ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

DataBase supported Reasoning System (DBRS)

ΣΥΣΤΗΜΑ ΑΠΟΘΗΚΕΥΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΣΗΜΑΣΙΟΛΟΓΙΚΩΝ ΣΧΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΛΙΑΓΟΥΡΗ ΓΙΑΝΝΗ
ΦΑΡΜΑΚΑΚΗ ΤΡΥΦΩΝΑ

Επιβλέπων: Ιωάννης Βασιλείου

Καθηγητής Ε.Μ.Π.

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

DataBase supported Reasoning System (DBRS)

ΣΥΣΤΗΜΑ ΑΠΟΘΗΚΕΥΣΗΣ ΚΑΙ ΔΙΑΧΕΙΡΙΣΗΣ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΣΗΜΑΣΙΟΛΟΓΙΚΩΝ ΣΧΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΛΙΑΓΟΥΡΗ ΓΙΑΝΝΗ
ΦΑΡΜΑΚΑΚΗ ΤΡΥΦΩΝΑ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τη 10η Ιουλίου 2008.

.....	.....	.....
Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Επικ. Καθηγητής Ε.Μ.Π.

.....

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

.....

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

Περίληψη

Στόχος της διπλωματικής εργασίας είναι η κατασκευή ενός συστήματος αποθήκευσης και διαχείρισης οντολογιών η εκφραστικότητα των οποίων ανήκει στο τμήμα $SHOIN^{(D)}$ της Περιγραφικής Λογικής. Κεντρική ιδέα της υλοποίησης αποτελεί ο συνδυασμός ενός σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (RDBMS) με μια Μηχανή Συλλογιστικής Ανάλυσης (Reasoner). Το σύστημα θα πρέπει να χαρακτηρίζεται από «ευελιξία» και επεκτασιμότητα. Με άλλα λόγια, θα πρέπει να είναι όσο το δυνατόν ανεξάρτητο από συγκεκριμένα εργαλεία και τεχνολογίες.

Επικεντρωνόμαστε στο πρόβλημα της ολοκληρωμένης και αποδοτικής διαχείρισης πολύ μεγάλου όγκου σημασιολογικής πληροφορίας (εκατομμύρια RDF τριάδες), το οποίο συναντάμε συχνά σε πραγματικές εφαρμογές (real-world applications). Σε αντίθεση με την πλειονότητα των υπαρχουσών πλατφόρμων, στοχεύουμε στην κατασκευή ενός συστήματος που όχι μόνο θα οργανώνει τα δεδομένα μίας οντολογίας θεωρώντας τα ως «ορθά», αλλά θα επιτελεί και όλες εκείνες τις απαραίτητες διεργασίες για τον έλεγχο της συνέπειας (consistency) της προς αποθήκευση γνώσης, αλλά και την πλήρη (sound & complete) αποτίμηση ερωτημάτων σχετικά με αυτή.

Ιδιαίτερη έμφαση δίνουμε στην κατασκευή ενός σχήματος βάσης δεδομένων που αφενός θα οργανώνει την πληροφορία χωρίς απώλειες (πληρότητα), αφετέρου θα επιτρέπει την γρήγορη αποτίμηση ερωτημάτων (απόδοση). Όσον αφορά στο τελευταίο, χρησιμοποιούμε μια σειρά από τεχνικές βελτιστοποίησης, εμπλουτίζοντας παράλληλα το σύστημα με μεθόδους που εκτελούν ένα είδος συλλογιστικής ανάλυσης στη βάση δεδομένων. Υπό αυτήν την έννοια, η τελευταία καθίσταται περισσότερο «αυτόνομη» σχετικά με ερωτήματα «ανοιχτού κόσμου» (open world), ενώ ταυτόχρονα υποβοηθά και τον Reasoner στην εξαγωγή υπονοούμενων (implicit) σχέσεων, στέλνοντας σε αυτόν μόνο τις περιπτώσεις που δεν μπορεί να αποκλείσει και για τις οποίες η εκτέλεση του Tableau αλγόριθμου είναι αναπόφευκτη.

Το σύστημα που αναπτύχθηκε με γλώσσα προγραμματισμού Java φέρει το όνομα «Database supported Reasoning System» (DBRS) και χρησιμοποιεί, στην παρούσα έκδοσή του, τον Reasoner Pellet και το DBMS PostgreSQL.

Λέξεις Κλειδιά: Σημασιολογικός Ιστός, Μεταδεδομένα, Οντολογία, Περιγραφική Λογική, Βάση Γνώσης, RDF/S, OWL, Συλλογιστική Ανάλυση, Tableau αλγόριθμος, Ψευδομοντέλα, Σχήματα Ετικετών, Μοντέλα βάσης δεδομένων για αποθήκευση σημασιολογικών σχημάτων, Συστήματα Διαχείρισης Οντολογιών.

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The aim of this work was the development of a scalable system able to store, manage and reason over very large ontologies (i.e. millions of RDF triples) of $SHOIN^{(D)}$ expressivity (in terms of Description Logic), which is the actual challenge in real-world semantic web applications.

Our main objective was to combine a Relational Database Management System with a Reasoner and exploit the benefits of bringing together current database technology and DL inference procedures. In contrast to the majority of the existent similar platforms, we focused on developing a system that does not only store the metadata considering them to be correct by default, but also performs light and heavy reasoning procedures both in the loading and querying phase.

Most of our effort was put on designing a data model that organizes the semantic information of $SHOIN^{(D)}$ expressivity without losses, while allowing efficient and fast querying. To accomplish these tasks, we used many optimization techniques, such as a Labeling Scheme for representing hierarchical structures and methods for storing and merging pseudomodels into the database. Through these techniques, the database becomes relatively more "independent" to open world queries and uses the Reasoner only for a limited set of queries that cannot be evaluated without the Tableau algorithm, achieving, in most of the cases, a performance far better than the stand-alone Reasoner.

The current version of the system named "Database supported Reasoning System" (DBRS) is developed in Java. It uses the well known Reasoner Pellet and the open source PostgreSQL DBMS. However, it is designed to be flexible and extensible, meaning that both the Reasoner and the DBMS can easily be replaced simply by changing the corresponding wrappers.

Keywords: Semantic Web, Metadata, Ontology, Description Logic, Knowledge Base, RDF/S, OWL, Reasoning, Tableau algorithm, Pseudomodels, Labeling Schemes, Database representations for storing semantics, Ontology Management Systems.

Η σελίδα αυτή είναι σκόπιμα λευκή.

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων (ΕΣΒΓΔ) του Εθνικού Μετσόβιου Πολυτεχνείου και αποτέλεσε μια πολύ καλή αφορμή για να ασχοληθούμε με ορισμένα εξαιρετικά ενδιαφέροντα προβλήματα που θέτει η προσπάθεια ανάπτυξης του Σημασιολογικού Ιστού. Στο σημείο αυτό, θα θέλαμε αρχικά να ευχαριστήσουμε τους καθηγητές Ι. Βασιλείου και Τ. Σελλή που μας έδωσαν την ευκαιρία να ασχοληθούμε με τα συγκεκριμένα θέματα, ενώ ιδιαιτέρως ευχαριστούμε τον υποψήφιο διδάκτορα Παναγιώτη Μπούρο για το ενδιαφέρον που έδειξε, τις πολύτιμες παρατηρήσεις του και την γενικώς πολύ καλή συνεργασία που είχαμε καθόλη τη διάρκεια της διπλωματικής.

