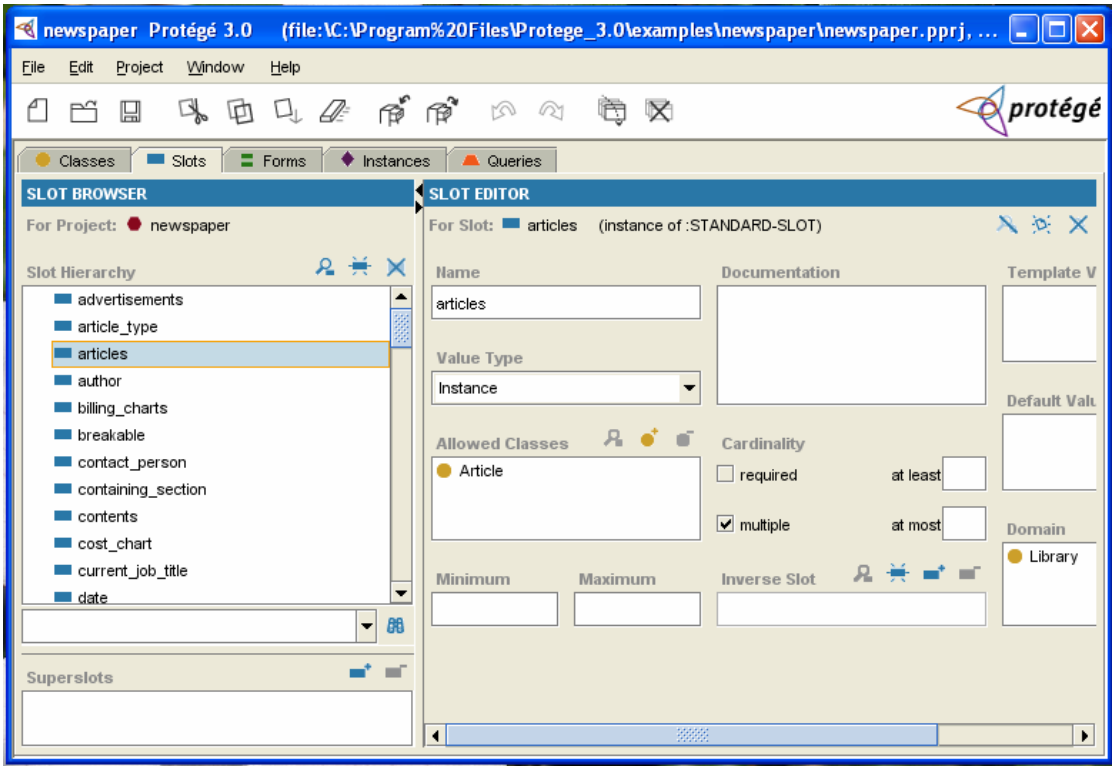
Εικόνα 3.1 Classes Tab

Slots Tab

Η καρτέλα των ιδιοτήτων παρέχει ένα απλό παράθυρο μέσω του οποίου ο χρήστης μπορεί να δει, δημιουργήσει και επεξεργαστεί **ιδιότητες**. Παρά το γεγονός ότι

συνήθως οι ιδιότητες θεωρούνται εξαρτημένες από μια κλάση, μπορούν να οριστούν και να χειριστούν ανεξάρτητα, ενώ δεν είναι απαραίτητο να έχουν σχέση με μια κλάση. Το παράθυρο αυτό αποτελείται από τρεις τομείς:

- Τον τομέα Slots Hierarchy στο πάνω αριστερό μέρος, στον οποίο φαίνονται όλες οι ιδιότητες της οντολογίας και προσφέρεται η δυνατότητα στο χρήστη να μετατρέψει τις υπάρχουσες, να δημιουργήσει νέες ιδιότητες, να δει πληροφορίες για την κάθε μία από αυτές, καθώς και να διαγράψει κάποια.
- Τον τομέα Superslots στο κάτω αριστερό μέρος, στον οποίο φαίνονται οι υπερ-ιδιότητες της εκάστοτε επιλεγμένης ιδιότητας.
- Τον τομέα Slot Editor, ο οποίος περιέχει μια φόρμα που επιτρέπει στο χρήστη να ονομάσει μια ιδιότητα, να επιλέξει την πληθικότητά της και τον τύπο της τιμής της, να ορίσει περιορισμούς, προεπιλογές, μέγιστες και ελάχιστες τιμές, καθώς και μια συνοπτική περιγραφή.



Εικόνα 3.2 Slots Tab

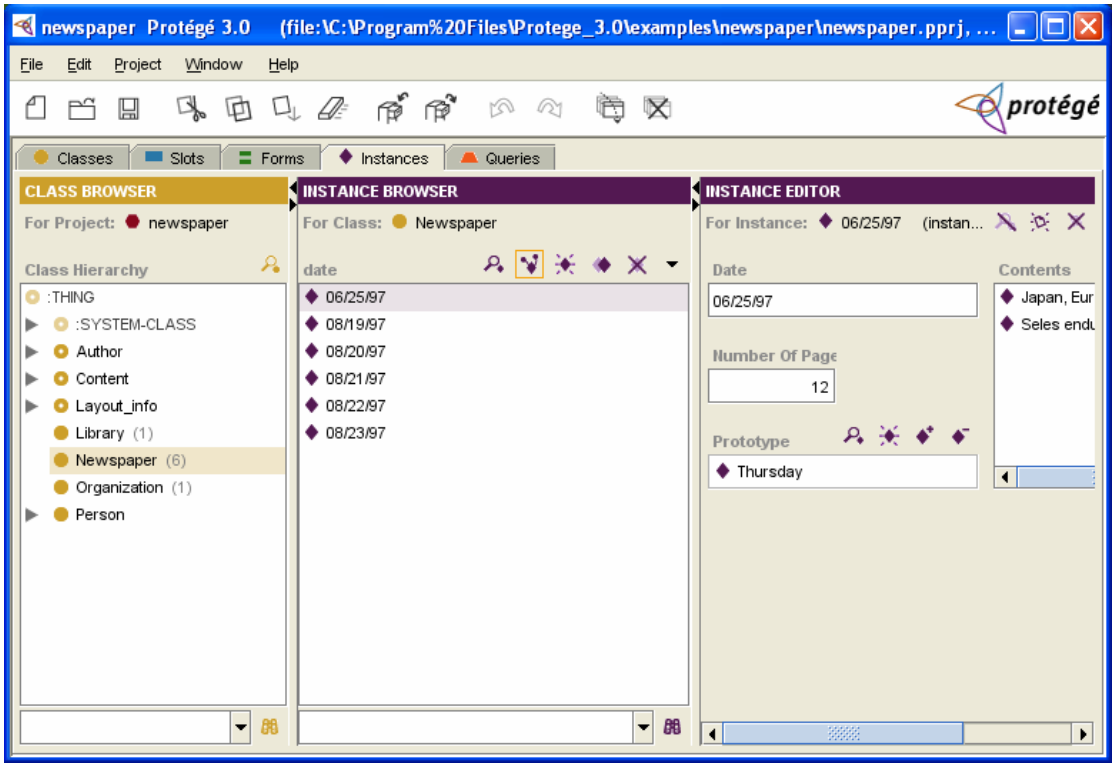
Instances Tab

Η καρτέλα των στιγμιοτύπων αποτελείται από ένα παράθυρο στο οποίο ο χρήστης μπορεί να δει, δημιουργήσει και να επεξεργαστεί **στιγμιότυπα**. Το παράθυρο αποτελείται από τρεις τομείς:

- Τον τομέα Class Browser στο αριστερό μέρος, ο οποίος εμφανίζει τις κλάσεις της οντολογίας και τις σχέσεις υπερκλάσης/υποκλάσης που υπάρχουν μεταξύ τους. Στην καρτέλα στιγμιοτύπων, δεν επιτρέπεται η επεξεργασία των κλάσεων ή η αλλαγή της ιεραρχίας τους.
- Τον τομέα Instance Browser στο κέντρο, που δείχνει τα άμεσα στιγμιότυπα για την επιλεγμένη κλάση και επιτρέπει στο χρήστη να

αντιγράψει, να δει λεπτομέρειες για κάποιο στιγμιότυπο ή και να το διαγράψει.

- Τον τομέα Instance Editor στο δεξί μέρος, όπου με τη βοήθεια μιας φόρμας ο χρήστης μπορεί να δει λεπτομέρειες για το επιλεγμένο στιγμιότυπο και να το επεξεργαστεί.

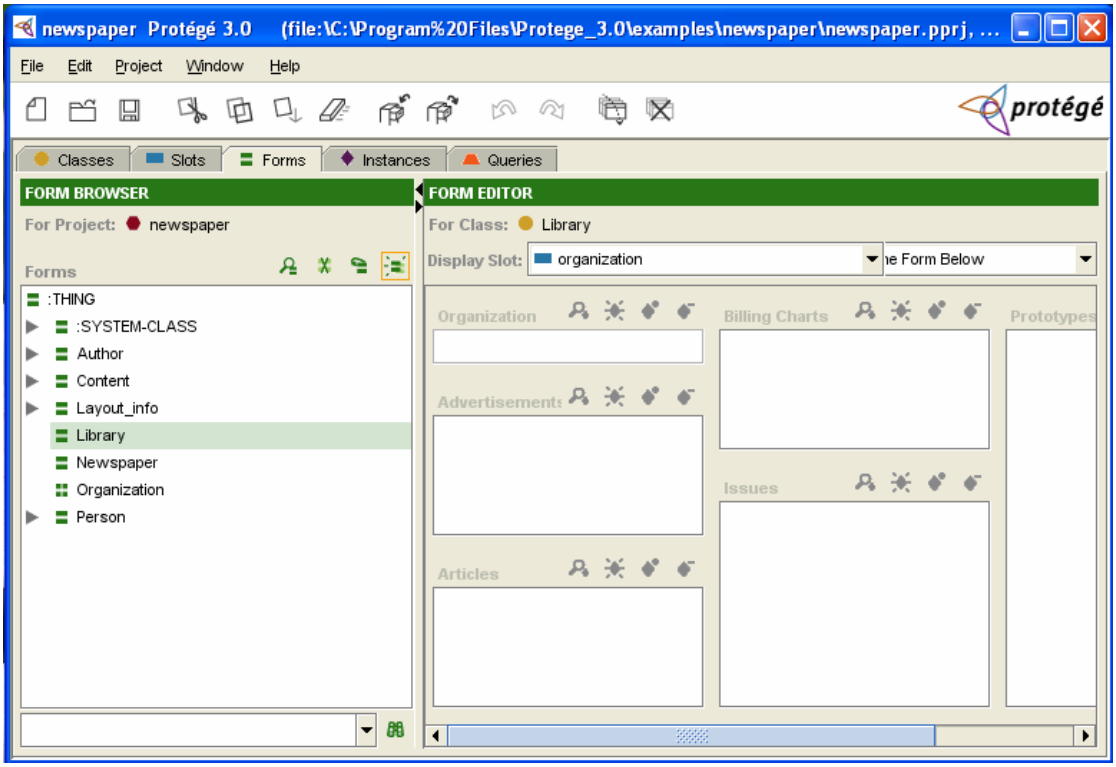


Εικόνα 3.3 Instances Tab

Forms Tab

Η καρτέλα αυτή επιτρέπει στο χρήστη να σχεδιάσει κάποιες **φόρμες**, η τελική μορφή των οποίων φαίνεται στην καρτέλα των στιγμιοτύπων. Αποτελείται από δύο τομείς:

- Τον τομέα Forms στα αριστερά, στον οποίο φαίνεται η ιεραρχία των κλάσεων της οντολογίας και ο οποίος επιτρέπει στο χρήστη να δημιουργήσει μια φόρμα για την επιλεγμένη κλάση, η οποία μπορεί να έχει τη μορφή της φόρμας κάποιας άλλης κλάσης.
- Τον τομέα Form Editor, ο οποίος δείχνει τη μορφή της φόρμας της επιλεγμένης κλάσης. Κάθε ιδιότητα της κλάσης είναι συσχετισμένη με μια γραφική διασύνδεση για ευκολία του χρήστη.

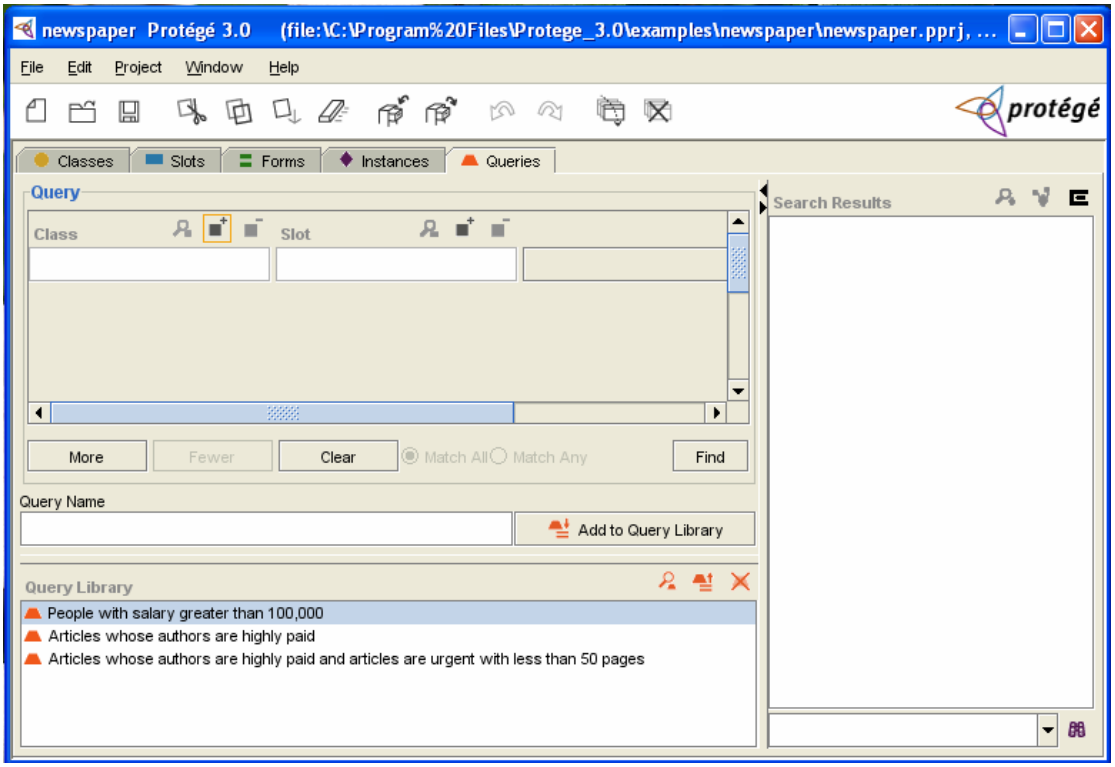


Εικόνα 3.4 Forms Tab

Queries Tab

Η καρτέλα των ερωτημάτων επιτρέπει στο χρήστη να δημιουργήσει, εκτελέσει και να αποθηκεύσει **ερωτήματα**, τα οποία αποτελούν έναν τρόπο επιλογής στιγμιοτύπων από μία οντολογία με βάση την τιμή ενός ή περισσοτέρων ιδιοτήτων. Τα ερωτήματα δεν είναι μέρος της βάσης γνώσης, αλλά είναι ένας τρόπος αναγνώρισης των στιγμιοτύπων της βάσης γνώσης που βασίζεται στα χαρακτηριστικά κλάσεων και ιδιοτήτων. Η καρτέλα αυτή αποτελείται από τρεις τομείς:

- Τον τομέα Query, όπου εισάγεται ή τροποποιείται ένα ερώτημα. Επιτρέπεται επίσης και συνδυασμός πολλαπλών ερωτημάτων.
- Τον τομέα Search Results, όπου εμφανίζονται τα αποτελέσματα
- Τον τομέα Query Library, που προσφέρει τη δυνατότητα αποθήκευσης και ανάκτησης ερωτημάτων. Τα αποθηκευμένα ερωτήματα μπορούν με τη σειρά τους να μεταβληθούν ή να συνδυαστούν με άλλα.

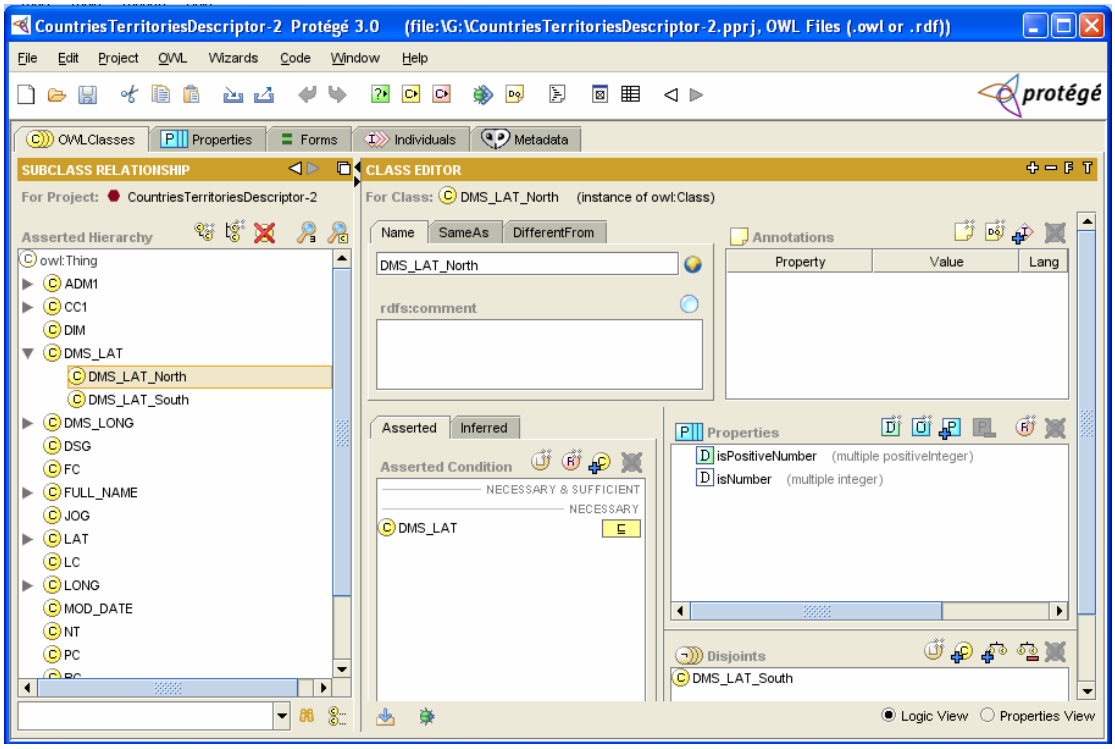


Εικόνα 3.5 Queries Tab

3.1.3 Επεκτάσεις

Το Protégé είναι ένα ευέλικτο εργαλείο ανοικτού κώδικα το οποίο αναπτύσσεται διαρκώς. Εκτός από το κυρίως μέρος της εφαρμογής που περιγράφηκε παραπάνω, υπάρχουν διαθέσιμες και αρκετές επεκτάσεις (**plugins**), οι οποίες φορτώνονται μέσω του Protégé και **επεκτείνουν τη λειτουργικότητά** του, προσθέτοντας καινούριες δυνατότητες και λειτουργίες που δεν περιλαμβάνονται στο κυρίως πρόγραμμα. Αυτές οι επεκτάσεις μπορούν να εμφανίζονται σε ξεχωριστό παράθυρο (tab), να έχουν τη μορφή ενός συγκεκριμένου συστατικού διασύνδεσης (widget) ή να εκτελούν οποιαδήποτε άλλη εργασία.

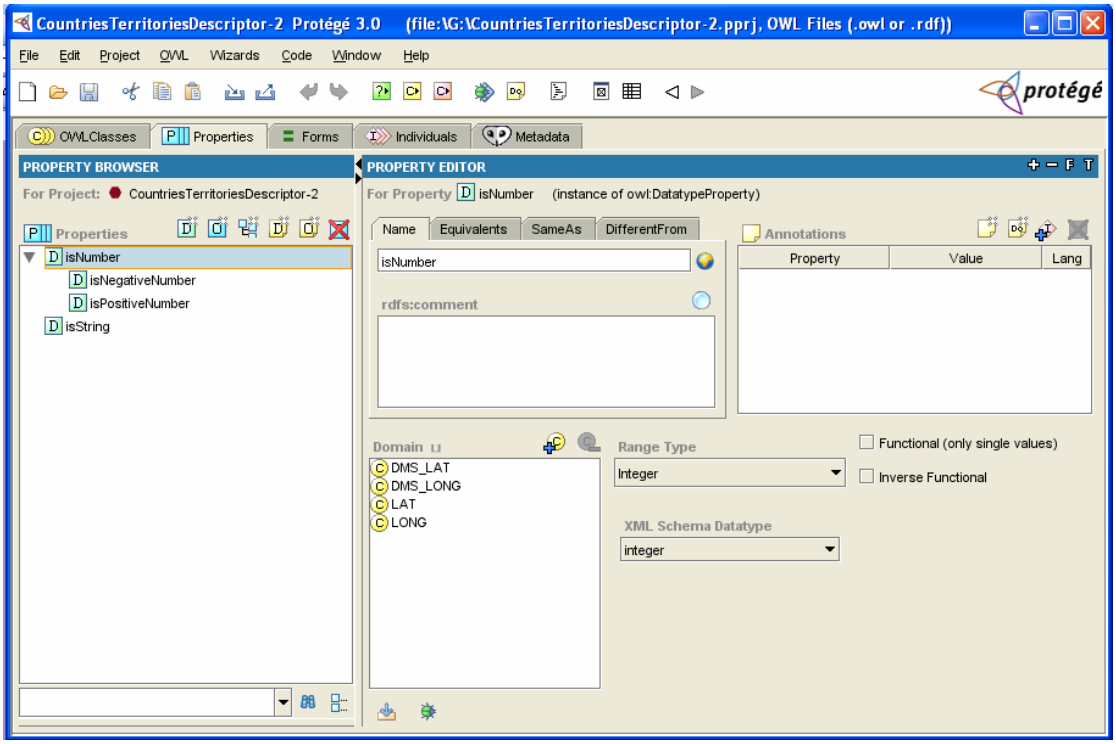
Η σημαντικότερη ίσως, από τις υπάρχουσες επεκτάσεις είναι το **Protégé OWL Plugin** [19][21], που παρέχει στο Protégé τη δυνατότητα να φορτώνει αλλά και να αποθηκεύει OWL αρχεία, να πραγματοποιεί ερωτήματα σε μοντέλα δεδομένων, να τα επεξεργάζεται καθώς και να εκτελεί αιτιολογήσεις (reasoning) βασιζόμενο σε μηχανές Περιγραφικής Λογικής. Το OWL Plugin παρέχει έναν αριθμό από συστατικά γραφικής διασύνδεσης που επιτρέπουν την επεξεργασία μιας OWL οντολογίας. Συγκεκριμένα, μόλις ο χρήστης επιλέξει ένα OWL αρχείο, εμφανίζονται πέντε καρτέλες (tabs), όπως φαίνονται στο παρακάτω σχήμα:



Εικόνα 3.6 OWL Classes Tab

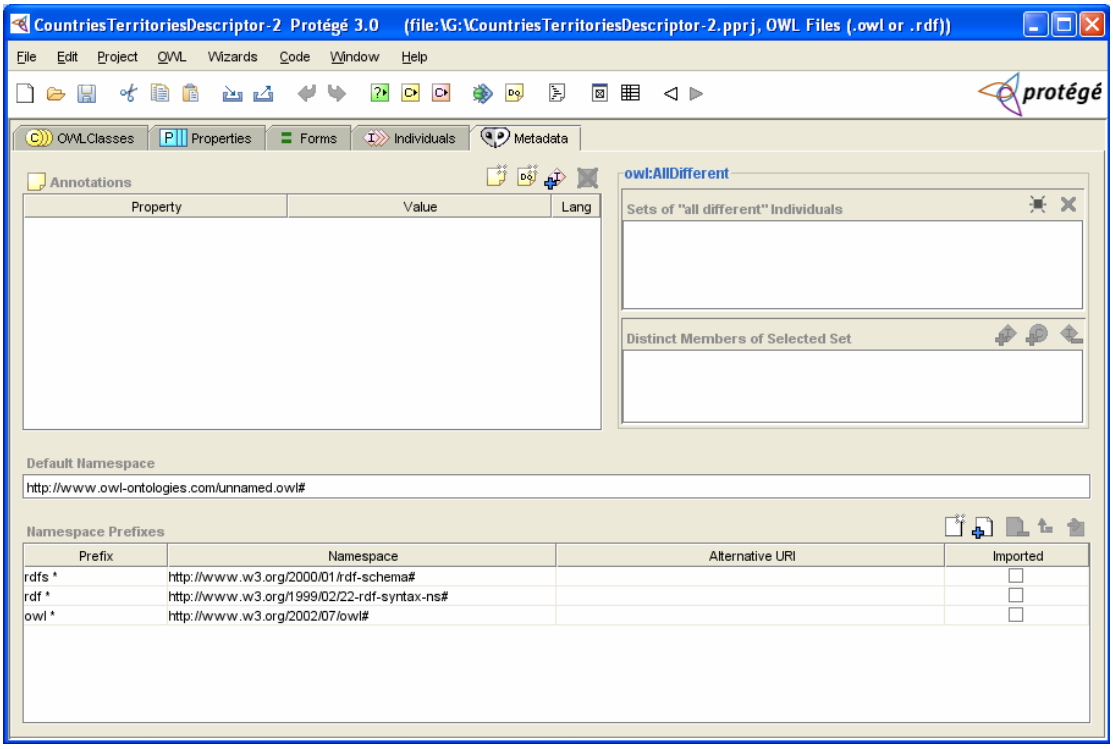
Το βασικότερο ίσως tab είναι το **OWLClasses** Tab, που φαίνεται στην παραπάνω εικόνα και το οποίο παρουσιάζει την ιεραρχία των κλάσεων της OWL οντολογίας και επιτρέπει τη διαγραφή ή τη δημιουργία νέων κλάσεων. Σε αυτήν την καρτέλα φαίνονται όλες οι πληροφορίες σχετικά με την επιλεγμένη κλάση, οι οποίες μπορούν και να μεταβληθούν. Έτσι, ο χρήστης μπορεί να επιβάλλει περιορισμούς, αναγκαίους ή ικανούς και αναγκαίους, τους οποίους θα πρέπει να ικανοποιούν τα άτομα (individuals) που αποτελούν την κλάση. Επίσης, επιτρέπεται ο καθορισμός των κλάσεων που είναι **ξένες** (disjoint) προς την επιλεγμένη κλάση, και οι οποίες δεν πρέπει να περιέχουν κοινά άτομα με την εν λόγω κλάση. Τέλος, προβάλλονται οι ιδιότητες που ανήκουν σε μια κλάση καθώς και τυχόν σχόλια για αυτήν την κλάση.

Όλες οι ιδιότητες της οντολογίας προβάλλονται στο δεύτερο από αριστερά tab, το **Properties** Tab. Σε αυτό το παράθυρο, ο χρήστης έχει τη δυνατότητα να δημιουργήσει καινούριες ιδιότητες, καθώς και να διαγράψει ή να επεξεργαστεί άλλες. Κατά τη δημιουργία μιας ιδιότητας, επιλέγεται ο τύπος της, ο οποίος μπορεί να είναι ένας από τους εξής: object, datatype ή annotation. Οι **object properties** συνδέουν ένα άτομο από το πεδίο ορισμού (domain) με ένα άλλο άτομο από το πεδίο τιμών (range). Οι **datatype properties** συνδέουν ένα άτομο μιας κλάσης με μία τιμή τύπου υποστηριζόμενου από το XML Schema, όπως integer ή string, για παράδειγμα. Οι annotation properties μπορούν να χρησιμοποιηθούν για να προσθέσουν μεταδεδομένα σε συστατικά της οντολογίας, όπως κλάσεις, άτομα ή ιδιότητες. Στο Properties Tab, ο χρήστης έχει τη δυνατότητα να ορίσει ρητά το πεδίο τιμών και το πεδίο ορισμού κάθε ιδιότητας, να δημιουργήσει **αντίστροφες ιδιότητες** (inverse properties), καθώς και να καθορίσει αν μια ιδιότητα είναι **μονότιμη** (functional) ή όχι.



Εικόνα 3.7 Properties Tab

Τα επόμενα δύο tabs είναι το **Forms** και το **Individuals** Tab, τα οποία είναι σχεδόν πανομοιότυπα στη μορφή και τη λειτουργικότητα με τα Forms και Instances Tab αντίστοιχα του απλού Protégé, όπως αυτά περιγράφηκαν παραπάνω. Τέλος, υπάρχει το **Metadata** Tab, το οποίο παρουσιάζει τα μεταδεδομένα της οντολογίας, όπως για παράδειγμα τους χώρους ονομάτων (namespaces). Ακόμα, παρουσιάζονται οι annotation properties, καθώς και τα άτομα της οντολογίας που έχουν δηλωθεί διαφορετικά το ένα από το άλλο.



Εικόνα 3.8 Metadata Tab

Εκτός των παραπάνω, το Protégé OWL Plugin επιτρέπει στο χρήστη να ορίσει κλάσεις με πιο πολύπλοκους τρόπους χρησιμοποιώντας συνθήκες οι οποίες σχηματίζονται με τη βοήθεια συμβόλων **Περιγραφικής Λογικής**. Αυτά τα σύμβολα αντιστοιχούν σε δομικά στοιχεία της OWL που έχουν περιγραφεί σε προηγούμενο κεφάλαιο, όπως για παράδειγμα οι owl:intersectionOf, owl:allValuesFrom, owl:cardinality. Αν μια κλάση περιγράφεται από ικανές (necessary) συνθήκες, τότε αυτό σημαίνει ότι κάθε άτομο που ανήκει σε αυτήν την κλάση ικανοποιεί αυτές τις συνθήκες. Μια τέτοια κλάση λέγεται **πρωταρχική** (primitive) κλάση. Αν όμως μία κλάση περιγράφεται από ικανές και αναγκαίες συνθήκες (necessary & sufficient), τότε κάθε άτομο που ανήκει σε αυτήν την κλάση θα τις ικανοποιεί, ενώ ισχύει και το αντίστροφο, δηλαδή κάθε άτομο που θα ικανοποιεί αυτές τις συνθήκες θα ανήκει στην κλάση. Μια τέτοια κλάση λέγεται **ορισμένη** (defined). Ο ορισμός και η περιγραφή των κλάσεων πραγματοποιείται στο OWLClasses Tab.

Ακόμα, το Protégé OWL Plugin παρέχει απευθείας πρόσβαση σε **μηχανές εξαγωγής συμπερασμάτων** (reasoners) βασισμένων σε Περιγραφική Λογική, όπως ο RACER. Προς το παρόν, υποστηρίζονται δύο τύποι αιτιολογήσεων: ο έλεγχος συνέπειας (consistency checking) και η κατάταξη (classification). Ο έλεγχος συνέπειας, δηλαδή ο έλεγχος αν μία κλάση θα μπορούσε να έχει έστω και ένα άτομο ως μέλος της, μπορεί να εφαρμοστεί για όλη την ιεραρχία των κλάσεων ή για μια επιλεγμένη ομάδα κλάσεων, επιλέγοντας το Check Consistency από το μενού OWL. Η κατάταξη, δηλαδή η δημιουργία ενός υπαγόμενου δέντρου που προκύπτει από τις οντολογικές δηλώσεις, μπορεί να εφαρμοστεί και πάλι είτε για όλη την ιεραρχία των κλάσεων είτε για έναν επιλεγμένο αριθμό κλάσεων, επιλέγοντας το Classify Taxonomy από το μενού OWL.

3.2 JENA

Η Jena [18] είναι ένα πλαίσιο εργασίας για την ανάπτυξη εφαρμογών του Σημασιολογικού Ιστού. Παρέχει ένα προγραμματιστικό περιβάλλον για τις γλώσσες RDF, RDFS και OWL, ενώ περιέχει και μια μηχανή εξαγωγής συμπερασμάτων (inference engine) βασισμένη σε κανόνες.

Η Jena είναι λογισμικό ανοικτού κώδικα (open source) και περιλαμβάνει:

- Ένα RDF API
- Μηχανισμούς ανάγνωσης και εγγραφής RDF σε μορφή RDF/XML, N3 και N-Triples
- Ένα OWL API
- Αποθήκευση στη μνήμη και μόνιμη (persistent) αποθήκευση
- Την RDQL, μια γλώσσα ερωτημάτων για μοντέλα RDF

3.2.1 Jena RDF API

Η Jena είναι ένα API της Java που μπορεί να χρησιμοποιηθεί για τη **δημιουργία και χειρισμό RDF γράφων**. Η Jena περιέχει κλάσεις για την αναπαράσταση γράφων, πόρων, ιδιοτήτων και κυριολεκτικών (literals). Οι διασυνδέσεις (interfaces) που αναπαριστούν τους πόρους, τις ιδιότητες και τα κυριολεκτικά λέγονται Resource, Property και Literal αντίστοιχα. Στη Jena, ο γράφος είναι γνωστός και ως μοντέλο και αναπαρίσταται από τη διασύνδεση Model. Ένα μοντέλο RDF είναι ουσιαστικά ένα σύνολο δηλώσεων (statements). Κάθε δήλωση βεβαιώνει ένα γεγονός σχετικό με κάποιον πόρο. Μια δήλωση αποτελείται από τρία μέρη: το υποκείμενο, το κατηγόρημα (ή ιδιότητα) και το αντικείμενο και για αυτό το λόγο αναφέρεται συχνά και ως **τριάδα** (triple).

Η Jena περιέχει μεθόδους για την ανάγνωση και εγγραφή της RDF ως XML, οι οποίες μπορούν να χρησιμοποιηθούν για να αποθηκευθεί ένα RDF μοντέλο σε ένα αρχείο και αργότερα, να διαβαστεί το μοντέλο από αυτό το αρχείο. Επίσης, περιέχει μεθόδους για την πρόσβαση και ανάκτηση πληροφορίας που είναι αποθηκευμένη σε ένα RDF μοντέλο. Ακόμα, η Jena περιέχει κάποιες βασικές μεθόδους για **αναζήτηση μέσα σε ένα RDF μοντέλο** και οι οποίες έχουν σαφώς λιγότερες δυνατότητες από τη γλώσσα RDQL – που περιγράφηκε στο προηγούμενο κεφάλαιο – μια πολύ ισχυρή γλώσσα ερωτημάτων για RDF μοντέλα. Η Jena παρέχει και τρεις λειτουργίες για τον χειρισμό μοντέλων: την **ένωση**, την **τομή** και τη **διαφορά**. Η ένωση δύο μοντέλων είναι η ένωση των συνόλων των δηλώσεων που αναπαριστούν το κάθε μοντέλο. Αυτή η λειτουργία επιτρέπει ουσιαστικά δεδομένα από διαφορετικές πηγές να συγχωνευθούν. Η τομή και η διαφορά ορίζονται με παρόμοιο τρόπο. Τέλος, η Jena μπορεί να χειριστεί και ένα ειδικό είδος πόρων που αντιπροσωπεύουν συλλογές πραγμάτων και ονομάζονται containers. Τα μέλη ενός container μπορεί να είναι είτε πόροι είτε κυριολεκτικά.

Το πακέτο (package) της Jena στο οποίο περιέχονται οι σχετικές με μοντέλα λειτουργίες είναι το com.hp.hpl.jena.rdf.model. Το API έχει βασιστεί σε διασυνδέσεις, έτσι ώστε μια εφαρμογή να μπορεί να τρέχει με διαφορετικές

υλοποιήσεις χωρίς κάποια αλλαγή. Αυτό το πακέτο περιέχει διασυνδέσεις για την αναπαράσταση μοντέλων, πόρων, ιδιοτήτων, κυριολεκτικών, δηλώσεων και όλων των άλλων κεντρικών εννοιών της RDF, όπως επίσης και μια διασύνδεση ModelFactory για τη δημιουργία μοντέλων.

3.2.2 Jena Ontology API

Με δεδομένο το ότι η Jena είναι μια πλατφόρμα της RDF, υποστηρίζει γλώσσες που έχουν αναπτυχθεί με βάση την RDF, όπως την RDFS, τις διάφορες υπογλώσσες της OWL και την DAML+OIL (γλώσσες που έχουν αναφερθεί στο προηγούμενο κεφάλαιο). Συνοπτικά, η RDFS είναι η πλέον ασθενής οντολογική γλώσσα που υποστηρίζεται από τη Jena και επιτρέπει την κατασκευή μιας απλής ιεραρχίας εννοιών και μια ιεραρχία ιδιοτήτων. Η OWL υποστηρίζει όλες τις δυνατότητες που έχει και η RDFS, ενώ εισάγει και πολυπλοκότερα στοιχεία. Στην OWL, τόσο οι κλάσεις όσο και οι ιδιότητες μπορούν να οριστούν με πιο σύνθετους τρόπους. Όσον αφορά στην DAML+OIL, αυτή μοιάζει σε πάρα πολλά σημεία με την OWL Full και διαθέτει σχεδόν την ίδια εκφραστικότητα, κάτι πολύ λογικό εφόσον σε αυτή βασίστηκε η ανάπτυξη της OWL.

Το Ontology API της Jena είναι το αποτέλεσμα της απαίτησης να εξασφαλίζεται η δυνατότητα χειρισμού πολλών οντολογικών γλωσσών και της υποστήριξης των αυξημένων δυνατοτήτων εξαγωγής συμπερασμάτων (inference) σε όλες αυτές τις γλώσσες. Για αυτό το λόγο, το Ontology API είναι ανεξάρτητο της γλώσσας. Για να επιτευχθεί αυτό, κάθε μία από τις γλώσσες έχει ένα πορτραίτο (profile), που απαριθμεί τα επιτρεπόμενα δομικά συστατικά, καθώς και τα URIs των κλάσεων και των ιδιοτήτων.

Κάθε πορτραίτο είναι συνδεδεμένο με ένα μοντέλο οντολογίας, το οποίο είναι μια επέκταση της κλάσης Model της Jena. Αυτή η γενική κλάση επιτρέπει πρόσβαση σε όλες τις δηλώσεις μιας συλλογής RDF δεδομένων. Η κλάση OntModel επεκτείνει αυτήν την κλάση προσθέτοντας υποστήριξη για όλα τα αντικείμενα που αναμένονται να βρίσκονται σε μια οντολογία, όπως κλάσεις, ιδιότητες και άτομα (individuals). Οι ιδιότητες που ορίζονται σε μια οντολογία αντιστοιχούν κατά κάποιο τρόπο σε μεθόδους πρόσβασης της Java. Για παράδειγμα, η κλάση OntClass περιγράφει μια κλάση της οντολογίας και έχει μια μέθοδο, τη listSuperClasses() για την απαρίθμηση των υπερκλάσεων μιας συγκεκριμένης κλάσης. Κάτι που πρέπει να τονιστεί είναι το γεγονός ότι καμιά πληροφορία δεν αποθηκεύεται στο αντικείμενο OntClass. Όταν κληθεί η μέθοδος listSuperClasses(), η πληροφορία ανακτάται από τις υποκείμενες της οντολογίας RDF δηλώσεις.

Οι δηλώσεις που είναι ορατές από τα αντικείμενα της οντολογίας στην Java αποτελούνται τόσο από τις βασικές δηλώσεις που βρίσκονται στον υποκείμενο της οντολογίας RDF γράφο όσο και από τις δηλώσεις που μπορούν να επαχθούν από τη χρήση ενός reasoner (μηχανής εξαγωγής συμπερασμάτων). Οι βασικές δηλώσεις είναι αποθηκευμένες στο βασικό γράφο, ο οποίος παριστάνεται από τη διασύνδεση Graph. Ο reasoner μπορεί να χρησιμοποιήσει τα περιεχόμενα του βασικού γράφου και τους σημασιολογικούς κανόνες της οντολογικής γλώσσας για να εμφανίσει ένα πληρέστερο σύνολο δηλώσεων, περιλαμβανομένων και αυτών που συνεπάγονται από

τις βασικές δηλώσεις. Αυτό το σύνολο επίσης παριστάνεται με τη διασύνδεση Graph, οπότε το μοντέλο λειτουργεί μόνο με αυτή τη διασύνδεση. Αυτό επιτρέπει την κατασκευή μοντέλων με ή χωρίς reasoner, χωρίς να χρειαστεί αλλαγή του οντολογικού μοντέλου.

Ο πολυμορφισμός στην RDF, δηλαδή το γεγονός ότι ένας πόρος μπορεί να θεωρηθεί, για παράδειγμα, και ως κλάση και ως περιορισμός, δεν επιτρέπει μια συνεπή ή μοναδική απεικόνιση μεταξύ ενός πόρου και μιας Java κλάσης. Η Jena δέχεται αυτό το βασικό χαρακτηριστικό του πολυμορφισμού στο επίπεδο της RDF θεωρώντας ότι μια κλάση της Java (είτε αυτή είναι η OntClass, είτε η Restriction είτε η DatatypeProperty) είναι απλά μια όψη (facet) του πόρου. Αυτό προσφέρει τη δυνατότητα στον προγραμματιστή να αναβάλλει τις αποφάσεις σχετικά με το ποια κλάση της Java θα χρησιμοποιηθεί μέχρι την εκτέλεση, ενώ αν δεν υποστηρίζεται η μετατροπή σε κάποια όψη, τότε θα εγερθεί μια εξαίρεση.

Σε γενικές γραμμές, το Jena Ontology API προσφέρει τις παρακάτω δυνατότητες:

Δημιουργία Οντολογικών Μοντέλων

Ένα οντολογικό μοντέλο είναι μια επέκταση του RDF μοντέλου της Jena που παρέχει επιπλέον δυνατότητες για το χειρισμό οντολογικών δεδομένων. Η Jena προσφέρει τη δυνατότητα δημιουργίας μοντέλου για συγκεκριμένη γλώσσα, οπότε στη μέθοδο createOntologyModel() της κλάσης ModelFactory θα πρέπει να τεθεί ως όρισμα το αντίστοιχο URI. Επίσης, επιτρέπεται ο καθορισμός και άλλων λεπτομερειών, όπως η χρήση reasoner.

Χειρισμός Οντολογικών Εγγράφων

Η Jena παρέχει μια σειρά μεθόδων για ανάγνωση μιας οντολογίας από ένα αρχείο και φόρτωσή της μέσα σε ένα οντολογικό μοντέλο. Εκτός από την ανάγνωση εγγράφων, ένα οντολογικό μοντέλο προσφέρει περισσότερες δυνατότητες, όπως η δυνατότητα στην OWL και στην DAML+OIL οι οντολογίες να χρησιμοποιηθούν ως επαναχρησιμοποιήσιμα στοιχεία και να εισαχθούν σε διαφορετικά έγγραφα. Πολύ σημαντικό είναι το γεγονός ότι κάθε οντολογικό έγγραφο που εισάγεται σε κάποιο άλλο κρατιέται σε διαφορετικό γράφο, έτσι ώστε να είναι γνωστή η προέλευση όλων των δηλώσεων.

Χειρισμός Οντολογικών Συστατικών

Οι κλάσεις είναι τα βασικά δομικά στοιχεία μιας οντολογίας. Μια κλάση της οντολογίας είναι η όψη ενός RDF πόρου. Η κλάση της Jena που αναπαριστά μια κλάση της οντολογίας είναι η OntClass και αυτή, με τη σειρά της, παρέχει έναν αριθμό μεθόδων για την επεξεργασία μιας κλάσης. Ακόμα, μια ιδιότητα σε ένα οντολογικό μοντέλο παριστάνεται από την κλάση OntProperty, η οποία είναι μια επέκταση της κλάσης Property της Java και η οποία επιτρέπει πρόσβαση στην πρόσθετη πληροφορία που μπορεί να δηλωθεί για μια ιδιότητα σε ένα οντολογικό μοντέλο. Ακόμα, η OWL και η DAML+OIL παρέχουν περισσότερα δομικά στοιχεία για τον ορισμό ιδιοτήτων, περιορισμών, λογικών πράξεων μεταξύ κλάσεων και ομαδοποίησης στοιχείων.

3.3 ΣΥΣΤΗΜΑΤΑ ΔΙΑΧΕΙΡΙΣΗΣ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

3.3.1 MySQL

3.3.1.1 Εισαγωγή

Η MySQL [35][36] είναι ένα πολύ γρήγορο και σταθερό σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (Relational Database Management System - **RDBMS**). Μία βάση δεδομένων μπορεί με αποτελεσματικότητα να αποθηκεύει, να ψάχνει, να ταξινομεί και να επιλέγει τα δεδομένα. Η MySQL ελέγχει την πρόσβαση στα δεδομένα και επιτρέπει σε πολλαπλούς χρήστες να δουλεύουν συγχρόνως με πολύ γρήγορη πρόσβαση και απαγορεύει σε μη εξουσιοδοτημένους χρήστες να έχουν πρόσβαση σε αυτήν. Αυτό σημαίνει ότι η MySQL είναι πολλαπλών χρηστών - πολλαπλών διεργασιών. Χρησιμοποιεί την SQL (Structured Query Language), που είναι το πρότυπο για τις επερωτήσεις (queries) προς τις βάσεις δεδομένων. Η MySQL διανεμήθηκε για πρώτη φορά το 1996 αλλά η ιστορία της ανάπτυξής της έχει ξεκινήσει από το 1979. Τέλος, η MySQL έχει άδεια ανοιχτού λογισμικού (open source licence), αλλά υπάρχουν και εμπορικές άδειες χρήσης εάν χρειάζονται.