Πίνακας περιεχομένων

1

Εισαγωγή

Ο όρος Σημασιολογικός Ιστός (Semantic Web) πρωτοεμφανίστηκε τη δεκαετία του 1990 αναφερόμενος στο όραμα εμπλουτισμού του (υπάρχοντος) Συντακτικού Ιστού (Syntactic Web) με σημασιολογική πληροφορία (semantics), δηλαδή με πληροφορία που αναφέρεται στα ίδια τα δεδομένα που υπάρχουν, οργανώνονται στο διαδίκτυο, παρουσιάζονται και μεταφέρονται μέσω αυτού [BLproposal]. Η σημασιολογική πληροφορία, ή αλλιώς μεταδεδομένα (metadata), στοχεύει στο να καταστήσει τους πόρους του διαδικτύου (web resources) προσπελάσιμους από αυτοματοποιημένες διαδικασίες, δηλαδή από αλγοριθμικές διαδικασίες (software) που δεν απαιτούν καθόλου ή τουλάχιστον περιορίζουν σε ένα βαθμό την «ανθρώπινη παρέμβαση». Με απλά λόγια, ο Σημασιολογικός Ιστός (ΣΙ) είναι ένας «πιο έξυπνος» ιστός όπου οι υπολογιστές θα μπορούν να «καταλαβαίνουν» την πληροφορία, να την οργανώνουν καλύτερα και συνεπώς να επιτελούν περισσότερα και πολυπλοκότερα καθήκοντα.

Η προσπάθεια προσθήκης των μεταδεδομένων στον Παγκόσμιο (συντακτικό) Ιστό (World Wide Web - WWW) γεννά μια σειρά από προκλήσεις που τα τελευταία χρόνια απασχολούν ιδιαιτέρως την επιστημονική έρευνα. Τα ζητήματα που ανακύπτουν είναι πολλά. Σε πρώτο βήμα, τα σημασιολογικά σχήματα πρέπει να περιγραφούν αυστηρώς με χρήση κατάλληλων τυπικών φορμαλισμών (να μετατραπούν δηλαδή σε γνώση), δεδομένου ότι η αυστηρότητα αυτή είναι που θα

εξασφαλίσει και τη δυνατότητα αυτόματης ανάλυσής τους από υπολογιστές. Στη συνέχεια, οι συγκεκριμένοι τυπικοί φορμαλισμοί πρέπει, με τη σειρά τους, να συνοδεύονται κι από τυπικές γλώσσες που αφενός θα μπορούν να «διαβαστούν» από υπολογιστές, αφετέρου θα διαθέτουν σύνταξη συμβατή με τα ισχύοντα πρότυπα του Παγκόσμιου Ιστού (π.χ. XML), έτσι ώστε να μπορούν εύκολα να ενσωματωθούν σε αυτά ή απλώς να συνδυαστούν μαζί τους.

Σε άμεση σχέση με τα προηγούμενα, ένα εξίσου κομβικό ζήτημα (και βασική προϋπόθεση για την περαιτέρω ανάπτυξη του ΣΙ) είναι η εύρεση αλγορίθμων που θα μπορούν να αναλύσουν[1] σημασιολογική πληροφορία πλούσιας εκφραστικότητας με όσο το δυνατόν αποδοτικότερο τρόπο. Ο λόγος για τον οποίο η σχετική έρευνα επικεντρώνεται στην αυξημένη απόδοση είναι προφανής. Οι διαδικασίες ανάλυσης πρέπει, αν λάβουμε υπόψη μας και την χρονική καθυστέρηση της μεταφοράς των δεδομένων πάνω στα πρωτόκολλα του διαδικτύου, να αποκρίνονται σε επιτρεπτούς για τον χρήστη χρόνους. Από την άλλη, ερχόμενοι στο δεύτερο σκέλος, οι συλλογιστικοί αλγόριθμοι πρέπει να μπορούν να αναλύουν πλήρως (sound & complete) γνώση μεγάλης εκφραστικότητας, έτσι ώστε να είναι σε θέση να χειριστούν όλα τα χαρακτηριστικά και τις (ενδεχομένως περίπλοκες) σχέσεις μεταξύ των δεδομένων του Παγκόσμιου Ιστού, όπως αυτές υφίστανται τώρα.

Ένα επίσης σημαντικό πρόβλημα που αναδύεται από την εμπειρία των πρώτων πραγματικών εφαρμογών (real-world applications) του ΣΙ είναι η ανάγκη ολοκληρωμένης διαχείρισης μεταδεδομένων πολύ μεγάλου όγκου. Στις πραγματικές εφαρμογές, ο όγκος (και η πολυπλοκότητα) των μεταδεδομένων τείνει να αυξάνεται, με αποτέλεσμα την αδυναμία συλλογιστικής ανάλυσής τους, λόγω αυξημένου κόστους σε κατανάλωση κύριας μνήμης (main memory)[2], ακόμα όμως και την αδυναμία, σε ορισμένες περιπτώσεις, απλής φόρτωσής τους (loading) σε αυτήν. Αν

---

[1] Η λειτουργία των αλγορίθμων συλλογιστικής ανάλυσης (όπως ονομάζονται) είναι διττή. Από τη μία, στοχεύουν στον έλεγχο της συνέπειας (consistency) των σημασιολογικών σχέσεων, ενώ από την άλλη, στην εξαγωγή νέας πληροφορίας, δηλαδή νέων σχέσεων και χαρακτηριστικών, που δε δηλώνεται ρητά στο αρχικό σύνολο γνώσης, αλλά προκύπτει από αυτό μέσω μιας λογικής επαγωγής (logical inference).

[2] Οι συλλογιστικοί αλγόριθμοι λειτουργούν αποκλειστικώς στην κύρια μνήμη.

μάλιστα στο τελευταίο προσθέσουμε και το γεγονός ότι οι σημασιολογικές σχέσεις είναι μέχρι στιγμής πολύ δύσκολο να κατατμηθούν και να αναλυθούν επιμέρους, αντιλαμβανόμαστε εύκολα το λόγο για τον οποίο η επιστημονική έρευνα προσανατολίζεται ήδη στην ανάπτυξη τεχνικών που θα εκμεταλλεύονται μηχανισμούς δευτερεύουσας μνήμης (second storage mechanisms), έτσι ώστε να ξεπεραστούν τα όποια απαγορευτικά όρια παρουσιάζονται.

1.1 Αντικείμενο διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία στοχεύει στην κατασκευή ενός συστήματος ολοκληρωμένης διαχείρισης σημασιολογικών σχημάτων με πολύ μεγάλο όγκο (εκατομμύρια RDF τριάδες) και με εκφραστικότητα που ανήκει στο τμήμα $SHOIN^{(D)}$ της Περιγραφικής Λογικής (Description Logic) [BCG+03].

Κεντρική ιδέα της υλοποίησης αποτελεί ο συνδυασμός ενός σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (RDBMS) με μια Μηχανή Συλλογιστικής Ανάλυσης (Reasoner). Η πρώτη λειτουργική μονάδα (RDBMS) χρησιμεύει στην οργάνωση και μόνιμη αποθήκευση της σημασιολογίας σε δευτερεύουσα μνήμη (βάση δεδομένων), με τη δεύτερη (Reasoner) να είναι απαραίτητη για την εκτέλεση των επαγωγικών διαδικασιών (inference procedures) που προϋποθέτει η ολοκληρωμένη διαχείριση των σημασιολογικών σχημάτων. Απώτερος σκοπός είναι να ξεπεραστεί ένα από τα βασικά προβλήματα που περιγράψαμε στην εισαγωγή του κεφαλαίου και το οποίο σχετίζεται με την άρση των απαγορευτικών ορίων που θέτει το περιορισμένο συνήθως μέγεθος της κύριας μνήμης, όσον αφορά στη συλλογιστική ανάλυση, αλλά και στη γενικότερη διαχείριση σημασιολογικών σχημάτων μεγάλου όγκου και πλούσιας εκφραστικότητας.

Η εργασία μας επικεντρώνεται σε δύο βασικούς άξονες. Ο πρώτος σχετίζεται, όπως προαναφέρθηκε, με την ολοκληρωμένη διαχείριση σημασιολογικής πληροφορίας εκφραστικότητας $SHOIN^{(D)}$, δηλαδή με την πληρότητα του συστήματος, ενώ ο δεύτερος με την απόδοσή του (performance). Με άλλα λόγια και σε αντίθεση με πολλές από τις υπάρχουσες πλατφόρμες, στοχεύουμε στην κατασκευή ενός συστήματος που όχι μόνο θα οργανώνει τα μεταδεδομένα θεωρώντας τα ως «ορθά», αλλά θα επιτελεί με αποδοτικό τρόπο και όλες εκείνες τις απαραίτητες διεργασίες, είτε «ελαφριάς» είτε «βαριάς» συλλογιστικής ανάλυσης, τόσο για τον

έλεγχο της συνέπειας (consistency) της προς αποθήκευση γνώσης, όσο και για την αποτίμηση ερωτημάτων σχετικά με αυτήν.

Ιδιαίτερη έμφαση δίνουμε στην κατασκευή ενός σχήματος βάσης δεδομένων που αφενός θα οργανώνει την $SHOIN^{(D)}$ πληροφορία χωρίς απώλειες (πληρότητα), αφετέρου θα επιτρέπει τη γρήγορη αποτίμηση ερωτημάτων (απόδοση). Σε αυτήν την κατεύθυνση, πέρα από τη χρήση της «κλασσικής» τεχνολογίας βάσεων δεδομένων (π.χ. Ευρετήρια), χρησιμοποιούμε και μια σειρά από άλλες τεχνικές βελτιστοποίησης. Μια από αυτές αφορά στην κατασκευή ενός Σχήματος Ετικετών (Labeling Scheme) για την αναπαράσταση και κωδικοποίηση των ιεραρχικών δομών (hierarchical structures) στη σχεσιακή βάση δεδομένων και κατά συνέπεια την επιτάχυνση των σχετικών ερωτημάτων. Παράλληλα, εμπλουτίζουμε το σύστημα με μεθόδους συγχώνευσης Ψευδομοντέλων (Pseudomodel merging), που αποτελούν ένα είδος συλλογιστικής ανάλυσης, στη βάση δεδομένων. Με αυτόν τον τρόπο, η τελευταία καθίσταται περισσότερο «αυτόνομη» σχετικά με ερωτήματα «ανοιχτού κόσμου» (open world), ενώ ταυτόχρονα υποβοηθά και τον Reasoner στην εξαγωγή υπονοούμενων (implicit) σχέσεων, στέλνοντας σε αυτόν μόνο τις περιπτώσεις που δεν μπορεί να αποκλείσει και για τις οποίες η εκτέλεση του (ακριβού) Tableau αλγόριθμου είναι αναπόφευκτη.

Τέλος, κατά την ανάπτυξη του συστήματος που φέρει το όνομα «Database supported Reasoning System» (DBRS), αναπτύξαμε μια σειρά από συνδυαστικούς αλγόριθμους που εκμεταλλεύονται τόσο το DBMS όσο και τον Reasoner, επιταχύνοντας έτσι τη λειτουργία του DBRS όχι μόνο στη φάση αποτίμησης ερωτημάτων (querying phase), αλλά και σε αυτή της φόρτωσης (loading phase).

1.2 Θέματα με τα οποία ασχοληθήκαμε

Τα θέματα με τα οποία ασχοληθήκαμε στα πλαίσια της διπλωματικής εργασίας συνοψίζονται στα παρακάτω:

i) Μελετήσαμε και παρουσιάσαμε την Περιγραφική Λογική (Description Logic - DL), μια οικογένεια λογικών φορμαλισμών που χρησιμοποιούνται για αναπαράσταση γνώσης.

ii) Μελετήσαμε δυο αλγορίθμους (Tableau αλγόριθμος, Προσυμπλήρωση) συλλογιστικής ανάλυσης γνώσης και περιγράψαμε εκτενώς τη λειτουργία τους μέσω παραδειγμάτων.

iii) Μελετήσαμε και αναλύσαμε τρόπους συνδυασμού «βαριάς» συλλογιστικής ανάλυσης με τεχνολογία βάσεων δεδομένων (Ψευδομοντέλα, DLP).

iv) Μελετήσαμε και παραθέσαμε πλεονεκτήματα και μειονεκτήματα τριών μοντέλων (σχημάτων βάσης δεδομένων) οργάνωσης και αποθήκευσης σημασιολογικών σχημάτων.

v) Μελετήσαμε και αναλύσαμε τεχνικές για εξαγωγή υπονοούμενης πληροφορίας (Backward/Forward chaining) και για αναπαράσταση ιεραρχικών δομών σε σχεσιακές βάσεις δεδομένων (ISA, NOISA, Labeling Schemes).

vi) Μελετήσαμε και περιγράψαμε συνοπτικά έντεκα συστήματα διαχείρισης οντολογιών.

vii) Ορίσαμε τις βασικές απαιτήσεις που πρέπει να ικανοποιεί ένα σύστημα διαχείρισης οντολογιών εκφραστικότητας $SHOIN^{(D)}$ και σύμφωνα με αυτές προσδιορίσαμε τα υποσυστήματα στα οποία πρέπει το σύστημα να διαιρείται.

viii) Κατασκευάσαμε ένα πλήρες μοντέλο Οντοτήτων-Συσχετίσεων για γνώση εκφραστικότητας $SHOIN^{(D)}$.

ix) Μελετήσαμε σε βάθος τη λειτουργία της ευρύτατα διαδεδομένης μηχανής συλλογιστικής ανάλυσης Pellet [Pellet], την οποία και ενσωματώσαμε στο πρότυπο σύστημα Database supported Reasoning System (DBRS).

x) Μελετήσαμε σε βάθος τη λειτουργία και τα χαρακτηριστικά του γνωστού αντικειμενο-σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (object relational DBMS) PostgreSQL [Postgres], με το οποίο υλοποιήσαμε τη βάση δεδομένων του DBRS.

xi) Αναπτύξαμε συνδυαστικούς[3] αλγόριθμους για την αποτίμηση ερωτημάτων σε γνώση εκφραστικότητας $SHOIN^{(D)}$.

xii) Αναπτύξαμε αλγόριθμους που επιταχύνουν τη συλλογιστική διαδικασία κατά τη φόρτωση οντολογιών $SHOIN^{(D)}$ στη βάση δεδομένων.

[3] Με την έννοια ότι συνδυάζουν DBMS και Reasoner.

xiii) Ενσωματώσαμε όλους τους παραπάνω αλγόριθμους στο σύστημα DBRS.

xiv) Ελέγξαμε την απόδοση του DBRS χρησιμοποιώντας δυο οντολογίες πολύ μεγάλου όγκου, με διαφορετικά όμως χαρακτηριστικά.

xv) Προτείναμε, βάσει της εμπειρίας μας, μελλοντικές βελτιώσεις και επεκτάσεις του συστήματος DBRS.