3.3.1.2 Δυνατότητες

Η MySQL είναι ένα σύστημα διαχείρισης βάσεων δεδομένων η οποία ακολουθεί το **σχεσιακό μοντέλο** (relational) και είναι συμβατή με ANSI-SQL. Η έκδοση MySQL Server είναι κατάλληλη για τη διαχείριση μεγαλύτερων βάσεων δεδομένων και την παρουσίαση δεδομένων σε web site στο Internet. Παρέχει υψηλή απόδοση και ασφάλεια. Επίσης, η MySQL μπορεί να χρησιμοποιηθεί σε ένα μεγάλο αριθμό λειτουργικών συστημάτων (Windows, Linux, AIX, Solaris κ.ά.).

Η MySQL υστερεί σε εφαρμογές με πολύπλοκες σχεσιακές βάσεις. Παράλληλα απουσιάζει υποστήριξη για αποθηκευμένες διαδικασίες (stored procedures, event procedures της Access) καθώς και για επεξεργασία δοσοληψιών (transaction processing).

Τα σημαντικότερα πλεονεκτήματα της MySQL σημειώνονται παρακάτω:

- **Επίδοση.** Η MySQL είναι χωρίς καμία αμφισβήτηση από τις πιο γρήγορες βάσεις δεδομένων. Εδώ βρίσκεται και ένα μειονέκτημα, η MySQL δεν υποστηρίζει κάποιες από τις σημαντικές λειτουργίες άλλων βάσεων δεδομένων, αυτές είναι οι stored procedures, τα triggers και το referential integrity. Έτσι όταν δεν πρόκειται για μεγάλου μεγέθους εφαρμογές όπως σύστημα διαχείρισης δεδομένων σε τράπεζα, τότε η MySQL είναι η ιδανικότερη.

- **Χαμηλό κόστος.** Η MySQL δεν κοστίζει τίποτα, έχει άδεια ανοιχτού λογισμικού (open source license) ή μικρό κόστος χρησιμοποιώντας μία εμπορική άδεια χρήσης.
- **Εύκολη στη χρήση.** Οι περισσότερες μοντέρνες βάσεις δεδομένων χρησιμοποιούν το πρότυπο SQL. Σημειώνεται ότι η MySQL είναι πολύ εύκολη στις ρυθμίσεις της σχετικά με άλλες βάσεις δεδομένων.
- **Υποστήριξη από πολλές πλατφόρμες**. Η MySQL μπορεί να τρέξει σε πάρα πολλά λειτουργικά συστήματα. Μεταξύ άλλων αυτά είναι το Linux, FreeBSD, NetBSD, Solaris, Microsoft Windows.
- **Ελεύθερος Πηγαίος Κώδικας προς το κοινό**. Στη MySQL μπορεί οποιοσδήποτε να χρησιμοποιήσει, αλλάξει, διανείμει των πηγαίο κώδικα της MySQL.

Τέλος, είναι πιο κατάλληλη για χρήση στο Internet (Internet Ready), είναι ιδιαίτερα βελτιστοποιημένη για ταχύτητα στην ανάκτηση δεδομένων και παρέχει ευκολίες στο backup.

3.3.1.3 Χωρικά Δεδομένα στη MySQL

Η έκδοση MySQL 4.1 εισάγει κάποιες επεκτάσεις για χωρικά δεδομένα που επιτρέπουν τη δημιουργία, αποθήκευση και ανάλυση γεωγραφικών χαρακτηριστικών. Αυτή τη στιγμή τα εν λόγω χαρακτηριστικά είναι διαθέσιμα μόνο για πίνακες τύπου MyIsam.

Η MySQL υλοποιεί την επέκταση στα χωρικά δεδομένα ακολουθώντας τις προδιαγραφές του Open GIS Consortium (OGC). Το OGC είναι μια διεθνής συνεργασία 250 εταιρειών, πρακτόρων και πανεπιστημίων που συμμετέχουν στην ανάπτυξη των δημοσίως διαθέσιμων εννοιολογικών λύσεων που σχετίζονται με όλες τις εφαρμογές που χειρίζονται χωρικά δεδομένα.

Το 1997, το Open GIS Consortium δημοσίευσε τα *OpenGIS® Simple Features Specifications For SQL,* ένα έγγραφο που προτείνει διάφορους εννοιολογικούς τρόπους για την επέκταση ενός σχεσιακού συστήματος διαχείρισης βάσεων δεδομένων (SQL RDBMS) ώστε να υποστηρίξει χωρικά δεδομένα.

Η MySQL υλοποιεί ένα υποσύνολο του περιβάλλοντος **Επαυξημένης SQL με Γεωμετρικούς Τύπους (SQL with Geometry Types)** που προτείνεται από το OGC. Ο όρος αυτός αναφέρεται σε ένα περιβάλλον SQL το οποίο επεκτείνεται με ένα σύνολο γεωμετρικών τύπων. Μία γεωμετρικά-αποτιμημένη στήλη SQL (geometry-valued SQL column) υλοποιείται ως μία στήλη που έχει γεωμετρικό τύπο. Η MySQL έχει ένα σύνολο γεωμετρικών τύπων, όπως επίσης και συναρτήσεις πάνω σε αυτούς τους τύπους για τη δημιουργία και ανάλυση των γεωμετρικών τιμών.

Οι γεωμετρικοί τύποι της MySQL αντιστοιχούν στις κλάσεις του OpenGIS. Παρακάτω παρατίθενται αυτοί οι οποίοι αποθηκεύουν απλές γεωμετρικές τιμές:

- GEOMETRY
- POINT

- LINESTRING
- POLYGON

Ο τύπος Geometry μπορεί να αποθηκεύσει γεωμετρικές τιμές οποιουδήποτε τύπου, ενώ οι Point, Linestring, Polygon περιορίζουν τις τιμές τους σε κάποιον συγκεκριμένο τύπο.

Οι υπόλοιποι γεωμετρικοί τύποι αποθηκεύουν συλλογές από τιμές (collections of values):

- MULTIPOINT
- MULTILINESTRING
- MULTIPOLYGON
- GEOMETRYCOLLECTION

Ο τύπος GeometryCollection μπορεί να αποθηκεύσει συλλογές από αντικείμενα οποιουδήποτε τύπου, ενώ οι Multipoint, Multilinestring, Multipolygon περιορίζουν τις τιμές τους σε αντικείμενα που έχουν κάποιον συγκεκριμένο τύπο.

Τέλος, παραθέτουμε την ιεραρχία που ακολουθούν οι γεωμετρικές κλάσεις:

- Geometry (non-instantiable)
    - Point (instantiable)
    - Curve (non-instantiable)
        - LineString (instantiable)
            - Line
            - LinearRing
    - Surface (non-instantiable)
        - Polygon (instantiable)
    - GeometryCollection (instantiable)
        - MultiPoint (instantiable)
        - MultiCurve (non-instantiable)
            - MultiLineString (instantiable)
        - MultiSurface (non-instantiable)
            - MultiPolygon (instantiable)

Δεν είναι δυνατή η δημιουργία ενός αντικειμένου από μια κλάση non-instantiable, δηλαδή μια κλάση που δε δίνει στιγμιότυπα, αλλά είναι δυνατή η δημιουργία ενός αντικειμένου από μια κλάση instantiable. Όλες οι κλάσεις έχουν ιδιότητες (properties) και μάλιστα αυτές που δίνουν στιγμιότυπα (instantiable) έχουν και ισχυρισμούς (assertions).

Η βασική κλάση είναι η κλάση **Geometry**, η οποία είναι αφηρημένη. Οι υποκλάσεις της Geometry που δίνουν στιγμιότυπα (instantiable) περιορίζονται σε αντικείμενα μηδενικής, μίας ή δύο διαστάσεων που υπάρχουν στο δισδιάστατο χώρο συντεταγμένων. Όλες οι instantiable γεωμετρικές κλάσεις ορίζονται ώστε έγκυρα στιγμιότυπα μιας γεωμετρικής κλάσης να θεωρούνται όσα είναι τοπολογικά κλειστά.

Η βασική κλάση Geometry έχει υποκλάσεις τις Point, Curve, Surface και GeometryCollection:

- Η **Point** αναπαριστά αντικείμενα μηδενικής διάστασης
- Η **Curve** αναπαριστά αντικείμενα μίας διάστασης και έχει υποκλάση την LineString με υποκλάσεις τις Line και LinearRing.
- Η **Surface** είναι σχεδιασμένη για αντικείμενα δύο διαστάσεων και υποκλάση την Polygon.
- Η **GeometryCollection** έχει εξειδικευμένες μηδενικής, μίας ή δύο διαστάσεων κλάσεις συλλογών (collection classes) με ονόματα MultiPoint, MultiLineString και MultiPolygon για τη μοντελοποίηση των γεωμετριών που αντιστοιχούν σε συλλογές από Points, LineStrings και Polygons αντίστοιχα. Οι MultiCurve και MultiSurface εισάγονται ως αφηρημένες υπερκλάσεις που γενικεύουν τη συλλογή των interfaces για τη διαχείριση Curves and Surfaces.

Τέλος, σημειώνεται ότι οι Geometry, Curve, Surface, MultiCurve και MultiSurface ορίζονται ως κλάσεις που δε δίνουν στιγμιότυπα (non-instantiable classes), ενώ οι Point, LineString, Polygon, GeometryCollection, MultiPoint, MultiLineString και MultiPolygon είναι κλάσεις που δίνουν στιγμιότυπα (instantiable classes).

3.3.2 PostgreSQL

3.3.2.1 Εισαγωγή

Η PostgreSQL [37] είναι ένα **αντικείμενο-σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων** και προέρχεται από το πακέτο της POSTGRES που υλοποιήθηκε στο πανεπιστήμιο της Καλιφόρνιας, Berkeley. Με μια δεκαετία ανάπτυξης πριν από την εμφάνιση της, και με πάνω από 350.000 γραμμές κώδικα, η PostgreSQL είναι τώρα η πιο προηγμένη βάση δεδομένων ελεύθερου λογισμικού υποστηρίζοντας τις προγενέστερες SQL92, SQL99 και SQL2003.

3.3.2.2 Ιστορία

Κατά το 1996, έγινε σαφές ότι το όνομα «Postgres95», το οποίο είχε υιοθετηθεί αρχικά, δε θα άντεχε με το πέρασμα του χρόνου. Για το λόγο αυτό, αλλά και προκειμένου να απεικονιστεί η σχέση μεταξύ της αρχικής POSTGRES και των πιο πρόσφατων εκδόσεων της που χρησιμοποιούν την SQL επιλέχτηκε το όνομα PostgreSQL . Η αρίθμηση έκδοσης ξεκινάει από το 6.0, με την αρχή να γίνεται από την εργασία POSTGRES του πανεπιστημίου Berkeley.

Η έμφαση κατά τη διάρκεια της ανάπτυξης των εκδόσεων v1.0.x της Postgres95 ήταν στην σταθεροποίηση του πίσω μέρους (backend) του κώδικα. Με τη σειρά εκδόσεων v6.x της PostgreSQL η έμφαση μεταφέρθηκε από τον προσδιορισμό και την κατανόηση των υπαρχόντων προβλημάτων στον κώδικα του πίσω μέρους, στην προσπάθεια αύξησης των χαρακτηριστικών γνωρισμάτων και των ικανοτήτων, αν και η εργασία συνεχίζεται διεξοδικά σε όλες τις περιοχές.

Σημαντικές **προσαυξήσεις** περιλαμβάνουν:

- Σημαντικά χαρακτηριστικά γνωρίσματα του πίσω μέρους, συμπεριλαμβανομένου εμφωλευμένων SELECT, προκαθορισμένων (default) τιμών, περιορισμούς, και σκανδαλιστές (triggers).
- Επιπρόσθετα SQL92-συμβατά γλωσσικά χαρακτηριστικά προστέθηκαν, όπως πρωταρχικά κλειδιά, αναγνωριστικά μέσα σε εισαγωγικά, μετατροπή τύπων, και δυαδική αλλά και δεκαεξαδική ακέραια είσοδος.
- Οι ενσωματωμένοι τύποι έχουν βελτιωθεί και παρέχουν νέους – ευρείας κλίμακας – τύπους ημερομηνίας και χρόνου. Επιπλέον, επιτυγχάνεται η υποστήριξη γεωμετρικών τύπων.
- Συνολικά, η ταχύτητα του κώδικα του πίσω μέρους έχει αυξηθεί κατά 20% και η ταχύτητα εκκίνησής του έχει μειωθεί κατά 80%.

3.3.2.3 Γεωμετρικοί τύποι στην PostgreSQL

Οι γεωμετρικοί τύποι στην PostgreSQL αναπαριστούν χωρικά αντικείμενα δύο διαστάσεων (two-dimensional spatial objects). Παρακάτω φαίνονται οι διαθέσιμοι γεωμετρικοί τύποι στην PostgreSQL:

Όνομα	Μέγεθος Αποθήκευσης	Περιγραφή	Αναπαράσταση
`point`	16 bytes	Σημείο στο επίπεδο	(x,y)
`line`	32 bytes	Άπειρη γραμμή (όχι πλήρως υλοποιημένο)	((x1,y1),(x2,y2))
`lseg`	32 bytes	Πεπερασμένο ευθύγραμμο τμήμα	((x1,y1),(x2,y2))
`box`	32 bytes	Ορθογώνιο Παραλληλεπίπεδο	((x1,y1),(x2,y2))
`path`	16+16n bytes	Κλειστό μονοπάτι (παρόμοιο με τον τύπο polygon)	((x1,y1),...)
`path`	16+16n bytes	Ανοιχτό μονοπάτι	[(x1,y1),...]
`polygon`	40+16n bytes	Πολύγωνο (παρόμοιο με το closed path)	((x1,y1),...)
`circle`	24 bytes	Κύκλος	<(x,y),r> (κέντρο και ακτίνα)

Σημεία (Points)

Ο πιο θεμελιώδης τύπος είναι το σημείο (point) και αποτελεί τη βάση για τους υπόλοιπους γεωμετρικούς τύπους. Οι τιμές των σημείων δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
(x , y)
x , y
όπου x και y είναι οι συντεταγμένες του σημείου ως αριθμοί κινητής υποδιαστολής.

Ευθύγραμμα τμήματα (Line segments)

Τα ευθύγραμμα τμήματα (lseg) αναπαρίστανται από ζεύγη σημείων. Οι τιμές των ευθυγράμμων τμημάτων δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
((x1 , y1) , (x2 , y2))
(x1 , y1) , (x2 , y2)
x1 , y1 , x2 , y2
όπου (x1,y1) και (x2,y2) είναι οι άκρες του ευθύγραμμου τμήματος.

Κουτιά (Boxes)

Τα κουτιά αναπαρίστανται από τα ζεύγη σημείων που βρίσκονται στις απέναντι γωνίες του κουτιού. Οι τιμές των κουτιών δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
((x1 , y1) , (x2 , y2))
(x1 , y1) , (x2 , y2)
x1 , y1 , x2 , y2
όπου (x1,y1) και (x2,y2) είναι οποιεσδήποτε απέναντι γωνίες του κουτιού.

Μονοπάτια (Paths)

Τα μονοπάτια αναπαρίστανται ως λίστες συνδεδεμένων σημείων. Τα μονοπάτια μπορεί να είναι *ανοιχτά*, όπου το πρώτο με το τελευταίο στοιχείο της λίστας δεν είναι συνδεδεμένα, ή *κλειστά*, όπου το πρώτο με το τελευταίο στοιχείο της λίστας είναι συνδεδεμένα. Οι τιμές των μονοπατιών δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
((x1 , y1) , ... , (xn , yn))
[(x1 , y1) , ... , (xn , yn)]
(x1 , y1) , ... , (xn , yn)
(x1 , y1 , ... , xn , yn)
x1 , y1 , ... , xn , yn
όπου τα σημεία είναι τα άκρα των ευθυγράμμων τμημάτων που συνθέτουν το μονοπάτι. Οι αγκύλες δηλώνουν ανοιχτό μονοπάτι, ενώ οι παρενθέσεις κλειστό.

Πολύγωνα (Polygons)

Τα πολύγωνα αναπαρίστανται ως λίστες σημείων (οι κορυφές των πολυγώνων). Θα μπορούσαν να θεωρούνται ως κλειστά μονοπάτια αλλά αποθηκεύονται με διαφορετικό τρόπο και έχουν τις δικές τους ρουτίνες που τα υποστηρίζουν.
Οι τιμές των μονοπατιών δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
((x1 , y1) , ... , (xn , yn))
(x1 , y1) , ... , (xn , yn)
(x1 , y1 , ... , xn , yn)
x1 , y1 , ... , xn , yn
όπου τα σημεία είναι τα άκρα των ευθυγράμμων τμημάτων που συνθέτουν το περίγραμμα του πολυγώνου.

Κύκλοι (Circles)

Οι κύκλοι αναπαρίστανται από ένα σημείο για το κέντρο και μια ακτίνα. Οι τιμές των μονοπατιών δηλώνονται σύμφωνα με την ακόλουθη σύνταξη:
< (x , y) , r >
((x , y) , r)
(x , y) , r
x , y , r
όπου (x,y) είναι το κέντρο του κύκλου και r η ακτίνα του.

Οι παραπάνω γεωμετρικοί τύποι έχουν ένα μεγάλο σύνολο συναρτήσεων και τελεστών που τους υποστηρίζουν. Παρακάτω φαίνονται οι **τελεστές**:

Τελεστής	**Περιγραφή**	**Παράδειγμα**
+	Μετατόπιση	box '((0,0),(1,1))' + point '(2.0,0)'
-	Μετατόπιση	box '((0,0),(1,1))' - point '(2.0,0)'
*	Κλιμάκωση / Περιστροφή	box '((0,0),(1,1))' * point '(2.0,0)'
/	Κλιμάκωση / Περιστροφή	box '((0,0),(2,2))' / point '(2.0,0)'
#	Σημείο ή κουτί τομής	'((1,-1),(-1,1))' # '((1,1),(-1,-1))'
#	Αριθμός σημείων σε μονοπάτι ή πολύγωνο	# '((1,0),(0,1),(-1,0))'
@-@	Μήκος ή περιφέρεια	@-@ path '((0,0),(1,0))'
@@	Κέντρο	@@ circle '((0,0),10)'
##	Σημείο του πρώτου τελεστή που είναι πλησιέστερο στο δεύτερο	point '(0,0)' ## lseg '((2,0),(0,2))'
<->	Απόσταση μεταξύ	circle '((0,0),1)' <-> circle '((5,0),1)'
&&	Υπάρχει επικάλυψη;	box '((0,0),(1,1))' && box '((0,0),(2,2))'
&<	Εκτείνεται στα δεξιά του...;	box '((0,0),(1,1))' &< box '((0,0),(2,2))'
&>	Εκτείνεται στα αριστερά του...;	box '((0,0),(3,3))' &> box '((0,0),(2,2))'
<<	Είναι αριστερά από...;	circle '((0,0),1)' << circle '((5,0),1)'
>>	Είναι δεξιά από...;	circle '((5,0),1)' >> circle '((0,0),1)'
<^	Είναι κάτω από...;	circle '((0,0),1)' <^ circle '((0,5),1)'
>^	Είναι πάνω από...;	circle '((0,5),1)' >^ circle '((0,0),1)'
?#	Τέμνει;	lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))'
?-	Είναι οριζόντιο;	?- lseg '((-1,0),(1,0))'
?-	Είναι ευθυγραμμισμένα οριζοντίως;	point '(1,0)' ?- point '(0,0)'
?|	Είναι κάθετο;	?| lseg '((-1,0),(1,0))'
?|	Είναι ευθυγραμμισμένα καθέτως;	point '(0,1)' ?| point '(0,0)'
?-|	Είναι κάθετα μεταξύ τους;	lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))'
?||	Είναι παράλληλα μεταξύ τους;	lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))'
~	Περιέχει;	circle '((0,0),2)' ~ point '(1,1)'

Τελεστής	Περιγραφή	Παράδειγμα
@	Περιέχεται σε...;	point '(1,1)' @ circle '((0,0),2)'
~=	Είναι ίδιο με...;	polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))'

Τέλος φαίνονται οι **συναρτήσεις**:

Συνάρτηση	Επιστρεφόμενος Τύπος	Περιγραφή	Παράδειγμα
area(*object*)	double precision	Εμβαδόν	area(box '((0,0),(1,1))')
box_intersect(box, box)	box	Κουτί τομής	box_intersect(box '((0,0),(1,1))',box '((0.5,0.5),(2,2))')
center(*object*)	point	Κέντρο	center(box '((0,0),(1,2))')
diameter(circle)	double precision	Διάμετρος κύκλου	diameter(circle '((0,0),2.0)')
height(box)	double precision	Κάθετο μήκος κουτιού	height(box '((0,0),(1,1))')
isclosed(path)	boolean	Είναι κλειστό μονοπάτι;	isclosed(path '((0,0),(1,1),(2,0))')
isopen(path)	boolean	Είναι ανοιχτό μονοπάτι;	isopen(path '[(0,0),(1,1),(2,0)]')
length(*object*)	double precision	Μήκος	length(path '((-1,0),(1,0))')
npoints(path)	integer	Αριθμός σημείων	npoints(path '[(0,0),(1,1),(2,0)]')
npoints(polygon)	integer	Αριθμός σημείων	npoints(polygon '((1,1),(0,0))')
pclose(path)	path	Μετατροπή μονοπατιού σε κλειστό	pclose(path '[(0,0),(1,1),(2,0)]')
popen(path)	path	Μετατροπή μονοπατιού σε ανοιχτό	popen(path '((0,0),(1,1),(2,0))')
radius(circle)	double precision	Ακτίνα κύκλου	radius(circle '((0,0),2.0)')
width(box)	double precision	Οριζόντιο μήκος κουτιού	width(box '((0,0),(1,1))')

4 ΑΝΤΙΣΤΟΙΧΙΣΗ ΟΝΤΟΛΟΓΙΩΝ

4.1 ΕΙΣΑΓΩΓΗ

Με την καθιέρωση της γλώσσας οντολογίας Ιστού OWL ως σύσταση για την αναπαράσταση των οντολογιών στον Ιστό, έγινε ένα σημαντικό βήμα προς την υλοποίηση του Σημασιολογικού Ιστού. Οι οντολογίες που αναπαρίστανται με την OWL μπορούν να χρησιμοποιηθούν για να επιτευχθεί διαλειτουργικότητα (inter-operability) μεταξύ διαφόρων εφαρμογών του Σημασιολογικού Ιστού. Εντούτοις, στα πλαίσια ενός ανοικτού περιβάλλοντος όπως ο Ιστός, είναι πολύ πιθανό ότι θα υπάρξουν πολλές οντολογίες οι οποίες θα μοιράζονται από πολλούς. Έτσι, θα υπάρξουν **ετερογενείς οντολογίες με επικαλυπτόμενες περιοχές-πεδία**.

Προκειμένου να αντιμετωπιστεί αυτή η ετερογένεια (heterogeneity) μεταξύ των οντολογιών στο Σημασιολογικό Ιστό και στις σχέσεις μεταξύ των οντολογιών υπάρχει **ανάγκη για αντιστοίχιση μεταξύ οντολογιών** ώστε να επιτευχθεί η διαλειτουργικότητα [38].

Η γλώσσα OWL προσφέρει σε περιορισμένο βαθμό αυτή τη δυνατότητα μέσω του owl:import το οποίο χρησιμοποιείται για την εισαγωγή μιας οντολογίας σε μια άλλη. Ο μηχανισμός αυτός που προσφέρει η OWL δεν είναι επαρκής για την πραγματοποίηση αντιστοίχισης μεταξύ οντολογιών στη γενική περίπτωση. Για το λόγο αυτό αναπτύσσονται αυτήν την περίοδο κάποια εργαλεία που έχουν τη δυνατότητα να **απεικονίζουν μία οντολογία σε μία άλλη**, καθώς και **γλώσσες για αντιστοίχιση οντολογιών** (ontology mapping languages).

Ιδιαίτερα σημαντική είναι και η **σημασιολογική διάθεση των βάσεων δεδομένων** ώστε να υπάρχει ένας αυτοματοποιημένος τρόπος για την έκφραση της δομής και της σημασιολογίας τους. Κάτι τέτοιο είναι εφικτό είτε με την αυτόματη δημιουργία οντολογιών οι οποίες αντιστοιχούνται στο περιεχόμενο μιας βάσης δεδομένων και μπορούν να το καταστήσουν διαθέσιμο, είτε με την **αντιστοίχιση δεδομένων από τη βάση σε κάποια ήδη υπάρχουσα οντολογία**

4.2 ΣΧΕΤΙΚΕΣ ΕΡΓΑΣΙΕΣ – ΕΡΓΑΛΕΙΑ

Η **αντιστοίχιση οντολογιών** (ontology mapping) φαίνεται ότι μπορεί να παρέχει λύση στο υπάρχον τοπίο της οντολογικής έρευνας. Όσο ο αριθμός των οντολογιών που δημοσιοποιούνται και είναι προσβάσιμες στο Διαδίκτυο αυξάνεται, τόσο αυξάνεται και ο αριθμός των εφαρμογών που θέλουν να τις χρησιμοποιήσουν. Μια μοναδική οντολογία δεν επαρκεί πλέον ώστε να υποστηρίξει τις στοιχειώδεις εργασίες που προβλέπονται από ένα κατανεμημένο περιβάλλον όπως ο Σημασιολογικός Ιστός.

Η αντιστοίχιση οντολογιών θα μπορούσε να παρέχει ένα κοινό στρώμα από το οποίο οι διάφορες οντολογίες θα μπορούσαν να προσπελαστούν και ως εκ τούτου θα μπορούσαν να ανταλλάξουν πληροφορίες με σημασιολογικό τρόπο.

Η ανάπτυξη τέτοιων αντιστοιχίσεων είναι στο επίκεντρο πολλών εργασιών που προέρχονται από διαφορετικές κοινότητες. Παρακάτω παρουσιάζονται αυτές οι εργασίες [39].

Γίνεται αναφορά στην μέθοδο **FCAMerge** για συγχώνευση οντολογιών (Stumme and Maedche 2001), στην μέθοδο **IF-Map** για αντιστοίχιση οντολογιών (Kalfoglou and Schorlemmer 2002), στα εργαλεία **PROMPT** και **PROMPTDIFF** του Protégé (Noy and Musen), στο εργαλείο **Chimaera** (McGuinness et al. 2000), στο σύστημα **GLUE** (Doan et al. 2002), στο σύστημα **CAIMAN** (Lacher and Groh 2001) και στο σύστημα **ONION** (Mitra and Wiederhold 2002).

4.2.1 FCAMerge

Οι Stumme και Maedche (Stumme and Maedche 2001) παρουσίασαν την μέθοδο **FCAMerge** για συγχώνευση οντολογιών. Οι συγγραφείς ενσωματώνουν **τεχνικές φυσικής γλώσσας** στη μέθοδο αυτή ώστε να παράγουν ένα πλέγμα (lattice) από έννοιες. Κατόπιν, το πλέγμα διατρέχεται χειροκίνητα από ένα μηχανικό γνώσης (knowledge engineer) ο οποίος δημιουργεί την οντολογία συγχώνευσης με την ημι-αυτόματη καθοδήγηση της FCAMerge. Συγκεκριμένα, η FCAMerge λειτουργεί ως εξής: είσοδος στη μέθοδο είναι ένα σύνολο εγγράφων από τα οποία αντλούνται οι έννοιες και οι οντολογίες προς συγχώνευση. Τα έγγραφα αυτά πρέπει να είναι αντιπροσωπευτικά του ζητούμενου τομέα και να είναι συσχετισμένα με τις οντολογίες. Επίσης πρέπει να καλύπτουν όλες τις έννοιες και από τις δύο οντολογίες. Οι ισχυρισμοί αυτοί πρέπει να ικανοποιούνται αυστηρά για τη σωστή εξαγωγή συμπερασμάτων από την FCAMerge. Παρόλο που η μέθοδος στηρίζεται στην ύπαρξη διαθέσιμων ταξινομημένων στιγμιοτύπων (classified instances) στις οντολογίες προς συγχώνευση, οι συγγραφείς υποστηρίζουν ότι κάτι τέτοιο δεν ισχύει στις περισσότερες περιπτώσεις, οπότε τάσσονται υπέρ της **εξαγωγής στιγμιότυπων** από τα έγγραφα:

«*Η εξαγωγή των στιγμιοτύπων από τα έγγραφα κειμένου παρακάμπτει το πρόβλημα που εντοπίζεται στις περισσότερες εφαρμογές όπου δεν υπάρχουν αντικείμενα που να*

είναι ταυτόχρονα στιγμιότυπα των οντολογιών πηγής και θα μπορούσαν να χρησιμοποιηθούν ως βάση για τον προσδιορισμό παρόμοιων εννοιών.»

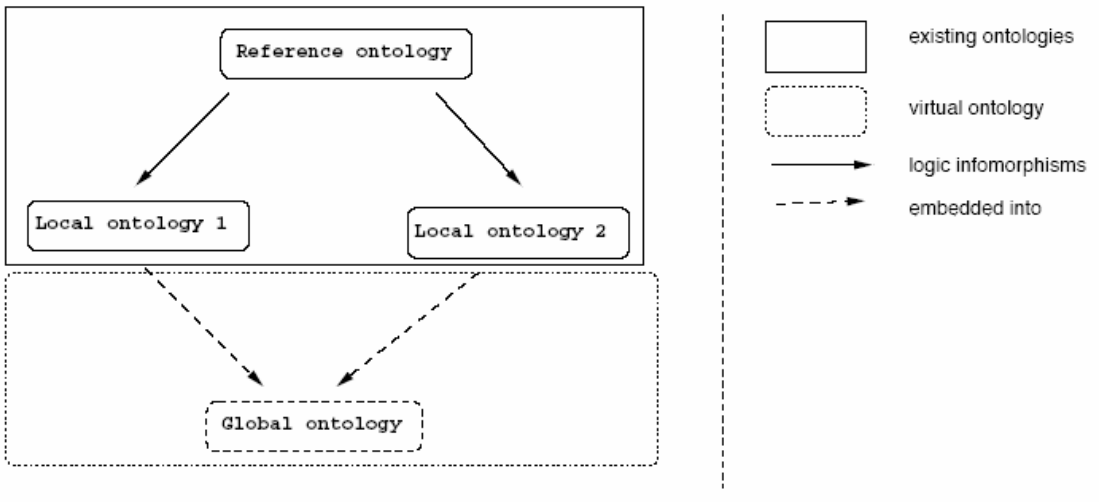
Έτσι, το πρώτο βήμα της μεθόδου FCAMerge θα μπορούσε να θεωρηθεί ως ένας **μηχανισμός οντολογικής κοινοποίησης** (ontology population mechanism). Αυτό το βήμα μπορεί να παραληφθεί αν υπάρχουν κοινά στιγμιότυπα και στις δύο οντολογίες. Όταν λοιπόν εξαχθούν τα στιγμιότυπα και παραχθεί το πλέγμα εννοιών οι Stumme και Maedche χρησιμοποιούν τεχνικές ανάλυσης τυπικών εννοιών (Formal Concept Analysis) για τη γέννηση των τυπικών εκφράσεων για κάθε οντολογία. Με τη λεκτική αυτή ανάλυση συσχετίζουν πολύπλοκες εκφράσεις όπως Hotel Schwarzer Adler με την έννοια Hotel.

Έπειτα οι δύο τυπικές εκφράσεις συγχωνεύονται για τη γέννηση ενός πλέγματος εννοιών (concept lattice). Τέλος, η μέθοδος FCAMerge φτάνει στο τελευταίο βήμα που είναι η μη αυτόματη κατασκευή της οντολογίας συγχώνευσης και χρειάζεται ανθρώπινη αλληλεπίδραση. Η κατασκευή αυτή είναι **ημι-αυτόματη** αφού χρειάζεται προηγούμενη γνώση στον τομέα και ο μηχανικός καλείται να επιλύσει πιθανές διπλοτυπίες ή συγκρούσεις.

4.2.2 IF-Map

Οι Kalfoglou και Schorlemmer (Kalfoglou and Schorlemmer 2002) ανέπτυξαν μια αυτόματη μέθοδο για οντολογική αντιστοίχιση, την μέθοδο **IF-Map**, η οποία βασίζεται στη θεωρία ροής της πληροφορίας (information flow theory) των Barwise και Seligman. Η μέθοδός τους στηρίζεται στο θεωρητικό υπόβαθρο της θεωρίας καναλιών (channel theory) των Barwise και Seligman και παρέχει ένα συστηματικό και μηχανιστικό τρόπο για την ανάπτυξή της σε ένα κατανεμημένο περιβάλλον για την τέλεση αντιστοιχίσεων μεταξύ διαφόρων οντολογιών.

Στο επόμενο σχήμα φαίνεται το πλαίσιο εργασίας του IF-Map για την υλοποίηση αντιστοιχίσεων μεταξύ οντολογιών:

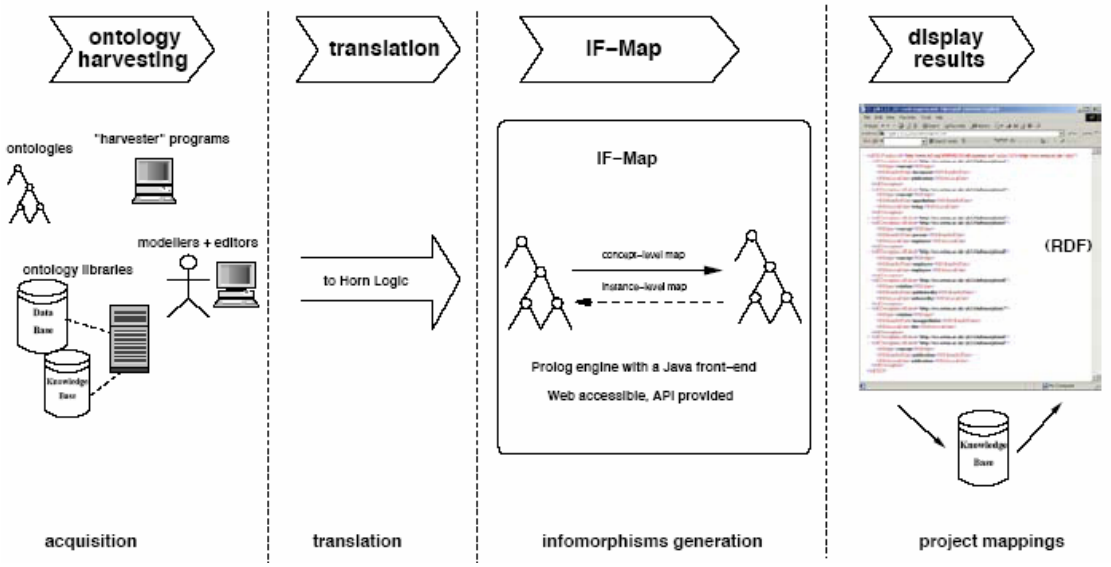


Εικόνα 4.1 Σενάριο της μεθόδου IF-Map

Το παραπάνω σχήμα προσομοιάζει τη διαδικασία δύο βημάτων που πρότεινε ο Kent για διαμοιρασμό μεταξύ οντολογιών (ontology sharing Kent 2000), αλλά έχει διαφορές στην υλοποίησή του.

Τα τετράγωνα που περικλείουν τη Reference ontology, τη Local ontology 1 και τη Local ontology 2 υποδηλώνουν υπάρχουσες οντολογίες. Υποθέτουμε ότι οι Local ontology 1 και η Local ontology 2 χρησιμοποιούνται από διαφορετικές κοινότητες και κοινοποιούνται με τα στιγμιότυπά τους, ενώ η Reference ontology είναι κοινά συμφωνημένη και ευνοεί το μοίρασμα της πληροφορίας. Το τετράγωνο με το διακεκομμένο περίγραμμα υποδηλώνει μια οντολογία που δεν υπάρχει από πριν αλλά θα κατασκευαστεί τη στιγμή αυτή για τους λόγους της συγχώνευσης. Τα συνεχόμενα βέλη που ενώνουν τη Reference ontology με τις Local ontology 1 και Local ontology 2 σημαίνουν ροή πληροφορίας μεταξύ των οντολογιών, ενώ τα διακεκομμένα βέλη δηλώνουν την ενσωμάτωση (embedding) από τη Local ontology 1 και τη Local ontology 2 στην Reference ontology.

Στο επόμενο σχήμα φαίνεται η αρχιτεκτονική της μεθόδου IF-Map:



Εικόνα 4.2 Αρχιτεκτονική της μεθόδου IF-Map

Η διαδικασία αποτελείται από τέσσερα βασικά **βήματα**: συγκομιδή οντολογιών (ontology harvesting), μετάφραση (translation), γέννηση πληροφοριο-μορφισμών (infomorphisms generation), παρουσίαση των αποτελεσμάτων.

Κατά τη συγκομιδή, οι οντολογίες αποκτούνται με πολλούς τρόπους όπως χρήση υπαρχόντων, φόρτωσή τους από βιβλιοθήκες οντολογιών (Ontolingua (Farquhar et al. 1997) ή WebOnto (Domingue 1998)), σύνταξή τους με κατάλληλα προγράμματα (Protégé) ή κατευθείαν από τον Παγκόσμιο Ιστό.

Η πολύπλευρη αυτή απόκτηση οντολογιών οδηγεί σε διάφορες τυποποιήσεις γλωσσών για οντολογίες όπως KIF (Genesereth and Fikes 1992), OCML (Motta 1999), RDF (Lassila and Swick 1999), Prolog, ακόμα και βάση γνώσης Protégé. Έτσι, ακολουθεί το βήμα της μετάφρασης. Οι συγγραφείς αναφέρουν:

«Αφού η μέθοδος IF-Map υιοθετεί τη λογική Horn (Horn logic) και εκτελείται με βάση μια μηχανή Prolog (Prolog Engine), πρέπει να μεταφραστούν οι παραπάνω τυποποιήσεις σε προτάσεις Prolog (Prolog clauses). »

Το επόμενο βήμα στη διαδικασία είναι ο κεντρικός μηχανισμός αντιστοίχισης κατά τον οποίο βρίσκονται λογικοί πληροφοριο-μορφισμοί (logic infomorphisms), αν υπάρχουν στις υπό εξέταση οντολογίες. Τέλος, αυτοί παρουσιάζονται ως αποτέλεσμα σε τυποποίηση RDF.

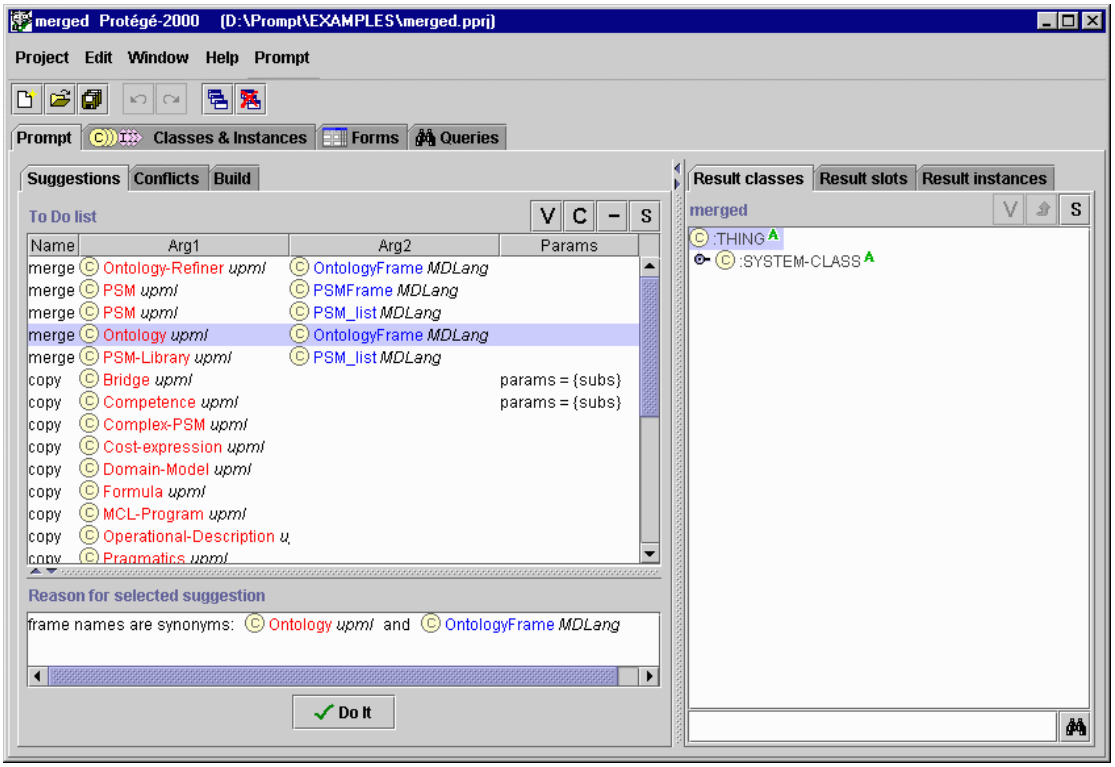
4.2.3 PROMPT - PROMPTDIFF

Οι Noy και Musen έχουν αναπτύξει μια σειρά εργαλείων που τελούν αντιστοίχιση (mapping) μεταξύ οντολογιών, συμμόρφωση (alignment) και σύγκριση εκδόσεων (versioning).

Τα εργαλεία αυτά είναι τα **PROMPT** (Noy and Musen 2000) και **PROMPTDIFF** (Noy and Musen 2002), τα οποία διατίθενται ως επεκτάσεις (plugins) του προγράμματος σύνταξης οντολογιών (ontology editor) Protégé. Όλα τα εργαλεία αναζητούν γ**λωσσολογική ομοιότητα** μεταξύ των εννοιών ώστε να αρχίσουν τη διαδικασία της συγχώνευσης ή της συμμόρφωσης, ενώ έπειτα χρησιμοποιούν τις θεμελιώδεις οντολογικές δομές του περιβάλλοντος του Protégé (classes, slots, facets) για την εύρεση περαιτέρω ταιριασμάτων μεταξύ των οντολογιών.

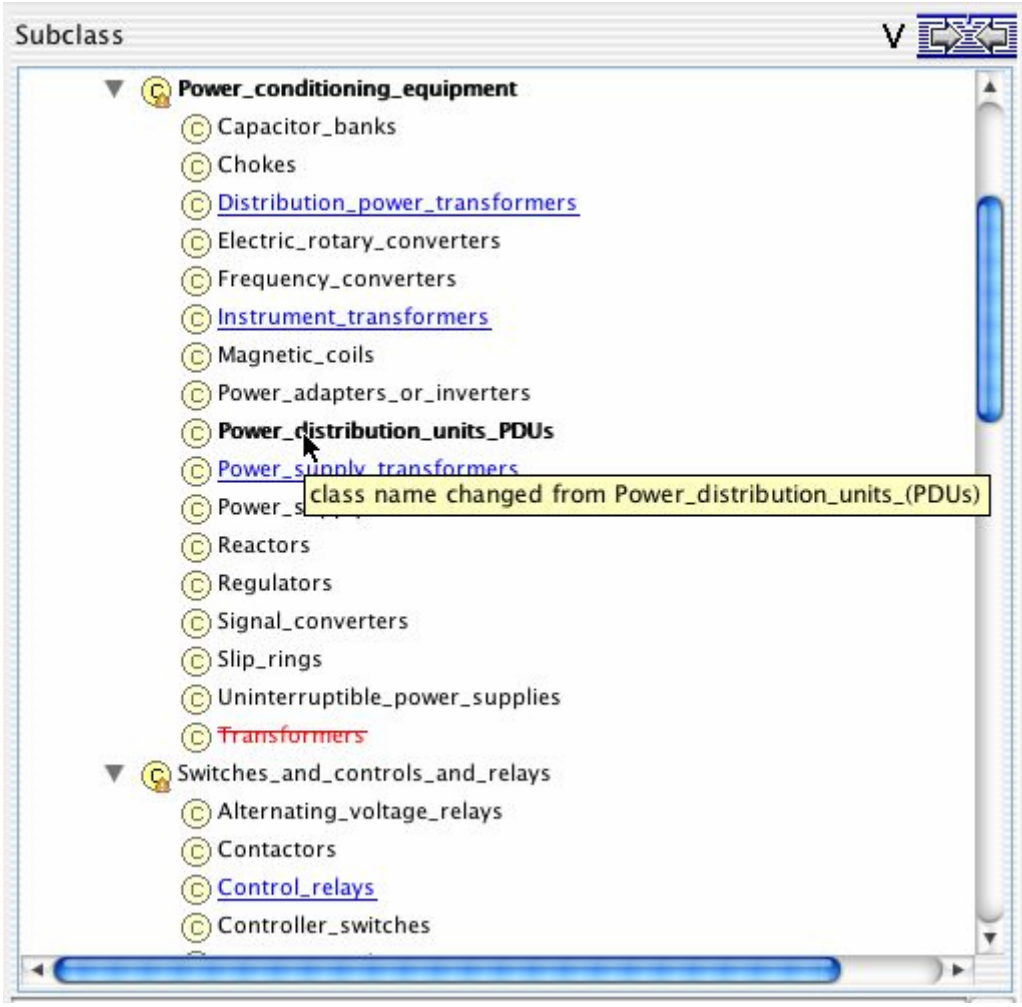
Οι συγγραφείς διαχωρίζουν τις έννοιες συγχώνευσης και συμμόρφωσης ορίζοντας :
«... συγχώνευση είναι η δημιουργία μιας μοναδικής συναφούς οντολογίας ενώ κατά τη συμμόρφωση εγκαθίστανται δεσμοί μεταξύ των οντολογιών που τις επιτρέπουν να επαναχρησιμοποιούν πληροφορία η μία από την άλλη»

Το PROMPT είναι ένα ημι-αυτόματο εργαλείο που παρέχει βοήθεια στο μηχανικό κατά τα βήματα της συγχώνευσης και της συμμόρφωσης:
«Εκεί όπου δεν μπορεί να ληφθεί αυτόματη απόφαση, ο αλγόριθμος οδηγεί το χρήστη στα σημεία της οντολογίας όπου είναι απαραίτητη η μεσολάβησή του, προτείνει πιθανές ενέργειες, υποδεικνύει τις συγκρούσεις μέσα στην οντολογία και προτείνει λύσεις.»



Εικόνα 4.3 Στιγμιότυπο εφαρμογής PROMPT

Το τελευταίο εργαλείο, το PROMPTDIFF, είναι ένας αλγόριθμος που ενσωματώνει διάφορες ευριστικές μεθόδους για τη σύγκριση μεταξύ εκδόσεων διαφορετικών οντολογιών. Οι συγγραφείς συνδυάζουν αυτούς τους «συγκριτές» με ένα καθορισμένο τρόπο, χρησιμοποιώντας τα αποτελέσματα του ενός ως είσοδο στον άλλο έως ότου δεν υπάρχουν πια διαφορές. Το PROMPTDIFF υλοποιεί σύγκριση βασισμένη στη δομή των οντολογιών και όχι στη σειριακή διάταξη των κειμένων που αυτές περιέχουν (text serialisation). Ο αλγόριθμος εργάζεται πάνω σε δύο εκδόσεις της ίδιας οντολογίας και βασίζεται στην εμπειρική παρατήρηση ότι ένα μεγάλο ποσοστό της οντολογίας παραμένει αμετάβλητο και στο ότι αν δύο σχήματα (frames) έχουν τον ίδιο τύπο και ίδιο ή παρόμοιο όνομα, το ένα είναι ένα αντίγραφο του άλλου.



Εικόνα 4.4 Στιγμιότυπο εφαρμογής PROMPTDIFF

4.2.4 CHIMAERA

Οι McGuinness και οι συνεργάτες της (McGuinness et al. 2000) ανέπτυξαν το εργαλείο **CHIMAERA** για το περιβάλλον σύνταξης οντολογιών Ontolingua, ένα εργαλείο αλληλεπίδρασης (interactive tool), όπου ο μηχανικός είναι υπεύθυνος για τη λήψη αποφάσεων που θα επηρεάσουν τη διαδικασία συγχώνευσης. Στο Chimaera αναλύονται οι οντολογίες προς συγχώνευση, και αν βρεθούν **γλωσσολογικά ταιριάσματα,** η συγχώνευση τελείται αυτόματα, αλλιώς καλείται η μεσολάβηση του χρήστη.

Συγκρίνοντας το εργαλείο Chimaera με το Prompt, παρατηρεί κανείς ότι είναι παρόμοια στο γεγονός ότι και τα δύο ενσωματώνονται σε περιβάλλοντα σύνταξης οντολογιών, αλλά διαφέρουν στις συστάσεις που κάνουν στο χρήστη κατά τη διαδικασία των βημάτων της συγχώνευσης.

4.2.5 GLUE

Οι Doan και οι συνεργάτες του (Doan et al. 2002) ανέπτυξαν ένα σύστημα, το **GLUE**, που χρησιμοποιεί τεχνικές εκμάθησης μηχανής (machine learning techniques) για την εύρεση αντιστοιχίσεων μεταξύ οντολογιών.

Δεδομένου δύο οντολογιών, για κάθε έννοια της μίας οντολογίας, το GLUE βρίσκει την πιο όμοια έννοια στην άλλη οντολογία χρησιμοποιώντας **πιθανολογικά** κάποια **μέτρα ομοιότητας**. Οι συγγραφείς υποστηρίζουν ότι εκεί έγκειται η διαφορά με τις άλλες τεχνικές εκμάθησης μηχανής όπου χρησιμοποιείται ένα μοναδικό μέτρο ομοιότητας.

Επίσης το GLUE,
«... χρησιμοποιεί **πολλαπλές τεχνικές εκμάθησης** κάθε μία από τις οποίες εκμεταλλεύεται ένα διαφορετικό τύπο πληροφορίας είτε στα δεδομένα στιγμιοτύπων είτε στην ταξινομική δομή των οντολογιών ...»

Η χρήση πολλαπλών τεχνικών εκμάθησης γίνεται γιατί υπάρχουν πολλοί διαφορετικοί τύποι πληροφορίας που μπορεί να συλλέξει η μηχανή εκμάθησης ώστε να κάνει προβλέψεις. Μπορεί να εκμεταλλευτεί τη συχνότητα των λέξεων στην τιμή κειμένου των στιγμιότυπων, το όνομα των στιγμιότυπων, την τυποποίηση των τιμών ή τα χαρακτηριστικά της διακύμανσης των τιμών. Για την αντιμετώπιση αυτής της ποικιλίας αναπτύχθηκαν δύο «μαθητές», ο «μαθητής περιεχομένου» και ο «μαθητής ονόματος». Ο πρώτος χρησιμοποιεί μια μέθοδο κατηγοριοποίησης κειμένου με όνομα Naïve Bayes εκμάθηση. Ο «μαθητής ονόματος» είναι παρόμοιος με τον πρώτο αλλά χρησιμοποιεί το πλήρες όνομα του στιγμιότυπου αντί για το περιεχόμενο του.

Τέλος, χρησιμοποιήθηκε μια τεχνική που αποδίδει ετικέτες στους κόμβους ενός γράφου δεδομένου ενός συνόλου περιορισμών. Η τεχνική αυτή στηρίζεται στην παρατήρηση ότι η ετικέτα ενός κόμβου επηρεάζεται από τα χαρακτηριστικά της γειτονιάς του κόμβου στο γράφο. Οι συγγραφείς εφάρμοσαν αυτήν την τεχνική για να αντιστοιχήσουν δύο ταξινομίες οντολογιών, την O1 στην O2, θεωρώντας τις έννοιες (κόμβους) στην O2 ως ετικέτες και ξαναδιατυπώνοντας το πρόβλημα ως την εύρεση της καλύτερης εκχώρησης ετικέτας σε έννοιες (κόμβους) της O1, δεδομένου της γνώσης που έχουν για τον τομέα και για τις δύο ταξινομίες. Η γνώση αυτή μπορεί να περιλαμβάνει περιορισμούς όπως «δύο κόμβοι ταιριάζουν αν ταιριάζουν επίσης οι κόμβοι στη γειτονιά τους» ή «αν ο Y είναι απόγονος του X».

Σημειώνεται ότι το σύστημα έχει αξιολογηθεί στην πράξη τελώντας την αντιστοίχιση δύο καταλόγων πανεπιστημιακών μαθημάτων.

4.2.6 CAIMAN

Οι Lacher και Groh (Lacher and Groh 2001) παρουσίασαν το **CAIMAN** (Community support Application Interoperability by MApping oNtologies), ένα ακόμα σύστημα που χρησιμοποιεί **τεχνικές εκμάθησης μηχανής** για αντιστοίχιση μεταξύ οντολογιών. Υιοθετούν το σενάριο κατά το οποίο τα μέλη μιας κοινότητας θα ήθελαν να διατηρήσουν τη δική τους προοπτική:

«Κάθε μέλος σε μια κοινότητα ενδιαφέροντος οργανώνει τα έγγραφά του ανάλογα με το δικό της σχήμα κατηγοριοποίησης (οντολογία)»

Αυτή η κάπως «αδύναμη» θεώρηση της οντολογίας δικαιολογεί τη χρήση των οντολογιών σε «προσωπικό» επίπεδο. Η αντιστοίχιση τότε έγκειται στη συμμόρφωση της οντολογίας με τη δομή του CiteSeer (http://citeseer.ist.psu.edu) (γνωστό και ως ResearchIndex). Η χρήση περισσότερο τυποποιημένων οντολογιών δεν υποστηρίζεται από τους συγγραφείς:

«Η πληροφορία πρέπει να κατηγοριοποιείται με τρόπο που ο χρήστης καταλαβαίνει και αποδέχεται. Αυτό θα μπορούσε να επιτευχθεί επιβάλλοντας μία τυποποιημένη κοινοτική οντολογία, με βάση την οποία θα οργανώνεται όλη η γνώση στην κοινότητα. Παρ' όλα αυτά κάτι τέτοιο δεν θα επιτευχθεί εξαιτίας της διαφωνίας των μελών της κοινότητας»

Ο μηχανισμός αντιστοίχισης χρησιμοποιεί τεχνικές εκμάθησης μηχανής για κατηγοριοποίηση κειμένου και μετράει την πιθανότητα να συμφωνούν δύο έννοιες. Για κάθε έννοια κόμβο στην «προσωπική» οντολογία, αναγνωρίζεται ένας κόμβος στην οντολογία κοινότητας που της αντιστοιχεί. Επίσης υποτίθεται ότι τα έγγραφα ή οι συνδέσεις στη φυσική θέση των εγγράφων μπορούν να αποθηκεύονται σε αποθήκες στην πλευρά του χρήστη αλλά και της κοινότητας. Τότε το CAIMAN προσφέρει δύο υπηρεσίες στο χρήστη: δημοσίευση εγγράφου (document publication), με την οποία δημοσιοποιείται το έγγραφο που ο χρήστης μόλις ανέθεσε σε μια έννοια κλάσης στην έννοια κλάσης της κοινότητας, και ανάκτηση σχετικών εγγράφων (retrieval of related documents) με την οποία μεταφέρονται από την αποθήκη της κοινότητας στο χρήστη έγγραφα που μόλις προστέθηκαν.

4.2.7 ONION

Οι Mitra και Wiederhold (Mitra and Wiederhold 2002) ανέπτυξαν το σύστημα σύνθεσης οντολογιών **ONION** (ONtology compositION system) με το οποίο επιλύεται η ετερογένεια (heterogeneity) μεταξύ διαφορετικών οντολογιών. Οι συγγραφείς υποστηρίζουν ότι η συγχώνευση οντολογιών είναι ανεπαρκής:

«Η προσέγγιση της συγχώνευσης δεν είναι κλιμακωτή και κοστίζει...Μία μονολιθική πηγή πληροφοριών δεν είναι εφικτή εξαιτίας άλυτων ασυνεπειών μεταξύ των οντολογιών»

Έπειτα επισημαίνουν ότι η σημασιολογική ετερογένεια μπορεί να λυθεί χρησιμοποιώντας **κανόνες άρθρωσης** (articulation rules) που εκφράζουν τη σχέση μεταξύ δύο ή περισσοτέρων εννοιών των οντολογιών. Η καθιέρωση τέτοιων κανόνων χειροκίνητα κοστίζει πολύ ενώ εξάλλου μια πλήρης αυτοματοποίηση δεν είναι δυνατή εξαιτίας της ανεπάρκειας της τεχνολογίας στην επεξεργασία της σημερινής φυσικής γλώσσας. Έτσι παίρνουν υπ' όψη τους σχέσεις (subclass of, part of, attribute of, instance of, value of) για να ορίσουν τους κανόνες άρθρωσης.

Τέλος, περιλαμβάνουν και ένα συστατικό εκμάθησης στο σύστημά τους, το οποίο δέχεται ανατροφοδότηση από τους χρήστες για να δημιουργήσει στο μέλλον καλύτερη σύνθεση οντολογιών. Σημειώνεται ότι οι αλγόριθμοι που χρησιμοποιούνται για την πραγματική αντιστοίχιση εννοιών βασίζονται σε γλωσσολογικά χαρακτηριστικά.

4.3 ΑΝΤΙΣΤΟΙΧΙΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΟΝΤΟΛΟΓΙΑ

Κατά τη διάρκεια των τελευταίων τεσσάρων δεκαετιών του εικοστού αιώνα, η χρήση των βάσεων δεδομένων αυξήθηκε σε όλες τις επιχειρήσεις. Η επανάσταση του Διαδικτύου (Internet Revolution) προς το τέλος της δεκαετίας του '90 αύξησε αισθητά την άμεση πρόσβαση χρηστών στις βάσεις δεδομένων. Οι περισσότεροι οργανισμοί υλοποίησαν διεπαφές Ιστού (web interfaces) για τις βάσεις δεδομένων τους και κατέστησαν διάφορες υπηρεσίες καθώς και μεγάλο όγκο πληροφορίας απευθείας προσβάσιμη στο Διαδίκτυο. Σήμερα, με την εμφάνιση του Σημασιολογικού Ιστού, το Διαδίκτυο είναι στα πρόθυρα μιας νέας επανάστασης. Ένα βασικό συστατικό αυτού είναι η δυνατότητα να γίνουν **οι βάσεις δεδομένων διαθέσιμες σημασιολογικά**, δηλαδή να βρεθεί ένας αυτοματοποιημένος τρόπος για την έκφραση της δομής και της σημασιολογίας των βάσεων δεδομένων. Μέσα σε αυτήν την προοπτική, δημιουργήθηκαν διάφορα εργαλεία που δημιουργούν αυτόματα τις οντολογίες οι οποίες αντιστοιχούν στο περιεχόμενο μιας βάσης δεδομένων και μπορούν να το καταστήσουν διαθέσιμο για τους ανθρώπους και τις μηχανές. Επίσης είναι δυνατή η **αντιστοίχιση δεδομένων από τη βάση σε κάποια υπάρχουσα οντολογία** [40].

4.3.1 KAON

4.3.1.1 Εισαγωγή

Το **KAON** [41] είναι ένα περιβάλλον διαχείρισης οντολογιών ανοικτού κώδικα, το οποίο στοχεύει κυρίως σε επιχειρησιακές εφαρμογές. Περιλαμβάνει μια μεγάλη συλλογή εργαλείων που επιτρέπουν την εύκολη δημιουργία και διαχείριση οντολογιών ενώ παρέχει επίσης ένα πλαίσιο εργασίας για τη δημιουργία εφαρμογών βασισμένες σε οντολογίες (ontology-based applications).

Το εργαλείο **KAON – REVERSE** [42] είναι ένα πρωτότυπο εργαλείο αντιστοίχισης περιεχομένου από σχεσιακές βάσεις δεδομένων σε οντολογίες επιτρέποντας αφ' ενός την αποθήκευση στιγμιοτύπων δεδομένων (instance data) σε τέτοιες βάσεις και αφ' ετέρου την άντληση πληροφοριών από τη βάση μέσω της σημασιολογίας της (conceptualisation).

4.3.1.2 KAON – REVERSE

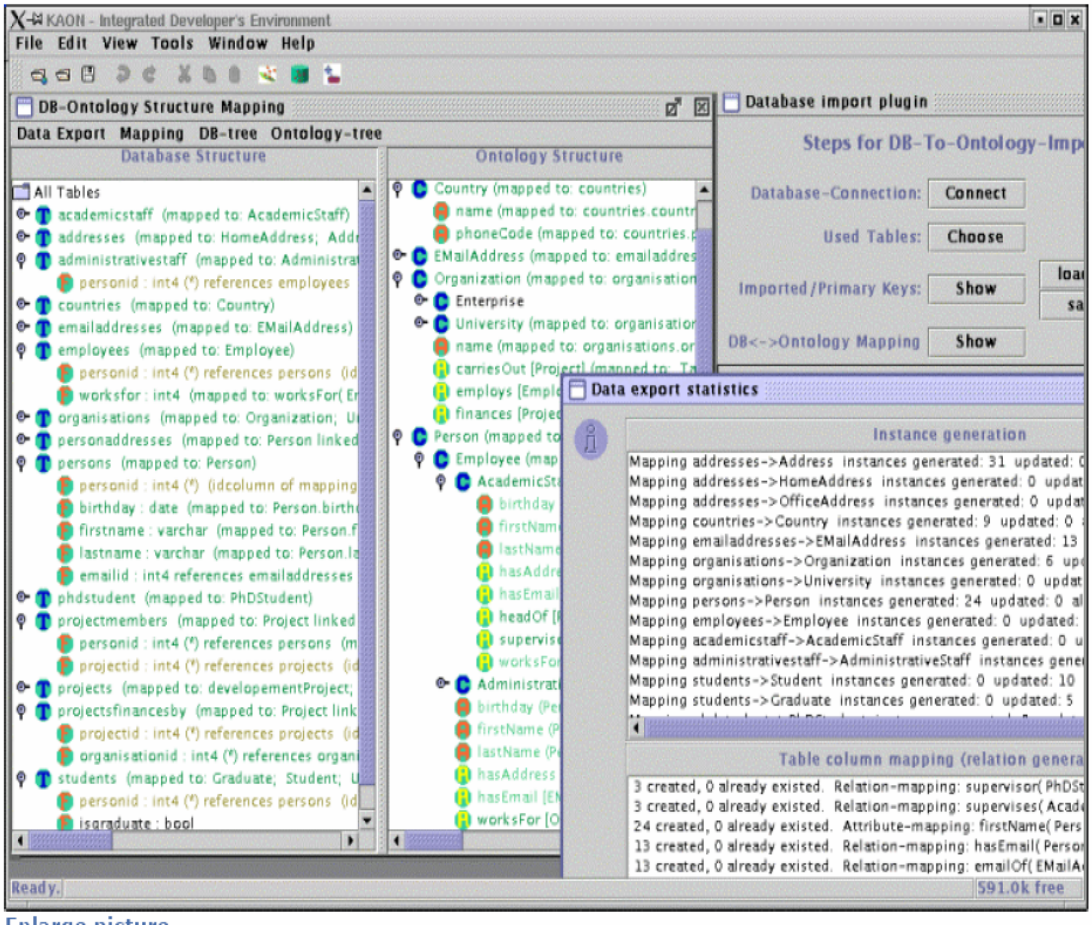
Το KAON – REVERSE επιτρέπει τη γέννηση RDF στιγμιοτύπων **αντιστοιχώντας περιεχόμενο που προέρχεται από σχεσιακές βάσεις δεδομένων, σε οντολογίες**. Το εργαλείο αυτό επιτρέπει το γέμισμα οντολογιών από υπάρχον περιεχόμενο που προέρχεται από κάποια σχεσιακή βάση δεδομένων. Συγκεκριμένα, δέχεται ένα σύνολο κανόνων καθορισμένων από τον χρήστη, που υποδεικνύουν ποιοι πίνακες αντιστοιχούνται σε ποιες έννοιες. Το εργαλείο μετασχηματίζει έπειτα τη βάση δεδομένων στην οντολογία. Αυτός ο μετασχηματισμός είναι ημι-στατικός αφού εάν η πηγαία βάση δεδομένων αλλάξει, ολόκληρη η διαδικασία μετασχηματισμού πρέπει να επαναληφθεί για να απεικονίσει αυτές τις αλλαγές στο μετασχηματισμένο μοντέλο [43].

Περιγραφή

Ο σκοπός του εργαλείου είναι, όπως αναφέρθηκε, η εισαγωγή πληροφορίας που προ-υπάρχει αποθηκευμένη σε μια βάση δεδομένων, σε μια ήδη υπάρχουσα οντολογία. Η διαδικασία αντιστοίχισης αποτελείται από τα επόμενα βήματα:

- Φόρτωση της οντολογίας στην οποία θα γίνει η αντιστοίχιση
- Σύνδεση μέσω JDBC (Java Database Connectivity) στη βάση δεδομένων που περιέχει τα δεδομένα προς εξαγωγή
- Καθορισμός των αντιστοιχίσεων ανάμεσα στη βάση δεδομένων και στη δομή της οντολογίας
- Εξαγωγή των δεδομένων σύμφωνα με τους κανόνες αντιστοίχισης από τη βάση δεδομένων στην οντολογία

Παρακάτω φαίνεται το γραφικό περιβάλλον του εργαλείου κατά τη διαδικασία της αντιστοίχισης:



Εικόνα 4.5 Γραφικό Περιβάλλον KAON

Στο αριστερό μέρος της εικόνας φαίνεται το παράθυρο της αντιστοίχισης το οποίο αποτελείται από δύο δενδρικές δομές, μία για τη βάση δεδομένων και μία για την οντολογία. Ο καθορισμός της αντιστοίχισης γίνεται σέρνοντας κάποιους πίνακες ή στήλες πάνω στο επιθυμητό στοιχείο της οντολογίας. Μετά τον καθορισμό των αντιστοιχίσεων τα δεδομένα μπορούν αυτόματα να εξαχθούν στην οντολογία, δημιουργώντας τα ανάλογα στατιστικά τα οποία φαίνονται στην εικόνα κάτω δεξιά.

Σημειώνεται ότι το KAON – REVERSE είναι πρωτότυπο και είναι μέρος του περιβάλλοντος KAON v1.0. Στο τρέχον σύστημα KAON v2.0 δεν ενσωματώνεται αφού θα αντικατασταθεί σύντομα από ένα πιο εξεζητημένο σύστημα που θα επιτρέπει ενσωμάτωση περιεχομένου από σχεσιακές βάσεις αλλά και από XML.

4.3.2 D2R MAP

4.3.2.1 Εισαγωγή

Η D2R [44][45] είναι μια δηλωτική, βασισμένη στην XML γλώσσα που περιγράφει τις αντιστοιχίσεις μεταξύ σχημάτων σχεσιακών βάσεων δεδομένων και οντολογιών

γλώσσας OWL. Οι αντιστοιχίσεις μπορούν να χρησιμοποιηθούν από έναν D2R επεξεργαστή ο οποίος εξάγει δεδομένα από μια σχεσιακή βάση δεδομένων σε RDF.

Ο κύριος στόχος της γλώσσας D2R είναι να επιτρέπει ευέλικτες αντιστοιχίσεις πολύπλοκων σχεσιακών δομών χωρίς πρώτα να χρειαστεί να αλλάξει το υπάρχον σχήμα της βάσης δεδομένων. Η ευελιξία αυτή επιτυγχάνεται **ενσωματώνοντας εντολές SQL, απευθείας στους κανόνες αντιστοίχισης**.

Όπως έχει ήδη αναφερθεί, ο Σημασιολογικός Ιστός είναι μια επέκταση του τρέχοντος Ιστού, στον οποίο τα δεδομένα έχουν μια σαφώς καθορισμένη έννοια αναπαριστώντας τα σε RDF και συνδέοντάς τα με κοινώς αποδεκτές οντολογίες. Αυτός ο σημασιολογικός εμπλουτισμός επιτρέπει στα δεδομένα να είναι διαμοιρασμένα και ενσωματωμένα σε διαφορετικές πηγές και επιτρέπει στις εφαρμογές να χρησιμοποιούν τα δεδομένα υπό διαφορετικά πλαίσια.

Επειδή όμως τα περισσότερα δεδομένα σήμερα αποθηκεύονται στις σχεσιακές βάσεις δεδομένων, για να είναι σε θέση να χρησιμοποιηθούν, με σημασιολογικό τρόπο, πρέπει να αντιστοιχηθούν σε RDF.

4.3.2.2 Διαδικασία Αντιστοίχισης D2R

Η D2R διαδικασία αντιστοίχισης που εκτελείται από τον επεξεργαστή D2R έχει τέσσερα λογικά **βήματα**:

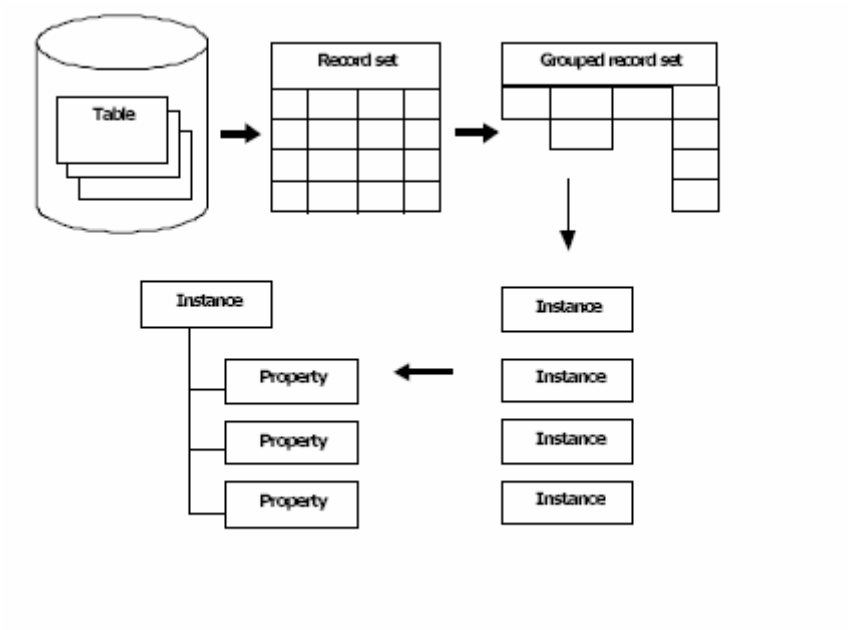
1. Επιλογή ενός συνόλου εγγραφών από τη βάση δεδομένων με τη χρήση της γλώσσας SQL

2. Ομαδοποίηση του συνόλου εγγραφών σύμφωνα με τις στήλες που ορίζονται στο δομικό στοιχείο d2r:groupBy.

3. Δημιουργία των στιγμιοτύπων κλάσης (class instances) και κατασκευή αναγνωριστικών (identifier construction).

4. Αντιστοίχιση των δεδομένων του ομαδοποιημένου συνόλου εγγραφών στις ιδιότητες στιγμιοτύπων (instance properties).

Η διαδικασία της αντιστοίχισης φαίνεται σχηματικά στο παρακάτω σχήμα:



Εικόνα 4.6 Η D2R Διαδικασία Αντιστοίχισης

Αρχικά, για κάθε κλάση ή ομάδα παρόμοιων κλάσεων, ένα σύνολο εγγραφών επιλέγεται από τη βάση δεδομένων. Έπειτα, το σύνολο των εγγραφών ομαδοποιείται σύμφωνα με τις στήλες groupBy της συγκεκριμένης κλάσης προς αντιστοίχιση (ClassMap). Κατόπιν δημιουργούνται τα στιγμιότυπα κλάσης (class instances) και τους ανατίθεται ένα URI ή ένα κενό αναγνωριστικό κόμβου (blank node identifier). Τέλος, δημιουργούνται οι ιδιότητες στιγμιοτύπου (instance properties) με χρήση γεφυρών ιδιοτήτων (datatype and object property bridges).

Σημειώνεται εδώ ότι η διαφοροποίηση μεταξύ του τρίτου και του τέταρτου βήματος επιτρέπει αναφορές σε κενούς κόμβους στο μοντέλο, όπως επίσης και τη δυναμική δημιουργία στιγμιοτύπων κατά τη διαδικασία αντιστοίχισης.

Ακόμα, η προσέγγιση αυτή επιτρέπει τη διαχείριση δυαδικών αλλά και υψηλότερης τάξης σχέσεων, ιδιοτήτων κλάσεων με πολλαπλές τιμές, πολύπλοκων συνθηκών και αυστηρά κανονικοποιημένων δομών πινάκων όπου τα δεδομένα είναι διασπαρμένα σε διάφορους πίνακες.

4.3.2.3 Ο επεξεργαστής D2R MAP

Ένα D2R πρωτότυπο επεξεργαστών είναι δημόσια διαθέσιμο υπό την άδεια GNU LGPL. Ο επεξεργαστής υλοποιείται σε Java και είναι βασισμένος στο Jena API. Εξάγει τα δεδομένα σαν RDF, N3, N-TRIPLES και μοντέλα της Jena. Είναι συμβατός με όλες τις σχεσιακές βάσεις δεδομένων με πρόσβαση JDBC ή ODBC.

4.3.2.4 Παράδειγμα Χρήσης D2R MAP

Το ακόλουθο παράδειγμα παρουσιάζει τη χρήση του D2R MAP για την εξαγωγή δεδομένων, αναφερόμενα σε συγγραφείς (authors) και στις δημοσιεύσεις (publications) τους, από μια βάση δεδομένων σε RDF. Επειδή οι συγγραφείς συνήθως έχουν περισσότερες από μία δημοσιεύσεις και κάποιες δημοσιεύσεις είναι δυνατό να γραφούν από πολλούς συγγραφείς, η όλη πληροφορία αποθηκεύεται τυπικά σε τρεις πίνακες: έναν για τους συγγραφείς, έναν για τις δημοσιεύσεις και έναν τρίτο για την ν:μ σχέση μεταξύ συγγραφέων και δημοσιεύσεων. Μια D2R MAP μετατροπή αυτών των πινάκων στις κλάσεις ex:Author και ex:Book είναι η παρακάτω:

```
<Map>
  <DBConnection odbcDSN="bookDB" />
  <ProcessorMessage outputFormat="RDF/XML-ABBREV"/>
  <Namespace prefix="ex" namespace="http://example.org#"/>
  <ClassMap type="ex:Book" sql="SELECT isbn, title FROM books;"
groupBy="isbn" uriPattern="ex:book@@isbn@@">
     <DatatypePropertyBridge property="ex:title" column="title" xml:lang="en"/>
  </ClassMap>
  <ClassMap type="ex:Author" sql="SELECT authors.aid, name, URL, isbn
FROM authors, bookauthor WHERE authors.aid = bookauthor.aid;"
groupBy="authors.aid">
     <DatatypePropertyBridge property="ex:fullname" column="name" />
     <ObjectPropertyBridge property="ex:homepage" column="URL" />
     <ObjectPropertyBridge property="ex:author_of" referredClass="ex:Book"
referredGroupBy="isbn" useContainer="rdf:Bag"/>
  </ClassMap>
</Map>
```

Τα πρώτα τρία υποστοιχεία (subelements) ορίζουν τη σύνδεση με τη βάση δεδομένων, την επιθυμητή τυποποίηση εξόδου (output format) και ένα παράδειγμα χώρου ονομάτων (namespace). Το πρώτο στοιχείο ClassMap περιγράφει την αντιστοίχιση για την κλάση ex:Book. Τα URIs των στιγμιοτύπων δημιουργούνται με τη χρήση ενός προτύπου, ενώ επίσης τίθεται και το χαρακτηριστικό xml:lang στην ιδιότητα. Το δεύτερο στοιχείο ClassMap περιγράφει τη δημιουργία των στιγμιότυπων ex:Author και συνδέει τους συγγραφείς με τις δημοσιεύσεις τους με τη χρήση ενός container rdf:bag για την ιδιότητα ex:author_of. Επειδή η αντιστοίχιση ex:Author ClassMap δεν περιέχει σχήμα κατασκευής URI (URI construction schema), τα στιγμιότυπα αναγνωρίζονται ως κενοί κόμβοι.

Παρακάτω ακολουθεί ένα παράδειγμα από ένα στιγμιότυπο το οποίο δημιουργήθηκε με την παραπάνω αντιστοίχιση:

```
<ex:Author rdf:nodeID='A465'>
  <ex:fullname>Chris Bizer</ex:fullname>
  <ex:homepage rdf:resource='http://www.bizer.de'/>
  <ex:author_of>
     <rdf:Bag>
       <rdf:li rdf:resource='http://example.org#book321230273'/>
      <rdf:li rdf:resource='http://example.org#book884237273'/>
```

```
    </rdf:Bag>
  </ex:author_of>
</ex:Author>
```

Το παραπάνω παράδειγμα δείχνει μόνο κάποιες λειτουργίες του D2R MAP. Ο πλήρης οδηγός της γλώσσας καθώς και περαιτέρω παραδείγματα μπορούν να βρεθούν στην ιστοσελίδα του D2R MAP [44].

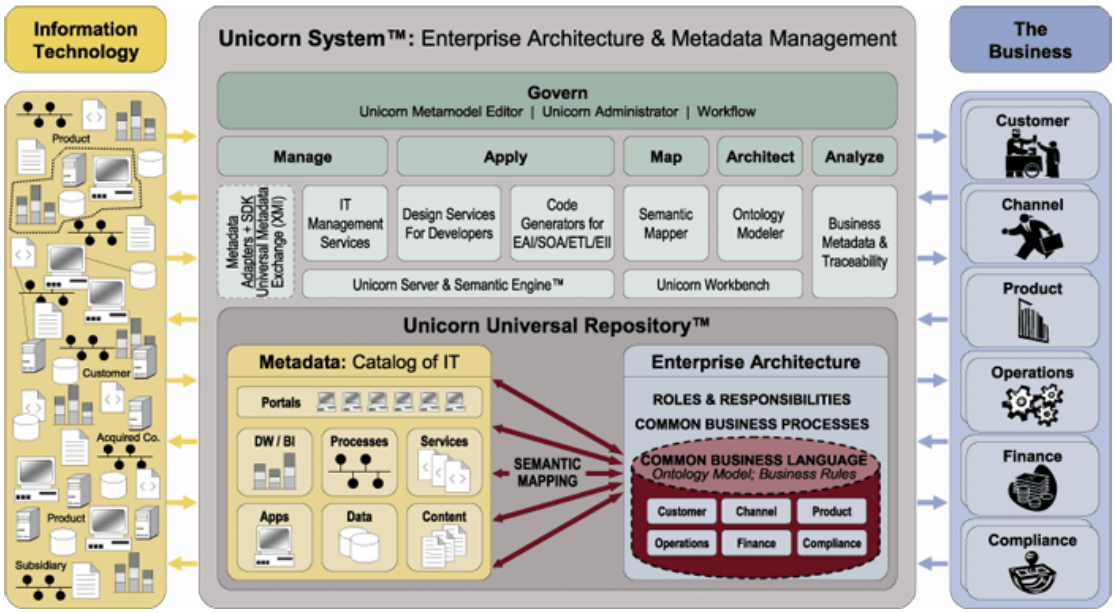
Στο μέλλον σχεδιάζεται η επέκταση της D2R MAP με αντιστοιχίσεις υπό συνθήκες και με δυνατότητες για πιο εξεζητημένες μετατροπές τιμών. Η επέκταση αυτή θα μπορούσε να βασιστεί στη RuleML [46], στην RDFT [47] ή σε άλλες γλώσσες που δανείζονται στοιχεία από την XSLT.

4.3.3 UNICORN

4.3.3.1 Εισαγωγή

Το σύστημα Unicorn (Unicorn System) [48] είναι μία πλατφόρμα για Επιχειρησιακή Αρχιτεκτονική (Enterprise Architecture) και Διαχείριση Μεταδεδομένων (Metadata Management).

Το περιβάλλον εργασίας Unicorn Workbench [49] είναι ενσωματωμένο στο σύστημα Unicorn και είναι ένα γραφικό εργαλείο σε Java για τη διαχείριση αποθήκης (Repository), τη μοντελοποίηση οντολογιών (Ontology Modelling) και την σημασιολογική αντιστοίχιση (Semantic Mapping).



Εικόνα 4.7 Σχηματική Αναπαράσταση Συστήματος Unicorn

Παρατηρούμε ότι ένα κεντρικό στοιχείο (core element) του συστήματος Unicorn είναι η αντιστοίχιση (Map). Παρέχονται προηγμένα εργαλεία σημασιολογικής

αντιστοίχισης με τα οποία κάθε σχήμα δεδομένων (data schema) αντιστοιχείται στο οντολογικό μοντέλο μέσω μιας αποτελεσματικής γραφικής μεθόδου δύο φάσεων.

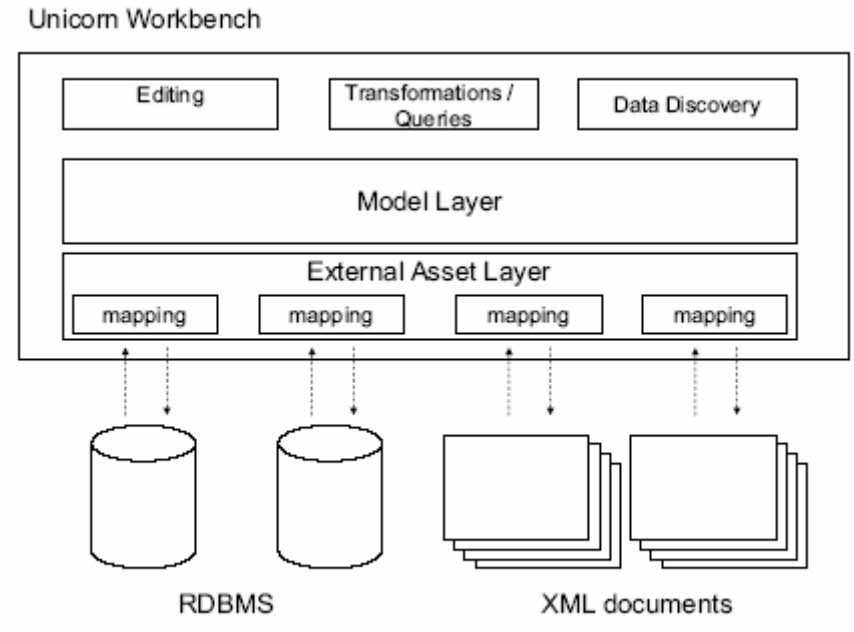
4.3.3.2 Unicorn Workbench

Παρακάτω παρουσιάζεται μια επισκόπηση του εργαλείου Unicorn Workbench όπως προκύπτει από τη μελέτη [50] του Jos de Bruijn στο πανεπιστήμιο του Innsbruck, όσον αφορά στη σημασιολογική ενσωμάτωση διαφόρων πηγών δεδομένων στις επιχειρήσεις.

Η μελέτη αυτή είναι μέρος του Corporate Ontology Grid (COG) project, στόχος του οποίου είναι η λύση του προβλήματος της σημασιολογικής ετερογένειας (semantic heterogeneity) μεταξύ πηγών δεδομένων, με τη σημασιολογική ολοκλήρωση των πηγών χρησιμοποιώντας ένα κεντρικό μοντέλο πληροφορίας (Information Model) όπως η οντολογία.

Περιγραφή

Το Unicorn Workbench δημιουργήθηκε για να υποστηρίξει τη Σημασιολογική Διαχείριση Πληροφορίας Unicorn (Unicorn Semantic Information Management (Schreiber, 2003) (SIM)) και να καταστήσει δυνατές τέτοιες υλοποιήσεις στις επιχειρήσεις.



Εικόνα 4.8 Αρχιτεκτονική Unicorn

Η **αρχιτεκτονική** του Unicorn αποτελείται από δύο βασικά επίπεδα:

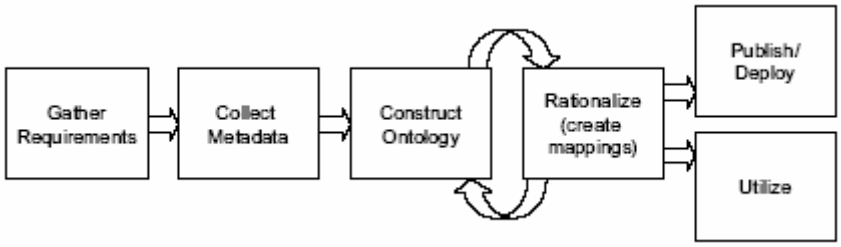
- Το επίπεδο εξωτερικών πόρων (External Assets Layer) περιέχει τις αντιστοιχίσεις με τις διάφορες πηγές δεδομένων. Όλα τα σχήματα δεδομένων μπορούν να εισαχθούν στο Unicorn εφόσον υπάρχει αναλυτής (parser) για

αυτά. Τα τρέχοντα σχήματα δεδομένων που υποστηρίζονται είναι σχεσιακά σχήματα βάσεων δεδομένων, σχήματα XML και COBOL Copybooks.

- Το επίπεδο μοντέλου πληροφορίας (Model Layer – Information Layer) περιέχει την οντολογία, η οποία περιλαμβάνει πακέτα, κλάσεις, ιδιότητες και κανόνες για την περιγραφή της έννοιας των δεδομένων που ανήκουν στις εξωτερικές πηγές.

Το επίπεδο μοντέλου πληροφορίας περιγράφει την έννοια των δεδομένων ενώ το επίπεδο εξωτερικών πόρων περιγράφει το πού βρίσκονται τα δεδομένα. Προκειμένου να καταστεί η οντολογία ενεργή, το Unicorn Workbench παρέχει τρεις λειτουργίες για το χρήστη. Μια **λειτουργία σύνταξης** (editing), η οποία χρησιμοποιείται για τη δημιουργία και τη συντήρηση οντολογιών καθώς και των αντιστοιχίσεων με τις διάφορες πηγές δεδομένων. Η δεύτερη λειτουργία είναι η **λειτουργία ανακάλυψης δεδομένων**, η οποία μπορεί να υιοθετηθεί από το χρήστη για να ανακαλύψει τη θέση των δεδομένων, που ανήκουν σε διαφορετικές πηγές. Τέλος, υπάρχει ένας μετασχηματισμός και μια **λειτουργία επερωτήσεων** με τα οποία ο χρήστης μπορεί να δημιουργήσει μετασχηματισμούς στιγμιότυπων (transformation of instances) μεταξύ των διαφορετικών πηγών δεδομένων και να θέσει ερωτήματα (queries) στην οντολογία.

Παρακάτω δίνεται η σχηματική απεικόνιση της μεθοδολογίας σημασιολογικής διαχείρισης πληροφορίας (Semantic Information Management Methodology) και έπειτα εξηγείται η **διαδικασία της αντιστοίχισης**:



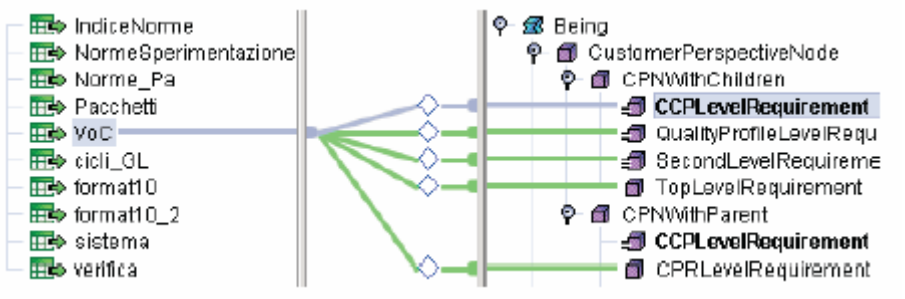
Εικόνα 4.9 Μεθοδολογία Σημασιολογικής Διαχείρισης Πληροφορίας

Κατά τη φάση συλλογής των μεταδεδομένων (metadata collection), διάφορα σχήματα δεδομένων έχουν εισαχθεί στο πρόγραμμα Unicorn, τα οποία χρησιμοποιούνται στην κατασκευή της οντολογίας (ontology construction). Αυτά τα σχήματα αντιστοιχίζονται στην οντολογία προκειμένου να είναι δυνατή η αποστολή επερωτήσεων στην οντολογία.

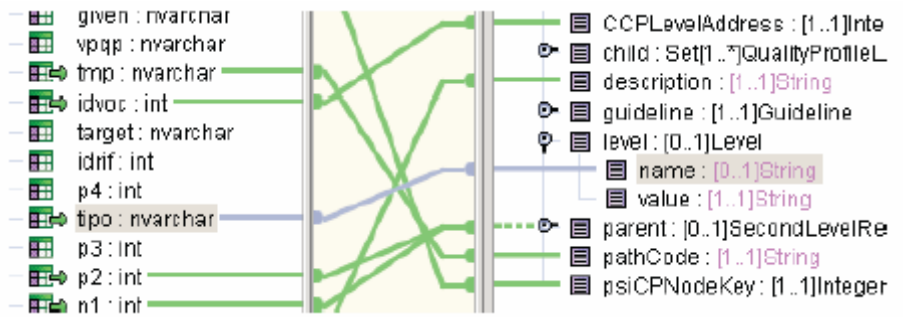
Κατά τη φάση της αντιστοίχισης ο χρήστης έχει τη βοήθεια του Unicorn Workbench στη δημιουργία των αντιστοιχίσεων μεταξύ των πόρων δεδομένων και της οντολογίας. Είναι δυνατό να φανούν οι αντιστοιχίσεις από την πλευρά των δεδομένων, έτσι ώστε να επιλεχθούν τα δεδομένα και να αποφασιστεί σε ποια κλάση ή ιδιότητα θα γίνει η αντιστοίχιση, αλλά είναι επίσης δυνατό να φανούν μέσω της ιεραρχίας κλάσεων ποιες αντιστοιχίσεις ήδη υπάρχουν ώστε να ληφθεί απόφαση για περαιτέρω αντιστοιχίσεις.

Οι αντιστοιχίσεις δημιουργούνται σε δύο **στάδια**. Αρχικά, δημιουργείται η **κοινή αντιστοίχιση** (Coarse Mapping) με την οποία συνδέονται κάποια συστατικά των δεδομένων, όπως πίνακες ή πολύπλοκοι τύποι, με κλάσεις της οντολογίας. Έπειτα,

δημιουργείται η **λεπτομερής αντιστοίχιση** (Detailed Mapping) με την οποία συνδέονται «άτομα» από τα πηγαία σχήματα δεδομένων με ιδιότητες (properties) της οντολογίας.