1.3 Οργάνωση κειμένου

Η διπλωματική εργασία οργανώνεται στα παρακάτω κεφάλαια:

- Στο Κεφάλαιο 2 παρουσιάζεται το γενικό θεωρητικό υπόβαθρο μαζί με τις βασικές έννοιες που είναι απαραίτητες για την κατανόηση της εργασίας.
- Στο Κεφάλαιο 3 παρουσιάζονται σχετικές, ως προς το θέμα της παρούσας διπλωματικής, εργασίες που αφορούν τόσο σε σχήματα βάσεων δεδομένων για αποθήκευση σημασιολογικής πληροφορίας, όσο και σε συστήματα ολοκληρωμένης διαχείρισης οντολογιών, παρόμοιας «φιλοσοφίας» με αυτή του DBRS.
- Στο Κεφάλαιο 4 αναλύονται οι βασικές απαιτήσεις που ικανοποιεί το σύστημά μας. Η μεθοδολογία παρουσίασης περιλαμβάνει το χωρισμό του DBRS σε υποσυστήματα και τη συνοπτική περιγραφή των λειτουργιών που το καθένα από αυτά επιτελεί. Στο τέλος του κεφαλαίου, δίνεται το μοντέλο Οντοτήτων-Συσχετίσεων της βάσης δεδομένων που χρησιμοποιεί εσωτερικά το DBRS.
- Στο Κεφάλαιο 5 αναφέρονται οι εφαρμογές που υλοποιούν τις λειτουργίες που περιγράφηκαν στο προηγούμενο κεφάλαιο. Στο τέλος, παρουσιάζεται το σχεσιακό διάγραμμα της βάσης δεδομένων του DBRS, το οποίο προέκυψε από την επεξεργασία του μοντέλου Οντοτήτων-Συσχετίσεων.
- Στο Κεφάλαιο 6 δίνονται οι λεπτομέρειες υλοποίησης του DBRS στην παρούσα έκδοσή του. Αρχικά, παρατίθενται επιλεκτικά μια σειρά από αλγόριθμοι που αναπτύχθηκαν. Στη συνέχεια, επεξηγούνται τα βήματα εγκατάστασης του συστήματος, ενώ επίσης γίνεται μια σύντομη αναφορά στις πλατφόρμες και τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την υλοποίησή του. Στο τέλος του κεφαλαίου, αναλύεται το Σχήμα Ετικετών,

αιτιολογείται η χρήση των ευρετηρίων (indices) της βάσης δεδομένων και περιγράφονται οι κλάσεις (με όρους αντικειμενοστρεφούς προγραμματισμού) του πηγαίου κώδικα (source code), τα μέλη τους (members) και οι μέθοδοί τους (methods).

- Στο Κεφάλαιο 7 παρουσιάζονται δυο σενάρια ελέγχου της λειτουργίας του συστήματος μαζί με συγκεντρωτικά αποτελέσματα που αφορούν στην απόδοσή του.
- Στο Κεφάλαιο 8 επιχειρείται μια σύνοψη της εργασίας και παρουσιάζονται τα συμπεράσματα αυτής. Επίσης, προτείνονται πιθανές μελλοντικές επεκτάσεις, αλλά και γενικές βελτιώσεις του συστήματος DBRS.
- Στο Κεφάλαιο 9 παρατίθεται η σχετική βιβλιογραφία.

2

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό δίνεται η περιγραφή εννοιών που αποτελούν το θεωρητικό υπόβαθρο της παρούσας εργασίας και που απαιτούνται για την κατανόησή της. Ήδη, στην εισαγωγή, έχουμε αναφερθεί στους όρους Σημασιολογικός Ιστός (ΣΙ) και μεταδεδομένα. Εδώ θα ασχοληθούμε με την έννοια της οντολογίας (ontology) και με μια οικογένεια λογικών φορμαλισμών που ονομάζεται Περιγραφική Λογική (Description Logic - DL). Θα εστιάσουμε το ενδιαφέρον μας σε μεθόδους συλλογιστικής ανάλυσης (reasoning) των μεταδεδομένων, δίνοντας έμφαση στον επικρατέστερο Tableau αλγόριθμο, καθώς και σε τεχνικές συνδυασμού τους με DBMSs. Οι δύο τελευταίες ενότητες αφορούν στις τυπικές γλώσσες (RDF και OWL) που χρησιμοποιούνται για την περιγραφή οντολογιών. Ο εξοικειωμένος αναγνώστης μπορεί να παραβλέψει αυτό το κεφάλαιο.

2.1 Η έννοια της οντολογίας

Η προσθήκη των μεταδεδομένων στο υπάρχον δικτυακό οικοδόμημα προϋποθέτει την οργάνωσή τους με τέτοιο τρόπο που να καθιστά τη διαχείρισή τους αποτελεσματική. Κλειδί στην επίτευξη αυτού του εγχειρήματος αποτελεί η χρήση των οντολογιών. Σαφής ορισμός για την γενικότερη έννοια της οντολογίας δεν υπάρχει

και, μάλιστα, ο όρος διαφοροποιείται αρκετά ανάλογα με τον επιστημονικό τομέα στον οποίο χρησιμοποιείται [Ontology]. Ωστόσο, αν προσπαθήσουμε να συμπυκνώσουμε τον τρόπο με τον οποίο αντιλαμβανόμαστε τις οντολογίες στα πλαίσια του ΣΙ, τότε καταλήγουμε στο εξής: Τυπική περιγραφή ενός συνόλου πληροφοριών και των συσχετίσεων μεταξύ τους. Κάθε οντολογία μπορεί να θεωρηθεί ως σύνθεση δύο βασικών μερών:

- Ένα λεξιλόγιο (intensional knowledge) που αποτελείται από ονόματα εννοιών (concepts) και σχέσεων (relationships). Χρησιμοποιείται για να περιγράψει την πληροφορία ή αλλιώς τον «κόσμο» που μοντελοποιούμε και διαφοροποιείται ανάλογα με αυτόν[4]. Στην ορολογία της Περιγραφικής Λογικής, όπως θα δούμε παρακάτω, το τμήμα αυτό ονομάζεται TBox (Terminology Box).
- Ένα σύνολο επιπλέον γνώσης (extensional knowledge) σχετικά με τον «κόσμο», που περιλαμβάνει δηλώσεις/ισχυρισμούς (assertions). Οι δηλώσεις αντιστοιχίζουν άτομα (individuals) σε έννοιες και ζεύγη ατόμων ή ζεύγη ατόμου-σταθεράς (literal) σε σχέσεις. Αναφέρονται συχνά στη βιβλιογραφία και ως στιγμιότυπα (instances). Στην Περιγραφική Λογική, το τμήμα αυτό ονομάζεται ABox (Assertion Box) και σημειώνουμε ότι σε μια οντολογία μπορεί να απουσιάζει τελείως.

Τα άτομα της οντολογίας αποτελούν, στην ουσία, τα αντικείμενα που θέλουμε να διαχειριστούμε και συχνά είναι αναγνωριστικά (URIs) πόρων του διαδικτύου. Οι έννοιες του λεξιλογίου ισοδυναμούν με σύνολα ατόμων που μοιράζονται ένα τουλάχιστον κοινό χαρακτηριστικό και αναφέρονται συχνά ως κλάσεις (classes). Οι σχέσεις αντιστοιχούν σε σύνολα από ζεύγη ατόμων (ή ζεύγη ατόμου-σταθεράς) και ονομάζονται ιδιότητες (properties), επειδή ακριβώς προσδίδουν ιδιότητες στα άτομα

[4] Για παράδειγμα, η μοντελοποίηση μιας ανθρώπινης οικογένειας απαιτεί διαφορετικές έννοιες και σχέσεις απ' ότι η μοντελοποίηση της γνώσης μας σχετικά με τις υπάρχουσες ποικιλίες κρασιών. Υπό αυτή την έννοια, κάθε οντολογία έχει το δικό της λεξιλόγιο.

συνδέοντάς τα μεταξύ τους (ή με κάποια σταθερά). Στην Περιγραφική Λογική, ονομάζονται ρόλοι (roles).

Οι οντολογίες χαρακτηρίζονται ως ο τυπικός (formal) προσδιορισμός των μεταδεδομένων ο οποίος εξασφαλίζει μια «κοινή αντίληψη» της περιοχής που μας ενδιαφέρει και παρέχει, δια της «φύσης» του, τη δυνατότητα συλλογιστικής ανάλυσης (reasoning), τόσο για την εξαγωγή νέων (υπονοούμενων) σχέσεων όσο και για τον έλεγχο της ισχύος ήδη υπαρχουσών (εύρεση αντιφάσεων). Η έννοια της τυπικότητας, όσον αφορά στον προσδιορισμό της μεταπληροφορίας, είναι καθοριστική για τη δυνατότητα των υπολογιστών να την αναλύσουν και άρα να διαχειριστούν αυτόματα και την ίδια την πληροφορία.

Η κατασκευή οντολογιών που μπορούν να «διαβαστούν» από υπολογιστές απαιτεί προφανώς την ύπαρξη τυπικών γλωσσών που θα χρησιμοποιούνται αποκλειστικά γι' αυτό το σκοπό και θα ικανοποιούν μια σειρά κριτηρίων. Ανάμεσα σ' αυτά είναι η ευκολία στη χρήση, η επαρκής εκφραστικότητα (expressivity), η δυνατότητα για συλλογιστική ανάλυση και φυσικά η συμβατότητα με τις ήδη υπάρχουσες τεχνολογίες του παγκόσμιου ιστού (π.χ. HTML, XML). Θα επανέλθουμε στα ζητήματα αυτά στις Ενότητες 2.3 και 2.4.

2.2 *Περιγραφική Λογική (Description Logic - DL)*

Η Περιγραφική Λογική (Description Logic - DL)[5] είναι μια από τις θεωρίες αναπαράστασης γνώσης. Η εκφραστικότητά της, το τι δηλαδή μπορεί να περιγραφεί, περιορίζεται σε ένα τμήμα μιας άλλης θεωρίας που ονομάζεται Λογική Πρώτης Τάξης (First Order Logic - FOL). Μια τρίτη θεωρία αναπαράστασης γνώσης είναι ο Λογικός Προγραμματισμός (Logic Programs - LP) ο οποίος επίσης «μοιράζεται ένα κοινό κομμάτι» με τη Λογική Πρώτης Τάξης, γνωστό ως Λογική Horn (Horn Logic). Κάθε μια από τις προαναφερθείσες θεωρίες έχει και διαφορετική «φιλοσοφία» που αποτυπώνεται και στη σύνταξή της. Στη FOL η αναπαράσταση γίνεται με

[5] Συχνά ο όρος συναντάται και στον πληθυντικό (Description Logics) επειδή ακριβώς μπορεί να θεωρηθεί ως σύνολο λογικών φορμαλισμών (γλωσσών αναπαράστασης) που εντάσσονται στη θεωρία της Περιγραφικής Λογικής, αλλά έχουν διαφορετικές δυνατότητες ως προς την εκφραστικότητα της γνώσης που μπορούν να μοντελοποιήσουν.

κατηγορήματα (predicates), μεταβλητές (variables) και σταθερές (constants), ενώ στον LP έχουμε κανόνες αιτίου-αιτιατού. Από την άλλη, η «φιλοσοφία» της DL είναι περισσότερο ανθρωποκεντρική (human-centred) και μοιάζει με το αντικειμενοστρεφές μοντέλο προγραμματισμού (object oriented programming - OOP). Η μοντελοποίηση στην DL γίνεται με έννοιες, ρόλους, άτομα και σταθερές (literals), όπως αυτά παρουσιάστηκαν στην προηγούμενη ενότητα.

Η DL, όπως κάθε θεωρία αναπαράστασης γνώσης, διαθέτει τρία βασικά στοιχεία:

- Λεξιλόγιο (Vocabulary)

  Απαρτίζεται από ονόματα εννοιών (κλάσεων) και ρόλων. Οι έννοιες χρησιμοποιούνται για να ομαδοποιήσουν άτομα με κοινά χαρακτηριστικά και οι ρόλοι για να εκφράσουν συσχετίσεις μεταξύ τους ή μεταξύ αυτών και σταθερών. Όπως αναφέρθηκε και στην Ενότητα 2.1, το λεξιλόγιο διαφοροποιείται ανάλογα με τη γνώση που κάθε φορά μοντελοποιείται.

- Συντακτικό (Syntax)

  Βασίζεται στους κατασκευαστές (constructors) και σε σύμβολα που εκφράζουν σχέσεις (π.χ. ιεραρχικές) μεταξύ στοιχείων του λεξιλογίου ή/και ατόμων. Οι κατασκευαστές δηλώνουν μια λειτουργία (operation) μεταξύ δύο ή περισσοτέρων εννοιών ή ρόλων (π.χ. ένωση). Ο συνδυασμός τους με τα σύμβολα του συντακτικού και τα στοιχεία του λεξιλογίου οδηγεί στην κατασκευή των αξιωμάτων (axioms). Η γνώση που μοντελοποιούμε με DL είναι, στην ουσία, ένα σύνολο αξιωμάτων.

- Σημασιολογία (Semantics)

  Αφορά στην ερμηνεία των στοιχείων του λεξιλογίου, των κατασκευαστών και των συμβόλων του συντακτικού. Η σημασιολογία στην DL είναι συνολοθεωρητική: Μια έννοια ερμηνεύεται ως σύνολο από αντικείμενα, ένας ρόλος ως σύνολο από ζεύγη αντικειμένων και ένα άτομο ως αντικείμενο.

Ο όρος μοντελοποιημένη γνώση, που χρησιμοποιήσαμε στα προηγούμενα, αναφέρεται στις θεωρίες αναπαράστασης γνώσης ως Βάση Γνώσης (Knowledge Base - KB) και αποτελεί θεμελιώδη έννοια την οποία θα χρησιμοποιούμε στο εξής.

Βάση Γνώσης (KB) ονομάζουμε ένα σύνολο γνώσης που περιγράφεται (μοντελοποιείται) με χρήση ενός τυπικού φορμαλισμού ή αλλιώς μιας γλώσσας αναπαράστασης (description language). Τα βασικά δομικά στοιχεία μιας KB είναι οι ατομικές έννοιες (atomic concepts), οι ατομικοί ρόλοι (atomic roles) και τα άτομα (individuals). Σύνθετες εκφράσεις (complex descriptions) μπορούν να κατασκευαστούν από αυτά χρησιμοποιώντας τους κατασκευαστές (constructors) που προσφέρει η εκάστοτε γλώσσα αναπαράστασης [BCG+03].