Εικόνα 4.10 Κοινή Αντιστοίχιση στο Unicorn Workbench



Εικόνα 4.11 Λεπτομερής Αντιστοίχιση στο Unicorn Workbench

Κατά την επιλογή «ατόμων» από το σχήμα δεδομένων μπορούν να εφαρμοστούν κάποιες συνθήκες οπότε με αυτόν τον τρόπο συγκεκριμένες ομάδες στιγμιότυπων μπορούν να αντιστοιχηθούν σε διαφορετικές κλάσεις της οντολογίας. Ανάλογη διαδικασία μπορεί να ακολουθηθεί και για την αντιστοίχιση ιδιοτήτων κατά τη λεπτομερή αντιστοίχιση. Συνήθως, στη φάση αυτή αντιστοιχίζονται «άτομα» σε άμεσες ιδιότητες (direct properties) της κλάσης στην οποία έχει εφαρμοστεί προηγουμένως η κοινή (coarse) αντιστοίχιση. Όμως είναι δυνατόν να γίνει το ίδιο και με έμμεσες ιδιότητες (indirect properties) οι οποίες μάλιστα μπορούν να βρίσκονται σε οποιοδήποτε βάθος.

Όταν χρειάζεται αντιστοίχιση στιγμιότυπων από διαφορετικά σχήματα δεδομένων σε μια μοναδική κλάση της οντολογίας, τότε δημιουργείται μια όψη αντιστοίχισης (mapping view). Μια τέτοια όψη μπορεί να περιέχει συνενώσεις (joins) διαφορετικών συστατικών από μια εξωτερική πηγή. Μία συνένωση μπορεί να οριστεί χρησιμοποιώντας ένα υπάρχον ή ένα συνεπαγόμενο (“implicit” foreign key) ξένο κλειδί. Τα ξένα κλειδιά της βάσης δεδομένων ή το συνεπαγόμενο κλειδί του Unicorn υποδηλώνουν είτε μια απλή σχέση με μια άλλη κλάση, είτε κληρονομικότητα. Στην πρώτη περίπτωση το ξένο κλειδί μπορεί να αντιστοιχηθεί σε μία ιδιότητα στην κλάση που δείχνει στην κλάση στόχο η οποία αντιπροσωπεύει τον τύπο στόχο του ξένου κλειδιού. Στην δεύτερη περίπτωση το ξένο κλειδί αντιστοιχίζεται απευθείας στη σχέση κληρονομικότητας. Η περίπτωση αυτή εφαρμόζεται όταν ένα συστατικό είναι υποσύνολο ενός άλλου και αυτό φαίνεται σαφώς με ένα ξένο κλειδί στη βάση δεδομένων.

4.3.4 R_2O

4.3.4.1 Εισαγωγή

Η R_2O (Relational to Ontology) είναι μία επεκτάσιμη, δηλωτική γλώσσα που περιγράφει αντιστοιχίσεις μεταξύ σχεσιακών σχημάτων βάσεων δεδομένων και οντολογιών που έχουν υλοποιηθεί σε RDF(S) ή OWL. Η γλώσσα αυτή παρέχει ένα επεκτάσιμο σύνολο αρχών με καλά ορισμένη σημασιολογία και θεωρείται αρκετά εκφραστική ώστε να μπορεί να αντιμετωπίσει περίπλοκες καταστάσεις αντιστοίχισης, στις οποίες υπάρχει ελάχιστη ομοιότητα μεταξύ της οντολογίας και του μοντέλου της βάσης δεδομένων [51].

4.3.4.2 Κίνητρα

Υπάρχει τεράστια ποσότητα δεδομένων στις σελίδες του Παγκόσμιου Ιστού που γεννιέται από τις σχεσιακές βάσεις δεδομένων. Η πληροφορία αυτή αναφέρεται ως βαθύς ιστός (Deep Web) [52] σε αντίθεση με τον επιφανειακό ιστό (Surface Web) που περιλαμβάνει όλες τις στατικές σελίδες του Παγκόσμιου Ιστού. Το πρόβλημα λοιπόν είναι η «αναβάθμιση» όλου αυτού του περιεχομένου σε περιεχόμενο του Σημασιολογικού Ιστού.

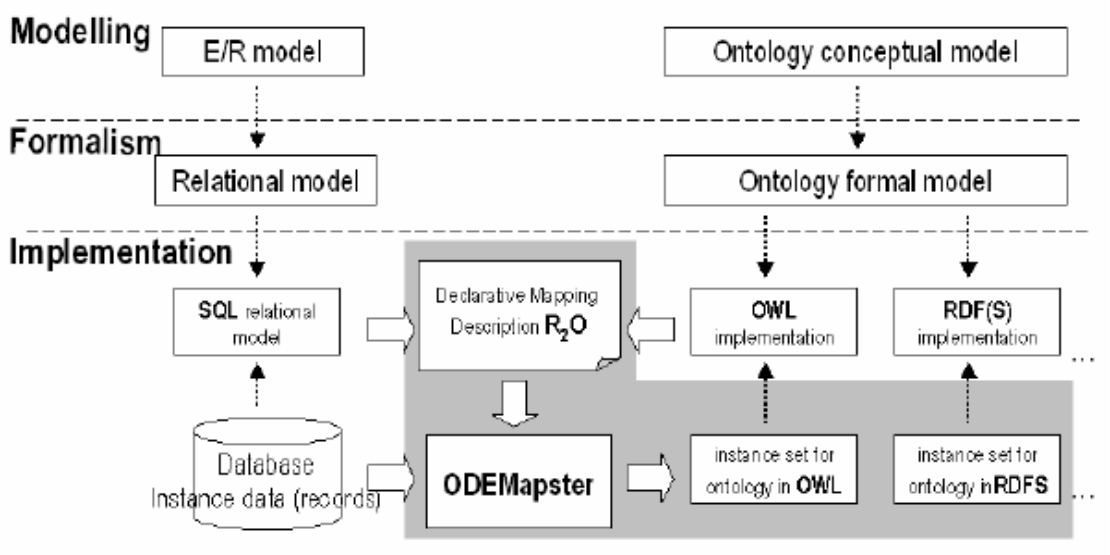
Για το λόγο αυτό έχουν αναφερθεί τρεις προσεγγίσεις. Η πρώτη [53][54] βασίζεται στην ημι-αυτόματη γέννηση μιας οντολογίας από ένα μοντέλο σχεσιακής βάσης δεδομένων. Έπειτα καθορίζονται οι απεικονίσεις μεταξύ της βάσης και της γεννηθείσας οντολογίας, αλλά επειδή υπάρχει μεγάλη ομοιότητα μεταξύ τους, δεν εμφανίζονται πολύπλοκες περιπτώσεις αντιστοίχισης. Η προσέγγιση αυτή δεν επιτρέπει αντιστοιχίσεις σε υπάρχουσα οντολογία, κάτι που την περιορίζει αρκετά. Μια δεύτερη προσέγγιση [55] προτείνει την προσθήκη σχολίων στις δυναμικές σελίδες του Ιστού με πληροφορίες για την υποκείμενη βάση δεδομένων. Η προσέγγιση αυτή δεν ασχολείται με πολύπλοκες καταστάσεις αντιστοίχισης και απαιτεί την κοινοποίηση του σχήματος της βάσης δεδομένων, κάτι που δεν είναι πάντα επιθυμητό. Η τρίτη προσέγγιση είναι αυτή που αναλύεται εδώ και περιγράφει **αντιστοιχίσεις μεταξύ σχεσιακών σχημάτων βάσεων δεδομένων που ήδη υπάρχουν και οντολογιών που ήδη υπάρχουν και έχουν υλοποιηθεί σε RDF(S) ή OWL**.

Με το ίδιο θέμα ασχολείται και η D2R MAP αλλά της λείπει εκφραστικότητα για τους πολύπλοκους μετασχηματισμούς, ενώ δεν είναι πλήρως δηλωτική.

4.3.4.3 Περιγραφή – Χαρακτηριστικά

Σκοπός της γλώσσας R_2O είναι η δημιουργία ενός εγγράφου που περιγράφει την αντιστοίχιση μεταξύ των συστατικών του SQL σχήματος της βάσης δεδομένων και της οντολογίας. Οι αντιστοιχίσεις αυτές επεξεργάζονται αυτόματα από τον επεξεργαστή αντιστοιχίσεων ODEMapster (ODEMapster mapping processor) ο οποίος εμπλουτίζει την οντολογία.

Παρακάτω φαίνεται η αρχιτεκτονική της αντιστοίχισης R_2O και στο γκρι πλαίσιο απεικονίζονται τα αποτελέσματα του ορισμού και της εκτέλεσης της αντιστοίχισης:



Εικόνα 4.12 Αρχιτεκτονική της αντιστοίχισης R_2O

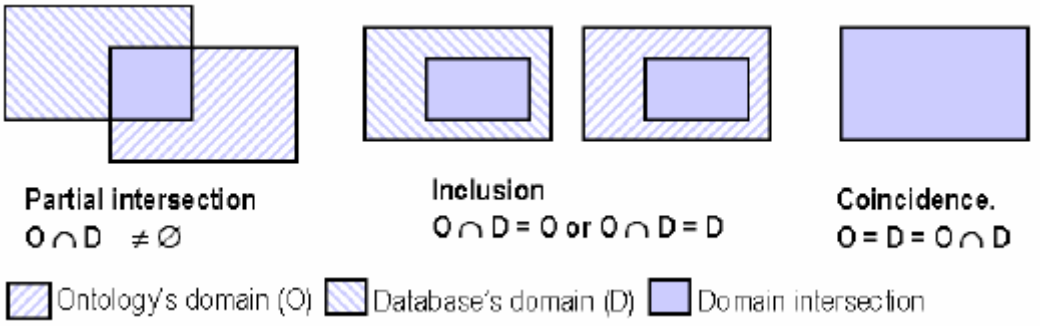
Η **R_2O** είναι μια γλώσσα **ανεξάρτητη συστήματος διαχείρισης βάσης δεδομένων** και υποστηρίζει οποιαδήποτε βάση υλοποιεί το πρότυπο SQL. Τα κύρια χαρακτηριστικά της είναι:

- Η R_2O καθορίζει τον τρόπο με τον οποίο θα **εμπλουτιστεί η οντολογία με στιγμιότυπα** (instances) προερχόμενα από το περιεχόμενο της βάσης δεδομένων. Έτσι, η βασική **ροή δεδομένων** είναι **από τη βάση προς την οντολογία**, κάτι που μπορεί να γίνει μαζικά ή κατά βούληση μέσω ερωτημάτων.
- Η R_2O μπορεί να χρησιμοποιηθεί για να εκφράσει τις αντιστοιχίσεις που προέρχονται από αυτόματα εργαλεία αντιστοίχισης όπως το Cupid [56].
- Ένας ορισμός αντιστοίχισης R_2O μπορεί να χρησιμοποιηθεί για αυτο-εξακρίβωση αφού χάρη στην πλήρως δηλωτική φύση της γλώσσας, οι ασυνέπειες και οι αμφιβολίες στον ορισμό της αντιστοίχισης μπορούν να ανιχνευθούν.
- Ένας ορισμός αντιστοίχισης R_2O μπορεί επίσης να χρησιμοποιηθεί για την εξακρίβωση της ακεραιότητας στοιχείων της βάσης, εφαρμόζοντας τα αξιώματα της οντολογίας στα στοιχεία της βάσης.

Παρακάτω παρουσιάζονται διάφορες **περιπτώσεις αντιστοίχισης** που προέρχονται από σενάρια αντιστοίχισης βάσης δεδομένων σε οντολογία (Database–to–Ontology Mapping) και καλύπτονται από τη γλώσσα R_2O.

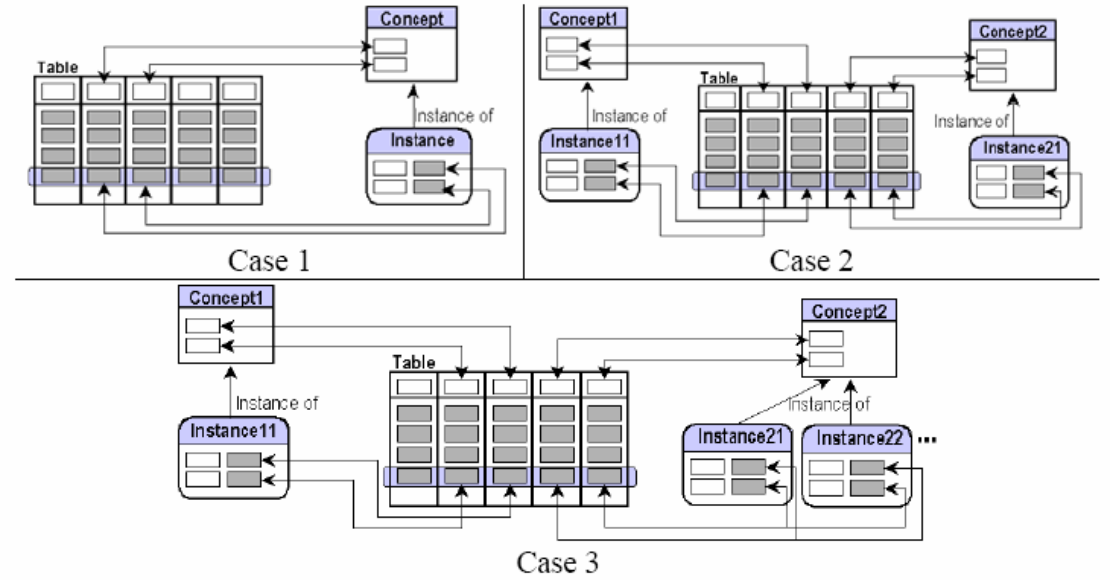
Η απεικόνιση βάσης δεδομένων σε οντολογία μπορεί να οριστεί ως ένα σύνολο αντιστοιχίσεων που σχετίζουν το λεξιλόγιο του σχεσιακού σχήματος της βάσης δεδομένων με αυτό της οντολογίας. Έτσι, πρέπει να συνδεθούν πίνακες, στήλες, πρωτεύοντα και ξένα κλειδιά της βάσης δεδομένων με κλάσεις, έννοιες, ιδιότητες της οντολογίας.

Σύμφωνα με το επίπεδο επικάλυψης των τομέων που καλύπτουν η βάση δεδομένων και η οντολογία διακρίνονται τρεις περιπτώσεις οι οποίες παρουσιάζονται γραφικά:



Εικόνα 4.13 Επίπεδα επικάλυψης μεταξύ τομέων που καλύπτονται από τη βάση δεδομένων και την οντολογία

Επειδή δεν συμπίπτουν πάντα οι τομείς, και τα κριτήρια μοντελοποίησης για τη δημιουργία της βάσης δεδομένων είναι διαφορετικά από αυτά της οντολογίας, οι αντιστοιχίσεις μεταξύ των στοιχείων μπορεί να είναι ευθείες ή απρόσμενες. Παρατηρώντας τον τρόπο με τον οποίο τα στοιχεία του σχήματος της βάσης δεδομένων αντιστοιχούνται σε έννοιες της οντολογίας, μπορούμε να διαχωρίσουμε τις ακόλουθες περιπτώσεις οι οποίες φαίνονται και γραφικά:



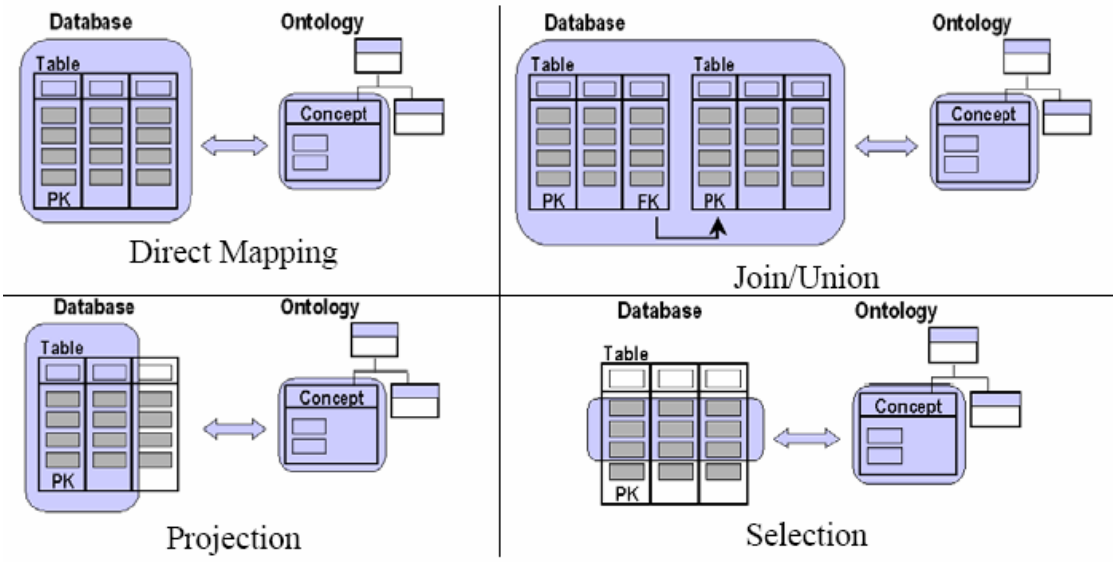
Εικόνα 4.14 Περιπτώσεις αντιστοίχισης

- Περίπτωση 1: Ένας πίνακας αντιστοιχίζεται σε μία έννοια της οντολογίας. Στην περίπτωση αυτή, οι στήλες του πίνακα αντιστοιχούνται στις ιδιότητες της έννοιας ενώ για κάθε εγγραφή του πίνακα δημιουργείται ένα στιγμιότυπο

της έννοιας και με τα δεδομένα της εγγραφής γεμίζουν οι τιμές του στιγμιότυπου.

- Περίπτωση 2: Ένας πίνακας χρησιμοποιείται για να δημιουργήσει στιγμιότυπα σε περισσότερες από μία έννοιες της οντολογίας, αλλά μόνο ένα στιγμιότυπο ανά έννοια. Στην περίπτωση αυτή, κάθε στήλη του πίνακα αντιστοιχεί στις ιδιότητες της έννοιας ενώ για κάθε εγγραφή του πίνακα δημιουργείται ένα στιγμιότυπο της έννοιας και με τα δεδομένα της εγγραφής γεμίζουν οι τιμές του στιγμιότυπου.
- Περίπτωση 3: Ένας πίνακας χρησιμοποιείται για να δημιουργήσει στιγμιότυπα σε περισσότερες από μία έννοιες της οντολογίας, αλλά μπορούν να δημιουργηθούν πολλαπλά στιγμιότυπα ανά έννοια. Στην περίπτωση αυτή, όπως και παραπάνω, οι στήλες του πίνακα αντιστοιχούνται στις ιδιότητες της έννοιας ενώ για κάθε εγγραφή του πίνακα δημιουργείται ένα στιγμιότυπο της έννοιας και με τα δεδομένα της εγγραφής γεμίζουν οι τιμές του στιγμιότυπου.

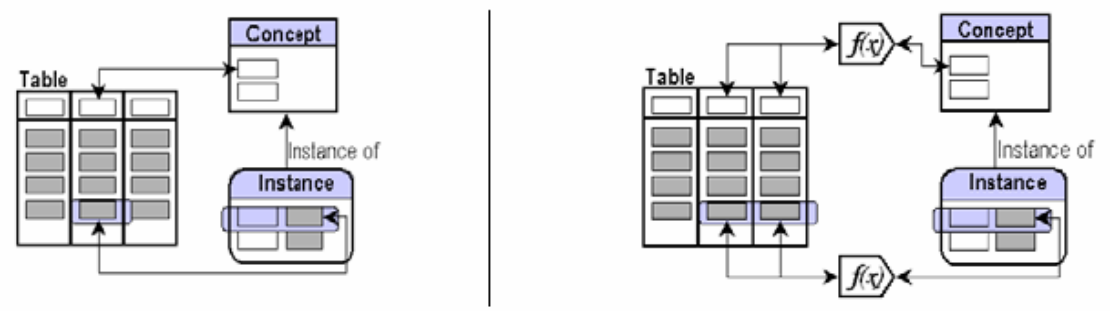
Αξίζει εδώ να αναφερθεί ότι πολλές φορές αντιστοιχούνται όλες οι στήλες ενός πίνακα σε ιδιότητες μιας έννοιας, αλλά στην πραγματικότητα λίγες από αυτές χρειάζονται, ενώ το ίδιο συμβαίνει και με τις εγγραφές. Έτσι, πριν τη διαδικασία της αντιστοίχισης είναι απαραίτητο να εκτελεστούν κάποιες αλγεβρικές λειτουργίες όπως προβολή ή επιλογή. Οι αντίστοιχες περιπτώσεις φαίνονται και γραφικά:



Εικόνα 4.15 Περιπτώσεις αντιστοίχισης με χρήση αλγεβρικών λειτουργιών

- Απευθείας αντιστοίχιση: Ο πίνακας της βάσης δεδομένων αντιστοιχίζεται απευθείας σε μια έννοια της οντολογίας και κάθε εγγραφή σε ένα στιγμιότυπο.
- Ένωση/Συνένωση: Ένα σύνολο πινάκων αντιστοιχίζεται σε μια έννοια της οντολογίας μετά την συνένωση και κάθε εγγραφή συνένωσης σε ένα στιγμιότυπο.
- Προβολή: Ένα υποσύνολο των στηλών του πίνακα της βάσης δεδομένων χρειάζεται να αντιστοιχηθεί.
- Επιλογή: Ένα υποσύνολο των γραμμών του πίνακα της βάσης δεδομένων χρειάζεται να αντιστοιχηθεί.
- Οποιοσδήποτε από τους παραπάνω συνδυασμούς.

Τέλος, οι τιμές των ιδιοτήτων μπορούν να συμπληρωθούν απευθείας από τα πεδία των εγγραφών μιας βάσης ή μετά την εφαρμογή μιας συνάρτησης μετασχηματισμού. Η συνάρτηση μετασχηματισμού μπορεί να επηρεάσει περισσότερα από ένα πεδίο δεδομένων. Στο επόμενο σχήμα φαίνεται η ιδέα:



Εικόνα 4.16 Περιπτώσεις αντιστοίχισης με απευθείας συμπλήρωση των πεδίων ή με χρήση μετασχηματισμού

Παρόλο που σχεσιακή άλγεβρα SQL καλύπτει πολλές περιπτώσεις, υπάρχουν άλλες για τις οποίες χρειάζονται πρόσθετοι μετασχηματισμοί. Παράδειγμα αποτελούν πολύπλοκες λειτουργίες όπως γλωσσικές τεχνικές επεξεργασίας, ταίριασμα κανονικών εκφράσεων κ.λ.π. Για τέτοιες περιπτώσεις η R_2O παρέχει τρόπους για να δηλωθούν ρητά οι επιλογές και οι μετασχηματισμοί.

5 ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΟΣ ΑΝΤΙΣΤΟΙΧΙΣΗΣ

5.1 ΕΙΣΑΓΩΓΗ

Η εφαρμογή που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής εργασίας προσφέρει τη δυνατότητα **αντιστοίχισης μιας βάσης δεδομένων σε μια ήδη υπάρχουσα οντολογία**. Παράλληλα, ως επαλήθευση της διαδικασίας αντιστοίχισης, υποστηρίζεται η δυνατότητα εκτέλεσης ενός **σημασιολογικού ερωτήματος** (RDQL Query) και η επιστροφή των πραγματικών δεδομένων που έχουν απεικονιστεί στις κλάσεις της οντολογίας οι οποίες απαντούν στο εν λόγω ερώτημα.

Σημειώνεται ότι η εφαρμογή υλοποιήθηκε σε Java και ενσωματώνεται στο Protégé (θα μπορούσε να είναι και αυτόνομη), το οποίο αποτελεί ένα από τα κυρίαρχα εργαλεία δημιουργίας και επεξεργασίας οντολογιών και το οποίο έχει αναφερθεί εκτενώς σε προηγούμενο κεφάλαιο. Συγκεκριμένα, ολόκληρη η εφαρμογή ενσωματώνεται στο φάκελο εγκατάστασης του προγράμματος Protégé, οπότε ο χρήστης είναι σε θέση, αφού εκτελέσει το Protégé να την επιλέξει από τα ήδη υπάρχοντα tab plugins. Εμφανίζεται δηλαδή σε μορφή καρτέλας το γραφικό περιβάλλον με τη βοήθεια του οποίου θα γίνει η αντιστοίχιση και η εκτέλεση του σημασιολογικού ερωτήματος.

5.2 ΦΙΛΟΣΟΦΙΑ - ΙΔΕΑ ΣΥΣΤΗΜΑΤΟΣ

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, υπάρχουν αρκετές περιπτώσεις και αρκετοί τρόποι απεικόνισης μιας κλάσης μιας οντολογίας σε στοιχεία μιας βάσης δεδομένων. Εκτός από την απλή περίπτωση αντιστοίχισης ενός ολόκληρου πίνακα μιας βάσης σε μια κλάση της οντολογίας, η οποία θα μπορούσε να αντιμετωπιστεί εύκολα, υπάρχει και η ανάγκη **αντιστοίχισης συγκεκριμένων στηλών ή γραμμών ενός ή περισσότερων πινάκων σε μια κλάση**, χρησιμοποιώντας και συνδυάζοντας

με αυτόν τον τρόπο τις αλγεβρικές πράξεις της προβολής, της επιλογής και της συνένωσης. Αυτή η ανάγκη οδήγησε στη χρήση ερωτημάτων σε γλώσσα βάσης δεδομένων (**SQL ερωτημάτων**), τα αποτελέσματα της εκτέλεσης των οποίων αποτελούν το σύνολο των πραγματικών δεδομένων που **θα αντιστοιχηθούν σε μια κλάση της οντολογίας**. Με αυτόν τον τρόπο, καλύπτονται οι περισσότερες περιπτώσεις αντιστοιχίσεων που χρησιμοποιούν τις αλγεβρικές πράξεις της επιλογής, της προβολής και της συνένωσης.

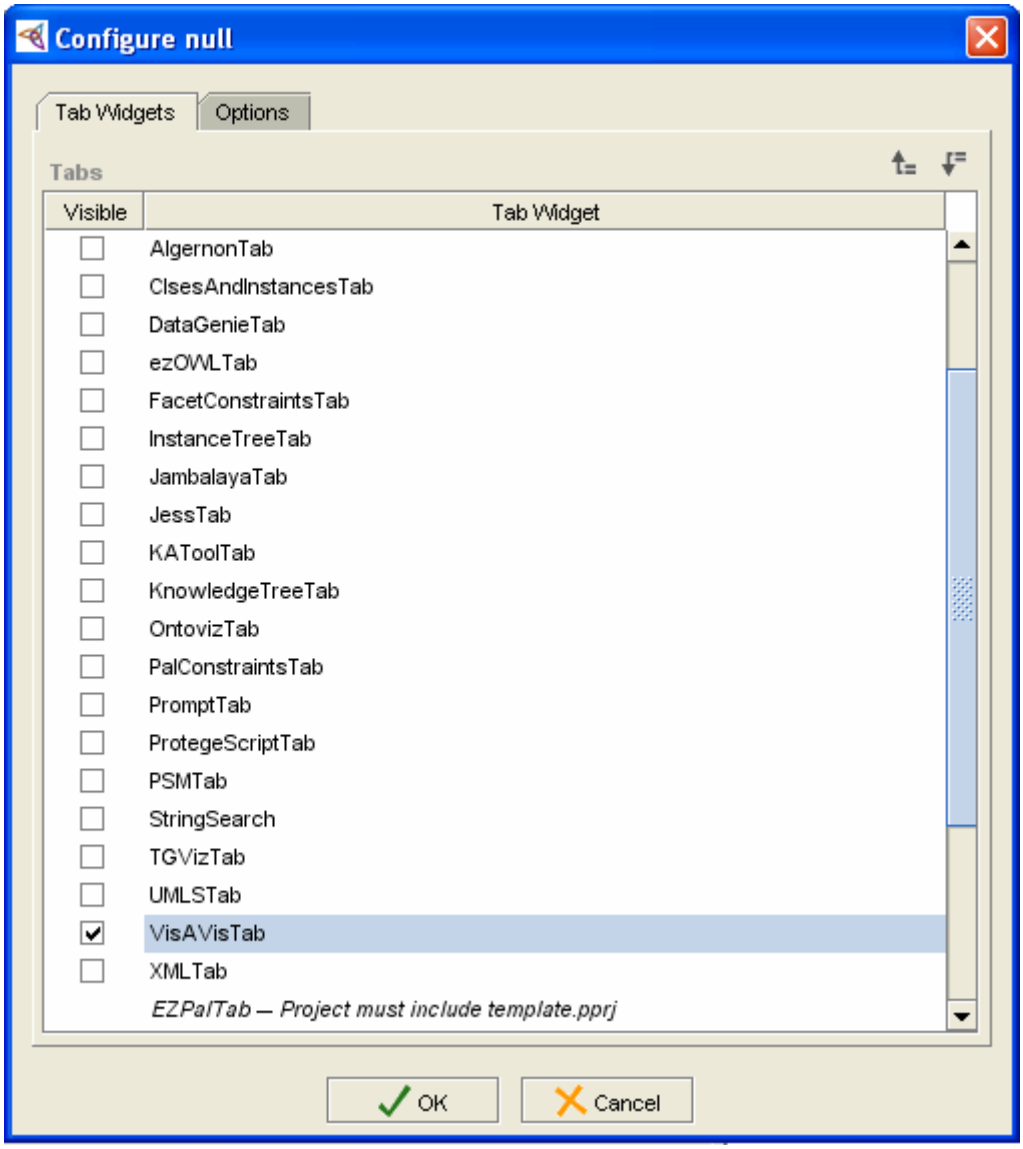
Βασικό σημείο της προσέγγισης είναι ο τρόπος αποθήκευσης των αντιστοιχίσεων. Ένας τρόπος υλοποίησης θα χρησιμοποιούσε γλώσσες επισήμανσης (XML, XSL κ.ά.) για την αποθήκευση των αντιστοιχίσεων σε διαφορετικό αρχείο από αυτό της οντολογίας. Στην τρέχουσα περίπτωση, επιλέχθηκε η **αποθήκευση των αντιστοιχίσεων σε ένα ξεχωριστό OWL αρχείο**, το οποίο περιέχει ένα αντίγραφο της αρχικής οντολογίας. Συνεπώς, το αρχείο που περιέχει την οντολογία αναφοράς παραμένει αναλλοίωτο, ενώ το αντίγραφό της εμπλουτίζεται με τις αντιστοιχίσεις.

Επίσης, ένα ακόμα σημαντικό πλεονέκτημα της χρήσης SQL ερωτήματος είναι το γεγονός ότι η μετατροπή (προσθήκη, αφαίρεση) δεδομένων από τη βάση συνεπάγεται **άμεση ενημέρωση** και των **δεδομένων** που έχουν αντιστοιχηθεί σε κάποια κλάση. Αυτό είναι λογικό, εφόσον για να βρεθούν τα δεδομένα που αντιστοιχούν, μια συγκεκριμένη χρονική στιγμή, σε μια κλάση, πρέπει την ίδια στιγμή να εκτελεστεί το συσχετισμένο με αυτήν την κλάση SQL ερώτημα, με αποτέλεσμα να επιστραφούν δεδομένα που βρίσκονται εκείνη τη δεδομένη στιγμή στη βάση. Τέλος, αξίζει να αναφερθεί ότι η ενσωμάτωση μόνο του SQL ερωτήματος για την επίτευξη της αντιστοίχισης έχει ως αποτέλεσμα τη λειτουργία της εφαρμογής με όσο το δυνατόν **μικρότερο κόστος αποθήκευσης**.

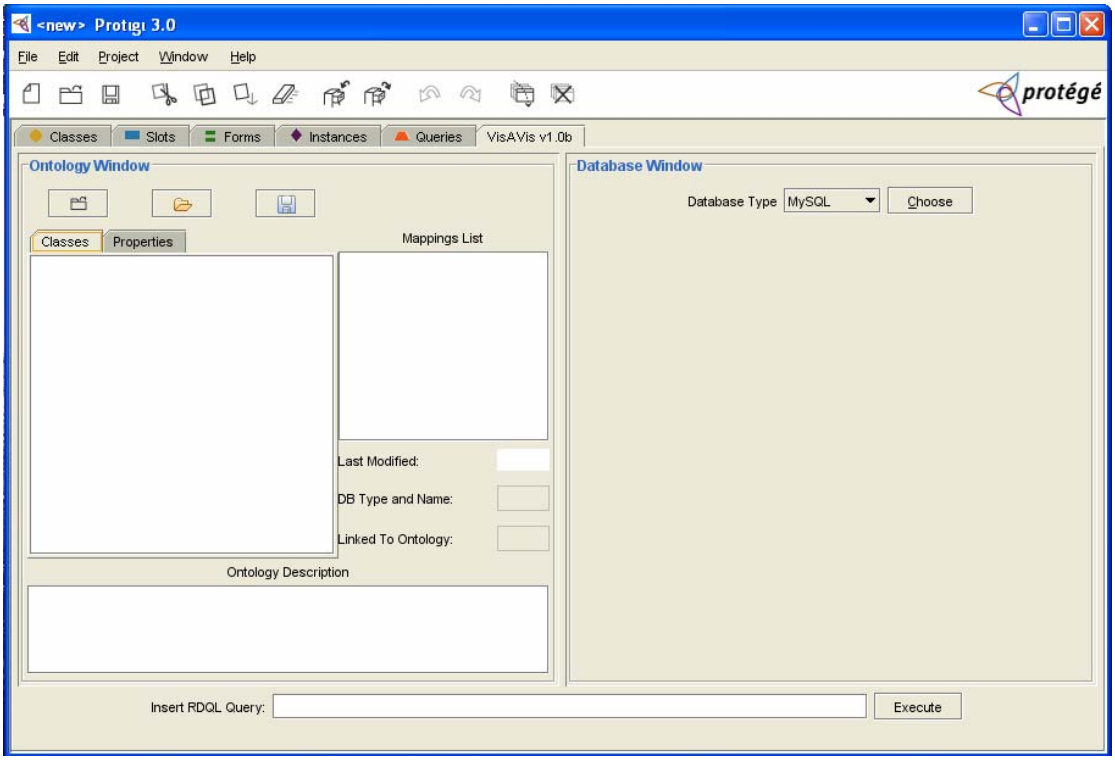
5.3 ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΥΣΤΗΜΑΤΟΣ

Όπως αναφέρθηκε και παραπάνω, η εφαρμογή που υλοποιήθηκε αποτελεί επέκταση σε μορφή καρτέλας (tab plugin) του προγράμματος Protégé και έχει ως στόχο την αντιστοίχιση κλάσεων μιας οντολογίας με συγκεκριμένα δεδομένα μιας βάσης δεδομένων. Παρακάτω, περιγράφεται αναλυτικά ο τρόπος χρήσης της εφαρμογής μέσω ενός απλού παραδείγματος στο οποίο χρησιμοποιείται μια **βάση χωρικών δεδομένων** και μια **οντολογία** που αναφέρεται στο συγκεκριμένο τομέα γνώσης.

Για να εμφανιστεί το γραφικό περιβάλλον της εφαρμογής αρκεί μέσα από το πρόγραμμα Protégé να επιλεχθεί το plugin VisAVisTab.



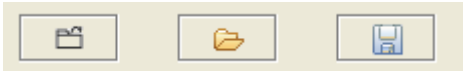
Εικόνα 5.1 Επιλογή του plugin VisAVisTab



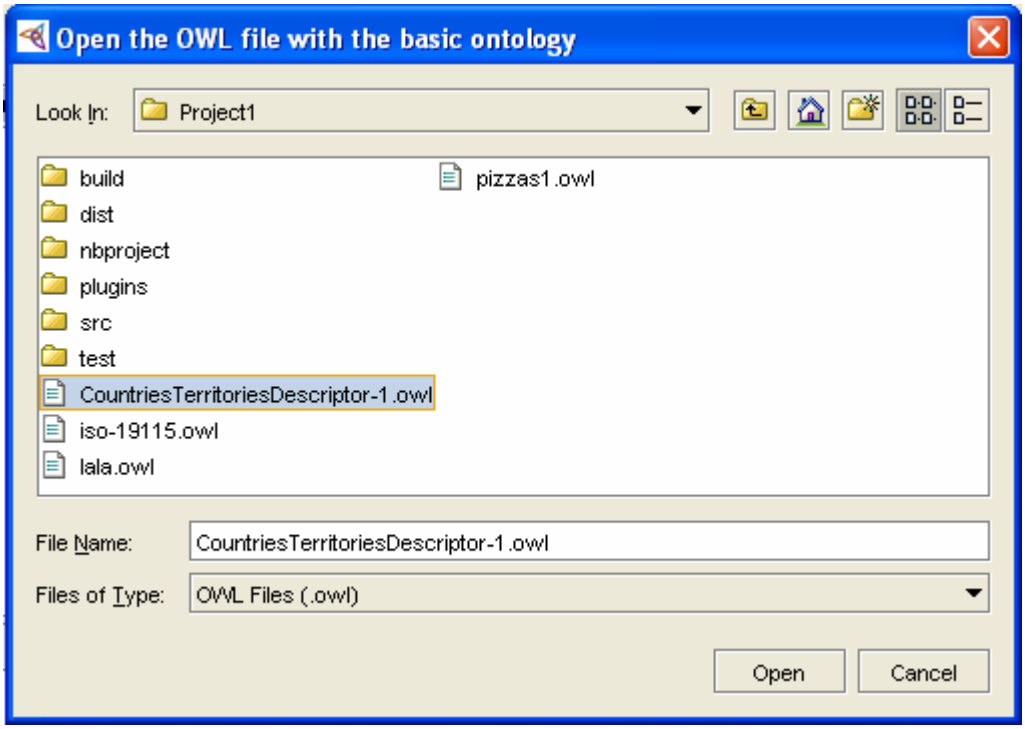
Εικόνα 5.2 Γραφικό περιβάλλον εφαρμογής

5.3.1 Φόρτωση και Απεικόνιση Οντολογίας

Το πρώτο βήμα για την αντιστοίχιση είναι η επιλογή της οντολογίας με βάση την οποία θα προχωρήσει η διαδικασία. Η επιλογή αυτή γίνεται με το πάτημα ενός κουμπιού και την εμφάνιση ενός παραθύρου διαλόγου.

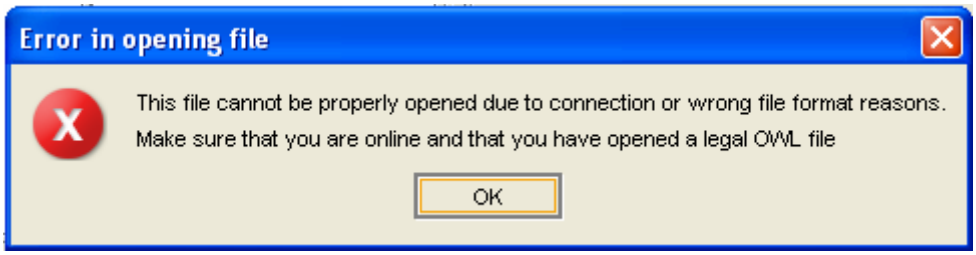


Εικόνα 5.3 Κουμπιά ανοίγματος οντολογίας, φόρτωσης και αποθήκευσης αντιστοιχίσεων



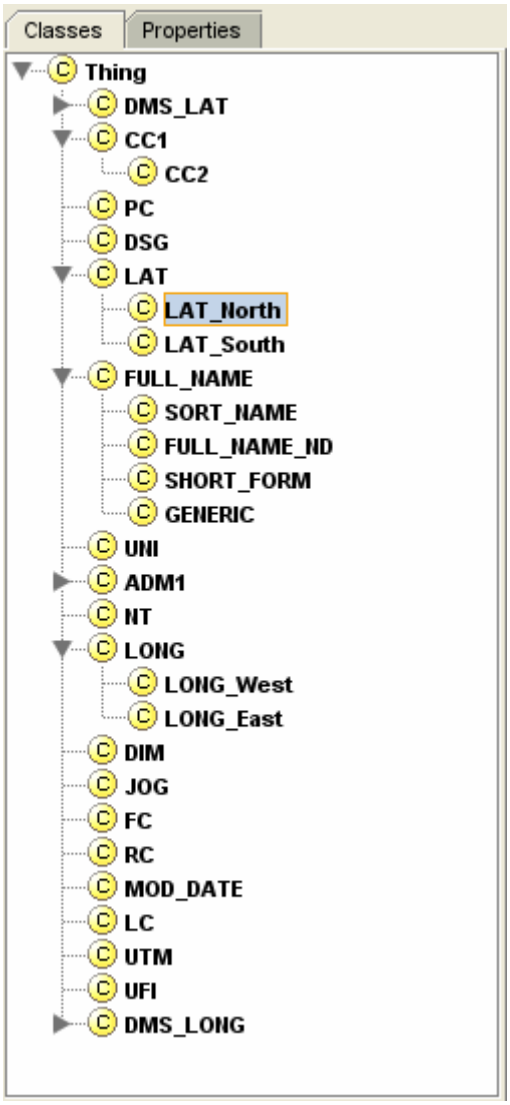
Εικόνα 5.4 Παράθυρο ανοίγματος οντολογίας

Αποδεκτά αρχεία οντολογίας είναι αυτά που έχουν κατάληξη .owl και συμμορφώνονται με τους κανόνες της γλώσσας OWL. Σε αντίθετη περίπτωση, εμφανίζεται ένα μήνυμα λάθους.

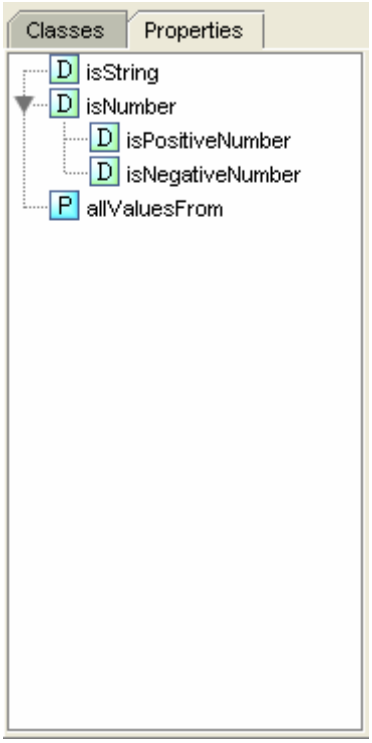


Εικόνα 5.5 Μήνυμα λάθους στο άνοιγμα έγκυρης οντολογίας

Όταν επιλεχθεί αποδεκτό αρχείο, απεικονίζεται σε **δενδρική μορφή** η οντολογία χρησιμοποιώντας μία καρτέλα για τις κλάσεις και μία για τις ιδιότητες.

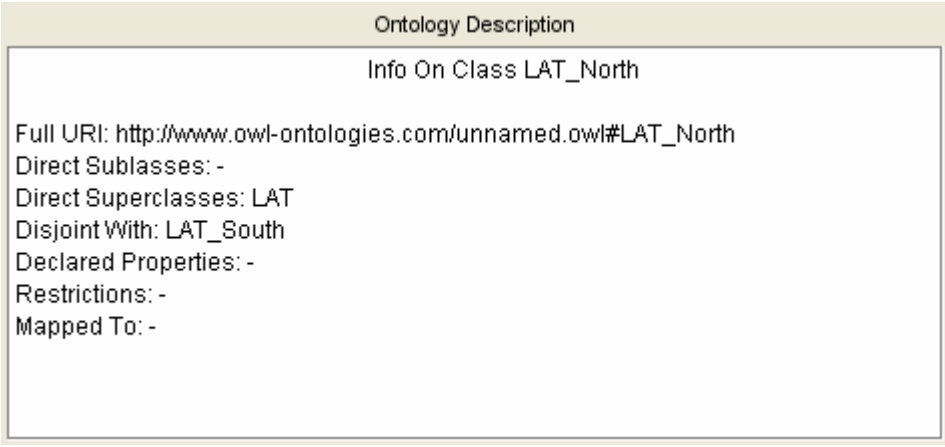


Εικόνα 5.6 Δενδρική μορφή κλάσεων οντολογίας

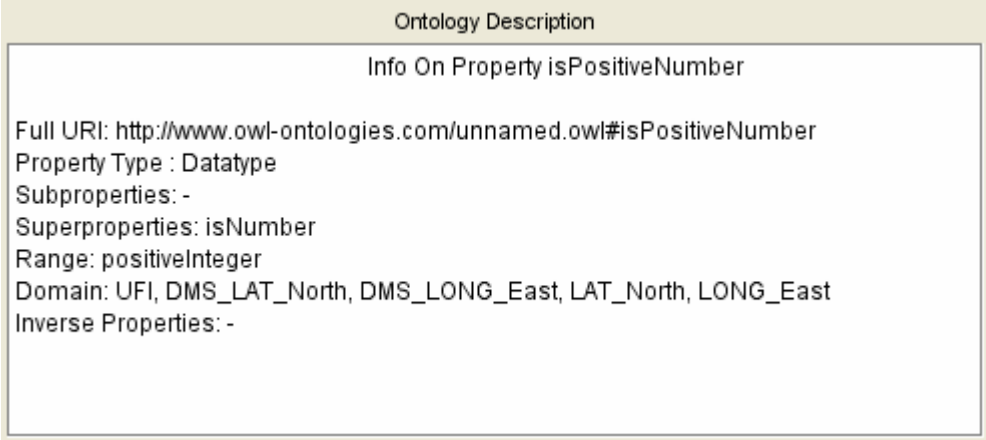


Εικόνα 5.7 Δενδρική μορφή ιδιοτήτων οντολογίας

Επιλέγοντας κάποιον κόμβο του δέντρου, εμφανίζονται σε ειδικό χώρο πληροφορίες για αυτόν τον κόμβο, είτε αυτός αναφέρεται σε κλάση είτε σε ιδιότητα, με σκοπό την πληρέστερη απεικόνιση της οντολογίας. Επίσης, οι πληροφορίες αυτές, περιγράφοντας τους κανόνες και τους περιορισμούς της οντολογίας, διευκολύνουν το χρήστη να βελτιστοποιήσει την αντιστοίχιση και να ελαχιστοποιήσει τα νοηματικά λάθη. Για τις **κλάσεις**, οι πληροφορίες που εμφανίζονται είναι οι υποκλάσεις (subclasses), υπερκλάσεις (superclasses) και οι ξένες μεταξύ τους κλάσεις (disjoint classes) της επιλεγμένης κλάσης, οι δηλωμένες ιδιότητές της καθώς και τυχόν σχετιζόμενοι περιορισμοί ή υπάρχουσες αντιστοιχίσεις με αυτήν. Για τις **ιδιότητες** αντίστοιχα, οι πληροφορίες που εμφανίζονται είναι ο τύπος τους (object, datatype, annotation), οι υπο-ιδιότητες (sub-properties), οι υπερ-ιδιότητες (super-properties), το πεδίο ορισμού (domain), το πεδίο τιμών (range) και οι αντίστροφες ιδιότητες (inverse properties) της επιλεγμένης ιδιότητας.