Η βασική γλώσσα αναπαράστασης, την οποία επεκτείνουν όλες οι άλλες, είναι η *AL* (Attributive Language) που περιγράφουμε στη συνέχεια. Συμβολίζουμε με A, B ατομικές έννοιες, με R, R_i (i=1,2,….) ατομικούς ρόλους, με C, C_i (i=1,2,….), D σύνθετες εκφράσεις και με a, b, c άτομα. Οι σύνθετες εκφράσεις στην *AL* σχηματίζονται βάσει του παρακάτω συντακτικού κανόνα:

$C, D \rightarrow A \mid T \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.T$, όπου

- T: Καθολική έννοια (Universial concept). Περιλαμβάνει όλα τα άτομα της KB.
- $\bot$: Κενή έννοια (Bottom concept). Δεν περιλαμβάνει κανένα άτομο της KB.
- $\neg A$: Ατομική άρνηση (Atomic negation). Η έννοια που περιλαμβάνει όλα εκείνα τα άτομα της KB που δεν ανήκουν στην ατομική έννοια A και μόνο αυτά.
- $C \sqcap D$: Τομή (Intersection) δύο σύνθετων εννοιών. Η έννοια που περιλαμβάνει μόνο τα άτομα εκείνα της KB που ανήκουν και στις δύο αρχικές έννοιες C και D.
- $\forall R.C$: Περιορισμός τιμής (Value restriction). Η έννοια που περιλαμβάνει όλα εκείνα τα άτομα της KB που συνδέονται μέσω του ατομικού ρόλου R μόνο με άτομα που ανήκουν στην σύνθετη έννοια C.
- $\exists R.T$: Περιορισμένη υπαρξιακή ποσοτικοποίηση (Limited existential quantification). Η έννοια που περιλαμβάνει όλα εκείνα τα άτομα της KB που συνδέονται μέσω του ατομικού ρόλου R με ένα τουλάχιστον άτομο της καθολικής έννοιας, δηλαδή της KB.

Οι επεξηγήσεις που δώσαμε για κάθε ένα από τα στοιχεία-έννοιες που συναντάμε στο συντακτικό κανόνα της *AL* αποτελούν στην ουσία μια ερμηνεία της σημασίας

τους, δηλαδή μια απόδοση της σημασιολογίας τους με φυσική γλώσσα. Προκειμένου όμως να ορίσουμε αυστηρά αυτή τη σημασιολογία, εισάγουμε την έννοια των διερμηνειών (interpretations) [BCG+03]. Μια διερμηνεία I αποτελείται από μια δομή $(\Delta^I , \cdot^I)$, όπου Δ^I το πεδίο της διερμηνείας (domain of interpretation) και $\cdot^I$ μια συνάρτηση αντιστοίχησης[6]. Η συνάρτηση αυτή αντιστοιχίζει σε κάθε ατομική έννοια A ένα σύνολο $A^I \subseteq \Delta^I$ και σε κάθε ατομικό ρόλο R μια δυαδική σχέση $R^I \subseteq \Delta^I \times \Delta^I$. Επεκτείνεται στις σύνθετες εκφράσεις της *AL* βάσει των παρακάτω:

$T^I = \Delta^I$

$\bot^I = \emptyset$

$(\neg A)^I = \Delta^I \setminus A^I$

$(C \sqcap D)^I = C^I \cap D^I$

$(\forall R.C)^I = \{a \in \Delta^I \mid \forall b. (a , b) \in R^I \rightarrow b \in C^I\}$

$(\exists R.T)^I = \{a \in \Delta^I \mid \exists b. (a , b) \in R^I\}$

Κάθε KB που βασίζεται σε DL διαιρείται σε δύο κύρια μέρη: TBox και ABox[7]. Οι όροι μας είναι ήδη γνωστοί από την προηγούμενη ενότητα που αφορούσε στις οντολογίες. Εδώ θα τους περιγράψουμε ως σύνολα αξιωμάτων από την οπτική γωνία της Περιγραφικής Λογικής:

- TBox (Terminology Box): Είναι εκείνο το σύνολο των αξιωμάτων της KB που αφορούν στην ορολογία, δηλαδή στο λεξιλόγιο που χρησιμοποιείται. Σε

[6] Αυτός ο τυπικός ορισμός της σημασιολογίας είναι γνωστός στη βιβλιογραφία ως Tarski-style.

[7] Η συγγένεια του όρου Βάση Γνώσης με αυτόν της Οντολογίας είναι κάτι παραπάνω από προφανής. Εκείνο που θέλουμε να σημειώσουμε εδώ είναι ότι, όταν (στα πλαίσια του ΣΙ) αναφερόμαστε σε οντολογίες, εννοούμε Βάσεις Γνώσεις (KBs) που έχουν «γραφτεί» με χρήση μιας τυπικής γλώσσας όπως οι RDF και OWL που θα δούμε στις Ενότητες 2.3 και 2.4.

περίπτωση εκφραστικότητας *AL* [8], το TBox περιλαμβάνει δύο ειδών αξιώματα:

- Αξιώματα υπαγωγής εννοιών (Concept inclusion axioms)

  Όπως υποδηλώνει και το όνομά τους, εκφράζουν σχέσεις ιεραρχίας μεταξύ κλάσεων της ΚΒ και έχουν τη μορφή $C \sqsubseteq D$. Ένα αξίωμα υπαγωγής $C \sqsubseteq D$ δηλώνει ότι τα άτομα που ανήκουν στην έννοια-υπαγωγέας (subsumee) C ανήκουν, επίσης, και στην έννοια-υπαγόμενος (subsumer) D, αλλά όχι το αντίστροφο. Η αυστηρή σημασιολογία των αξιωμάτων αυτού του είδους ορίζεται ως εξής:

  - Μια διερμηνεία *I* ικανοποιεί το αξίωμα $C \sqsubseteq D$ ανν $C^I \subseteq D^I$

- Αξιώματα Ισοδυναμίας εννοιών (Concept equality axioms)[9]

  Εκφράζουν σχέσεις ισοδυναμίας μεταξύ κλάσεων της ΚΒ και έχουν τη μορφή

  $C \equiv D$. Ένα αξίωμα ισοδυναμίας $C \equiv D$ δηλώνει ότι τα άτομα που ανήκουν στην έννοια C ανήκουν, επίσης, και στην έννοια D και αντιστρόφως. Η αυστηρή σημασιολογία των αξιωμάτων αυτού του είδους ορίζεται ως εξής:

  - Μια διερμηνεία *I* ικανοποιεί το αξίωμα $C \equiv D$ ανν $C^I \equiv D^I$.

  Αυτομάτως καταλαβαίνουμε, από τη σημασιολογία τους, ότι τα αξιώματα ισοδυναμίας μπορούν να εκφραστούν και ως αξιώματα υπαγωγής. Στην ουσία, κάθε αξίωμα ισοδυναμίας εκφράζεται με δύο αξιώματα υπαγωγής ως εξής[10]: $C \equiv D \Leftrightarrow C \sqsubseteq D$ και $D \sqsubseteq C$

---

[8] Το εύρος της γνώσης μιας ΚΒ δηλώνεται με το όνομα της γλώσσας αναπαράστασης που χρησιμοποιείται.

[9] Τα αξιώματα αυτά, όταν έχουν τη μορφή Όνομα έννοιας ≡ Σύνθετη έκφραση, αναφέρονται και ως αξιώματα ορισμού (definition axioms), θεωρώντας ότι ο ορισμός της έννοιας του πρώτου μέλους είναι η σύνθετη έκφραση του δεύτερου.

[10] Αυτό ισχύει μόνο αν στο TBox δεν έχουμε κύκλους (cycles), δηλαδή το όνομα μιας έννοιας δεν υπάρχει στον ορισμό της όπως π.χ. στο αξίωμα: Άνθρωπος ≡ Ζώο Π ∃Έχει_Πατέρα.Άνθρωπος.

- ABox (Assertion Box): Είναι εκείνο το σύνολο των αξιωμάτων που αφορούν στις δηλώσεις σχετικά με τα άτομα της KB. Σε περίπτωση εκφραστικότητας *AL*, το ABox περιλαμβάνει τα εξής είδη αξιωμάτων:
    - Αξιώματα δηλώσεων εννοιών (Concept assertion axioms)

      Εκφράζουν αντιστοιχήσεις ατόμων της KB σε έννοιες και έχουν τη μορφή a : C. Ένα αξίωμα a : C δηλώνει ότι το άτομο a είναι τύπου C. Η αυστηρή σημασιολογία των αξιωμάτων αυτού του είδους ορίζεται ως εξής:

      - Μια διερμηνεία *I* ικανοποιεί το αξίωμα a : C ανν $a^I \in C^I$
    - Αξιώματα δηλώσεων ρόλων (Role assertion axioms)

      Εκφράζουν αντιστοιχήσεις ζευγών ατόμων σε ατομικούς ρόλους και έχουν τη μορφή (a , b) : R, όπου a υποκείμενο (subject) και b αντικείμενο (object) της δήλωσης. Ένα αξίωμα (a , b) : R δηλώνει ότι τα άτομα a και b συνδέονται μεταξύ τους μέσω της σχέσης R. Η αυστηρή σημασιολογία των αξιωμάτων αυτού του είδους ορίζεται ως εξής:

      - Μια διερμηνεία *I* ικανοποιεί το αξίωμα (a , b) : R ανν $(a^I , b^I) \in R^I$

Συνοψίζοντας, μια διερμηνεία *I* που ικανοποιεί όλα τα αξιώματα του TBox, με τον τρόπο που ορίσαμε παραπάνω, λέμε ότι ικανοποιεί το TBox της KB και αποτελεί ένα μοντέλο (model) του. Αν επίσης ικανοποιεί κατ' αντιστοιχία και το ABox, τότε η *I* ονομάζεται μοντέλο της Βάσης Γνώσης [BCG+03].

Όπως αναφέραμε στα προηγούμενα, η εκφραστικότητα μιας γλώσσας αναπαράστασης εξαρτάται από τους κατασκευαστές που αυτή παρέχει. Προσθέτοντας λοιπόν νέους κατασκευαστές στην βασική γλώσσα *AL*, παίρνουμε γλώσσες μεγαλύτερης εκφραστικότητας. Παραδείγματα τέτοιων κατασκευαστών είναι:

- Ένωση εννοιών (Concept union)

  Δηλώνεται με το γράμμα *U*. Η ένωση δύο σύνθετων εννοιών συμβολίζεται με C D και αποτελεί την έννοια εκείνη η οποία περιλαμβάνει όλα τα άτομα που ανήκουν είτε στην C είτε στην D είτε και στις δύο έννοιες μαζί. Η σημασιολογία της ορίζεται ως εξής: $(C \sqcup D)^I \Leftrightarrow C^I \cup D^I$.

- Πλήρης υπαρξιακή ποσοτικοποίηση (Full existential quantification)

  Δηλώνεται με το γράμμα *E*. Η πλήρης υπαρξιακή ποσοτικοποίηση συμβολίζεται με $\exists$R.C και αποτελεί την έννοια εκείνη η οποία περιλαμβάνει όλα εκείνα τα άτομα που συνδέονται, μέσω του ατομικού ρόλου R, με ένα τουλάχιστον άτομο της σύνθετης έννοιας C. Η σημασιολογία της ορίζεται ως εξής: $(\exists R.C)^I = \{a \in \Delta^I \mid \exists b.\ (a, b) \in R^I \wedge b \in C^I\}$.

- Περιορισμός αριθμού (Number restriction)[11]

  Δηλώνεται με το γράμμα *N*. Συμβολίζεται με ≥ n R και με ≤ n R. Ο περιορισμός ≥ n R αποτελεί την έννοια που περιλαμβάνει όλα εκείνα τα άτομα που μετέχουν το λιγότερο (at least) n φορές στις δηλώσεις του ατομικού ρόλου R. Κατ' αντιστοιχία, ο περιορισμός ≤ n R αποτελεί την έννοια που περιλαμβάνει όλα εκείνα τα άτομα που μετέχουν το πολύ (at most) n φορές στις δηλώσεις του ατομικού ρόλου R. Η σημασιολογία τους ορίζεται ως εξής:

  $(\geq n\ R)^I = \{a \in \Delta^I \mid |\{b \mid (a, b) \in R^I\}| \geq n\}$

  $(\leq n\ R)^I = \{a \in \Delta^I \mid |\{b \mid (a, b) \in R^I\}| \leq n\}$

  όπου ο συμβολισμός **|** ···· **|** δηλώνει τον αριθμό των μελών ενός συνόλου (cardinality of set). Ένας ρόλος για τον οποίο ισχύει μέγιστος περιορισμός αριθμού ≤n R με n=1 ονομάζεται λειτουργικός ρόλος (functional role) και η ύπαρξή του συμβολίζεται με *F*.

- Σύνθετη άρνηση (Complex negation)[12]

  Δηλώνεται με το γράμμα *C* και συμβολίζεται με ¬C. Η σύνθετη άρνηση ¬C είναι η

  σύνθετη εκείνη έννοια που περιλαμβάνει όλα τα άτομα της KB που δεν ανήκουν στην έννοια C. Η σημασιολογία της ορίζεται ως εξής:

  $(\neg C)^I = \Delta^I \setminus C^I$.

Επεκτείνοντας την *AL* με οποιοδήποτε υποσύνολο των παραπάνω κατασκευαστών, παίρνουμε μια νέα γλώσσα αναπαράστασης (μεγαλύτερης εκφραστικότητας από την αρχική) που ανήκει στη λεγόμενη οικογένεια *AL*-γλωσσών.

---

[11] Αναφέρεται συχνά και ως περιορισμός μεγέθους συνόλου (cardinality restriction)

[12] Αναφέρεται συχνά και ως πλήρης άρνηση (full negation)

Για παράδειγμα, επεκτείνοντας την *AL* με περιορισμό αριθμού, παίρνουμε τη γλώσσα *ALN*. Εκείνο που αξίζει να σημειώσουμε εδώ είναι ότι ο συνδυασμός δύο κατασκευαστών μπορεί να προσφέρει εκφραστικότητα ίδια με αυτή ενός άλλου κατασκευαστή. Χαρακτηριστική περίπτωση αποτελεί η σύνθετη άρνηση που, όταν προστεθεί στη γλώσσα *AL*, την εξοπλίζει με ένωση εννοιών, αφού όπως γνωρίζουμε από τη θεωρία συνόλων:

$\neg(C^I \cap D^I) = (\neg C)^I \cup (\neg D)^I$ και άρα: $\neg(C \sqcap D) = (\neg C) \sqcup (\neg D)$.

Έτσι, γράφουμε *ALC* και όχι *ALEC*.