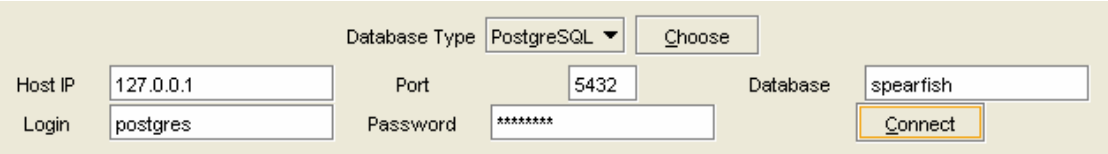
Εικόνα 5.8 Πεδίο πληροφοριών για μια κλάση της οντολογίας



Εικόνα 5.9 Πεδίο πληροφοριών για μια ιδιότητα της οντολογίας

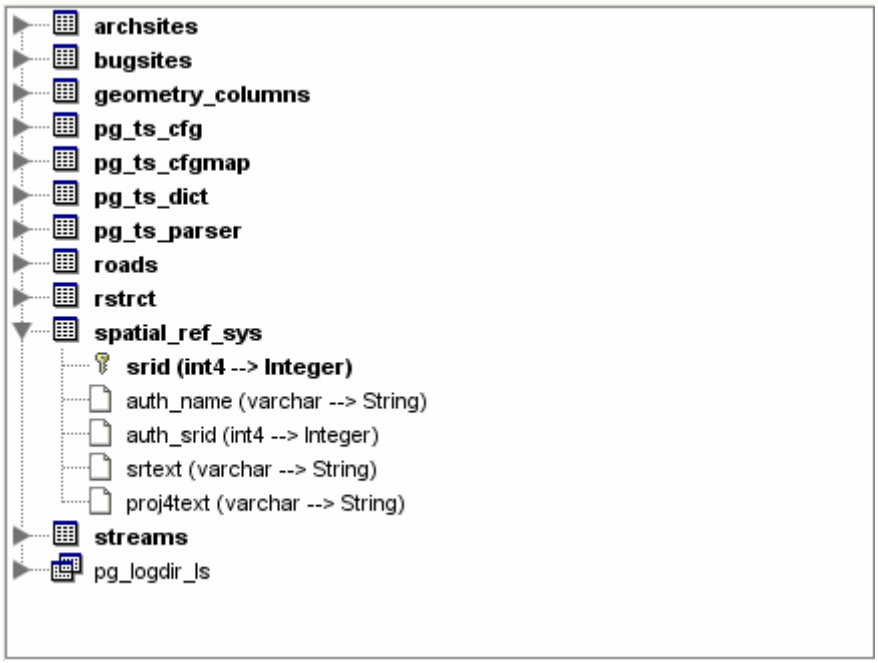
5.3.2 Φόρτωση και Απεικόνιση Βάσης Δεδομένων

Το σύστημα επιτρέπει την επιλογή βάσης δεδομένων μεταξύ των δύο υποστηριζόμενων βάσεων (MySQL και PostgreSQL). Οποιαδήποτε επιλογή οδηγεί στο αντίστοιχο παράθυρο σύνδεσης στο οποίο αν ο χρήστης εισάγει τα σωστά στοιχεία, θα συνδεθεί επιτυχώς με τη βάση που επιθυμεί.



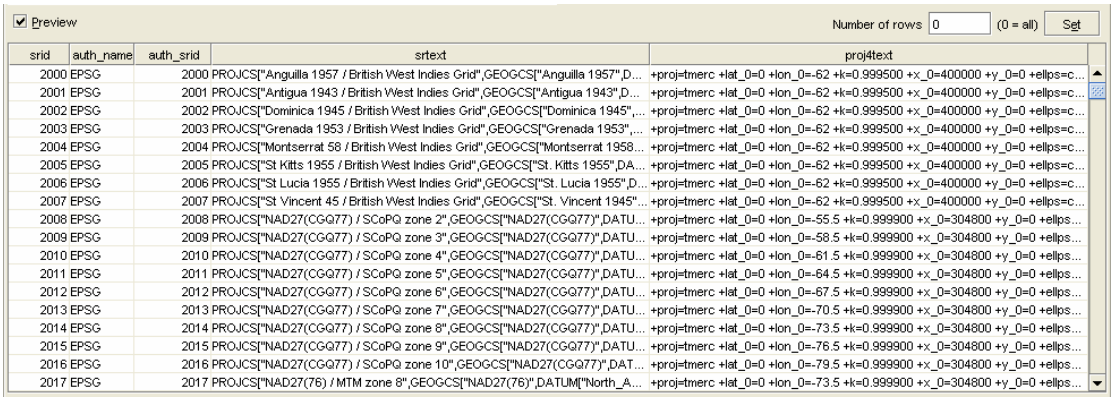
Εικόνα 5.10 Πεδία σύνδεσης με τη βάση δεδομένων

Μόλις επιτευχθεί η σύνδεση, εμφανίζεται το **σχήμα της βάσης σε δενδρική μορφή**, οπότε διατρέχοντας την ιεραρχία της βάσης, φαίνονται οι πίνακες, οι στήλες κάθε πίνακα και οι περιορισμοί κλειδιών.



Εικόνα 5.11 Δενδρική Μορφή Βάσης Δεδομένων

Επιπλέον, υποστηρίζεται η δυνατότητα **προεπισκόπησης των δεδομένων** κάθε πίνακα, όταν αυτός είναι επιλεγμένος. Η δυνατότητα αυτή ενεργοποιείται ή απενεργοποιείται αντίστοιχα ανάλογα με τις απαιτήσεις του χρήστη.


Preview

Number of rows 0 (0 = all) Set

srid	auth_name	auth_srid	srtext	proj4text
2000	EPSG	2000	PROJCS["Anguilla 1957 / British West Indies Grid",GEOGCS["Anguilla 1957",D...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2001	EPSG	2001	PROJCS["Antigua 1943 / British West Indies Grid",GEOGCS["Antigua 1943",D...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2002	EPSG	2002	PROJCS["Dominica 1945 / British West Indies Grid",GEOGCS["Dominica 1945",...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2003	EPSG	2003	PROJCS["Grenada 1953 / British West Indies Grid",GEOGCS["Grenada 1953",...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2004	EPSG	2004	PROJCS["Montserrat 58 / British West Indies Grid",GEOGCS["Montserrat 1958...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2005	EPSG	2005	PROJCS["St Kitts 1955 / British West Indies Grid",GEOGCS["St. Kitts 1955",DA...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2006	EPSG	2006	PROJCS["St Lucia 1955 / British West Indies Grid",GEOGCS["St. Lucia 1955",D...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2007	EPSG	2007	PROJCS["St Vincent 45 / British West Indies Grid",GEOGCS["St. Vincent 1945"...	+proj=tmerc +lat_0=0 +lon_0=-62 +k=0.999500 +x_0=400000 +y_0=0 +ellps=c...
2008	EPSG	2008	PROJCS["NAD27(CGQ77) / SCoPQ zone 2",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-55.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2009	EPSG	2009	PROJCS["NAD27(CGQ77) / SCoPQ zone 3",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-58.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2010	EPSG	2010	PROJCS["NAD27(CGQ77) / SCoPQ zone 4",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-61.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2011	EPSG	2011	PROJCS["NAD27(CGQ77) / SCoPQ zone 5",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-64.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2012	EPSG	2012	PROJCS["NAD27(CGQ77) / SCoPQ zone 6",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-67.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2013	EPSG	2013	PROJCS["NAD27(CGQ77) / SCoPQ zone 7",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-70.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2014	EPSG	2014	PROJCS["NAD27(CGQ77) / SCoPQ zone 8",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2015	EPSG	2015	PROJCS["NAD27(CGQ77) / SCoPQ zone 9",GEOGCS["NAD27(CGQ77)",DATU...	+proj=tmerc +lat_0=0 +lon_0=-76.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2016	EPSG	2016	PROJCS["NAD27(CGQ77) / SCoPQ zone 10",GEOGCS["NAD27(CGQ77)",DAT...	+proj=tmerc +lat_0=0 +lon_0=-79.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...
2017	EPSG	2017	PROJCS["NAD27(76) / MTM zone 8",GEOGCS["NAD27(76)",DATUM["North_A...	+proj=tmerc +lat_0=0 +lon_0=-73.5 +k=0.999900 +x_0=304800 +y_0=0 +ellps...

Εικόνα 5.12 Προεπισκόπηση Δεδομένων ενός Πίνακα της Βάσης Δεδομένων

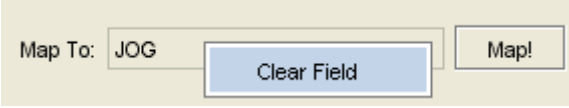
Σημειώνεται ότι είναι ορατή και η μετατροπή τύπου δεδομένων κάθε στήλης από τον τύπο της βάσης σε τύπο δεδομένων που υποστηρίζει το Protégé.

5.3.3 Αντιστοίχιση (Mapping)

Με την επιτυχή φόρτωση της οντολογίας και της βάσης δεδομένων, ο χρήστης έχει τη δυνατότητα να αρχίσει τη διαδικασία της αντιστοίχισης. Η διαδικασία της αντιστοίχισης περιλαμβάνει την επιλογή κάποιων δεδομένων της βάσης και την επιλογή της κλάσης της οντολογίας στην οποία θα αντιστοιχηθούν αυτά τα δεδομένα.

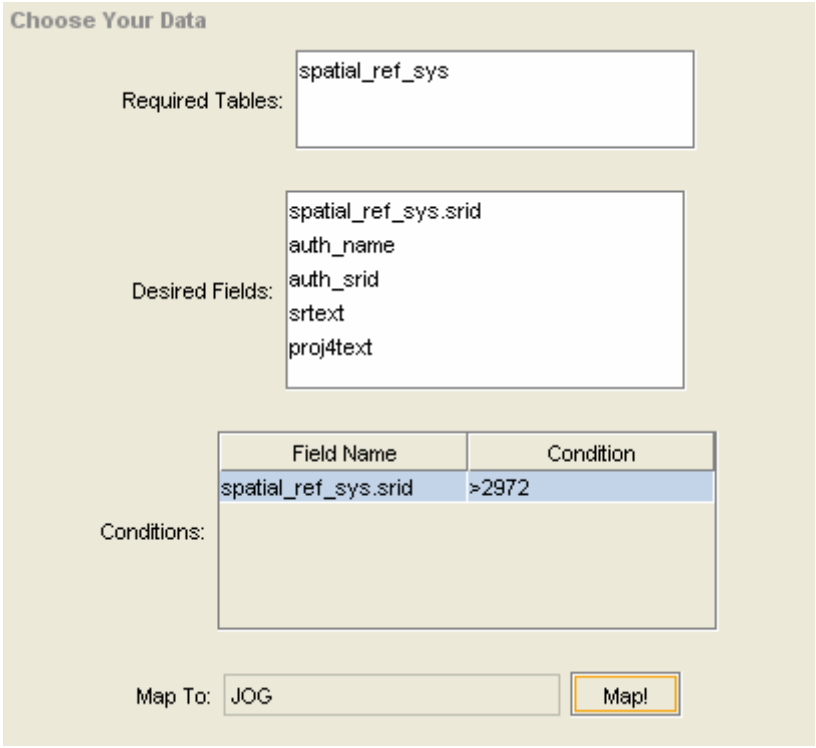
Η **επιλογή της κλάσης προς αντιστοίχιση** γίνεται με γραφικό τρόπο, διαλέγοντάς την από τη δενδρική ιεραρχία και σέρνοντάς τη (Drag & Drop) στο κατάλληλο πεδίο

αντιστοίχισης. Σε περίπτωση λάθους δίνεται η δυνατότητα διαγραφής του πεδίου αντιστοίχισης.



Εικόνα 5.13 Πεδίο αντιστοίχισης και δυνατότητα διαγραφής του

Όσον αφορά στην **επιλογή των δεδομένων**, στόχος είναι η γέννηση ενός ερωτήματος σε γλώσσα βάσης δεδομένων (SQL Query) το οποίο θα επιστρέφει τα επιθυμητά δεδομένα. Για τη διευκόλυνση του χρήστη, η διαδικασία αυτή γίνεται με **γραφικό τρόπο**. Για το λόγο αυτό, διατίθενται τα κατάλληλα πεδία που θα περιέχουν τους πίνακες, τις στήλες, καθώς και τυχόν συνθήκες που χαρακτηρίζουν τα δεδομένα. Οι στήλες στις οποίες ανήκουν τα επιθυμητά δεδομένα μπορούν να επιλεγούν από τη δενδρική ιεραρχία της βάσης και να συρθούν (Drag & Drop) στο αντίστοιχο πεδίο, ενώ το ίδιο ισχύει και για τους πίνακες των δεδομένων. Επίσης, σε περίπτωση που χρειάζονται να τεθούν κάποιοι περιορισμοί, αρκεί ο χρήστης να σύρει τη στήλη που θέλει στο πεδίο κριτηρίων και να εισάγει δίπλα της τον επιθυμητό περιορισμό.

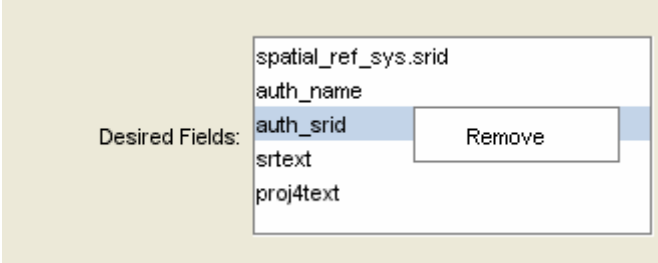


Εικόνα 5.14 Διαδικασία επιλογής δεδομένων προς αντιστοίχιση

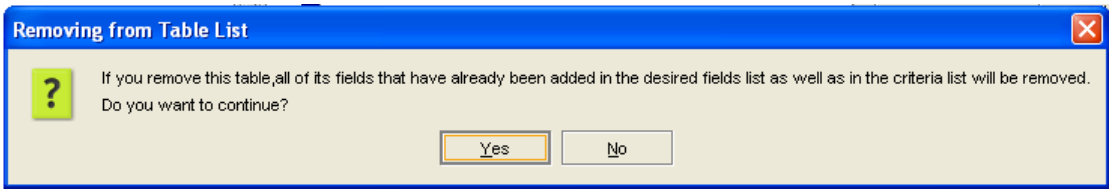
Σημειώνεται ότι η παραπάνω διαδικασία είναι απόλυτα **αυτοματοποιημένη**, οπότε εισάγοντας έναν πίνακα στο πεδίο πινάκων προστίθενται όλες οι στήλες του στο πεδίο στηλών. Αυτό είναι χρήσιμο στην περίπτωση που ένας πίνακας έχει μεγάλο αριθμό στηλών και ο χρήστης θέλει να τις εισάγει όλες, αποφεύγοντας να τις σύρει μία προς μία στο πεδίο στηλών.

Εικόνα 5.15 Παράθυρο Διαλόγου για την εισαγωγή πίνακα στην αντίστοιχη λίστα

Επίσης, τοποθετώντας απλώς μία στήλη στο πεδίο στηλών ή στο πεδίο κριτηρίων, τίθεται αυτόματα και ο αντίστοιχος πίνακας στο πεδίο πινάκων. Ακόμα, σε περίπτωση ύπαρξης **στηλών με το ίδιο όνομα** που ανήκουν σε διαφορετικούς πίνακες, ελήφθη μέριμνα ώστε να τοποθετείται μπροστά στο όνομα της στήλης το όνομα του πίνακά της. Με αυτόν τον τρόπο, διασφαλίζεται ότι κάθε στήλη θα έχει ένα μοναδικό όνομα, έτσι ώστε η επιλογή των δεδομένων να γίνει σωστά. Τέλος, υπάρχει η δυνατότητα διαγραφής ενός η περισσοτέρων στοιχείων από οποιοδήποτε πεδίο, η οποία και αυτή ακολουθεί κατάλληλη αυτοματοποίηση για την αποφυγή λαθών.

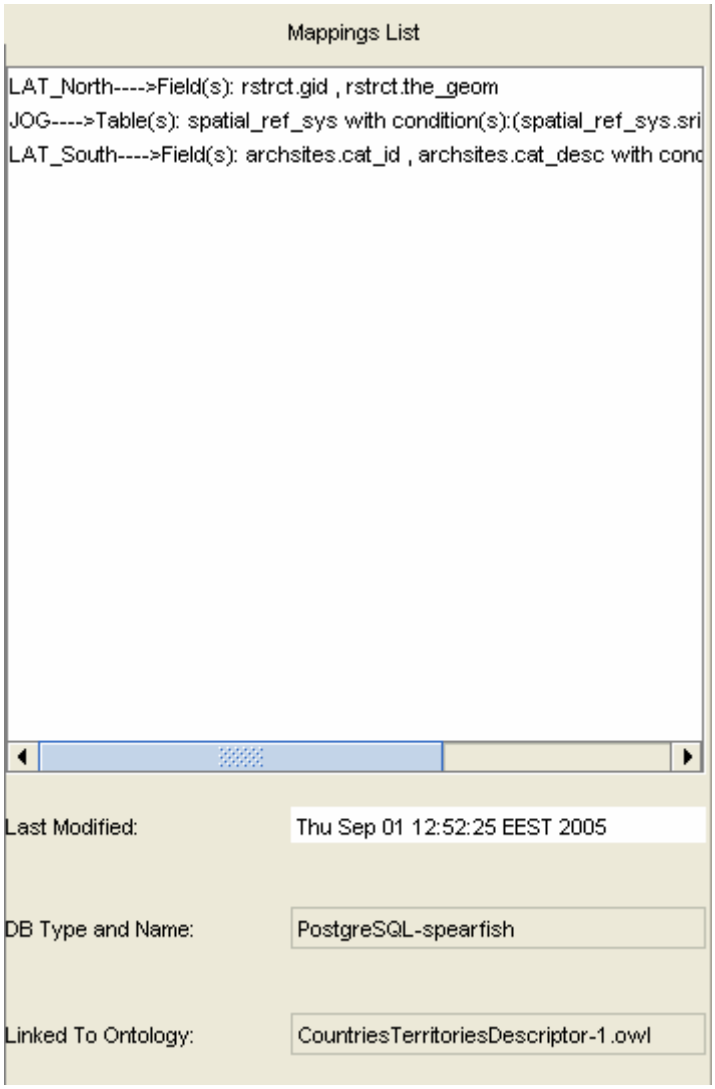


Εικόνα 5.16 Μενού αφαίρεσης πεδίων από την αντίστοιχη λίστα



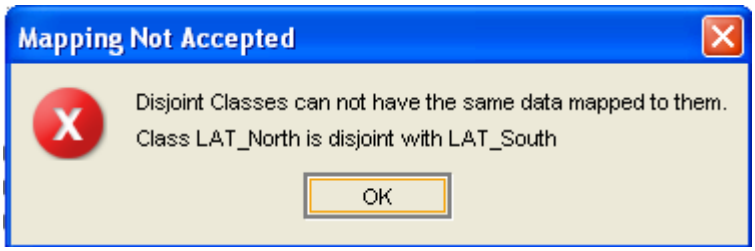
Εικόνα 5.17 Παράθυρο διαλόγου για την αφαίρεση πίνακα από την αντίσοτιχη λίστα

Μόλις η διαδικασία επιλογής δεδομένων ολοκληρωθεί και με την προϋπόθεση υπάρχει στο πεδίο αντιστοίχισης το όνομα μίας κλάσης της οντολογίας, το πάτημα του κουμπιού αντιστοίχισης ολοκληρώνει την όλη διαδικασία. **Η αντιστοίχιση γίνεται δεκτή, εκτός αν οι κανόνες και οι περιορισμοί της οντολογίας δεν την επιτρέπουν**, ενώ ταυτόχρονα προστίθεται μια λεπτομερής περιγραφή της αντιστοίχισης σε κατάλληλο πεδίο, όπου συγκεντρώνονται όλες οι αντιστοιχίσεις. Το πεδίο αυτό βοηθά, καθώς απεικονίζονται όλες οι αντιστοιχίσεις που έχουν γίνει και προσφέρεται η δυνατότητα αφαίρεσης οποιασδήποτε από αυτές.



Εικόνα 5.18 Λίστα αντιστοιχίσεων και συμπληρωματικές πληροφορίες

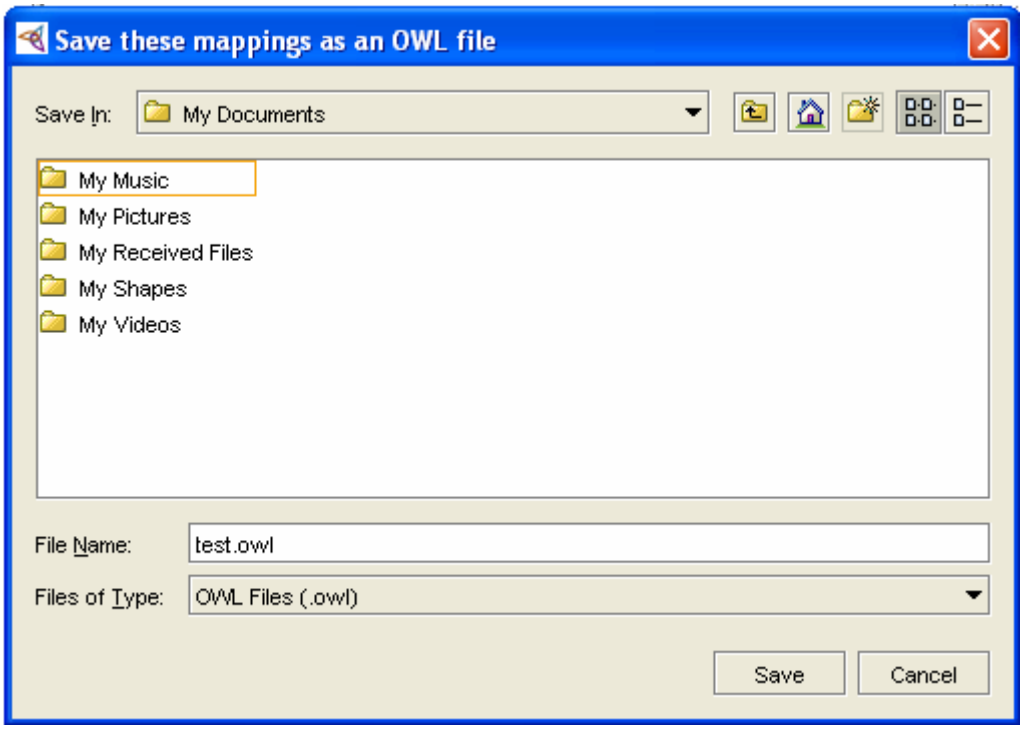
Οι **έλεγχοι** που πραγματοποιούνται συνίστανται σε σύγκριση των δεδομένων που έχουν αντιστοιχηθεί σε ξένες μεταξύ τους κλάσεις, υποκλάσεις και υπερκλάσεις. Σε περίπτωση που η αντιστοίχιση απορριφθεί, εμφανίζεται μήνυμα λάθους, το οποίο περιγράφει σαφώς το λόγο απόρριψης.



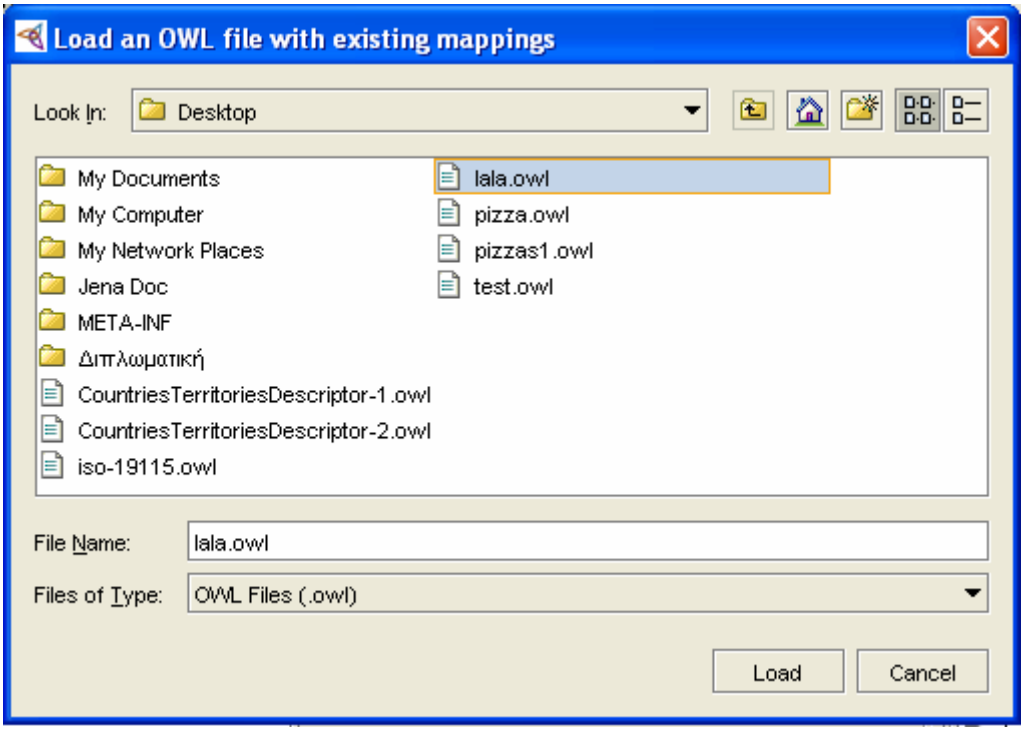
Εικόνα 5.19 Μήνυμα απόρριψης αντιστοίχισης (σημασιολογικός έλεγχος ξένων κλάσεων)

Εικόνα 5.20 Μήνυμα απόρριψης αντιστοίχισης (σημασιολογικός έλεγχος υποκλάσεων)

Σε οποιαδήποτε στιγμή της διαδικασίας και εφόσον έχει γίνει τουλάχιστον μία αντιστοίχιση, ο χρήστης μπορεί να **αποθηκεύσει** σε ξεχωριστό αρχείο κατάληξης .owl τις αντιστοιχίσεις που έχει πραγματοποιήσει. Με τον τρόπο αυτό, η διαδικασία της απεικόνισης μπορεί να συνεχιστεί κάποια άλλη στιγμή. Η δυνατότητα αυτή παρέχεται μέσω της λειτουργίας **φόρτωσης** αρχείου με κατάληξη .owl που περιέχει υπάρχουσες απεικονίσεις. Το σύστημα δεν επιτρέπει τη φόρτωση τέτοιου αρχείου χωρίς να έχει προηγηθεί άνοιγμα της βασικής οντολογίας. Ακόμα, πραγματοποιείται έλεγχος για το αν το αρχείο με τις αντιστοιχίσεις που πρόκειται να φορτωθεί συμφωνεί (ως προς το όνομα αρχείου) με τη βασική οντολογία που είναι ανοικτή.

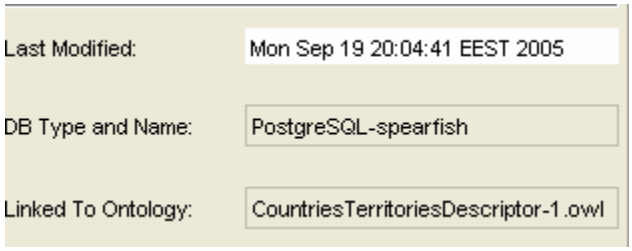


Εικόνα 5.21 Παράθυρο αποθήκευσης αντιστοιχίσεων



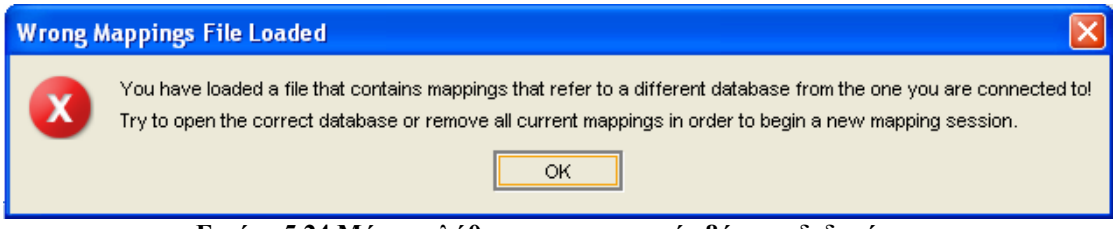
Εικόνα 5.22 Παράθυρο φόρτωσης αντιστοιχίσεων

Επίσης, παρουσιάζονται πληροφορίες που χαρακτηρίζουν το αρχείο, όπως η **ημερομηνία τελευταίας αντιστοίχισης**, ο **τύπος** και το **όνομα** της **βάσης δεδομένων** στην οποία αναφέρονται οι αντιστοιχίσεις, και το **όνομα αρχείου** της **οντολογίας** στην οποία βασίστηκε η όλη διαδικασία.

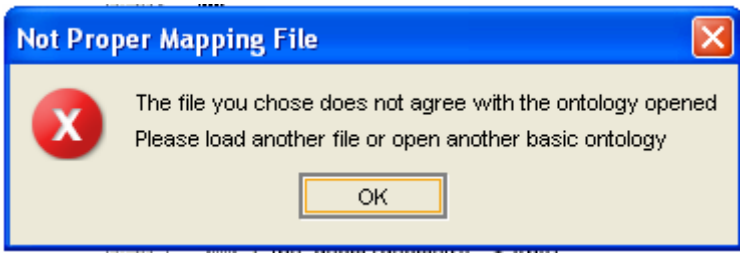


Εικόνα 5.23 Πληροφορίες αντιστοίχισης

Αξίζει να σημειωθεί ότι η εφαρμογή δεν επιτρέπει την αποθήκευση στο ίδιο αρχείο αντιστοιχίσεων που αναφέρονται σε διαφορετικές βάσεις δεδομένων ή σε διαφορετικές οντολογίες. Σε περίπτωση που ο χρήστης προσπαθήσει κάτι τέτοιο, θα εμφανιστεί μήνυμα λάθους.



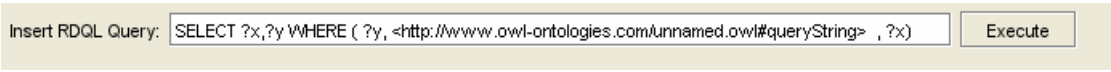
Εικόνα 5.24 Μήνυμα λάθους για ασυμφωνία βάσεων δεδομένων



Εικόνα 5.25 Μήνυμα λάθους για ασυμφωνία οντολογίας και αρχείου αντιστοιχίσεων

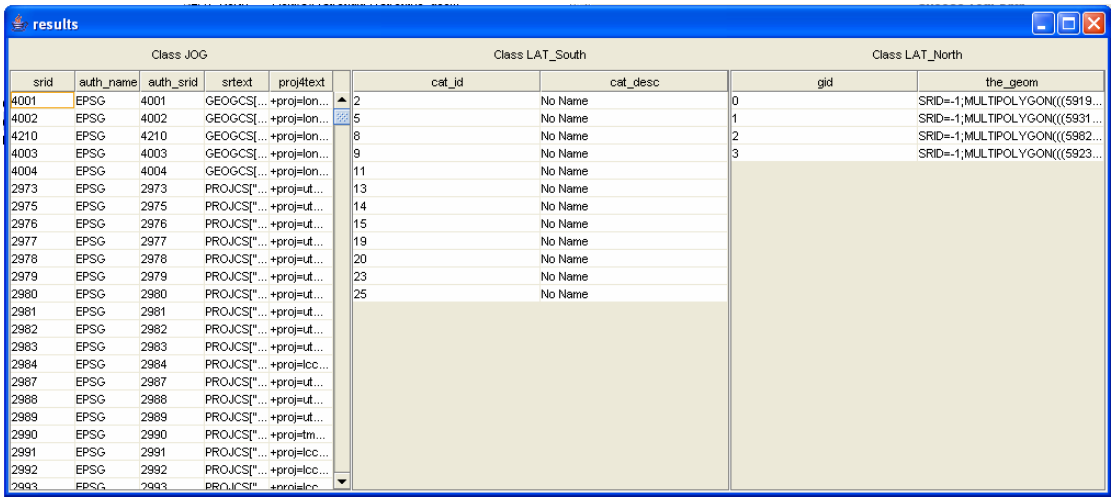
5.3.4 Εκτέλεση Σημασιολογικού Ερωτήματος (RDQL Query)

Για τον **έλεγχο της ορθότητας** της αντιστοίχισης κλάσεων μιας οντολογίας με δεδομένα μιας βάσης, υπάρχει πεδίο κειμένου στο οποίο εισάγεται ένα σημασιολογικό ερώτημα (RDQL Query).



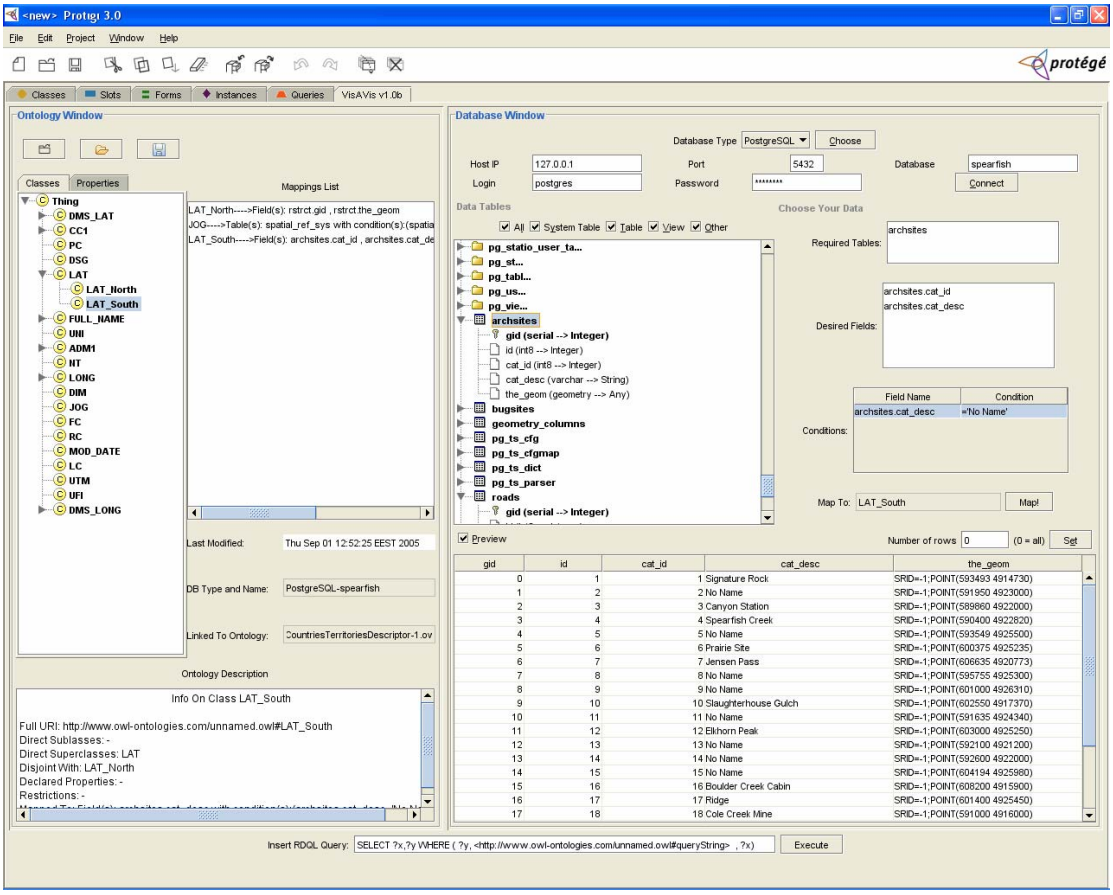
Εικόνα 5.26 Πεδίο κειμένου για εκτέλεση RDQL Query

Με την εκτέλεση του ερωτήματος, επιστρέφονται τα ονόματα των κλάσεων που απαντούν στο ερώτημα και που έχουν προηγουμένως αντιστοιχηθεί σε κάποια δεδομένα. Τα δεδομένα αυτά μαζί με τις αντίστοιχες κλάσεις της οντολογίας εμφανίζονται σε ξεχωριστό παράθυρο αποτελεσμάτων. Έτσι επιτυγχάνεται η εξόρυξη δεδομένων από τη βάση μέσω του σημασιολογικού ερωτήματος που εφαρμόζεται στην οντολογία.



Εικόνα 5.27 Αποτελέσματα εκτέλεσης RDQL Query

Σημειώνεται ότι για να εκτελεστεί σωστά το σημασιολογικό ερώτημα, θα πρέπει να είναι ανοικτή η επιθυμητή οντολογία, να έχουν γίνει κάποιες αντιστοιχήσεις και να έχει επιτευχθεί σύνδεση με τη σωστή βάση δεδομένων. Σε αντίθετη περίπτωση, εμφανίζεται μήνυμα λάθους.



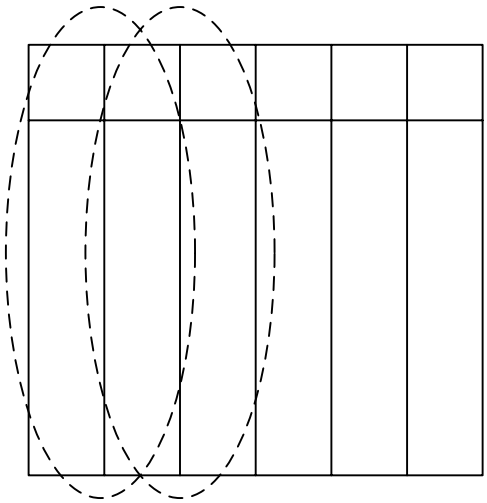
Εικόνα 5.28 Συνολική εικόνα εφαρμογής

5.4 ΕΛΕΓΧΟΙ ΣΗΜΑΣΙΟΛΟΓΙΚΗΣ ΟΡΘΟΤΗΤΑΣ

Κατά το στάδιο της αντιστοίχισης, πραγματοποιούνται έλεγχοι σημασιολογικής ορθότητας προκειμένου να καταχωρηθεί μια αντιστοίχιση ως αποδεκτή ή όχι. Οι περιπτώσεις που εξετάζονται είναι ο έλεγχος των δεδομένων που απεικονίζονται σε ξένες μεταξύ τους κλάσεις, καθώς και ο έλεγχος των δεδομένων που απεικονίζονται σε κλάσεις που συνδέονται με μια σχέση υπερκλάσης/υποκλάσης.

5.4.1 Έλεγχος των ξένων κλάσεων

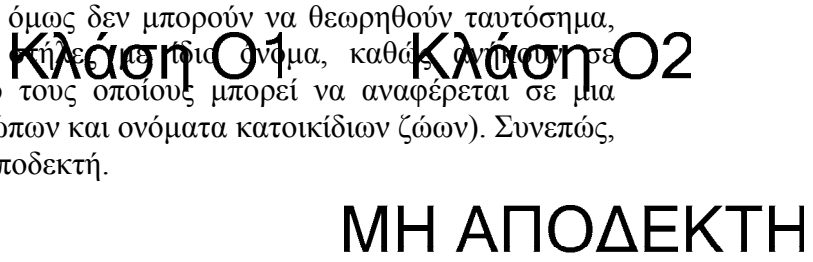
Ο πρώτος έλεγχος που επιτελείται προκειμένου να εξακριβωθεί αν μια δεδομένη αντιστοίχιση είναι σημασιολογικά ορθή είναι ο έλεγχος των δεδομένων που έχουν αντιστοιχηθεί σε **κλάσεις ξένες προς την τρέχουσα απεικονιζόμενη κλάση**. Δύο κλάσεις ονομάζονται ξένες ή μη επικαλυπτόμενες (disjoint), όταν δεν έχουν κοινά στοιχεία. Συνεπώς, είναι λογικό τα δεδομένα στα οποία αντιστοιχούνται δύο ξένες μεταξύ τους κλάσεις να είναι διαφορετικά, δηλαδή **να μην επικαλύπτονται**.

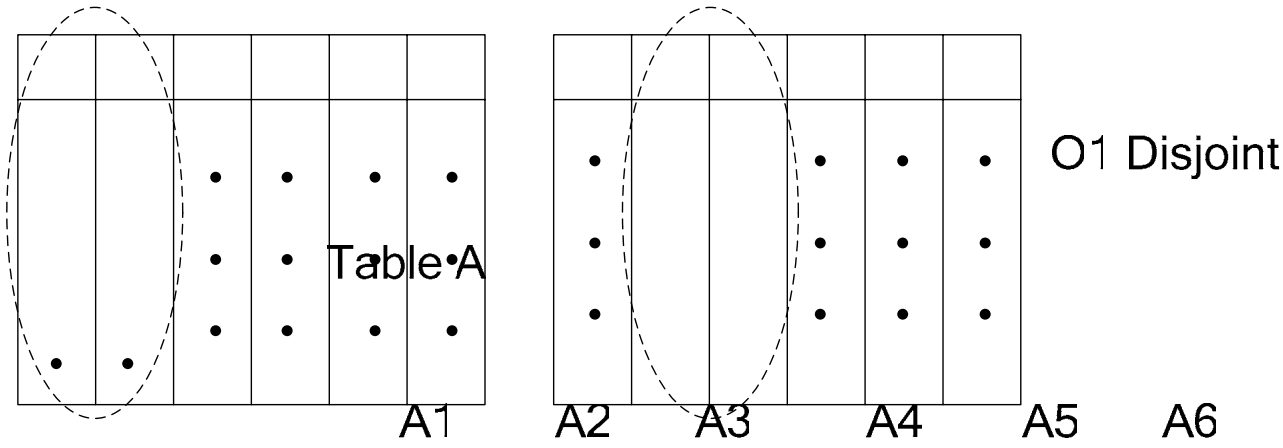


Εικόνα 5.29 Μη αποδεκτή αντιστοίχιση λόγω κοινών δεδομένων

Αρχικά, ανακτώνται όλες οι ξένες κλάσεις της υπό έλεγχο κλάσης έτσι ώστε να συγκριθούν τα δεδομένα που έχουν αντιστοιχηθεί σε αυτές με αυτά που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση. Τα δεδομένα που έχουν απεικονιστεί στις ξένες κλάσεις της υπό έλεγχο κλάσης συγκρίνονται ένα προς ένα με τα δεδομένα που πρόκειται να απεικονιστούν στην υπό έλεγχο κλάση και αν βρεθεί **έστω και ένα κοινό δεδομένο** μεταξύ των δύο ομάδων δεδομένων, **η αντιστοίχιση απορρίπτεται**. Συγκεκριμένα, για να θεωρηθούν δύο δεδομένα ταυτόσημα, πρέπει να έχουν την ίδια τιμή, να ανήκουν στην ίδια στήλη και στον ίδιο πίνακα. Συνεπώς, η εύρεση δύο δεδομένων με την ίδια τιμή δε συνεπάγεται ταυτότητα αυτών των δεδομένων, αφού πρέπει να ελεγχθούν παράλληλα τα ονόματα των στηλών και των πινάκων στα οποία αυτά ανήκουν.

Για παράδειγμα, στο παρακάτω σχήμα, τόσο στην κλάση Ο1 όσο και στην κλάση Ο2, οι οποίες είναι ξένες μεταξύ τους, έχουν αντιστοιχηθεί δύο δεδομένα με τις ίδιες τιμές (george, jack). Τα εν λόγω δεδομένα όμως δεν μπορούν να θεωρηθούν ταυτόσημα, παρά το γεγονός ότι ανήκουν σε στήλες με ίδιο όνομα, καθώς ανήκουν σε διαφορετικούς πίνακες, καθένας από τους οποίους μπορεί να αναφέρεται σε μια ξεχωριστή έννοια (π.χ. ονόματα ανθρώπων και ονόματα κατοικίδιων ζώων). Συνεπώς, η αντιστοίχιση είναι σημασιολογικά αποδεκτή.

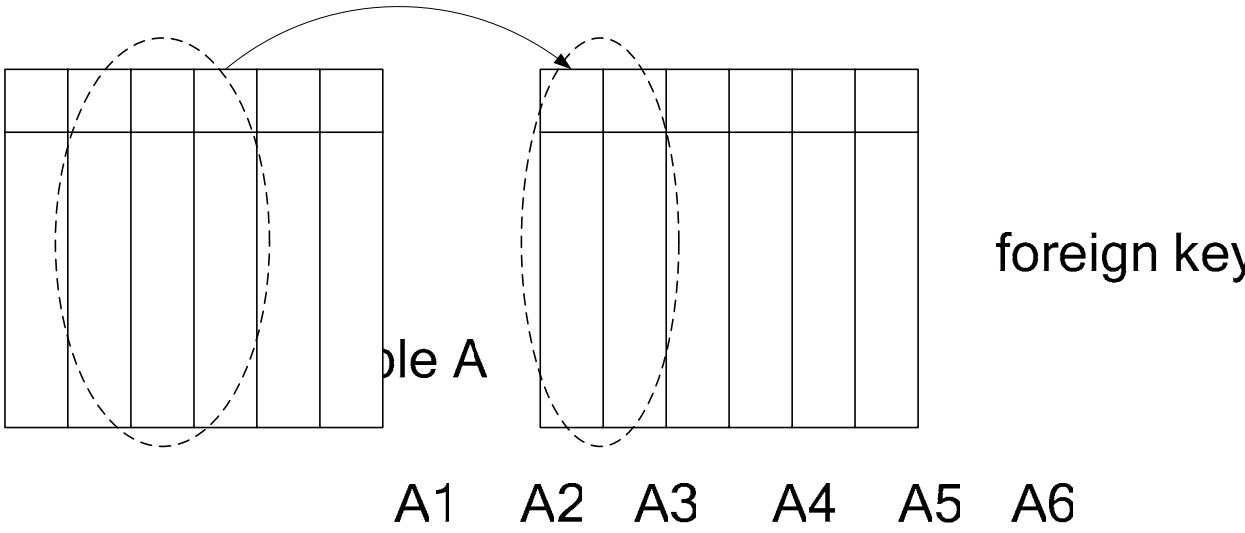




Εικόνα 5.30 Αποδεκτή αντιστοίχιση παρά την ύπαρξη ίδιων τιμών

Σημειώνεται ότι υπάρχει περίπτωση δύο δεδομένα να θεωρηθούν **ταυτόσημα**, ακόμα και αν έχουν την **ίδια τιμή αλλά ανήκουν σε διαφορετικές στήλες**. Κάτι τέτοιο συναντάται όταν μια στήλη είναι ξένο κλειδί σε μια άλλη στήλη, οπότε τα δεδομένα που περιέχονται σε αυτήν είναι ουσιαστικά τα ίδια. Επομένως, εκτός από τον έλεγχο τιμής των δεδομένων, ελέγχεται και αν η στήλη(-ες) στην οποία ανήκουν τα δεδομένα προς αντιστοίχιση με την υπό έλεγχο κλάση, αποτελεί ξένο κλειδί σε μια στήλη άλλου πίνακα η οποία περιέχει δεδομένα που έχουν ήδη απεικονιστεί σε μια ξένη προς αυτή κλάση. Επίσης, χρειάζεται να εξεταστεί και η αντίστροφη περίπτωση. Συγκεκριμένα, ελέγχεται αν κάποια στήλη που περιέχει δεδομένα που έχουν απεικονιστεί σε μια ξένη κλάση προς την υπό έλεγχο κλάση αποτελεί ξένο κλειδί σε μια στήλη στην οποία περιέχονται δεδομένα που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση.

Ένα απλό παράδειγμα φαίνεται στο παρακάτω σχήμα, όπου οι ξένες κλάσεις Ο1 και Ο2 έχουν απεικονιστεί σε στήλες που ανήκουν σε διαφορετικούς πίνακες και συνεπώς, εκ πρώτης όψεως η αντιστοίχιση αυτή θα έπρεπε να είναι αποδεκτή. Όμως, η στήλη Α4 είναι ξένο κλειδί στη στήλη Β1 και συνεπώς, θα περιέχει τιμές που επίσης περιέχονται στη Β1 και που αναφέρονται σημασιολογικά στην ίδια έννοια. Άρα, γίνεται φανερό ότι αυτή η αντιστοίχιση δεν είναι σημασιολογικά αποδεκτή.



Εικόνα 5.31 Μη αποδεκτή αντιστοίχιση λόγω κοινών δεδομένων (μέσω σχέσης ξένου κλειδιού)

Αξίζει να αναφερθεί ότι ο παραπάνω έλεγχος δεν είναι αρκετός για να εξασφαλιστεί η σημασιολογική ορθότητα της αντιστοίχισης, σε ό,τι αφορά στις ξένες κλάσεις. Είναι απαραίτητο να **επεκταθεί ο έλεγχος και στα ανώτερα επίπεδα ιεραρχίας κλάσεων** και να εξεταστούν οι ξένες κλάσεις της υπερκλάσης της υπό έλεγχο κλάσης.

Στην περίπτωση που μια ξένη κλάση της υπό έλεγχο κλάσης δεν έχει απεικονιστεί σε κάποια δεδομένα, ο έλεγχος επεκτείνεται στις υποκλάσεις της, οι οποίες εξ ορισμού είναι και αυτές ξένες με την υπό έλεγχο κλάση. Ο έλεγχος γίνεται για κάθε μία από τις υποκλάσεις στις οποίες έχουν ήδη αντιστοιχηθεί κάποια δεδομένα, ενώ στην περίπτωση που κάποια υποκλάση δεν έχει απεικονιστεί σε κάποια δεδομένα, ο έλεγχος επεκτείνεται και σε χαμηλότερο επίπεδο και ούτω καθεξής.

Συμπερασματικά, μια αντιστοίχιση μιας ομάδας δεδομένων σε μια κλάση γίνεται δεκτή μόνο αν δε βρεθεί μία ξένη σε αυτήν κλάση, η οποία να έχει αντιστοιχηθεί σε τουλάχιστον ένα δεδομένο που πρόκειται να αντιστοιχηθεί στην υπό έλεγχο κλάση.

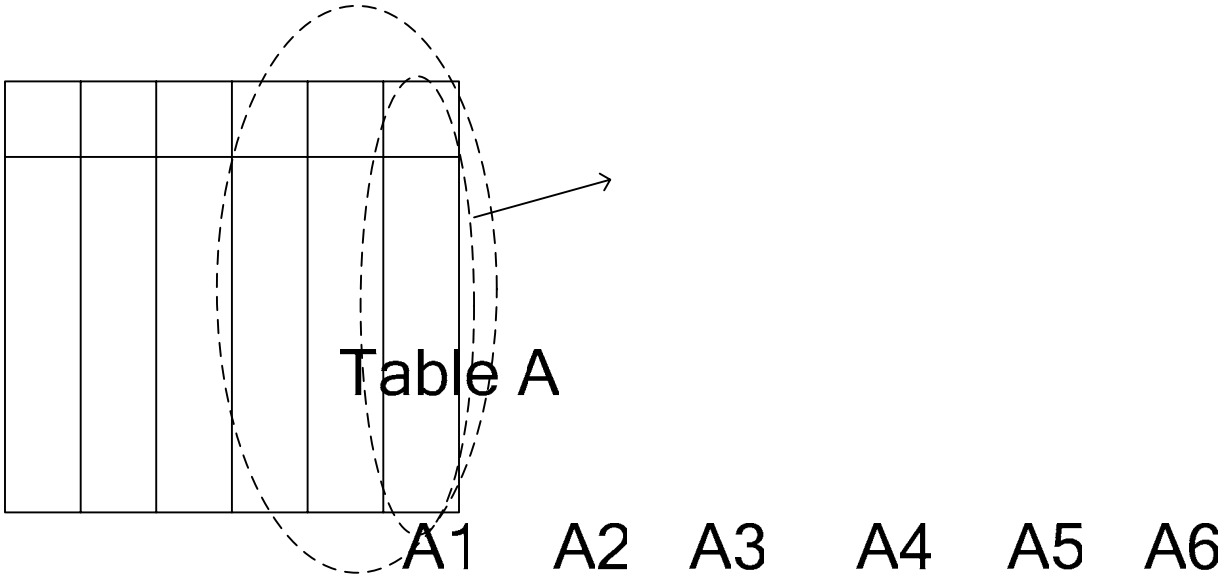
Κλάση Ο1

5.4.2 Έλεγχος υπερκλάσεων

ΜΗ Α

Ο επόμενος έλεγχος που επιτελείται για την εξασφάλιση της σημασιολογικής ορθότητας μιας αντιστοίχισης δεδομένων σε μια κλάση εξετάζει τις υπερκλάσεις αυτής της κλάσης και τα δεδομένα στα οποία αυτές έχουν τυχόν αντιστοιχηθεί. Σε αυτό το σημείο, υπενθυμίζεται ότι αν μια κλάση Α είναι υπερκλάση μιας κλάσης Β, αυτό σημαίνει ότι όλα τα άτομα που ανήκουν στην κλάση Β θα ανήκουν υποχρεωτικά και στην κλάση Α. Με βάση τον παραπάνω ορισμό, είναι λογικό ότι **τα δεδομένα που πρόκειται να αντιστοιχηθούν** σε μια κλάση πρέπει να αποτελούν **υποσύνολο**

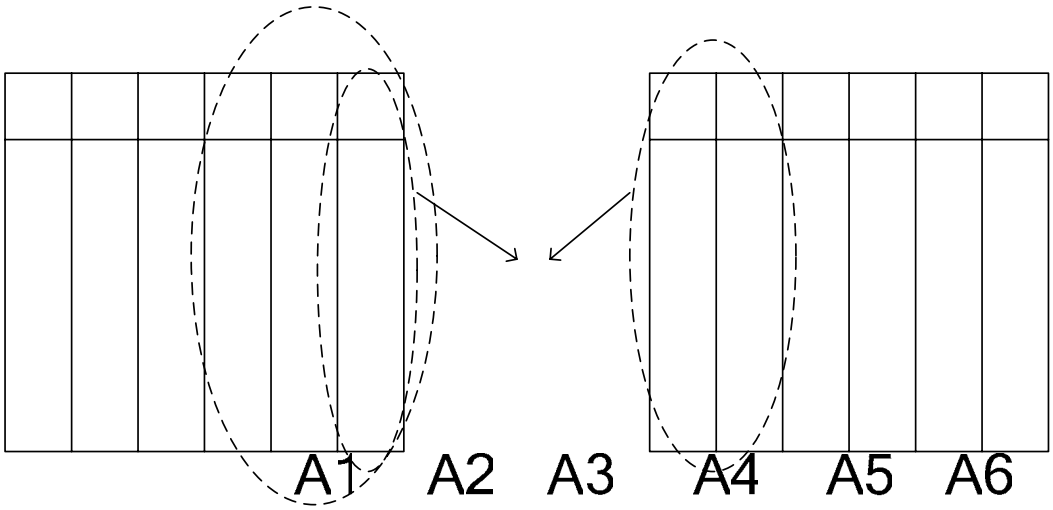
των δεδομένων που έχουν αντιστοιχηθεί σε όλες τις υπερκλάσεις αυτής της κλάσης. Αν αυτό δεν ισχύει, η αντιστοίχιση απορρίπτεται.



Εικόνα 5.32 Αποδεκτή αντιστοίχιση στην περίπτωση ελέγχου υπερκλάσεων

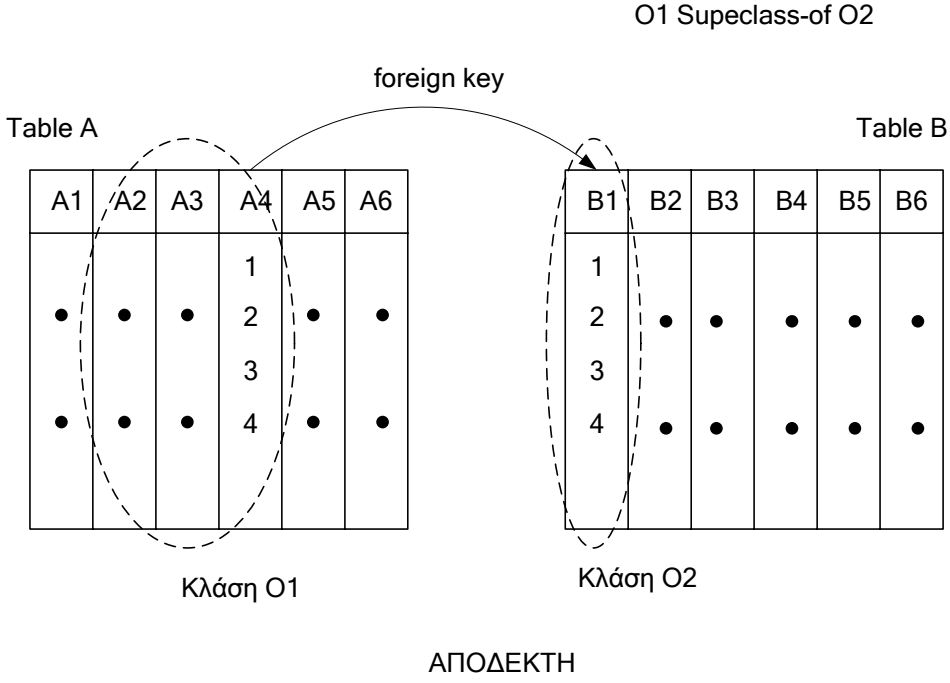
Αρχικά, ανακτώνται όλες οι άμεσες υπερκλάσεις της υπό έλεγχο κλάσης προκειμένου να συγκριθούν τα δεδομένα που έχουν αντιστοιχηθεί σε αυτές με τα δεδομένα που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση. Σημειώνεται ότι άμεσες υπερκλάσεις μιας κλάσης είναι αυτές που ιεραρχικά βρίσκονται στο αμέσως ανώτερο επίπεδο από το επίπεδο της κλάσης αυτής. Τα δεδομένα που πρόκειται να απεικονιστούν στην υπό έλεγχο κλάση εξετάζονται ένα προς ένα για να εξακριβωθεί αν υπάγονται στο σύνολο των δεδομένων που έχουν αντιστοιχηθεί σε μια άμεση υπερκλάση της υπό έλεγχο κλάσης. Αυτή η διαδικασία εκτελείται για όλες τις άμεσες υπερκλάσεις της υπό έλεγχο κλάσης. Αν βρεθεί έστω και ένα δεδομένο που πρόκειται να αντιστοιχηθεί στην υπό έλεγχο κλάση και που δε βρίσκεται μεταξύ των δεδομένων που έχουν αντιστοιχηθεί σε κάποια άμεση υπερκλάση της υπό έλεγχο κλάσης, η αντιστοίχιση απορρίπτεται.

Τυπικό παράδειγμα αποτελεί το επόμενο σχήμα, όπου η κλάση O1, η οποία είναι υπερκλάση της O2, έχει αντιστοιχηθεί σε δεδομένα που δεν περιέχουν εξ ολοκλήρου τα δεδομένα που έχουν αντιστοιχηθεί στην O2. Για αυτόν το λόγο, η αντιστοίχιση δεν είναι σημασιολογικά αποδεκτή.



Εικόνα 5.33 Μη αποδεκτή αντιστοίχιση

Όπως και στον έλεγχο ξένων κλάσεων, για να θεωρηθούν δύο δεδομένα ταυτόσημα, χρειάζεται να έχουν την ίδια τιμή, να ανήκουν στην ίδια στήλη και στον ίδιο πίνακα. Εξαίρεση αποτελεί η περίπτωση στην οποία τα δύο δεδομένα έχουν την ίδια τιμή και ανήκουν σε στήλες, η μία εκ των οποίων είναι **ξένο κλειδί** στην άλλη, οπότε και πάλι τα δεδομένα αυτά θεωρούνται ταυτόσημα. Αυτή η περίπτωση φαίνεται στο παρακάτω σχήμα, όπου η κλάση O2 έχει αντιστοιχηθεί στη στήλη B1, ενώ η O1, η οποία είναι υπερκλάση της O2, έχει αντιστοιχηθεί στις στήλες A2, A3, A4. Εκ πρώτης όψεως, η αντιστοίχιση δε θα έπρεπε να επιτραπεί, όμως δεδομένου του ότι η στήλη A4 είναι ξένο κλειδί στη B1, περιέχει τις ίδιες τιμές με αυτή και οι τιμές αυτές αναφέρονται σημασιολογικά στην ίδια έννοια, θεωρείται ότι τα δεδομένα που έχουν αντιστοιχηθεί στην O2 (στήλη B1) περιέχονται εξ ολοκλήρου στα δεδομένα που έχουν αντιστοιχηθεί στην O1 (στήλες A2, A3, A4). Συνεπώς, η αντιστοίχιση είναι σημασιολογικά αποδεκτή.



Εικόνα 5.34 Αποδεκτή αντιστοίχιση (περίπτωση ξένων κλειδιών)

Στην περίπτωση που μια άμεση υπερκλάση της υπό έλεγχο κλάσης δεν έχει απεικονιστεί σε κάποια δεδομένα, τότε ο έλεγχος πρέπει να μεταφερθεί στο αμέσως ανώτερο επίπεδο, ελέγχοντας όλες τις άμεσες υπερκλάσεις αυτής της μη αντιστοιχησθείσας κλάσης και ούτω καθεξής.

Συμπερασματικά, για να γίνει αποδεκτή μια αντιστοίχιση δεδομένων σε μια κλάση, πρέπει τα δεδομένα αυτά να είναι υποσύνολο των δεδομένων που έχουν αντιστοιχηθεί στις υπερκλάσεις αυτής της κλάσης.

5.4.3 Έλεγχος υποκλάσεων

Ο τελευταίος έλεγχος σημασιολογικής ορθότητας μιας αντιστοίχισης δεδομένων με μια κλάση μιας οντολογίας αναφέρεται στη σύγκριση των δεδομένων που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση με τα δεδομένα που έχουν αντιστοιχηθεί στις υποκλάσεις της. Όπως και προηγουμένως, τονίζεται ότι μια κλάση Α είναι υποκλάση μιας κλάσης Β, όταν όλα τα άτομα που ανήκουν στην κλάση Α ανήκουν και στην κλάση Β. Συνεπώς, για να γίνει αποδεκτή μια αντιστοίχιση, θα πρέπει **τα δεδομένα που πρόκειται να αντιστοιχηθούν σε μια κλάση να αποτελούν υπερσύνολο των δεδομένων** ή, με άλλα λόγια, να «περιέχουν» όλα τα δεδομένα **που έχουν ήδη απεικονιστεί** σε όλες τις υποκλάσεις της υπό έλεγχο κλάσης. Ουσιαστικά, ο έλεγχος αυτός εφαρμόζει τους ίδιους κανόνες με τον έλεγχο υπερκλάσεων και για αυτόν το λόγο, τα σχηματικά παραδείγματα που δόθηκαν στην προηγούμενη παράγραφο ισχύουν και για το συγκεκριμένο είδος ελέγχου.

Ο έλεγχος των υποκλάσεων της υπό έλεγχο κλάσης είναι αντίστοιχος του ελέγχου υπερκλάσεων με τη μόνη διαφορά ότι, αυτή τη φορά, θα πρέπει τα δεδομένα που

έχουν αντιστοιχηθεί στις υποκλάσεις μιας κλάσης να αποτελούν υποσύνολο των δεδομένων που πρόκειται να αντιστοιχηθούν σε αυτήν την κλάση. Συνεπώς, όπως και στον έλεγχο υπερκλάσεων, ανακτώνται όλες οι άμεσες υποκλάσεις της υπό έλεγχο κλάσης, δηλαδή όλες οι υποκλάσεις που βρίσκονται ιεραρχικά στο αμέσως κατώτερο επίπεδο προκειμένου να συγκριθούν τα δεδομένα που έχουν αντιστοιχηθεί σε αυτές, με τα δεδομένα που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση. Τα δεδομένα που έχουν αντιστοιχηθεί σε καθεμία από τις άμεσες υποκλάσεις της υπό έλεγχο κλάσης συγκρίνονται ένα προς ένα με τα δεδομένα που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση, προκειμένου να εξασφαλιστεί ότι τα πρώτα αποτελούν υποσύνολο των δεύτερων. Αν βρεθεί έστω και ένα δεδομένο που έχει ήδη απεικονιστεί σε κάποια υποκλάση, αλλά όχι και στην υπό έλεγχο κλάση, τότε η εν λόγω απεικόνιση είναι σημασιολογικά απορριπτέα.

Δύο δεδομένα, όπως και στους δύο παραπάνω ελέγχους, θεωρούνται ταυτόσημα, όταν έχουν την ίδια τιμή και ανήκουν στην ίδια στήλη και στον ίδιο πίνακα ή όταν έχουν την ίδια τιμή και ανήκουν σε στήλες, η μία εκ των οποίων είναι ξένο κλειδί στην άλλη.

Στην περίπτωση που μια άμεση υποκλάση της υπό έλεγχο κλάσης δεν έχει απεικονιστεί σε κάποια δεδομένα, τότε ο έλεγχος μεταφέρεται ένα επίπεδο χαμηλότερα, εξετάζοντας τις υποκλάσεις της μη αντιστοιχησθείσας κλάσης και ούτω καθεξής.

Συνοπτικά, για να γίνει αποδεκτή μια αντιστοίχιση, θα πρέπει τα δεδομένα που πρόκειται να αντιστοιχηθούν στην υπό έλεγχο κλάση να περιέχουν όλα τα δεδομένα που έχουν ήδη αντιστοιχηθεί σε όλες τις υποκλάσεις της.

5.5 ΑΠΟΘΗΚΕΥΣΗ ΑΝΤΙΣΤΟΙΧΙΣΕΩΝ

Όπως αναφέρθηκε παραπάνω, οι αντιστοιχίσεις μεταξύ δεδομένων από μια βάση και κλάσεων μιας OWL οντολογίας γίνονται μέσω SQL ερωτημάτων, καθώς μέσω των SQL ερωτημάτων μπορεί να γίνουν όχι μόνο απλές επιλογές δεδομένων (ολόκληρος πίνακας, συγκεκριμένες στήλες ενός πίνακα), αλλά και συνθετότερες επιλογές που εμπλέκουν περιορισμούς επί των γραμμών ενός πίνακα. Ουσιαστικά δηλαδή, η κλάση μιας οντολογίας αντιστοιχείται σε ένα SQL ερώτημα και μέσω αυτού, με τα δεδομένα που προκύπτουν από την εκτέλεσή του. Συνεπώς, πρόκειται για μια μάλλον έμμεση αντιστοίχιση οντολογικών κλάσεων και δεδομένων που προέρχονται από βάσεις δεδομένων.

Ο τρόπος με τον οποίο επιτυγχάνεται η αντιστοίχιση είναι η **προσθήκη μιας** αυθαίρετης **ιδιότητας** στην υπό αντιστοίχιση κλάση. Η ιδιότητα αυτή ονομάζεται queryString και παίρνει ως τιμή της το SQL ερώτημα, το οποίο θα επιστρέψει τα επιθυμητά προς αντιστοίχιση δεδομένα. Η γλώσσα OWL προσφέρει και ένα σύνολο στοιχείων και μηχανισμών, όπως τα owl:versionInfo, rfds:comment και rdfs:label, τα οποία έχουν χρησιμοποιηθεί για να αποθηκευθούν η στιγμή της τελευταίας αντιστοίχισης, το όνομα του OWL αρχείου στο οποίο ανήκουν όσες κλάσεις έχουν αντιστοιχηθεί και το όνομα της βάσης δεδομένων από την οποία προέρχονται τα δεδομένα αντίστοιχα. Βέβαια, υπάρχει και το στοιχείο rdfs:domain, το οποίο δηλώνει,

ως γνωστόν, το πεδίο ορισμού μιας ιδιότητας, οπότε στην περίπτωση της ιδιότητας queryString, αναμένεται ότι ως πεδίο ορισμού έχουν τεθεί οι κλάσεις της οντολογίας που έχουν αντιστοιχηθεί σε κάποια δεδομένα.

Συνεχίζοντας το παράδειγμα χρήσης της εφαρμογής, παρατίθεται ένα τμήμα του αρχικού OWL αρχείου που περιέχει την οντολογία που χρησιμοποιήθηκε για τη διαδικασία αντιστοίχισης:

```
<owl:Ontology rdf:about="" />
  <owl:Class rdf:ID="JOG" />
  <owl:Class rdf:ID="LAT" />
- <owl:Class rdf:ID="LAT_North">
        <rdfs:subClassOf rdf:resource="#LAT" />
-       <owl:disjointWith>
              <owl:Class rdf:ID="LAT_South" />
        </owl:disjointWith>
  </owl:Class>
- <owl:Class rdf:about="#LAT_South">
        <owl:disjointWith rdf:resource="#LAT_North" />
        <rdfs:subClassOf rdf:resource="#LAT" />
  </owl:Class>
```

Πλαίσιο 5.1 Ορισμοί κλάσεων (Αρχικό αρχείο οντολογίας)

Αν πραγματοποιηθούν οι αντιστοιχίσεις που περιγράφηκαν στο παράδειγμα χρήσης και αποθηκευθούν σε ξεχωριστό OWL αρχείο, τότε το αντίστοιχο με το παραπάνω τμήμα αυτού του αρχείου θα έχει ως εξής:

```
<owl:Ontology rdf:about="" />
- <owl:Class rdf:ID="JOG">
        <queryString>SELECT * FROM spatial_ref_sys WHERE
(spatial_ref_sys.srid>2972)</queryString>
  </owl:Class>
  <owl:Class rdf:ID="LAT" />
- <owl:Class rdf:ID="LAT_North">
        <rdfs:subClassOf rdf:resource="#LAT" />
        <queryString>SELECT rstrct.gid , rstrct.the_geom FROM
rstrct</queryString>
-       <owl:disjointWith>
              <owl:Class rdf:ID="LAT_South" />
        </owl:disjointWith>
  </owl:Class>
- <owl:Class rdf:about="#LAT_South">
        <rdfs:subClassOf rdf:resource="#LAT" />
        <owl:disjointWith rdf:resource="#LAT_North" />
        <queryString>SELECT archsites.cat_desc FROM archsites WHERE
(archsites.cat_desc='No Name')</queryString>
  </owl:Class>
```

Πλαίσιο 5.2 Ορισμοί κλάσεων (αρχείο αντιστοιχίσεων)

Επιπλέον, θα έχουν προστεθεί οι παρακάτω γραμμές στο OWL αρχείο που περιέχουν τον ορισμό της νέας ιδιότητας queryString:

```
- <owl:DatatypeProperty rdf:ID="queryString">
        <rdfs:domain rdf:resource="#JOG" />
        <rdfs:comment>CountriesTerritoriesDescriptor-1.owl</rdfs:comment>
        <owl:versionInfo>Thu Sep 01 12:04:27 EEST 2005</owl:versionInfo>
        <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty" />
        <rdfs:domain rdf:resource="#LAT_South" />
        <rdfs:domain rdf:resource="#LAT_North" />
        <rdfs:label>PostgreSQL-spearfish</rdfs:label>
  </owl:DatatypeProperty>
```

Πλαίσιο 5.3 Ορισμός ιδιότητας queryString (αρχείο αντιστοιχίσεων)

5.6 ΠΕΡΙΟΡΙΣΜΟΙ – ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

Προς το παρόν, η εφαρμογή υποστηρίζει μόνο δύο συστήματα διαχείρισης βάσεων δεδομένων, τη **MySQL** και την **PostgreSQL**, οι οποίες υποστηρίζουν την αποθήκευση χωρικών και γεωγραφικών δεδομένων και ταυτόχρονα αποτελούν και οι δύο μη εμπορικά προϊόντα. Στο μέλλον, η εφαρμογή θα μπορούσε να υποστηρίξει και άλλα συστήματα διαχείρισης, όπως Oracle, DB2, SQLServer κ.ά.

Ακόμα, εισάγονται διάφοροι περιορισμοί στη **λειτουργικότητα** της εφαρμογής που οφείλονται στο γεγονός ότι ορισμένες μέθοδοι της Java δε λειτουργούν με τον επιθυμητό τρόπο είτε στη MySQL είτε στην PostgreSQL. Ακόμα, το γεγονός ότι, μέχρι την παρούσα στιγμή, δεν έχει γενικευθεί η χρήση ξένων κλειδιών σε όλους τους τύπους πινάκων της MySQL δεν επιτρέπει την πληρότητα του ελέγχου σημασιολογικής ορθότητας των αντιστοιχίσεων οντολογικών κλάσεων με δεδομένα που είναι αποθηκευμένα σε μια MySQL βάση δεδομένων. Τα εν λόγω προβλήματα, τα οποία ίσως ξεπεραστούν στο μέλλον με την κυκλοφορία αναβαθμισμένων εκδόσεων των συγκεκριμένων συστημάτων διαχείρισης βάσεων δεδομένων, αναφέρονται λεπτομερώς στο επόμενο κεφάλαιο.

Επίσης, υπάρχουν διάφοροι περιορισμοί που αναφέρονται στην **απεικόνιση** της οντολογίας. Συγκεκριμένα, η εφαρμογή μπορεί να απεικονίσει μόνο OWL οντολογίες που είναι αποθηκευμένες σε ένα αρχείο κατάληξης .owl, ενώ παράλληλα δεν μπορούν να απεικονιστούν σωστά οντολογίες που περιέχουν ορισμούς κλάσεων αυξημένης πολυπλοκότητας (με πολλαπλές εμφωλεύσεις δομικών στοιχείων της OWL, όπως restriction, union, intersection, subclass κ.ά.).

Όσον αφορά τις αντιστοιχίσεις, θεωρήθηκε επαρκής η χρήση ενός **SQL ερωτήματος** για την επιλογή των δεδομένων, εφόσον αυτό μπορεί να καλύψει τις συνηθέστερες περιπτώσεις επιλογής δεδομένων, ενώ ταυτόχρονα η χρήση πιο πολύπλοκων ερωτημάτων θα δυσκόλευε και τη φιλικότητα της εφαρμογής προς το χρήστη. Βέβαια, στο μέλλον, θα μπορούσε η διαδικασία της αντιστοίχισης να βασιστεί σε

περισσότερο πολύπλοκα SQL ερωτήματα, που θα δημιουργούνται από τη σύνθεση απλών ερωτημάτων και τη χρήση όλων των δυνατοτήτων της γλώσσας SQL.

Ένας περιορισμός της εφαρμογής που αξίζει αναφοράς είναι η συχνή εκτέλεση SQL ερωτημάτων κατά τη διάρκεια των σημασιολογικών ελέγχων. Το παραπάνω κρίνεται ως μειονέκτημα, ιδιαίτερα στις περιπτώσεις των χωρικών βάσεων δεδομένων, στις οποίες ο όγκος των δεδομένων είναι τεράστιος, αλλά και σε κάθε άλλη περίπτωση ανάγκης αντιστοίχισης μεγάλου όγκου δεδομένων. Σε μια τέτοια περίπτωση, ο χρόνος απόκρισης της εφαρμογής αυξάνεται, ενώ υπάρχει πιθανότητα να μην ολοκληρωθεί ο σημασιολογικός έλεγχος.

Επίσης, στην παρούσα εφαρμογή, επιτρέπεται μόνο η αντιστοίχιση δεδομένων με κλάσεις της οντολογίας. Μια μελλοντική επέκταση της εφαρμογής θα μπορούσε να περιλαμβάνει την **αντιστοίχιση ιδιοτήτων της οντολογίας** με κάποια στοιχεία και συστατικά μιας βάσης δεδομένων. Με αυτόν τον τρόπο, θα μπορούσαν να επεκταθούν και οι έλεγχοι σημασιολογικής ορθότητας μιας αντιστοίχισης που, προς το παρόν, περιορίζονται στους ελέγχους ξένων μεταξύ τους κλάσεων και κλάσεων που συνδέονται με τη σχέση υπερκλάσης / υποκλάσης.

Επιπλέον, όπως αναφέρθηκε, προσφέρεται η δυνατότητα **αποθήκευσης** των αντιστοιχίσεων που πραγματοποιούνται μέσω της εφαρμογής. Όμως, η υλοποίηση στην παρούσα φάση δεν επιτρέπει την αποθήκευση στο ίδιο αρχείο αντιστοιχίσεων που αναφέρονται σε δεδομένα που ανήκουν σε διαφορετικές βάσεις δεδομένων. Μελλοντικά, εφόσον αυτό κριθεί σκόπιμο, θα μπορούσαν να γίνουν προσπάθειες προς την άρση αυτού του περιορισμού, όπως βέβαια θα μπορούσαν να γίνουν προσπάθειες ώστε η εφαρμογή να γίνει περισσότερο φιλική προς το χρήστη.

6 ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ

Στο κεφάλαιο αυτό, εξηγείται λεπτομερώς ο κώδικας της εφαρμογής με παράθεση αρκετών τμημάτων του.

6.1 ΦΟΡΤΩΣΗ ΚΑΙ ΑΠΕΙΚΟΝΙΣΗ ΟΝΤΟΛΟΓΙΑΣ

Η φόρτωση μιας οντολογίας επιτυγχάνεται με το πάτημα του κουμπιού ανοίγματος, οπότε ανοίγει ένα παράθυρο προς επιλογή του κατάλληλου αρχείου με κατάληξη .owl. Αν ο χρήστης επιλέξει είτε ένα αρχείο με κατάληξη .owl που δε συμμορφώνεται με τους κανόνες της γλώσσας OWL είτε ένα αρχείο που έχει διαφορετική κατάληξη, τότε εμφανίζεται ένα μήνυμα λάθους σε παράθυρο που ενημερώνει το χρήστη.

```
if (source==openButton){
        ...
        chooser.setDialogTitle("Open the OWL file with the basic ontology");

        …

           int returnVal = chooser.showOpenDialog(this);
           if (returnVal == JFileChooser.APPROVE_OPTION) {
              …
              newFilePath="file:/";
              newFilePath=newFilePath.concat(chooser.getSelectedFile().getPath());
              String suffix = new String();
              if (newFilePath.contains(".")) suffix = newFilePath.substring(newFilePath.lastIndexOf('.') + 1);
              if (!suffix.equalsIgnoreCase("owl")){
                 JOptionPane.showMessageDialog(null,"Make sure the file name ends in .owl",
                       "File Format Not Supported",
                       JOptionPane.ERROR_MESSAGE);
              } else{
                 //if a different OWL file is opened, initialize the GUI components
                 if (!oldFilePath.equals(newFilePath)){

                    …

                    ontModel= ModelFactory.createOntologyModel( OntModelSpec.OWL_MEM, null );
                    try{
                       ontModel.read( newFilePath,"",null );
                    }catch(Exception ex){
                       JOptionPane.showMessageDialog(null,"This file cannot be properly opened due to connection or wrong file format reasons.\nMake sure that you are online and that you have opened a legal OWL file",
                             "Error in opening file",
                             JOptionPane.ERROR_MESSAGE);
                    }
                 }
              }
           }
        }
```

Πλαίσιο 6.1 Άνοιγμα οντολογίας (OntologyPanel.java)

Όταν επιλεγεί ένα αποδεκτό αρχείο οντολογίας, διαβάζεται το αρχείο αυτό και αποθηκεύεται στη μεταβλητή κλάσης ontModel της κλάσης OntModel. Προκειμένου να απεικονιστεί η ιεραρχία των κλάσεων και των ιδιοτήτων της οντολογίας, καλούνται οι μέθοδοι showHierarchy() της κλάσης ClassHierarchy και showHierarchy() της κλάσης PropertyHierarchy. Οι μέθοδοι αυτές δημιουργούν αντικείμενα τύπου DefaultTreeModel τα οποία τίθενται στα δύο δένδρα, έτσι ώστε να απεικονιστούν στις δύο καρτέλες (tabs) κλάσεων και ιδιοτήτων.

```
                    //show the class hierarchy
                    ch.showHierarchy( ontModel );
                    treeModel= ch.treeModel;
                    ontTree.setModel(treeModel);
                    //show the property hierarchy
                    ph.showHierarchy( ontModel );
                    propModel= ph.treeModel;
```

```
                propTree.setModel(propModel);
```

Πλαίσιο 6.2 Απεικόνιση ιεραρχίας κλάσεων και ιδιοτήτων (OntologyPanel.java)

Η μέθοδος showHierarchy() της κλάσης ClassHierarchy καλεί τη μέθοδο showClass() για κάθε κλάση της οντολογίας που είναι άμεση υποκλάση (direct subclass) της κλάσης owl:Thing. Αυτές οι κλάσεις επιστρέφονται από τη μέθοδο rootClasses().

```
        //recurse down to the sub-classes,calling method showClass for the classes that are direct
sub-classes of class Thing
            for (i = rootClasses( m );  i.hasNext();  ) {
               showClass((OntClass) i.next(),rootNode, new ArrayList(),propOnTop );
            }
```

Πλαίσιο 6.3 Επαναληπτική κλήση μεθόδου showClass() (ClassHierarchy.java)

```
protected Iterator rootClasses( OntModel m ) {
    List roots = new ArrayList();

    for (Iterator i = m.listClasses();  i.hasNext(); ) {
       OntClass c = (OntClass) i.next();

       if (c.isAnon()) {
          continue;
       }

       Iterator isuper=findSuperclasses(c);
       if (!isuper.hasNext()||(c.hasSuperClass( m.getProfile().THING(), true ) )) roots.add(c);

    }

    return roots.iterator();
  }
```

Πλαίσιο 6.4 Μέθοδος rootClasses() (ClassHierarchy.java)

Η μέθοδος findSuperclasses() που καλείται από τη rootClasses() δημιουργήθηκε με σκοπό να υπερκαλύψει τη μέθοδο listSuperClasses() της Jena, αφού η τελευταία αναζητά υπερκλάσεις που έχουν δηλωθεί ρητά στην οντολογία με χρήση του rdfs:subClassOf μηχανισμού. Αντίθετα, η findSuperclasses() επεκτείνει την παραπάνω καλύπτοντας και την περίπτωση **έμμεσης δήλωσης υπερκλάσεων** μέσα σε owl:equivalentClass tag.

```
public Iterator findSuperclasses(OntClass cls){

    /*find the super-classes that are defined inside an <equivalentClass> tag,which are not
returned by
     * the listSuperclasses() method
     *They usually represent necessary and sufficient conditions for an individual to belong to a
certain class
     */
    Vector v=new Vector();
    Iterator i=cls.listEquivalentClasses();
    while (i.hasNext()){
       OntClass current= (OntClass)i.next();
       if (current.isIntersectionClass()){
          IntersectionClass inter=current.asIntersectionClass();
          Iterator i2=inter.listOperands();
          while (i2.hasNext()){
```

```
                OntClass oper=(OntClass)i2.next();
                if (!oper.isAnon()){
                   v.addElement(oper);
                }
             }
          }

      }

      //find only the super-classes which are declared through a <subclassOf> tag
      Iterator i3= cls.listSuperClasses();
      while (i3.hasNext()){
         OntClass current=(OntClass)i3.next();
         if (!current.isAnon()){
            v.addElement(current);
         }

      }
      return v.iterator();
   }
```

Πλαίσιο 6.5 Μέθοδος findSuperclasses() (ClassHierarchy.java)

Η μέθοδος showClass() είναι **αναδρομική**, παίρνει ως όρισμα μια κλάση της οντολογίας, συλλέγει διάφορες πληροφορίες για αυτήν την κλάση και δημιουργεί για αυτήν έναν κόμβο του δέντρου ο οποίος ενσωματώνει αυτές τις πληροφορίες. Προκειμένου να κληθεί για κάθε κλάση της οντολογίας, η showClass() καλείται αρχικά για κάθε κλάση της οντολογίας που είναι άμεση υποκλάση της owl:Thing και στη συνέχεια, υπάρχει αναδρομική κλήση για όλες τις άμεσες υποκλάσεις κάθε μίας από αυτές. Για κάθε κλάση, η showClass() καλεί τη findSubclasses(), μια μέθοδο αντίστοιχη της findSuperclasses, η οποία βρίσκει τις άμεσες υποκλάσεις μιας κλάσης, υπερκαλύπτοντας έτσι τη μέθοδο listSubClasses() της Jena. Ο λόγος που δημιουργήθηκε αυτή η μέθοδος είναι η κάλυψη της **έμμεσης περίπτωσης δήλωσης υποκλάσεων** μέσα σε ένα owl:equivalentClass tag, όπως έγινε και στην περίπτωση των υπερκλάσεων. Ακολούθως, καλείται η μέθοδος findSuperclasses() για να βρεθούν οι άμεσες υπερκλάσεις της τρέχουσας κλάσης, η listDisjointWith() της Jena για να βρεθούν οι κλάσεις που είναι ξένες με αυτήν την κλάση, η listDeclaredProperties() της Jena για να ανακτηθούν οι ιδιότητες που έχουν δηλωθεί για την κλάση αυτή και τέλος, η μέθοδος findRestrictions(), σκοπός της οποίας είναι να βρει τους περιορισμούς που σχετίζονται με την τρέχουσα κλάση. Σημειώνεται ότι οι περιορισμοί μπορούν να τεθούν σε μία κλάση είτε μέσα σε ένα owl:equivalentClass tag είτε μέσα σε ένα rdfs:subClassOf tag και σε αυτό το γεγονός βασίζεται η μέθοδος findRestrictions(). Αφού έχουν συλλεχθεί λοιπόν, οι παραπάνω πληροφορίες (υποκλάσεις, υπερκλάσεις, ξένες κλάσεις, δηλωθείσες ιδιότητες, περιορισμοί), κατασκευάζεται με βάση αυτές ένα αντικείμενο ClassInfo, που συσχετίζεται με έναν **καινούριο κόμβο δέντρου** ο οποίος τοποθετείται στην κατάλληλη θέση.

```
protected void showClass (OntClass cls, DefaultMutableTreeNode parentNode, List occurs,
Vector propOnTop ) {


      /** find the direct sub-classes,super-classes,disjoint classes,declared properties
       * and restrictions related to this class,in order to create and fill a new structure ClassInfo
       * which will be added to the tree
```

```
      */
      Iterator isub=findSubclasses(cls);
      … //fill a vector with the subclasses


      Vector superClasses=new Vector();
      …. //fill a vector with the superclasses

      //add owl:Thing as superclass to the classes returned by rootClasses() method
      ...

      Iterator idis=cls.listDisjointWith();
      …//fill a vector with the disjoints

      Iterator iprop=cls.listDeclaredProperties(true);
      … //fill a vector with the declared properties

      Iterator ires = findRestrictions(cls);
      … //fill a vector with the restrictions



      ClassInfo nodeInfo = new ClassInfo (cls.getURI(), ClassInfo.CLASS, subClasses,
superClasses, disjoint, properties, restrictions);
      NodeInfo myInfo=new NodeInfo(NodeInfo.CLASS,cls.getURI(),nodeInfo,null);
      DefaultMutableTreeNode childNode = new DefaultMutableTreeNode(myInfo);
      treeModel.insertNodeInto(childNode, parentNode, parentNode.getChildCount());


      // recurse to the next level down
      if (cls.canAs( OntClass.class )  &&  !occurs.contains( cls )) {
         for (Iterator i = findSubclasses( cls );  i.hasNext(); ) {
            OntClass sub = (OntClass) i.next();

            // we push this expression on the occurs list before we recurse
            occurs.add( cls );
            showClass(sub,childNode, occurs,propOnTop );
            occurs.remove( cls );
         }
      }
   }
```

Πλαίσιο 6.6 Μέθοδος showClass() (ClassHierarchy.java)

```
public Iterator findRestrictions(OntClass cls){

      //find restrictions in <equivalentClass> tag
      Vector v=new Vector();
      Iterator i=cls.listEquivalentClasses();
      while (i.hasNext()){
         OntClass current= (OntClass)i.next();
         if (current.isIntersectionClass()){
            IntersectionClass inter=current.asIntersectionClass();
            Iterator i2=inter.listOperands();
            while (i2.hasNext()){
               OntClass oper=(OntClass)i2.next();
               if (oper.isRestriction())  v.addElement(oper.asRestriction());
            }
         }
```

```
    }

    //find restrictions in <subclassOf> tags
    i=cls.listSuperClasses(true);
    while (i.hasNext()){
       OntClass current = (OntClass)i.next();
       if (current.isRestriction()) v.addElement(current.asRestriction());
    }

    return v.iterator();
  }
```

Πλαίσιο 6.7 Μέθοδος findRestrictions() (ClassHierarchy.java)

```
/*A class that contains all the information for a certain class
 **/
class ClassInfo extends Object {

  public ClassInfo(String text, int type, Vector vsub,Vector vsup,Vector vdis,Vector vprop,Vector
vres) {
     … //standard constructor giving values to the class variables
  }


  public String toString() {
     return name;
  }

  ... //variable declarations

  public static final int CLASS = 1;
  public static final int THING = 2;

  public static final int OTHER = -1;
}
```

Πλαίσιο 6.8 Κλάση ClassInfo (ClassHierarchy.java)

Ανάλογες ενέργειες πραγματοποιεί και η μέθοδος showHierarchy() της κλάσης PropertyHierarchy, η οποία έχει την ίδια δομή με τη showHierarchy() της ClassHierarchy. Η μόνη διαφορά έγκειται στις πληροφορίες που συσχετίζονται με κάθε κόμβο του δέντρου. Συγκεκριμένα, για κάθε ιδιότητα αποθηκεύονται σε ένα αντικείμενο τύπου PropertyInfo ο τύπος της, οι υπο-ιδιότητές της, οι υπερ-ιδιότητές της, το πεδίο τιμών της (range), το πεδίο ορισμού της (domain) και οι αντίστροφές της ιδιότητες. Η κλάση OntTreeCellRenderer αναλαμβάνει την απεικόνιση και των δύο δένδρων, θέτοντας ξεχωριστό εικονίδιο σε κάθε κόμβο του δένδρου ανάλογα με το αν είναι κλάση ή ιδιότητα.

Από τη στιγμή που έχει απεικονιστεί η οντολογία με τη μορφή δύο δέντρων στις δύο καρτέλες (tabs) που αναφέρθηκαν παραπάνω, ο χρήστης έχει τη δυνατότητα επιλέγοντας κάποιον κόμβο οποιουδήποτε από τα δύο δέντρα να δει **πληροφορίες** σχετικά με την επιλεγμένη κλάση ή ιδιότητα σε μια περιοχή κειμένου. Αυτό επιτυγχάνεται με την κλήση της μεθόδου fillInfoArea(), η οποία δέχεται ως είσοδο ένα αντικείμενο τύπου NodeInfo που περιέχει όλες τις πληροφορίες που σχετίζονται με τον τρέχοντα επιλεγμένο κόμβο του δέντρου. Ανάλογα με το αν το αντικείμενο NodeInfo περιέχει ένα αντικείμενο ClassInfo ή PropertyInfo, δηλαδή ανάλογα με το

αν ο επιλεγμένος κόμβος αναφέρεται σε κλάση ή ιδιότητα, παρουσιάζονται διαφορετικές πληροφορίες. Αν έχει επιλεχθεί μια **κλάση**, παρουσιάζονται οι άμεσες υπερκλάσεις της, οι άμεσες υποκλάσεις της, οι κλάσεις που είναι ξένες προς αυτήν, οι ιδιότητες που έχουν δηλωθεί για αυτήν την κλάση, καθώς και οι περιορισμοί που έχουν τεθεί σε αυτήν, όπως περιέχονται στο αντικείμενο ClassInfo που δημιουργήθηκε κατά το διάβασμα της οντολογίας. Επίσης, αναφέρονται και τυχόν αντιστοιχίσεις που έχουν γίνει για τη συγκεκριμένη κλάση. Αντίστοιχα, αν ο επιλεγμένος κόμβος αναφέρεται σε **ιδιότητα**, ο τύπος της ιδιότητας, οι υπερ-ιδιότητές της, οι υπο-ιδιότητές της, το πεδίο τιμών της, το πεδίο ορισμού της, καθώς και οι αντίστροφές της, εξάγονται από το αντικείμενο PropertyInfo που έχει συσχετιστεί με τον κόμβο.

```
        //whenever a node of the JTree is selected,fill the info area with information about the
selected node
            public void fillInfoArea(NodeInfo nodeInfo){
               infoArea.setText(null);
               Font font = getFont();
               //if the selected node is a class
               if (nodeInfo.type==NodeInfo.CLASS){
                  ClassInfo classInfo = nodeInfo.classInfo;
                  infoArea.setFont(font.deriveFont(Font.PLAIN));
                  String localName=classInfo.name.substring(classInfo.name.indexOf('#')+1);
                  String totalInfo = "\t\tInfo On Class " + localName +"\n\n";
                  totalInfo = totalInfo.concat("Full URI: "+classInfo.name+"\n");

                  totalInfo = totalInfo.concat("Direct Sublasses: ");
                  … //write down the subclasses

                  if (classInfo.type==ClassInfo.CLASS){
                     totalInfo= totalInfo.concat("\nDirect Superclasses: ");
                  … //write down the superclasses


                     totalInfo= totalInfo.concat("\nDisjoint With: ");
                  … //write down the disjoints


                     … and so on for the rest information

                  infoArea.setText(totalInfo);

                  //if the selected node is a property
               } else if (nodeInfo.type==NodeInfo.PROPERTY){
                  … likewise gather and write down all the information about this property

                  infoArea.setText(totalInfo);
               }
            }
```

Πλαίσιο 6.9 Μέθοδος fillInfoArea() (OntologyPanel.java)

6.2 ΦΟΡΤΩΣΗ ΚΑΙ ΑΠΕΙΚΟΝΙΣΗ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ

Η τρέχουσα εφαρμογή υποστηρίζει δύο είδη συστημάτων διαχείρισης βάσεων δεδομένων: MySQL και PostgreSQL. Ο χρήστης μπορεί να επιλέξει από μια

αναπτυσσόμενη λίστα το σύστημα με το οποίο επιθυμεί να εργαστεί και να πατήσει το κουμπί επιλογής. Ανάλογα με το σύστημα που επιλέγεται, φορτώνεται και ένα διαφορετικό πλαίσιο με τα αντίστοιχα στοιχεία διεπαφής. Τα πλαίσια που αντιστοιχούν στα δύο συστήματα μοιάζουν αρκετά μεταξύ τους και διαφέρουν κυρίως στη διάταξη των στοιχείων εκείνων που επιτρέπουν τη σύνδεση του χρήστη με τη βάση δεδομένων.

Στην περίπτωση επιλογής της **MySQL** φορτώνεται το πλαίσιο MySQLPanel. Αυτό περιέχει τέσσερα πεδία κειμένου για τη διεύθυνση IP του server στον οποίο βρίσκεται η βάση, τη θύρα στην οποία θα επικοινωνήσει ο JDBC (Java Database Connectivity) Driver, το όνομα (login) και το συνθηματικό (password) του χρήστη. Όταν ο χρήστης πατήσει το κουμπί σύνδεσης και όλα τα στοιχεία είναι σωστά, τότε θα επιτευχθεί η σύνδεση και μέσω της μεθόδου getMySQLDB(), η εφαρμογή θα αναγνωρίσει ποιες βάσεις δεδομένων είναι διαθέσιμες και θα τις τοποθετήσει σε μια αναπτυσσόμενη λίστα, από όπου ο χρήστης καλείται να επιλέξει μία και να πατήσει ένα δεύτερο κουμπί σύνδεσης (το κουμπί «Go»). Η getMySQLDB() δέχεται ως είσοδο ένα αντικείμενο της κλάσης Connection που αναπαριστά την τρέχουσα σύνδεση και επιστρέφει ένα ResultSet, στο οποίο περιέχονται τα ονόματα των διαθέσιμων βάσεων δεδομένων. Επίσης, σημειώνεται ότι με την επίτευξη της σύνδεσης, αποθηκεύονται τα στοιχεία της (διεύθυνση IP, θύρα, όνομα, συνθηματικό χρήστη) σε ένα αντικείμενο τύπου ConnectInfo, έτσι ώστε να είναι διαθέσιμα ανά πάσα στιγμή.

```
//setup the connection
        String strURL="";
           strDriver="org.gjt.mm.mysql.Driver";
           strURL = "jdbc:mysql://"+txtHostIP.getText()+":"+txtPort.getText()+"/";

           // load the JDBC driver
           Class.forName(strDriver);

           con = DriverManager.getConnection(strURL,
                 txtUserID.getText(), new String(txtPW.getPassword()));

           … //construct a new ConnectInfo instance to store all the information about the
connection

           //fill the JComboBox containing the MySQL databases with the ones referring to the
new connection
           ResultSet resDB = getMySQLDB(con);
           while (resDB.next()){
              cmbDB.addItem(resDB.getString(1));
           }
```

Πλαίσιο 6.10 Σύνδεση με MySQL βάση δεδομένων (MySQLPanel.java)

```
//Find the list of databases that are written in MySQL

   public ResultSet getMySQLDB(Connection con) throws SQLException{

      String strQuery="SHOW DATABASES";
      PreparedStatement myStmt;
      ResultSet ret = null;

      try {
         myStmt = con.prepareStatement(strQuery);
         ret = myStmt.executeQuery(strQuery);
```

```
    } catch (SQLException e) {
       System.out.println(e.getMessage());
    }
    return ret;
  }
```

Πλαίσιο 6.11 Μέθοδος getMySQLDB() (My SQLPanel.java)

Στην περίπτωση επιλογής της **PostgreSQL** φορτώνεται το πλαίσιο PostgresPanel, το οποίο σε γενικές γραμμές είναι ίδιο με το MySQLPanel. Μια διαφορά εντοπίζεται στα στοιχεία που απαιτούνται για να γίνει σύνδεση σε μια βάση δεδομένων, καθώς ο χρήστης εκτός των άλλων χρειάζεται να γράψει σε ένα πεδίο κειμένου το όνομα της βάσης με την οποία επιθυμεί να συνδεθεί. Όταν ο χρήστης πατήσει το κουμπί σύνδεσης και τα στοιχεία είναι σωστά, τότε συνδέεται κατευθείαν στη συγκεκριμένη βάση που αυτός έχει υποδείξει. Με άλλα λόγια, ο χρήστης πρέπει να γνωρίζει εκ των προτέρων το όνομα της βάσης με την οποία επιθυμεί να συνδεθεί, καθώς ο JDBC Driver της PostgreSQL το απαιτεί για να επιτευχθεί η σύνδεση.

Μετά τη σύνδεση με τη βάση, επόμενο βήμα είναι η **απεικόνιση σε δενδρική μορφή του σχήματός της**. Χρησιμοποιώντας τη μέθοδο getMetadata() της κλάσης Connection, η εφαρμογή ανακτά τις απαραίτητες πληροφορίες μέσω ενός αντικειμένου DatabaseMetaData. Μέσα σε έναν επαναληπτικό βρόχο, αποθηκεύονται το όνομα, ο τύπος, τα πρωταρχικά και τα ξένα κλειδιά κάθε πίνακα, ενώ σε έναν εσωτερικό επαναληπτικό βρόχο καθορίζονται το όνομα και ο τύπος δεδομένων κάθε στήλης του πίνακα όπως και το αν η στήλη είναι ξένο ή πρωταρχικό κλειδί. Ενώ γίνεται η προσπέλαση πινάκων και στηλών, δημιουργείται κάθε φορά ένας κόμβος του δένδρου, ο οποίος συσχετίζεται με ένα αντικείμενο τύπου TreeNodeInfo, το οποίο περιέχει το όνομα και τον τύπο του πίνακα ή της στήλης.

```
con = DriverManager.getConnection (connectInfo.m_DSN, connectInfo.m_UID, connectInfo.m_PWD);
            DatabaseMetaData dbMetaData = con.getMetaData();


            // Obtain all the available table types from the data source
            ...

            // Set up the data tables tree


            ResultSet rsTables = dbMetaData.getTables(databaseSelected,null,null,tableTypes);
            boolean bMore = rsTables.next();
            while (bMore) {

               ... //determine each table type

               // Get primary keys
               if (dsInfo.m_supportsCoreSQLGrammar) {
                  rs = dbMetaData.getPrimaryKeys(databaseSelected,null,strTableName);
                  more = rs.next();
                  while (more) {
                     FieldInfo primary= new FieldInfo();
                     primary.fieldName=rs.getString("COLUMN_NAME");
                     primary.fieldParent=strTableName;
                     primaryKeys.add(primary);
                     more = rs.next();
```

```
                    }
                    rs.close();
                }

                //add the table to the tree
                TreeNodeInfo nodeInfo = new TreeNodeInfo(strTableName,tableType);
                DefaultMutableTreeNode childNode = new DefaultMutableTreeNode(nodeInfo);
                treeModel.insertNodeInto(childNode, rootNode, rootNode.getChildCount());
                treeTables.scrollPathToVisible(new TreePath(childNode.getPath()));
                ResultSet rsColumns = dbMetaData.getColumns (databaseSelected, null,
strTableName, null);


                boolean bMore2 = rsColumns.next();
                while (bMore2) {
                   … //determine the data types of the columns of each table

                   // Determine whether this column is a primary key
                   int colType = TreeNodeInfo.COLUMN;
                   for (int i = 0; i < primaryKeys.size(); i++) {

                      String key = ((FieldInfo)primaryKeys.elementAt(i)).fieldName;
                      String keyTable = ((FieldInfo)primaryKeys.elementAt(i)).fieldParent;
                      if ((key.compareToIgnoreCase(strColumnName) == 0) &&
(keyTable.compareToIgnoreCase(strTableName) == 0)) {
                         colType = TreeNodeInfo.COLUMN_PK;
                         break;
                      }

                   }

                   //add each column to the tree under the table it belongs
                   TreeNodeInfo nodeInfo2 = new TreeNodeInfo(nodeText,colType);
                   DefaultMutableTreeNode childColumnNode = new DefaultMutableTreeNode
(nodeInfo2);
                   treeModel.insertNodeInto (childColumnNode, childNode,
childNode.getChildCount() );

                   bMore2 = rsColumns.next();
                }
                rsColumns.close();

                tables.add(nodeInfo);

                bMore = rsTables.next();
             }

            rsTables.close();
```

Πλαίσιο 6.12 Απεικόνιση μιας MySQL βάσης δεδομένων με λήψη πληροφοριών για πίνακες και στήλες (MySQLPanel.java)

Αυτή η πληροφορία χρησιμοποιείται από την κλάση DBTreeCellRenderer, η οποία καθορίζει τις λεπτομέρειες της απεικόνισης του δέντρου (εικονίδια, γραμματοσειρά) ανάλογα με τον τύπο της στήλης ή του πίνακα. Σε αυτό το σημείο, πρέπει να σημειωθεί ότι η MySQL δεν υποστηρίζει ξένα κλειδιά και συνεπώς, η μέθοδος getImportedKeys() της κλάσης DatabaseMetaData δε λειτουργεί και δεν επιστρέφει ένα ResultSet που υπό κανονικές συνθήκες θα περιείχε τα ξένα κλειδιά. Ακόμα, ο χρήστης έχει τη δυνατότητα, μέσω κάποιων κουμπιών επιλογής, να επιλέξει τους

τύπους των πινάκων που επιθυμεί να απεικονιστούν στο δένδρο. Αυτή η δυνατότητα παρέχεται από τη μέθοδο selectTableTypesForView(), η οποία παίρνει ως είσοδο έναν πίνακα με τα ονόματα των τύπων των πινάκων που θα περιληφθούν στο δένδρο της βάσης, το οποίο και σχεδιάζει επαναλαμβάνοντας σε μεγάλο βαθμό τη διαδικασία που περιγράφηκε παραπάνω. Οι μέθοδοι setConnectedGUI() και itemStateChanged() των κλάσεων MySQLPanel και PostgresPanel καθορίζουν ποια κουμπιά επιλογής θα είναι ενεργά και ποια όχι.

Ανά πάσα στιγμή, ο χρήστης μπορεί να επιλέξει κάποιον πίνακα από το δένδρο της βάσης και να εμφανιστούν σε έναν πίνακα τα **περιεχόμενα** αυτού του πίνακα. Αυτό επιτυγχάνεται με τη μέθοδο valueChanged, η οποία καλείται όταν ο ακροατής συμβάντων TreeSelectionListener ανιχνεύσει κάποιο συμβάν που σχετίζεται με το δένδρο της βάσης. Αυτή δημιουργεί ένα SQL ερώτημα ανάλογα με τον επιλεγμένο πίνακα, το οποίο θα ανακτήσει όλα τα δεδομένα του. Η valueChanged() καλεί με τη σειρά της την updateTable() της κλάσης DataPreviewPanel, σκοπός της οποίας είναι να ενσωματώσει έναν πίνακα δεδομένων σε ένα πλαίσιο.

```
// A new table has been selected from the tree and we want to display
   // a preview of it in the preview table

public void valueChanged(TreeSelectionEvent e) {
      //to check whether the tree corresponds to the currently working database
      if (treeModel.getChildCount(treeModel.getRoot())!=0){
         DefaultMutableTreeNode node = (DefaultMutableTreeNode)
treeTables.getLastSelectedPathComponent();
         if (node!=null){
               DefaultMutableTreeNode parentNode=(DefaultMutableTreeNode)node.getParent();
               TreeNodeInfo parentName=(TreeNodeInfo)parentNode.getUserObject();
               if (parentName.toString().equals("Data Source")){
                  TreeNodeInfo nodeInfo = (TreeNodeInfo)node.getUserObject();
                  String strTableName = getDBMSSpecificNamingStyle(nodeInfo.m_text);

                  String strQuery = "SELECT * FROM " + strTableName;
                  previewPanel.updateTable(connectInfo,strQuery,databaseSelected);
               }
         }
         if (node == null)
            return;
      }
   }
```

Πλαίσιο 6.13 Μέθοδος valueChanged() (MySQLPanel.java)

Το ερώτημα εκτελείται από την executeQuery() της κλάσης JDBCDataTable, που έχει κληθεί από την updateTable2() της DataPreviewPanel. Η κλάση JDBCDataTable επεκτείνει την κλάση AbstractTableModel και η μέθοδος executeQuery() ουσιαστικά γεμίζει τον πίνακα με τα δεδομένα που έχουν ανακτηθεί από το SQL ερώτημα που έχει προηγουμένως εκτελέσει. Τέλος, παρέχεται η δυνατότητα στο χρήστη να επιλέξει τον αριθμό των γραμμών των δεδομένων που θέλει να περιέχει ο πίνακας.

```
Class.forName(connectInfo.m_driver);
         con = DriverManager.getConnection(connectInfo.m_DSN, connectInfo.m_UID,
connectInfo.m_PWD);

         //in case of a mySQL connection,first select which database to use
```

```
if (connectInfo.m_DB.equals("MySQL")){
   PreparedStatement myStmt;
   String query ="USE "+databaseSelected;
   myStmt = con.prepareStatement(query);
   myStmt.execute(query);
}

   stmt = con.createStatement();
   resultSet = stmt.executeQuery(strQuery);


rsMetaData = resultSet.getMetaData();

int numColumns = rsMetaData.getColumnCount();
columnNames = new String[numColumns];
for (int i = 0; i < numColumns; i++)
   columnNames[i] = rsMetaData.getColumnLabel(i+1);

rows.removeAllElements();
int count = 0;
while (resultSet.next()) {
   Vector newRow = new Vector();
   for (int i = 0; i < numColumns; i++) {
      … //get the value of the current element and add it to vector newRow
   }
   rows.addElement(newRow);
   count++;
   if (count == numRows)
      break;
}
resultSet.close();
stmt.close();
con.close();

fireTableChanged(null);
return true;
```

Πλαίσιο 6.14 Συμπλήρωση πίνακα μιας βάσης δεδομένων (JDBCDataTable.java)

6.3 ΑΝΤΙΣΤΟΙΧΙΣΗ (MAPPING)

Η διαδικασία της αντιστοίχισης περιλαμβάνει την **επιλογή μιας κλάσης** από το δένδρο κλάσεων της οντολογίας, καθώς και την **επιλογή δεδομένων** από τη βάση δεδομένων. Η επιλογή μιας κλάσης της οντολογίας γίνεται με Drag από το δένδρο κλάσεων και Drop στο ειδικό πεδίο αντιστοίχισης (mapping). Για το σκοπό αυτό, στην κλάση OntologyPanel, έχουν ενσωματωθεί οι ακροατές DragSourceListener, DragGestureListener, ενώ στις κλάσεις QueryPanelMySQL και QueryPanelPostgres (όπου μεταξύ άλλων περιέχεται και το πεδίο αντιστοίχισης) έχει ενσωματωθεί ένας DropTargetListener.

Η επιλογή των δεδομένων από τη βάση δεδομένων βασίζεται και αυτή στη μέθοδο Drag & Drop και επιτυγχάνεται μέσω της (μη αντιληπτής από το χρήστη) δημιουργίας ενός SQL ερωτήματος, το οποίο όταν εκτελεστεί θα επιστρέψει τα επιλεγμένα δεδομένα. Υπάρχουν τρεις λίστες, στις οποίες μπορούν να προστεθούν είτε πίνακες είτε στήλες από τη βάση δεδομένων: η λίστα πινάκων, η λίστα στηλών

και η λίστα συνθηκών. Η λίστα πινάκων περιέχει τους πίνακες που θα συμμετέχουν στη διαδικασία επιλογής των δεδομένων, η λίστα στηλών τις στήλες στις οποίες ανήκουν τα επιθυμητά δεδομένα και η λίστα συνθηκών περιορισμούς που θα τεθούν στις επιλεγμένες στήλες, ώστε η επιλογή να είναι περισσότερο εξειδικευμένη. Με λίγα λόγια, σε ό,τι αφορά το SQL ερώτημα, η λίστα πινάκων περιλαμβάνει τους πίνακες που ακολουθούν το From, η λίστα στηλών τις στήλες που ακολουθούν το Select και η λίστα συνθηκών, τις συνθήκες που τοποθετούνται μετά το Where. Όλες οι προσθήκες στις λίστες διαχειρίζονται από τη μέθοδο drop() των κλάσεων QueryPanelMySQL και QueryPanelPostgres. Σε αυτή τη μέθοδο, γίνεται ο έλεγχος για το αν το στοιχείο που σύρεται από το δένδρο της βάσης δεδομένων επιτρέπεται να προστεθεί σε μια λίστα. Όταν ο χρήστης προσπαθεί να προσθέσει ένα στοιχείο στη λίστα πινάκων, ελέγχεται αν αυτό το στοιχείο είναι πράγματι πίνακας και στην περίπτωση αυτή, προστίθεται όχι μόνο ο συγκεκριμένος πίνακας στη λίστα πινάκων, αλλά και όλες οι στήλες του στη λίστα στηλών (συγκεκριμένα, όλες όσες δε βρίσκονται ήδη στη λίστα στηλών). Με αυτόν τον τρόπο, υπονοείται ότι **η άμεση προσθήκη ενός πίνακα στη λίστα από μέρους του χρήστη ισοδυναμεί με την επιλογή όλων των δεδομένων του πίνακα**. Στην περίπτωση που ο χρήστης προσπαθεί να προσθέσει ένα στοιχείο στη λίστα στηλών, η εφαρμογή πρώτα ελέγχει αν το στοιχείο αυτό είναι μια στήλη και αμέσως μετά, την προσθέτει στη λίστα. Επίσης, **αν ο πίνακας στον οποίο ανήκει η στήλη δε βρίσκεται ήδη στη λίστα πινάκων, προστίθεται αυτόματα**. Με αυτόν τον τρόπο, είναι γνωστό ανά πάσα στιγμή ποιοι πίνακες συμμετέχουν στη διαδικασία επιλογής των δεδομένων (SQL ερώτημα). Όσον αφορά στη λίστα συνθηκών, αυτή στην πραγματικότητα υλοποιείται με ένα αντικείμενο τύπου JTable. Η εφαρμογή επιτρέπει στο χρήστη να αφήσει με drop μόνο στήλες της βάσης δεδομένων στον πίνακα, οπότε προστίθεται σε αυτόν μια καινούρια γραμμή με δύο στήλες: η πρώτη περιέχει το όνομα της στήλης που έκανε drop ο χρήστης και η δεύτερη είναι κενή, έτσι ώστε ο χρήστης να συμπληρώσει μόνος του τον περιορισμό που επιθυμεί να θέσει στην επιλογή των δεδομένων. Όπως και στην περίπτωση της λίστας στηλών, αν ο πίνακας στον οποίο ανήκει η επιλεγμένη στήλη δε βρίσκεται ήδη στη λίστα πινάκων, τότε προστίθεται.

```
public void drop(DropTargetDropEvent event) {

        Transferable transferable = event.getTransferable();
        // we accept only Strings
        if (transferable.isDataFlavorSupported(DataFlavor.stringFlavor)){

           event.getDropTargetContext().dropComplete(true);


           if (event.getDropTargetContext().getComponent()==tableList){

              JTree treeTables=parentPanel.treeTables;
              String s = (String)transferable.getTransferData( DataFlavor.stringFlavor);
              DefaultMutableTreeNode node = (DefaultMutableTreeNode)
treeTables.getLastSelectedPathComponent();
              String nodeName=((TreeNodeInfo)node.getUserObject()).toString();
              DefaultMutableTreeNode parentNode=(DefaultMutableTreeNode)node.getParent();
              Enumeration children = node.children();
              TreeNodeInfo parentName=(TreeNodeInfo)parentNode.getUserObject();

              //search the table list to check if the dragged table is already in the list
```

```
            if (parentName.toString().equals("Data Source")){


                if (!tablesVector.contains(nodeName)) tablesVector.addElement(nodeName);
                //add in the fields list all the fields of the added table (that are not yet added)
                while (children.hasMoreElements()){
                   DefaultMutableTreeNode child = (DefaultMutableTreeNode)
children.nextElement();
                   TreeNodeInfo childInfo = (TreeNodeInfo)child.getUserObject();
                   String fieldName=childInfo.toString();
                   ...

                   //add to the field list all the fields of the table dragged, unless they are already
in it
                   ...
                }

              tableList.setListData(tablesVector);
              fieldList.setListData(fieldsVector);

            }
            else {
              JOptionPane.showMessageDialog(null,"Please drop only tables from the database
in this field",
                 "No Table Dragged",
                 JOptionPane.ERROR_MESSAGE);
            }
         }


         else if (event.getDropTargetContext().getComponent()==fieldList){
            …
             //likewise in the tables case
             //accept only columns dropped
            if (!parentName.toString().equals("Data Source")){
               …

               //add the table that this field belongs to, to the table list, unless it already exists
               if (!tablesVector.contains(parentName.toString())) {
                  tablesVector.addElement(parentName.toString());
               }

               //add in front of every field name that has a synonymy, its parent table name
               …

               //search the field list to check if the dragged field is already in the list
               …

               fieldList.setListData(fieldsVector);
               tableList.setListData(tablesVector);
            }
            else {
              JOptionPane.showMessageDialog(null,"Please drop only columns from the
database in this field",
                 "No Column Dragged",
                 JOptionPane.ERROR_MESSAGE);
            }

         }
```

```
        else if ((event.getDropTargetContext().getComponent()==scrollTable) ||
(event.getDropTargetContext().getComponent()==criteriaTable)){

            JTree treeTables=parentPanel.treeTables;
            String s = (String)transferable.getTransferData( DataFlavor.stringFlavor);
            DefaultMutableTreeNode
node=(DefaultMutableTreeNode)treeTables.getLastSelectedPathComponent();
            DefaultMutableTreeNode parentNode=(DefaultMutableTreeNode)node.getParent();
            TreeNodeInfo parentName=(TreeNodeInfo)parentNode.getUserObject();
            //accept only columns dropped
            if (!parentName.toString().equals("Data Source")){
               …

               criteriaVector.addElement(fieldInfo);
               Vector temp = new Vector();
               temp.addElement(fieldInfo.fieldName);
               dm.addRow(temp);
               criteriaTable.setModel(dm);

               //add the table to which this field belongs to the table list as well
               if (!tablesVector.contains(parentName.toString()))
                  tablesVector.addElement(parentName.toString());
               tableList.setListData(tablesVector);

            }
            else {
               JOptionPane.showMessageDialog(null,"Please drop only columns from the
database in this field",
                  "No Column Dragged",
                  JOptionPane.ERROR_MESSAGE);
            }

         }
        else{
            event.rejectDrop();
         }


      } else{
         event.rejectDrop();
      }
   }
```

Πλαίσιο 6.15 Μέθοδος drop() (QueryPanelMySQL.java)

Προκειμένου το SQL ερώτημα να είναι απόλυτα σαφές και να αποφευχθούν προβλήματα από πιθανές **συνωνυμίες στηλών**, σε κάθε προσθήκη σε λίστα, ελέγχεται πρώτα αν στο δένδρο της βάσης υπάρχει άλλη στήλη με το ίδιο όνομα και στην περίπτωση που υπάρχει, προστίθεται το όνομα του πίνακα μπροστά από το όνομα της στήλης. Ο έλεγχος συνωνυμιών επιτελείται από τη μέθοδο hasSynonymy() των κλάσεων QueryPanelMySQL και QueryPanelPostgres, η οποία παίρνει ως είσοδο ένα αντικείμενο FieldInfo και επιστρέφει μια Boolean τιμή, ανάλογα με το αν υπάρχει στήλη με το ίδιο όνομα σε κάποιον άλλο πίνακα ή όχι. Η κλάση FieldInfo είναι μια απλή κλάση που περιέχει δύο strings, που δηλώνουν το όνομα της στήλης και το όνομα του πίνακα στο οποίο αυτή ανήκει. Κάθε φορά που ο χρήστης αφήνει μια στήλη σε κάποια λίστα, δημιουργείται ένα αντικείμενο FieldInfo, που περιγράφει την επιλεγμένη στήλη και έτσι γίνεται γνωστό στην εφαρμογή σε ποιον πίνακα ανήκει η εν λόγω στήλη. Όπως αναφέρθηκε, η hasSynonymy() διατρέχει όλο το δένδρο της

βάσης και αν βρει μια στήλη με το ίδιο όνομα με τη στήλη-όρισμα, αλλά με διαφορετικό πατέρα (πίνακα), τότε επιστρέφει true, κάτι που σημαίνει ότι υπάρχει στο δένδρο τουλάχιστον μία συνωνυμία.

```
/* check if a field of a table has synonymy with one of another table.
   *If there is a synonymy,add in front of the field's name the table's name it belongs to
   */
  public boolean hasSynonymy(FieldInfo f){

     String parentName=f.fieldParent;
     String name=f.fieldName;
     DefaultTreeModel treeModel=parentPanel.treeModel;
     boolean hasSyn=false;

     DefaultMutableTreeNode root=(DefaultMutableTreeNode)treeModel.getRoot();
     int tablesCount=treeModel.getChildCount(root);

     for (int i=0;i<tablesCount;i++){
        DefaultMutableTreeNode currentTable = (DefaultMutableTreeNode)
treeModel.getChild(root,i);
        TreeNodeInfo tableNode=(TreeNodeInfo)(currentTable).getUserObject();
        String tableName=tableNode.toString();
        if (!tableName.equals(parentName)){
           int fieldsCount=treeModel.getChildCount(treeModel.getChild(root,i));
           for (int j=0;j<fieldsCount;j++){
              DefaultMutableTreeNode currentField = (DefaultMutableTreeNode)
treeModel.getChild(currentTable,j);
              String fieldName=((TreeNodeInfo)currentField.getUserObject()).toString();
              fieldName=fixFieldsName(fieldName);
              if (fieldName.equals(name)) hasSyn=true;
           }
        }
     }
     return hasSyn;

  }
```

Πλαίσιο 6.16 Μέθοδος hasSynonymy() (QueryPanelMySQL.java)

Επίσης, προσφέρεται η δυνατότητα **αφαίρεσης** κάποιων στοιχείων από τις λίστες που αναφέρθηκαν παραπάνω. Αυτή η διαδικασία πραγματοποιείται μέσα στη μέθοδο actionPerformed() των κλάσεων QueryPanelMySQL και QueryPanelPostgres. Αν ζητηθεί από το χρήστη η αφαίρεση ενός πίνακα από τη λίστα πινάκων, τότε αφαιρείται όχι μόνο ο πίνακας, αλλά και όλες οι στήλες του πίνακα που έχουν προστεθεί σε κάποια από τις άλλες δύο λίστες. Αντίστοιχα, αξίζει να σημειωθεί ότι αν αφαιρεθεί μια στήλη από τη λίστα στηλών ή μια συνθήκη (που αφορά σε μια στήλη) από τη λίστα συνθηκών, τότε αφαιρείται και ο πίνακας στον οποίο ανήκει αυτή η στήλη, με την προϋπόθεση βέβαια ότι δεν υπάρχουν άλλες στήλες από τον εν λόγω πίνακα σε κάποια από τις δύο λίστες.

```
//if Remove in the Table List is pressed
     if (source==removeMenuItemT){
        String tobeRemoved=(String)tableList.getSelectedValue();

//remove table from the table list
              tablesVector.remove(tobeRemoved);
```

```
            try{

               DatabaseMetaData dbMetaData =parentPanel.con.getMetaData();

               //remove all of its fields in the fieldList and criteria list
               ResultSet rsColumns = dbMetaData.getColumns(parentPanel.databaseSelected,
null, tobeRemoved, null);
               boolean bMore = rsColumns.next();
               while (bMore) {
                  String strColumnName = rsColumns.getString("COLUMN_NAME");
                  for (int i=0;i<fieldInfoVector.size();i++){
                     FieldInfo current=(FieldInfo)fieldInfoVector.elementAt(i);
                     FieldInfo temp=new FieldInfo();
                     temp.fieldName=strColumnName;
                     temp.fieldParent=tobeRemoved;
                     boolean hasSyn=hasSynonymy(temp);
                     if (hasSyn){
                        if (current.fieldName.equals(tobeRemoved+"."+strColumnName)) {
                           fieldsVector.remove(current.fieldName);
                           fieldInfoVector.remove(current);
                        }
                     }       else       if       (strColumnName.equals(current.fieldName)       &&
(tobeRemoved.equals(current.fieldParent))){
                        fieldsVector.remove(strColumnName);
                        fieldInfoVector.remove(current);
                     }
                  }
                  for (int i=0;i<criteriaVector.size();i++){
                     FieldInfo current=(FieldInfo)criteriaVector.elementAt(i);
                     FieldInfo temp=new FieldInfo();
                     temp.fieldName=strColumnName;
                     temp.fieldParent=tobeRemoved;
                     boolean hasSyn=hasSynonymy(temp);
                     if (hasSyn){
                        if (current.fieldName.equals(tobeRemoved+"."+strColumnName)) {
                           criteriaVector.remove(current);
                           for (int j=0;j<dm.getRowCount();j++){
                              if       (current.fieldName.equals       ((String)dm.getValueAt(j,0)))
dm.removeRow(j);
                           }
                        }
                     }       else       if       (strColumnName.equals(current.fieldName)       &&
(tobeRemoved.equals(current.fieldParent))){
                        criteriaVector.remove(current);
                        for (int j=0;j<dm.getRowCount();j++){
                           if       (strColumnName.equals       ((String)dm.getValueAt(j,0)))
dm.removeRow(j);
                        }
                     }
                  }
                  bMore = rsColumns.next();
               }
               rsColumns.close();

            }catch(SQLException ex){

            }
```

```
            fieldList.setListData(fieldsVector);
            tableList.setListData(tablesVector);

        }
```

Πλαίσιο 6.17 Αφαίρεση πίνακα από τη λίστα πινάκων (QueryPanelMySQL.java)

```
class FieldInfo {

                    String fieldName;
                    String fieldParent;
}
```

Πλαίσιο 6.18 Κλάση FieldInfo (QueryPanelPostgres.java)

Από τη στιγμή που έχει επιλεγεί τόσο μια κλάση της οντολογίας (εξαιρείται η κλάση owl:Thing) όσο και κάποια δεδομένα από τη βάση, ο χρήστης πατώντας το κουμπί **αντιστοίχισης** (map) μπορεί να την πραγματοποιήσει. Πρώτα, η εφαρμογή **δημιουργεί το SQL ερώτημα** το οποίο θα πρέπει να συσχετιστεί με την εν λόγω κλάση, έχοντας ως δεδομένα τα περιεχόμενα των τριών λιστών. Αρχικά, ελέγχεται αν υπάρχει δυνατότητα απλοποίησης του SQL ερωτήματος με τη χρήση αστερίσκου, κάτι που ισχύει όταν η λίστα στηλών περιέχει όλες τις στήλες των πινάκων που βρίσκονται στη λίστα πινάκων.

```
queryString=new String();
            queryString=queryString.concat("SELECT ");

            boolean needStar=true;

            //check if star is needed in the query
               DatabaseMetaData dbMetaData =parentPanel.con.getMetaData();
               Vector allFields=new Vector();
               for (int i=0;i<tablesVector.size();i++){
                  String tableName=(String)tablesVector.elementAt(i);
                  ResultSet rsColumns = dbMetaData.getColumns(parentPanel.databaseSelected,
null, tableName, null);
                  boolean bMore = rsColumns.next();
                  while (bMore) {
                     String strColumnName = rsColumns.getString("COLUMN_NAME");
                     FieldInfo fi=new FieldInfo();
                     fi.fieldName=strColumnName;
                     fi.fieldParent=tableName;
                     allFields.addElement(fi);
                     bMore=rsColumns.next();
                  }
               }
               for (int i=0;i<allFields.size();i++){
                  FieldInfo fi=(FieldInfo)allFields.elementAt(i);
                  String fieldName=fi.fieldName;
                  boolean hasSyn=hasSynonymy(fi);
                  if (hasSyn) fieldName=fi.fieldParent+"."+fi.fieldName;
                  if (!fieldsVector.contains(fieldName)){
                     needStar=false;
                     break;
                  }
               }
               if (fieldsVector.size()==0) needStar=true;
```

Πλαίσιο 6.19 Έλεγχος για τη χρήση αστερίσκου στο SQL ερώτημα (QueryPanelMySQL.java)

Μετά τη λέξη Select προστίθενται τα ονόματα των στηλών που περιέχονται στη λίστα στηλών (ή αστερίσκος, αν το επιτρέπει η περίπτωση), ακολουθεί η λέξη From και τα ονόματα των πινάκων της λίστας πινάκων και τέλος, αν η λίστα συνθηκών δεν είναι άδεια, προστίθεται η λέξη Where μαζί με τις υπάρχουσες συνθήκες. Αξίζει να σημειωθεί ότι στην περίπτωση της PostgreSQL, η οποία υποστηρίζει ξένα κλειδιά, εκτός από τις ρητά καθορισμένες από το χρήστη συνθήκες, χρειάζεται να προστεθούν και αυτές που επιβάλλονται από την ύπαρξη ξένων κλειδιών στους επιλεγμένους πίνακες. Επομένως, χρειάζεται μια διερεύνηση για αυτές τις **επιπλέον συνθήκες** που θα προστεθούν. Η εφαρμογή διατρέχει μια λίστα με τα ξένα κλειδιά όλων των πινάκων της βάσης δεδομένων και για κάθε ένα από αυτά, εξετάζει αν τόσο ο πίνακας στον οποίο ανήκει όσο και ο πίνακας στον οποίο δείχνει συμπεριλαμβάνονται στη λίστα πινάκων. Σε αυτήν την περίπτωση προστίθεται η κατάλληλη συνθήκη μετά τη λέξη Where.

```
if (needStar) queryString=queryString.concat("*");
 else {
    Iterator it=fieldsVector.iterator();
    while (it.hasNext()){
       queryString=queryString.concat((String)it.next());
       if (it.hasNext()) queryString=queryString.concat(" , ");
    }
 }

 queryString=queryString.concat(" FROM ");
 Iterator it2=tablesVector.iterator();
 while (it2.hasNext()){
    queryString=queryString.concat((String)it2.next());
    if (it2.hasNext()) queryString=queryString.concat(" , ");
 }

 //add WHERE only if there are criteria in the criteria table
 if (criteriaVector.size()>0){
    queryString=queryString.concat(" WHERE ");
    String criteriaString=new String();

    for (int i=0;i<dm.getRowCount();i++) {
       criteriaString=criteriaString.concat("(");
       String currentField =(String)dm.getValueAt(i,0);
       String currentCondition=(String)dm.getValueAt(i,1);
       criteriaString=criteriaString.concat(currentField+currentCondition+")");
       if (i!=dm.getRowCount()-1){
          criteriaString=criteriaString.concat(" AND ");
       }

    }
    queryString=queryString.concat(criteriaString);
```

Πλαίσιο 6.20 Δημιουργία SQL ερωτήματος (QueryPanelMySQL.java)

```
Iterator it3=parentPanel.foreignKeys.iterator();
 Iterator it4=tablesVector.iterator();
 boolean needWhere=false;
 while (it3.hasNext()){
    ForeignInfo currentForeign=(ForeignInfo)it3.next();
    String fk_table=currentForeign.fk_table;
    String pk_table=currentForeign.pk_table;
    boolean foundfk=false;
```

```
                boolean foundpk=false;
                while (it4.hasNext()){
                   String currentTable=it4.next().toString();
                   if (fk_table.equals(currentTable)) foundfk=true;
                   else if (pk_table.equals(currentTable)) foundpk=true;
                }
                if (foundpk&&foundfk) {
                   inclForeign.addElement(currentForeign);
                   needWhere=true;
                }

             }

             //add WHERE only if there are criteria in the criteria table or if some foreign keys need
to be taken into account
             if ((criteriaVector.size()>0)||needWhere){
                queryString=queryString.concat(" WHERE ");
                for (int j=0; j<inclForeign.size();j++ ){
                   String fk_name=((ForeignInfo)inclForeign.elementAt(j)).fk_name;
                   String pk_name=((ForeignInfo)inclForeign.elementAt(j)).pk_name;
                   if (j!=0) queryString=queryString.concat("AND ");
                   queryString=queryString.concat("("+fk_name+"="+pk_name+") ");
                   if ((j==inclForeign.size()-1)&&(criteriaVector.size()!=0))
queryString=queryString.concat("AND ");

                }
               ... likewise in MySQL
             }
```

Πλαίσιο 6.21 Έλεγχος ξένων κλειδιών κατά το σχηματισμό SQL ερωτήματος στην περίπτωση της PostgreSQL (QueryPanelPostgres.java)

```
//a class containing all the information needed about a foreign key
class ForeignInfo{

   public ForeignInfo(String fk_name,String fk_table,String pk_name,String pk_table){
      this.fk_name=fk_name;
      this.fk_table=fk_table;
      this.pk_name=pk_name;
      this.pk_table=pk_table;
   }
```

Πλαίσιο 6.22 Κλάση ForeignInfo (PostgresPanel.java)

Μόλις ολοκληρωθεί η σύνταξη του SQL ερωτήματος, εκτελείται για να ελεγχθεί η **εγκυρότητά** του και αν δεν υπάρχει κάποιο λάθος, η εφαρμογή προχωρεί σε κάποιους **σημασιολογικούς ελέγχους** που σχετίζονται με την αντιστοίχιση του ερωτήματος στην επιλεγμένη κλάση. Αρχικά, καλείται η μέθοδος disjCheckMapDown(), η οποία ελέγχει σε ποια δεδομένα έχουν αντιστοιχηθεί οι ξένες προς την κλάση αυτή κλάσεις της οντολογίας ή οι υποκλάσεις τους και αν είναι ίδια με τα δεδομένα με τα οποία πρόκειται η τρέχουσα κλάση να αντιστοιχηθεί. Η disjCheckMapDown() δέχεται ως ορίσματα την επιλεγμένη κλάση καθώς και ένα διάνυσμα με τα δεδομένα στα οποία πρόκειται αυτή να αντιστοιχηθεί, όπως αυτά έχουν προκύψει μετά την εκτέλεση του SQL ερωτήματος. Συγκεκριμένα, διατρέχονται όλες οι ξένες προς την επιλεγμένη κλάση κλάσεις και για κάθε μία από αυτές, εκτελείται το SQL ερώτημα με το οποίο αυτές έχουν συσχετιστεί (αν έχουν ήδη απεικονιστεί κάπου) και τα αποτελέσματα του ερωτήματος συγκρίνονται ένα προς ένα με τα δεδομένα που έχουν περάσει ως όρισμα στη μέθοδο.

```
//check whether disjoint classes (and their subclasses) contain the same data
   public boolean disjCheckMapDown(OntClass cls,Vector v){

      boolean mapAccepted=true;
      Iterator idis=cls.listDisjointWith();
      while (idis.hasNext()&&mapAccepted){
         OntClass cur=(OntClass)idis.next();
         OntProperty map=ontModel.getOntProperty(basePrefix+"queryString");

         mapAccepted=recurseDown(cur,map,cls,v);
      }

      return mapAccepted;
   }
```

Πλαίσιο 6.23 Μέθοδος disjCheckMapDown (QueryPanelPostgres.java)

Η παραπάνω σύγκριση των δεδομένων γίνεται με την κλήση της μεθόδου compareData(), η οποία καλείται στη μέθοδο recurseDown(), στην περίπτωση που μια ξένη κλάση έχει απεικονιστεί σε κάποια δεδομένα. Αν βρεθεί έστω και ένα ταίριασμα, τότε η αντιστοίχιση απορρίπτεται καθώς ξένες μεταξύ τους κλάσεις δεν επιτρέπεται να έχουν αντιστοιχηθεί σε ίδια δεδομένα. Στην περίπτωση της **MySQL**, η οποία δεν υποστηρίζει ξένα κλειδιά, αρκεί ο έλεγχος της τιμής δύο δεδομένων και των ονομάτων της στήλης και του πίνακα στον οποίο αυτά ανήκουν. Κάθε δεδομένο λοιπόν, αντιπροσωπεύεται από ένα αντικείμενο DataInfo, μια απλή κλάση που περιέχει την τιμή του δεδομένου, το όνομα της στήλης και το όνομα του πίνακα στο οποίο αυτό ανήκει. Αν υπάρξει απόλυτη ταύτιση μεταξύ δύο αντικειμένων DataInfo, αυτό σημαίνει ότι είναι το ίδιο δεδομένο και συνεπώς, η αντιστοίχιση απορρίπτεται για σημασιολογικούς λόγους. Στην περίπτωση της **PostgreSQL**, εκτός από τον απλό παραπάνω έλεγχο, εισάγεται και ένας επιπλέον σχετιζόμενος με τα ξένα κλειδιά. Δηλαδή, αν μια στήλη-ξένο κλειδί έχει αντιστοιχηθεί σε μια κλάση και η στήλη στην οποία δείχνει αυτό το ξένο κλειδί έχει αντιστοιχηθεί σε μια ξένη προς αυτή κλάση, τότε η αντιστοίχιση θεωρείται απορριπτέα. Δυστυχώς, στην PostgreSQL το πρόβλημα έγκειται στη χρήση της μεθόδου getTableName() της κλάσης ResultSetMetaData, η οποία θα έπρεπε να επιστρέφει το όνομα του πίνακα στον οποίο ανήκει μια συγκεκριμένη στήλη του ResultSet που περιέχει δεδομένα, αλλά στη συγκεκριμένη περίπτωση δεν επιστρέφει τίποτα, σε αντίθεση με τη MySQL. Έτσι, ο έλεγχος ταυτότητας δύο δεδομένων δεν είναι απόλυτα ακριβής, εφόσον δεν υπάρχει τρόπος ανάκτησης του ονόματος του πίνακα στον οποίο ανήκει κάποιο δεδομένο. Βέβαια, ο έλεγχος τιμής και ονόματος στήλης τις περισσότερες φορές κρίνεται επαρκής και μόνο σε περίπτωση ύπαρξης στηλών με το ίδιο όνομα σε διαφορετικούς πίνακες, γίνεται απόρριψη μιας αντιστοίχισης που θα έπρεπε, υπό κανονικές συνθήκες, να επιτρέπεται.

```
//compare the data mapped to classes cur and cls
   public boolean compareData(OntClass cur, OntProperty map,OntClass cls, Vector v ){
      boolean mapAccepted=true;
      StmtIterator i = ontModel.listStatements(cur, map, (RDFNode)null);
            while (i.hasNext()){
               com.hp.hpl.jena.rdf.model.Statement s=i.nextStatement();
               String execquery=s.getString();
               ResultSet ret;

               /*
                *execute the query for OntClass cur and store the results in another vector temp
```

```
            */
              java.sql.Statement myStmt = parentPanel.con.createStatement();
              ret = myStmt.executeQuery(execquery);

              ResultSetMetaData rmeta=ret.getMetaData();
              int numColumns=rmeta.getColumnCount();

              Vector temp=new Vector();

              while(ret.next()){
                 for(int j=1;j<=numColumns;j++) {
                    FieldInfo newFI=new FieldInfo();
                    newFI.fieldName=ret.getString(j);
                    newFI.fieldParent=rmeta.getColumnName(j);
                    temp.addElement(newFI);
                 }
              }

              //compare the two vectors

              for (int j=0;j<v.size();j++){
                 FieldInfo orcur =(FieldInfo)v.elementAt(j);
                 for (int k=0;k<temp.size();k++){
                    FieldInfo tempcur=(FieldInfo)temp.elementAt(k);
                    //map not accepted,because at least one cell of a table has been mapped to
both classes.
                    if ((tempcur.fieldName.equals(orcur.fieldName)) &&
(tempcur.fieldParent.equals(orcur.fieldParent))){
                       mapAccepted=false;
                       break;
                    }
                    for (int l=0;l<parentPanel.foreignKeys.size();l++){
                       ForeignInfo curFI=(ForeignInfo)parentPanel.foreignKeys.elementAt(l);
                       //the class being checked has been mapped to data that belong in a foreign
key column that points to a table mapped to a disjoint class
                       if ((tempcur.fieldName.equals(orcur.fieldName)) &&
(curFI.fk_name.equals(orcur.fieldParent)) && (curFI.pk_name.equals(tempcur.fieldParent))){
                          mapAccepted=false;
                          break;
                       }
                       //the class being checked has been mapped to data that belong in a foreign
key column of a table that has been mapped to a disjoint class
                       if ((tempcur.fieldName.equals(orcur.fieldName)) &&
(curFI.pk_name.equals(orcur.fieldParent)) && (curFI.fk_name.equals(tempcur.fieldParent))){
                          mapAccepted=false;
                          break;
                       }
                    }

                 }
                 if (!mapAccepted){
                    JOptionPane.showMessageDialog(null,"Disjoint Classes can not have the
same data mapped to them.\nClass "+cur.getLocalName()+" is disjoint with
"+cls.getLocalName(),
                          "Mapping Not Accepted", JOptionPane.ERROR_MESSAGE);
                    break;
                 }


              }
```

```
            }
        return mapAccepted;
    }
```

Πλαίσιο 6.24 Μέθοδος compareData() (QueryPanelPostgres.java)

```
//class containing information about every element of the tables
class DataInfo{
    String data;    //the element's value
    String column;  //the column it belongs to
    String table;   //its table

}
```

Πλαίσιο 6.25 Κλάση DataInfo (QueryPanelMySQL.java)

Αν κάποια από τις ξένες προς την επιλεγμένη κλάση κλάσεις της οντολογίας δεν έχει αντιστοιχηθεί σε κάποιο σύνολο δεδομένων, τότε ο **έλεγχος επεκτείνεται στις υποκλάσεις** της κλάσης αυτής με την αναδρομική κλήση της recurseDown().

```
//recurse further down to the subclasses of cur and so on
    public boolean recurseDown (OntClass cur,OntProperty map,OntClass cls,Vector v){
        boolean mapAccepted=true;
        if (map.hasDomain(cur)){
            mapAccepted=compareData(cur,map,cls,v);
          }
          else {
            Iterator isub=ontPanel.ch.findSubclasses(cur);
            while (isub.hasNext()){
              mapAccepted=recurseDown((OntClass)isub.next(),map,cls,v);
              if (!mapAccepted) break;
            }
          }
        return mapAccepted;
    }
```

Πλαίσιο 6.26 Μέθοδος recurseDown() (QueryPanelPostgres.java)

Μόλις ολοκληρωθεί ο έλεγχος των ξένων κλάσεων και των υποκλάσεών τους, αν δεν έχει βρεθεί κάποιο σημασιολογικό λάθος, επεκτείνεται ο ίδιος έλεγχος και για τις υπερκλάσεις της επιλεγμένης κλάσης και τις ξένες κλάσεις αυτών, καλώντας τη μέθοδο disjCheckMapUp(). Αυτή παρουσιάζει πολλές ομοιότητες με την disjCheckMapDown() και καλεί τη recurseUp() για όλες τις υπερκλάσεις της επιλεγμένης κλάσης. Η recurseUp(), με τη σειρά της, καλεί την compareData() για σύγκριση των δεδομένων που έχουν αντιστοιχηθεί στις ξένες κλάσεις των υπερκλάσεων της αρχικής κλάσης. Αν μια από αυτές τις ξένες κλάσεις δεν έχει αντιστοιχηθεί, καλείται πρώτα η recurseDown(), για να ερευνήσει τις υποκλάσεις της, προτού επεκταθεί ο έλεγχος στο αμέσως ανώτερο επίπεδο με την αναδρομική κλήση της recurseUp().

```
    public boolean disjCheckMapUp(OntClass cls,Vector v){
        boolean mapAccepted=true;
        OntProperty map=ontModel.getOntProperty(basePrefix+"queryString");

        //recurse for the disjoints of the class's superclass
          Iterator isuper=ontPanel.ch.findSuperclasses(cls);
          while (isuper.hasNext()){
```

```
            OntClass cur=(OntClass)isuper.next();
            mapAccepted=recurseUp(cur,map,cls,v);
            if (!mapAccepted) break;
          }

      return mapAccepted;
    }
```

Πλαίσιο 6.27 Μέθοδος disjCheckMapUp() (QueryPanelPostgres.java)

```
//move up and check the disjoints of cur and their superclasses as well as the subclasses of every
one of them
    public boolean recurseUp (OntClass cur,OntProperty map,OntClass cls,Vector v){
      boolean mapAccepted=true;
      Iterator idis=cur.listDisjointWith();
      while ((idis.hasNext())&&(mapAccepted)){
         OntClass current=(OntClass)idis.next();
         if (map.hasDomain(current)){
            mapAccepted=compareData(current,map,cls,v);
            if (!mapAccepted) break;
         }
         else {
            mapAccepted=recurseDown(current,map,cls,v);
            Iterator isuper=ontPanel.ch.findSuperclasses(current);
            while (isuper.hasNext()&&mapAccepted){
               mapAccepted=recurseUp((OntClass)isuper.next(),map,cls,v);
               if (!mapAccepted) break;
            }
         }
      }
      idis=cur.listDisjointWith();
      if (!idis.hasNext()){
         Iterator isuper=ontPanel.ch.findSuperclasses(cur);
         while (isuper.hasNext()&&mapAccepted){
              mapAccepted=recurseUp((OntClass)isuper.next(),map,cls,v);
              if (!mapAccepted) break;
           }
      }
      return mapAccepted;
    }
```

Πλαίσιο 6.28 Μέθοδος recurseUp() (QueryPanelPostgres.java)

Μετά τις disjCheckMapDown() και disjCheckMapUp(), καλείται η μέθοδος superCheckMap(), η οποία ελέγχει αν τα δεδομένα που πρόκειται να αντιστοιχηθούν σε μια κλάση έχουν επίσης αντιστοιχηθεί στις υπερκλάσεις της. Όπως οι disjCheckMapDown() και disjCheckMapUp(), η superCheckMap() παίρνει ως ορίσματα μια κλάση της οντολογίας και ένα διάνυσμα με τα δεδομένα που πρόκειται να αντιστοιχηθούν σε αυτή. Για κάθε υπερκλάση της τρέχουσας κλάσης, εκτελείται το αντίστοιχο SQL ερώτημα (αν υπάρχει) και γίνεται έλεγχος αν τα ανακτηθέντα δεδομένα περιέχουν τα δεδομένα στα οποία η τρέχουσα κλάση θα αντιστοιχηθεί. Αν όχι, τότε η αντιστοίχιση απορρίπτεται, καθώς δεν επιτρέπεται μια υπερκλάση να μην περιέχει όλα τα δεδομένα που έχουν αντιστοιχηθεί σε μια υποκλάση της. Και εδώ, υπάρχει το ίδιο πρόβλημα με την PostgreSQL και τη χρήση της getTableName(). Επίσης, στην PostgreSQL, ο έλεγχος χρησιμοποιεί και τα ξένα κλειδιά όπως παραπάνω, σε αντίθεση με τη MySQL. Αν κάποια από τις άμεσες υπερκλάσεις της τρέχουσας κλάσης δεν έχει αντιστοιχηθεί σε κάποια δεδομένα, τότε αναγκαστικά ο έλεγχος πρέπει να μεταφερθεί στις υπερκλάσεις του αμέσως επόμενου επιπέδου και

ούτω καθεξής. Αυτό επιτυγχάνεται με την επαναληπτική κλήση της superCheckMap() για όλες τις άμεσες υπερκλάσεις της τρέχουσας κλάσης που δεν έχουν αντιστοιχηθεί σε κάποια δεδομένα.

```
//check whether superclasses of the class being checked contain all of the data
   public boolean superCheckMap(OntClass cls,Vector v){

      boolean mapAccepted=true;
      Vector superChecked=new Vector();
      Iterator isuper = ontPanel.ch.findSuperclasses(cls);
      while (isuper.hasNext()&&mapAccepted){
         OntClass cur=(OntClass)isuper.next();
         OntProperty map=ontModel.getOntProperty(basePrefix+"queryString");

         //if this class has been mapped to a query
         if (map.hasDomain(cur)){
            superChecked.addElement(cur.getURI());
            StmtIterator i = ontModel.listStatements(cur, map, (RDFNode)null);
            while (i.hasNext()){
               com.hp.hpl.jena.rdf.model.Statement s=i.nextStatement();
               String execquery=s.getString();
               ResultSet ret;

               /**for every superclass
                *execute the query and store the results in another vector temp
                */
                  java.sql.Statement myStmt = parentPanel.con.createStatement();
                  ret = myStmt.executeQuery(execquery);

                  ResultSetMetaData rmeta=ret.getMetaData();
                  int numColumns=rmeta.getColumnCount();

                  Vector temp=new Vector();
                  ... //store the data in vector temp

                  //compare the two vectors
                   for (int j=0;j<v.size();j++){
                     FieldInfo orcur =(FieldInfo)v.elementAt(j);
                     boolean dataFound=false;
                     for (int k=0;k<temp.size();k++){
                        FieldInfo tempcur=(FieldInfo)temp.elementAt(k);
                        if                ((tempcur.fieldName.equals(orcur.fieldName))                &&
(tempcur.fieldParent.equals(orcur.fieldParent))){
                           dataFound=true;
                           break;
                        }
                        for (int l=0;l<parentPanel.foreignKeys.size();l++){
                           ForeignInfo curFI=(ForeignInfo)parentPanel.foreignKeys.elementAt(l);
                           //the class being checked has been mapped to data that belong in a foreign
key column that points to a table mapped to a superclass
                           if                ((tempcur.fieldName.equals(orcur.fieldName))                &&
(curFI.fk_name.equals(orcur.fieldParent)) && (curFI.pk_name.equals(tempcur.fieldParent))){
                              dataFound=true;
                              break;
                           }
                           //the class being checked has been mapped to data that belong in a foreign
key column of a table that has been mapped to a superclass
                           if                ((tempcur.fieldName.equals(orcur.fieldName))                &&
(curFI.pk_name.equals(orcur.fieldParent)) && (curFI.fk_name.equals(tempcur.fieldParent))){
```

```
                        dataFound=true;
                        break;
                    }
                }
            }
            //map not accepted,because at least one cell of a table is not found on the data
mapped to one of the superclasses
            if (!dataFound) mapAccepted=false;
            if (!mapAccepted){
               JOptionPane.showMessageDialog(null,"Subclasses must contain no data that
are different from the ones mapped to their superclasses.\nClass "+cls.getLocalName()+" is
subclass of "+cur.getLocalName(),
                        "Mapping Not Accepted", JOptionPane.ERROR_MESSAGE);
               break;
            }
          }

         }

      }

   }
   //at the end, recurse for (indirect) superclasses
   if (mapAccepted){
      isuper=ontPanel.ch.findSuperclasses(cls);
      while (isuper.hasNext()){
         OntClass cur=(OntClass)isuper.next();
         if (!superChecked.contains(cur.getURI())) mapAccepted=superCheckMap(cur,v);
         if (!mapAccepted) break;
      }
   }
   return mapAccepted;
}
```

Πλαίσιο 6.29 Μέθοδος superCheckMap() (QueryPanelPostgres.java)

Ο τρίτος κατά σειρά σημασιολογικός έλεγχος εκτελείται από την subCheckMap(), η οποία παίρνει τα ίδια ορίσματα με τις δύο προηγούμενες και ελέγχει αν οι υποκλάσεις της τρέχουσας κλάσης έχουν αντιστοιχηθεί σε δεδομένα που περιέχονται στα δεδομένα που πρόκειται να αντιστοιχηθούν στην τρέχουσα κλάση. Η δομή της subCheckMap() είναι ίδια με αυτή της superCheckMap(), δηλαδή εκτελεί τα SQL ερωτήματα που πιθανόν έχουν συσχετιστεί με κάθε μία από τις υποκλάσεις της τρέχουσας κλάσης και συγκρίνει τα δεδομένα που προκύπτουν με τα δεδομένα που πρόκειται να αντιστοιχηθούν στην τρέχουσα κλάση. Αν μια υποκλάση έχει αντιστοιχηθεί σε διαφορετικά ή περισσότερα δεδομένα από την τρέχουσα κλάση, τότε η αντιστοίχιση απορρίπτεται. Όπως και στην superCheckMap(), καλείται επαναληπτικά η subCheckMap() για όλες τις άμεσες υποκλάσεις που δεν έχουν απεικονιστεί κάπου για να επεκταθεί ο έλεγχος και στα κατώτερα επίπεδα.

Αν όλοι οι έλεγχοι είναι θετικοί, τότε επόμενο βήμα είναι ο συσχετισμός της επιλεγμένης κλάσης με το SQL ερώτημα που μόλις δημιουργήθηκε. Για το σκοπό αυτό δημιουργείται μια datatype property με το όνομα queryString, η οποία παίρνει τιμή το SQL ερώτημα για τη συγκεκριμένη κλάση. Με άλλα λόγια, δημιουργείται μια δήλωση (statement) που έχει ως υποκείμενο την κλάση, ως κατηγόρημα την queryString και ως αντικείμενο ένα string που αντιπροσωπεύει το SQL ερώτημα. Στην περίπτωση που η **ίδια κλάση έχει προηγουμένως αντιστοιχηθεί** σε κάποια

δεδομένα, ο χρήστης ερωτάται με ένα παράθυρο διαλόγου αν επιθυμεί να αντικαταστήσει τα υπάρχοντα δεδομένα και σε περίπτωση καταφατικής απάντησης, απλά τίθεται ως τιμή στην υπάρχουσα ιδιότητα το νέο SQL ερώτημα. Επίσης, η τρέχουσα κλάση προστίθεται στο πεδίο ορισμού (domain) της ιδιότητας queryString, το version Info της τίθεται ίσο με την τρέχουσα χρονική στιγμή, το είδος (MySQL, PostgreSQL) και το όνομα της βάσης δεδομένων αποθηκεύονται στην ετικέτα (label) αυτής της ιδιότητας και τέλος, ως σχόλιο για την ιδιότητα αυτή τίθεται το όνομα αρχείου της οντολογίας, από την οποία επιλέχθηκε η κλάση που αντιστοιχείται. Αυτές οι προσθήκες πραγματοποιούνται σε ένα αντικείμενο της κλάσης OntModel, το οποίο περιέχει όλες τις πληροφορίες για την οντολογία που έχει φορτωθεί από το χρήστη. Ταυτόχρονα, γεμίζουν τα ειδικά πεδία που περιέχουν τις λεπτομέρειες της αντιστοίχισης, όπως το version Info, η ετικέτα και το σχόλιο της queryString, που αναφέρθηκαν παραπάνω. Ακόμα, καλείται η μέθοδος describeQuery() της κλάσης OntologyPanel, που παίρνει ως όρισμα ένα string (και συγκεκριμένα, το SQL ερώτημα) και περιγράφει περιφραστικά την αντιστοίχιση.

```
                DatatypeProperty odp = ontModel.createDatatypeProperty(basePrefix + "queryString", true);
                ...

                //sets the domain for property "queryString"
                odp.addDomain(selectedClass);
                //set the property value
                selectedClass.setPropertyValue(odp, (RDFNode) (ontModel.createLiteral(queryString,null) ) );
                //pass a timestamp as version info for this property
                Calendar cal=Calendar.getInstance();
                odp.setVersionInfo(cal.getTime().toString());
                //save the database type and the database name in the property's label
                odp.setLabel("PostgreSQL-"+parentPanel.databaseSelected,null);
                //save the ontology file name as the property's comment
                String localName = ontPanel.oldFilePath.substring (ontPanel.oldFilePath.lastIndexOf('\\') + 1);
                odp.setComment(localName,null);


                ontPanel.describeQuery(selectedClass.getLocalName(),queryString);
                ontPanel.txtVersion.setText(odp.getVersionInfo());
                ontPanel.txtDB.setText(odp.getLabel(null));
                ontPanel.txtFile.setText(odp.getComment(null));
```

Πλαίσιο 6.30 Συσχετισμός κλάσης με SQL ερώτημα (QueryPanelPostgres.java)

Ανά πάσα στιγμή, ο χρήστης μπορεί να πατήσει το κουμπί αποθήκευσης (save) για να **αποθηκεύσει** τις υπάρχουσες αντιστοιχίσεις σε ένα .owl αρχείο. Στην πραγματικότητα, αποθηκεύεται το αντικείμενο OntModel, όπως αυτό έχει διαμορφωθεί ως εκείνη τη στιγμή.

```
else if ((source==saveButton)&&(ontModel!=null)){
        JFileChooser chooser=new JFileChooser();
        OWLFileFilter filter=new OWLFileFilter();
        filter.addExtension("owl");
        filter.setDescription("OWL Files");
        chooser.setFileFilter(filter);
        chooser.setDialogTitle("Save these mappings as an OWL file");
```

```
        int returnVal = chooser.showSaveDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
           String saveFile=chooser.getSelectedFile().getPath();
           File f;
           if ((!saveFile.endsWith(".owl")) && (!saveFile.endsWith(".OWL"))) f=new File
(saveFile+".owl");
           else f=new File(saveFile);
              String basePrefix=ontModel.getNsPrefixURI("");
              FileWriter fw=new FileWriter(f);

              RDFWriter st=ontModel.getWriter("RDF/XML-ABBREV");
              st.setProperty("xmlbase",basePrefix);
              st.write(ontModel,fw,basePrefix);
              fw.close();

        }
     }
```

Πλαίσιο 6.31 Αποθήκευση αντιστοιχίσεων σε αρχείο (OntologyPanel.java)

Επίσης, πριν αρχίσει η διαδικασία της αντιστοίχισης, ο χρήστης μπορεί, πατώντας το κουμπί **φόρτωσης** (load) να φορτώσει αντιστοιχίσεις που έχει πραγματοποιήσει παλιότερα και είναι αποθηκευμένες σε ένα αρχείο κατάληξης .owl. Απαραίτητη προϋπόθεση βέβαια, για να συμβεί αυτό, είναι να έχει ανοιχτεί προηγουμένως η οντολογία πάνω στην οποία θα βασιστεί η όλη διαδικασία και επίσης, το αρχείο .owl που περιέχει τις αντιστοιχίσεις να συμφωνεί με τη βασική οντολογία. Ακόμα, με δεξί κλικ πάνω σε κάποια αντιστοίχιση, που απεικονίζεται στη λίστα αντιστοιχίσεων, ο χρήστης μπορεί να την αφαιρέσει από το μοντέλο της οντολογίας ή να αφαιρέσει όλες τις αντιστοιχίσεις που περιέχονται στο αντικείμενο τύπου OntModel. Όλες αυτές οι λειτουργίες πραγματοποιούνται μέσω της μεθόδου actionPerformed() της κλάσης OntologyPanel.

```
else if (source==removeMenuItem){
           String tobeRemoved=(String) mapList.getSelectedValue();
           mapVector.removeElement(mapList.getSelectedValue());
           mapList.setListData(mapVector);
           String basePrefix=ontModel.getNsPrefixURI("");
           String classtoRemove=tobeRemoved.substring(0,tobeRemoved.indexOf('-'));
           OntClass myClass= ontModel.getOntClass(basePrefix+classtoRemove);
           OntProperty myProp = ontModel.getOntProperty(basePrefix+"queryString");
           Property myProp2 = ontModel.getProperty(basePrefix,"queryString");
           StmtIterator i = ontModel.listStatements(myClass, myProp2, (RDFNode)null);
           Vector tmp=new Vector();
           while (i.hasNext()){
              Statement s = i.nextStatement();
              tmp.addElement(s.getSubject());
           }
           for (int j=0;j<tmp.size();j++){
              Resource res=(Resource)tmp.elementAt(j);
              res.removeAll(myProp);
              myProp.removeDomain(res);
           }
           //if there are no more mappings, remove completely the property "queryString"
           if (mapVector.size()==0) {
              myProp.remove();
              txtVersion.setText(null);
              txtDB.setText(null);
```

```
            txtFile.setText(null);
        }

    }
```

Πλαίσιο 6.32 Αφαίρεση αντιστοίχισης (OntologyPanel.java)

Τέλος, αξίζει να σημειωθεί ότι δεν επιτρέπεται η αποθήκευση στο ίδιο αρχείο (ισοδύναμα, μέσα στο ίδιο μοντέλο OntModel) αντιστοιχίσεων που αναφέρονται σε διαφορετικές οντολογίες ή σε διαφορετικές βάσεις δεδομένων. Αν επιχειρηθεί κάτι τέτοιο, η εφαρμογή θα προειδοποιήσει το χρήστη με κατάλληλα μηνύματα.

```
//check whether the mappings loaded refer to the ontology opened
                DatatypeProperty dp = loadedOntModel.getDatatypeProperty(basePrefix +
"queryString");
                if (dp!=null){
                   String fullName=dp.getComment(null);
                   String localName=fullName.substring(fullName.lastIndexOf('\\')+1);
                   if (!localName.equals(oldLocalName)){
                      acceptLoading=false;
                      JOptionPane.showMessageDialog(null,"The file you chose does not agree
with the ontology opened\n" +
                            "Please load another file or open another basic ontology",
                            "Not Proper Mapping File",
                            JOptionPane.ERROR_MESSAGE);
                   }

                }
```

Πλαίσιο 6.33 Έλεγχος συμβατότητας του αρχείου αντιστοιχίσεων με την ανοικτή οντολογία (OntologyPanel.java)

6.4 ΕΚΤΕΛΕΣΗ ΣΗΜΑΣΙΟΛΟΓΙΚΟΥ ΕΡΩΤΗΜΑΤΟΣ (RDQL QUERY)

Ως επαλήθευση της διαδικασία αντιστοίχισης που έχει προηγηθεί, ο χρήστης μπορεί να εισάγει ένα σημασιολογικό ερώτημα (RDQL Query) σε ειδικό πεδίο, να πατήσει το κουμπί εκτέλεσης και να ανακτήσει τα ονόματα των κλάσεων που δίνουν απάντηση στο ερώτημα, αλλά και τα δεδομένα που έχουν αντιστοιχηθεί σε αυτές. Αυτές τις λειτουργίες επιτελεί η κλάση OntQueryPanel και απαραίτητη προϋπόθεση για να ληφθούν αποτελέσματα είναι η ύπαρξη ανοικτής οντολογίας και η σύνδεση με την κατάλληλη βάση δεδομένων.

Πρώτα, εκτελείται το RDQL ερώτημα και επιστρέφονται τα αποτελέσματά του σε μορφή αντικειμένων ResultBinding, τα οποία η εφαρμογή αποθηκεύει σε διανύσματα διανυσμάτων τα οποία έχουν διάσταση όσες και οι μεταβλητές του ερωτήματος. Στη συνέχεια, γίνεται έλεγχος αν κάποια από τις κλάσεις που έχουν ήδη αντιστοιχηθεί σε κάποια δεδομένα περιλαμβάνεται μεταξύ των αποτελεσμάτων του σημασιολογικού ερωτήματος και αν ναι, αποθηκεύεται στο διάνυσμα resourcesFound.

```
            Query query = new Query(ontQuery);
```

```
            // Need to set the source if the query does not.
            query.setSource(ontPanel.ontModel);
            QueryExecution qe = new QueryEngine(query) ;
            QueryResults results = qe.exec() ;
            Iterator iter = results;
            Vector v=new Vector();
            Vector vres=new Vector();
            while (iter.hasNext()) {
               ResultBinding res = (ResultBinding)iter.next() ;
               Iterator nam=res.names();
               //find the number of variables in the query and store it to v
               while(nam.hasNext()){
                  v.addElement(nam.next().toString()) ;
               }

               //store the results of the RDQL query in a vector
               for (int i=0;i<v.size();i++){
                  Vector temp=new Vector();
                  temp.addElement(((RDFNode)res.get(v.elementAt(i).toString())).toString());
                  vres.add(temp);
               }



            }
            results.close() ;
            Iterator i=dp.listDomain();

            //add to a vector all the results of the RDQL query that are classes and are already
mapped
            while (i.hasNext()){
               Resource current =(Resource)i.next();
               for (int j=0;j<vres.size();j++){
                  Vector cur=(Vector)vres.elementAt(j);
                  if
((cur.contains(current.getURI()))&&(!resourcesFound.contains(current.getURI())))
                     resourcesFound.addElement(current.getURI());
               }
            }
```

Πλαίσιο 6.34 Εκτέλεση RDQL ερωτήματος και εύρεση των κλάσεων που απαντούν σε αυτό και έχουν αντιστοιχηθεί σε κάποια δεδομένα (OntQueryPanel.java)

Επόμενο βήμα είναι η **ανάκτηση των δεδομένων** για τις κλάσεις που βρίσκονται στο εν λόγω διάνυσμα. Πριν γίνει αυτό, πρέπει να ελεγχθεί ότι ο χρήστης είναι ήδη συνδεδεμένος στη βάση δεδομένων στην οποία αναφέρονται οι αντιστοιχίσεις. Αν δεν ισχύει κάτι τέτοιο, η εφαρμογή εμφανίζει μηνύματα λάθους και ο χρήστης καλείται να συνδεθεί στη σωστή βάση. Το είδος καθώς και το όνομα της βάσης δεδομένων στην οποία αναφέρονται οι αντιστοιχίσεις είναι αποθηκευμένα, όπως έχει αναφερθεί στην ετικέτα της ιδιότητας queryString και με αυτόν τον τρόπο, ανακτώνται εύκολα.

```
         String basePrefix=ontPanel.ontModel.getNsPrefixURI("");
        DatatypeProperty dp = ontPanel.ontModel.getDatatypeProperty(basePrefix +
"queryString");
         if (dp!=null){
            ontQuery=queryField.getText();
            String base=dp.getLabel(null);
            ...
```

```
            //check whether the mappings refer to the database connected
            boolean agreement=false;
            if (base.contains("PostgreSQL")&&(pgPanel!=null)){
               if (base.equals("PostgreSQL-"+pgPanel.databaseSelected)) agreement=true;
               else  JOptionPane.showMessageDialog(null,"Make sure you are connected to the
correct PostgreSQL database",
                     "Wrong Connection",
                     JOptionPane.ERROR_MESSAGE);
            } else if (base.contains("MySQL")&&(mySQLPanel!=null)){
               if (base.equals("MySQL-"+mySQLPanel.databaseSelected)) agreement=true;
               else  JOptionPane.showMessageDialog(null,"Make sure you are connected to the
correct MySQL database",
                     "Wrong Connection",
                     JOptionPane.ERROR_MESSAGE);
            }
```

Πλαίσιο 6.35 Έλεγχος αν οι αντιστοιχίσεις αναφέρονται στη βάση δεδομένων με την οποία η εφαρμογή είναι συνδεδεμένη (OntQueryPanel.java)

Όταν έχει εξασφαλιστεί ότι ο χρήστης έχει συνδεθεί στη βάση στην οποία αναφέρονται οι αντιστοιχήσεις, για κάθε κλάση που περιέχεται στο διάνυσμα resourcesFound, **ανακτάται η τιμή της ιδιότητας queryString**, δηλαδή το συσχετισμένο SQL ερώτημα, το οποίο και εκτελείται. Τα αποτελέσματα κάθε SQL ερωτήματος οργανώνονται σε πίνακες και όλοι αυτοί οι πίνακες μαζί με το όνομα της αντίστοιχης κλάσης της οντολογίας παρουσιάζονται σε ξεχωριστό παράθυρο.

```
if (agreement){
            //for each RDQL query result retrieve the database query,execute it and show the
results in a new frame
            for (int j=0;j<resourcesFound.size();j++){
               //retrieve the database query which is an underlying result of the RDQL query
               DefaultTableModel dm = new DefaultTableModel();
               JTable resultsTable=new JTable(dm);
               OntResource resource = ontPanel.ontModel.getOntResource
(resourcesFound.elementAt(j).toString());
               String dbQuery=resource.getPropertyValue(dp).toString();

                  //execute the database query
                  ResultSet ret = null;
                  if ((base.contains("PostgreSQL-"))&&(pgPanel!=null)){
                     …//execute the SQL query
                     }
                  } else if ((base.contains("MySQL-"))&&(mySQLPanel!=null)){
                      … //execute the SQL query
                  }
                  ResultSetMetaData rmeta=ret.getMetaData();
                  int numColumns=rmeta.getColumnCount();

                  //fill the table
                  Vector colNames=new Vector();
                  for(int k=1;k<=numColumns;k++) {
                     colNames.addElement(rmeta.getColumnName(k));
                  }

                  Vector rows=new Vector();
                  while(ret.next()){
                     Vector newRow=new Vector();
                     for(int k=1;k<=numColumns;k++)
                        newRow.addElement(ret.getString(k));
```

```
                    rows.addElement(newRow);
                }
                ret.close();
                dm.setDataVector(null,colNames);
                for (int k=0;k<rows.size();k++){
                    Vector currentRow=(Vector)rows.elementAt(k);
                    dm.addRow(currentRow);

                }
                resultsTable.setModel(dm);
                tables.addElement(resultsTable);

            }
        }
```

Πλαίσιο 6.36 Ανάκτηση δεδομένων που έχουν αντιστοιχηθεί στις κλάσεις που απαντούν στο RDQL ερώτημα (OntQueryPanel.java)

7 ΣΥΜΠΕΡΑΣΜΑΤΑ - ΕΠΕΚΤΑΣΕΙΣ

Κατά την εκπόνηση της παρούσας διπλωματικής εργασίας εξήχθησαν τα παρακάτω συμπεράσματα:

Η διαδικασία της αντιστοίχισης δεδομένων από μία βάση δεδομένων με μία κλάση μιας οντολογίας είναι πλήρως **δυναμική**. Αυτό σημαίνει ότι δε χρειάζεται να είναι γνωστή εκ των προτέρων ούτε η βάση δεδομένων ούτε η οντολογία που θα χρησιμοποιηθούν. Βέβαια, για να έχει κάποιο νόημα η αντιστοίχιση θα πρέπει τα δεδομένα της βάσης και οι κλάσεις της οντολογίας να συμβαδίζουν σημασιολογικά, με την ευθύνη αυτή να εναπόκειται αποκλειστικά στον εκάστοτε χρήστη της εφαρμογής.

Η αντιστοίχιση επιτεύχθηκε με την προσθήκη επιπλέον πληροφορίας στο αρχείο της οντολογίας. Η πληροφορία αυτή είναι ένα **SQL ερώτημα**, δηλαδή ένα αλφαριθμητικό (string) του οποίου το μέγεθος αποθήκευσης κρίνεται αμελητέο σε σύγκριση με το μέγεθος ενός συνηθισμένου αρχείου οντολογίας. Επομένως, η απεικόνιση κάποιων δεδομένων σε κάποια οντολογική κλάση επιτυγχάνεται με πολύ μικρό κόστος. Ακόμα, η χρήση του SQL ερωτήματος για την αντιστοίχιση εξασφαλίζει και τη διαρκή ενημέρωση των δεδομένων που αντιστοιχούν σε κάποια οντολογική κλάση, μετά από κάποια μετατροπή, προσθήκη ή αφαίρεση δεδομένων από τη βάση. Αυτό βέβαια ίσως να μην ισχύει, αν αλλάξει το σχήμα της βάσης δεδομένων με τέτοιο τρόπο, ώστε το SQL ερώτημα να μην είναι πλέον έγκυρο. Επίσης, πιθανή μετατροπή των περιεχομένων της βάσης μπορεί να καταστήσει ήδη υπάρχουσες αντιστοιχίσεις σημασιολογικά λανθασμένες, καθώς αυτή η μετατροπή μπορεί να παραβίασε έναν ή περισσότερους σημασιολογικούς ελέγχους.

Η Γλώσσα Οντολογιών Ιστού **OWL**, η οποία προς το παρόν είναι η επικρατέστερη γλώσσα συγγραφής οντολογιών, προσφέρει αρκετά δομικά στοιχεία τα οποία χρησιμοποιήθηκαν στην παρούσα εργασία με τέτοιο τρόπο ώστε να αποθηκευθούν και άλλες συμπληρωματικές πληροφορίες, όπως το όνομα και ο τύπος της βάσης δεδομένων από την οποία προέρχονται τα δεδομένα. Στο μέλλον, είναι πιθανό η γλώσσα OWL να αποκτήσει μεγαλύτερη ευελιξία και περισσότερες δυνατότητες, έτσι ώστε να καταστεί δυνατή η αποθήκευση στο ίδιο αρχείο αντιστοιχίσεων από διαφορετικές βάσεις δεδομένων με πλήρη και σαφή διάκριση μεταξύ τους.

Η **διαδικασία απεικόνισης** άλλων στοιχείων ή δομικών στοιχείων μιας οντολογίας, όπως οι **ιδιότητες**, σε μία βάση δεδομένων δεν είναι εξίσου προφανής με τη διαδικασία απεικόνισης κλάσεων. Στην απλουστευμένη περίπτωση που κάθε κλάση μιας οντολογίας απεικονίζεται σε έναν ολόκληρο πίνακα μιας βάσης δεδομένων, ή έστω σε μερικές στήλες ενός πίνακα, μπορεί να θεωρηθεί ότι μια ιδιότητα απεικονίζεται ως μια στήλη ενός πίνακα ή ως μια αναφορά ξένου κλειδιού. Στη γενικότερη όμως περίπτωση που εξετάστηκε στην παρούσα διπλωματική εργασία, δεν μπορεί να γίνει μια τέτοια υπόθεση. Σίγουρα, το θέμα της αντιστοίχισης ιδιοτήτων είναι από τα πλέον σημαντικά και θεμελιώδη ζητήματα που χρειάζονται να επιλυθούν μελλοντικά, ώστε να υπάρξει μια ολοκληρωμένη λύση του προβλήματος της απεικόνισης μιας οντολογίας σε μια βάση δεδομένων.

Το ζήτημα της αντιστοίχισης μιας οντολογίας σε μια βάση δεδομένων αναμένεται να παίξει πολύ σημαντικό ρόλο για τις μελλοντικές εφαρμογές του Σημασιολογικού Ιστού. Ως ελάχιστη ένδειξη για το παραπάνω, η εφαρμογή η οποία αναπτύχθηκε δίνει τη δυνατότητα εκτέλεσης ενός **σημασιολογικού ερωτήματος** και επιστρέφει **πραγματικά δεδομένα**, τα οποία απαντούν σε αυτό. Μια γενίκευση αυτής της εφαρμογής θα μπορούσε να αντικαταστήσει τις τρέχουσες μηχανές αναζήτησης του Διαδικτύου, δεχόμενη ως είσοδο ένα γενικό ερώτημα που δε θα βασίζεται σε λέξεις – κλειδιά, αλλά σε έννοιες και θα επιστρέφει συγκεκριμένα έγγραφα που θα σχετίζονται εννοιολογικά με το ερώτημα.

8 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] G. Antoniou, F. van Harmelen: A Semantic Web Primer, The MIT Press

[2] M.Smith, C.Welty, and D. McGuinness, eds. "OWL Web Ontology Language: Guide", February 10, 2004, http://www.w3.org/TR/owl-guide/

[3] T. Berners-Lee, "Semantic Web Road Map", September 1998, http://www.w3.org/DesignIssues/Semantic.html

[4] "Semantic Web", Wikipedia, http://en.wikipedia.org

[5] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web", Scientific American, May 2001.

[6] Ελληνικό Γραφείο του W3C, http://www.w3c.gr

[7] "The Semantic Web: An Interview with Tim Berners-Lee", Consortium Standards Bulletin, June 2005, http://www.consortiuminfo.org/bulletins

[8] T. Gruber, "What is an Ontology?", http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

[9] J. Heflin, "OWL Web Ontology Language: Use Cases and Requirements", February 10, 2004. http://www.w3.org/TR/webont-req/

[10] H. Boley, "The Semantic Web in Ten Passages", September 23, 2004, http://www.dfki.uni-kl.de/~boley/sw10pass/sw10pass-en.htm

[11] O. Lassila, R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", February 22, 1999, http://www.w3.org/TR/REC-rdf-syntax

[12] D. Brickley, R.V. Guha, "RDF Vocabulary Description Language: RDF Schema", November 12, 2002, http://www.w3.org/TR/rdf-schema

[13] S. Bechhofer, F. van Harmelen, J. Hendler et al., "OWL Web Ontology Language: Reference", February 10, 2004, http://www.w3.org/TR/owl-ref/

[14] D. McGuiness, F. van Harmelen, "OWL Web Ontology Language: Overview", February 10, 2004, http://www.w3.org/TR/owl-features/

[15] A. Seaborne, Jena Tutorial, "A Programmer's Introduction to RDQL", April 2002, Updated February 2004, http://jena.sourceforge.net/tutorial/RDQL/index.html

[16] R. Fikes, P. Hayes, I. Horrocks, "OWL-QL – A Language for Deductive Query Answering on the Semantic Web", http://ksl.stanford.edu/projects/owl-ql/

[17] RQL v2.1 User Manual, http://139.91.183.30:9090/RDF/RQL/Manual.html

[18] Jena: A Semantic Web Framework, http://jena.sourceforge.net/

[19] Protégé OWL Plugin, http://protege.stanford.edu/plugins/owl/

[20] Protégé User's Guide, http://protege.stanford.edu

[21] H. Knublauch, R. Fergerson, N. Noy et al., "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications", http://protege.stanford.edu/plugins/owl/documentation.html

[22] OntoWeb: Ontology-based Information Exchange for Knowledge Management and Electronic Commerce IST Project IST-2000-29243, 31 May 2002, http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf

[23] OILEd, http://oiled.man.ac.uk

[24] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens, "OILEd: a Reason-able Ontology Editor for the Semantic Web", Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, p 396-408, 2001.

[25] I. Horrocks, U. Sattler, S. Tobies, "Practical reasoning for expressive description logics", 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99) (LNAI, Springer-Verlag, 1999), P. 161-180.

[26] Ontoprise GmbH, http://www.ontoprise.de

[27] Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke, "OntoEdit: Collaborative Ontology Engineering for the Semantic Web", Proceedings of the International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.

[28] Y. Sure, S. Staab, J. Angele, D. Wenke and A. Maedche, "OntoEdit: Guiding Ontology Development by Methodology and Inferencing", submitted 2002, http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/2002_odbase_ontoedit.pdf

[29] S. Handschuh. "Ontoplugins – a flexible component framework", Technical report, University of Karlsruhe, May 2001, http://www.aifb.uni-karlsruhe.de/WBS/sha/papers/ontoplugin.pdf

[30] Ontolingua Server, http://ontolingua.stanford.edu/

[31] A. Farquhar, R. Fikes, J. Rice, "The Ontolingua Server: A Tool for Collaborative Ontology Construction", Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, (Banff, Alberta, Canada 1996) 44.1-44.19.

[32] WebOnto, http://kmi.open.ac.uk/projects/webonto

[33] R. Elmasri, S.B.Navathe (2000), "Θεμελιώδεις Αρχές Συστημάτων Βάσεων Δεδομένων (Τόμος Α' και Β')", Μετάφραση: Μιχάλης Χατζόπουλος, Εκδόσεις Δίαυλος

[34] Τίμος Σελλής, Προχωρημένα Θέματα Βάσεων Δεδομένων, Εθνικό Μετσόβιο Πολυτεχνείο

[35] MySQL Documentation, http://dev.mysql.com

[36] MySQL Reference Manual, Spatial Extensions in MySQL, http://dev.mysql.com/doc/mysql/en/spatial-extensions-in-mysql.html

[37] The PostgreSQL Global Development Group, PostgreSQL 8.0, PostgreSQL documentation, http://www.postgresql.org/docs/

[38] J. de Bruijn, A. Polleres, “Towards an Ontology Mapping Specification Language for the Semantic Web”, DERI Technical Report 2004-06-30 June 2004, http://www.deri.at/publications/techpapers/documents/DERI-TR-2004-06-30.pdf

[39] Y. Kalfoglou and M. Schorlemmer, “Ontology Mapping: The State of the Art”, http://eprints.ecs.soton.ac.uk/10519/01/ker02-ontomap.pdf

[40] Adapting Ontologies From Relational Databases – The WebAI Wiki, http://www.mindswap.org

[41] KAON, http://kaon.semanticweb.org

[42] KAON Reverse, http://kaon.semanticweb.org/alphaworld/reverse/view

[43] KAON Reverse Documentation, FZI, AIFB, University of Karlsruhe, Germany

[44] C. Bizer, D2R MAP – Map Language Specification, FU Berlin, http://www.wiwiss.fu-berlin.de/suhl/bizer/d2rmap/D2Rmap.htm

[45] C. Bizer, D2R MAP - A Database to RDF Mapping Language, Freie Universität Berlin, http://www2003.org/cdrom/papers/poster/p004/p4-bizer.html

[46] RuleML**,** http://www.dfki.uni-kl.de/ruleml/

[47] B. Omelayenko. “RDFT: A Mapping Meta-Ontology for Business Integration”, ECAI2002(Lyon,2002), http://www.cs.vu.nl/~borys/papers/rdft4ktsw02.pdf

[48] The Unicorn System, http://unicorn.com/index.htm

[49] Unicorn Workbench, http://unicorn.com/products/unicornsystem/workbench.htm

[50] J. de Bruijn, “Semantic Integration of Disparate Data Sources in the COG Project”, Digital Enterprise Research Institute University of Innsbruck, http://www.cogproject.org/publications/COG-ICEIS2004.pdf

[51] J. Barrasa, O. Corcho, A. Gómez-Pérez, “R2O an Extensible and Semantically Based Database-to-ontology Mapping Language”, http://www.cs.man.ac.uk/~ocorcho/documents/SWDB2004_BarrasaEtAl.pdf

[52] M.K. Bergman, “The Deep Web: Surfacing hidden value”, Journal of Electronic Publishing, University of Michigan, September 2001, http://www.press.umich.edu/jep/07-01/bergman.html

[53] L. Stojanovic, N. Stojanovic, R. Volz, “Migrating data-intensive Web Sites into the Semantic Web”, Symposium on Applied Computing, Madrid, Spain, March 2002

[54] N. Stojanovic, L. Stojanovic, R. Volz, “A Reverse Engineering Approach for Migrating Dataintensive Web Sites to the Semantic Web”, Intelligent Information Processing, August 25-30, 2002, Montreal, Canada (Part of the IFIP World Computer Congress WCC2002)

[55] S. Handschuh, S. Staab, R. Volz, “On deep annotation”, 12th International World Wide Web Conference, Budapest, May 2003

[56] J. Madhavan, P. Bernstein, E. Rahm, “Generic Schema Matching with Cupid”, Proceedings of the 27th VLDB Conference. Roma, Italy, 2001.