Άλλοι κατασκευαστές που προσδίδουν επιπλέον εκφραστικότητα είναι οι:

- Ποιοτικός περιορισμός αριθμού (Qualified number restriction)[13]

  Δηλώνεται με το γράμμα *Q* και συμβολίζεται με ≥ n R.C και ≤ n R.C. Ο περιορισμός ≥ n R.C αποτελεί την έννοια που περιλαμβάνει όλα εκείνα τα άτομα που μετέχουν ως υποκείμενα το λιγότερο (at least) n φορές στις δηλώσεις του ατομικού ρόλου R, έχοντας ως αντικείμενο άτομο της έννοιας C. Κατ' αντιστοιχία, ο περιορισμός ≤ n R αποτελεί την έννοια που περιλαμβάνει όλα εκείνα τα άτομα που μετέχουν ως υποκείμενα το πολύ (at most) n φορές στις δηλώσεις του ατομικού ρόλου R, έχοντας ως αντικείμενο άτομο της έννοιας C. Η σημασιολογία τους ορίζεται ως εξής:

  $(\geq n\ R.C)^I = \{a \in \Delta^I \mid |\{b \mid (a, b) \in R^I \wedge b \in C^I\}| \geq n\}$

  $(\leq n\ R.C)^I = \{a \in \Delta^I \mid |\{b \mid (a, b) \in R^I \wedge b \in C^I\}| \leq n\}$

- Μεταβατικός ρόλος (Transitive role)

  Συμβολίζεται με R_+ και, συνήθως, στη βιβλιογραφία η ύπαρξή του υποδηλώνεται με το γράμμα *S* που αποτελεί σύντμηση των *AL*, *C* και μεταβατικών ρόλων. Ο R_+ είναι εκείνος ο ρόλος του οποίου οι δηλώσεις έχουν μεταβατική ιδιότητα, δηλαδή μπορεί να σχηματίζουν μια «αλυσίδα» ή, όπως συχνά ονομάζεται, ένα μεταβατικό κλείσιμο (Transitive Closure - TC). Η σημασιολογία του μεταβατικού ρόλου ορίζεται ως εξής:

  $(R_+)^I = \{a,b,c \in \Delta^I \mid \{(a,b),(b,c)\} \subseteq R_+^I \rightarrow (a,c) \in R_+^I\}$

---

[13] Αναφέρεται συχνά και ως ποιοτικός περιορισμός μεγέθους συνόλου (qualified cardinality restriction)

- Αντίστροφος ρόλος (Inverse role)

  Δηλώνεται με το γράμμα *I* και συμβολίζεται με R^-. Ο R^- είναι εκείνος ο ρόλος που περιλαμβάνει ακριβώς τα αντίστροφα ζεύγη ατόμων σε σχέση με τον ατομικό ρόλο R. Η σημασιολογία του R^- ορίζεται ως εξής: $(R^-)^I = \{a,b \in \Delta^I \mid (a,b) \in R^I \leftrightarrow (b,a) \in R^{-I}\}$. Ένας ρόλος R που είναι αντίστροφος με τον εαυτό του ονομάζεται συμμετρικός ρόλος (symmetric role).

- Απαριθμητικά (Nominals)

  Δηλώνονται με το γράμμα *O* και συμβολίζονται με {a,b,…}, όπου a,b,… άτομα της KB. Χρησιμεύουν στον ορισμό απαριθμημένων εννοιών (enumerated concepts), δηλαδή εννοιών που ορίζονται ως ένα σύνολο συγκεκριμένων ατόμων της KB. Η σημασιολογία τους ορίζεται ως εξής: $(\{a\})^I = \{a^I\}$.

Σε εκφραστικές γλώσσες αναπαράστασης συναντάμε και άλλους κατασκευαστές όπως ένωση ρόλων (role union), τομή ρόλων (role intersection), ανακλαστικό ρόλο (reflexive role), μη ανακλαστικό ρόλο (irreflexive role) κ.α. Δεν έχουμε σκοπό να αναλύσουμε εξαντλητικά όλες τις περιπτώσεις. Εκείνο, όμως, που αξίζει να σημειώσουμε εδώ είναι ένα ενδιαφέρον είδος αξιωμάτων που δεν αναφέραμε μέχρι τώρα καθότι δεν υπάρχει στην *AL*. Ο λόγος για τα αξιώματα υπαγωγής ρόλων (role inclusion axioms), η ύπαρξη των οποίων σε μια KB δηλώνεται με το γράμμα *H*. Τα αξιώματα αυτά έχουν τη μορφή $R_1 \sqsubseteq R_2$, όπου R_1 και R_2 ατομικοί ρόλοι. Ένα αξίωμα $R_1 \sqsubseteq R_2$ δηλώνει ότι τα ζεύγη ατόμων του ρόλου R_1 είναι υποσύνολο των ζευγών του ρόλου R_2, αλλά όχι το αντίστροφο. Η σημασιολογία των αξιωμάτων αυτού του είδους ορίζεται ως εξής:

- Μια διερμηνεία *I* ικανοποιεί το αξίωμα $R_1 \sqsubseteq R_2$ αν $R_1^I \subseteq R_2^I$

Κατ' αντιστοιχία, ορίζονται και αξιώματα ισοδυναμίας ρόλων (role equivalence) με τον τρόπο που περιγράψαμε και για τα αξιώματα ισοδυναμίας των εννοιών στην *AL*.

Πολλά συστήματα διαχείρισης οντολογιών, όπως και το DBRS, χειρίζονται γλώσσες αναπαράστασης που υποστηρίζουν και τύπους δεδομένων (data types), όπως αυτοί ορίζονται αυστηρά με χρήση της XML [XML]. Η ύπαρξη τύπων δεδομένων και τιμών δεδομένων - literals) δηλώνεται με το γράμμα $^{(D)}$ και συνοδεύ-

Συμβολισμός	Εκφραστικότητα - Κατασκευαστές
AL	Ατομική άρνηση Τομή εννοιών Περιορισμός τιμής Περιορισμένη υπαρξιακή ποσοτικοποίηση
FL–	*AL* χωρίς ατομική άρνηση
FL_o	*FL–* χωρίς περιορ. υπαρξιακή ποσοτικοποίηση
U	Ένωση εννοιών
E	Πλήρης υπαρξιακή ποσοτικοποίηση
N	Περιορισμοί αριθμών
C	Σύνθετη άρνηση
Q	Ποιοτικοί περιορισμοί αριθμών
S	*AL* , *C* και μεταβατικοί ρόλοι
I	Αντίστροφοι ρόλοι
O	Απαριθμητικά
H	Ιεραρχία ατομικών ρόλων
R	Ανακλαστικοί ρόλοι Μη ανακλαστικοί ρόλοι Ασύμβατοι ρόλοι Αξιώματα υπαγωγής σύνθετων ρόλων
F	Λειτουργικοί ρόλοι
(D)	Τιμές δεδομένων Τύποι δεδομένων Ρόλοι τύπου δεδομένων

Πίνακας 2.1: Σύμβολα εκφραστικότητας στην DL

εται από ρόλους τύπων δεδομένων (datatype roles), οι οποίοι συνδέουν άτομα με σταθερές εκφράζοντας μια σχέση μεταξύ των δύο.

2.2.1 Συλλογιστική ανάλυση στην Περιγραφική Λογική

Όπως καταστήσαμε σαφές και στα προηγούμενα, η Περιγραφική Λογική δεν αποσκοπεί μόνο στη μοντελοποίηση της γνώσης, αλλά και στην εύκολη ανάλυσή της. Η ανάλυση αυτή περιλαμβάνει δύο σκέλη. Το πρώτο αφορά στον έλεγχο της ισχύος των αξιωμάτων της KB, ενώ το δεύτερο στην εξαγωγή νέων αξιωμάτων που προκύπτουν από τα υπάρχοντα μέσω μιας αλγοριθμικής διαδικασίας.

Οι επαγωγικές διαδικασίες (inference procedures) σε μια KB εκφραστικότητας *AL* διακρίνονται σε:

- TBox επαγωγικές διαδικασίες

  Πρόκειται για επαγωγικές διαδικασίες που σχετίζονται με την ορολογία της KB. Αυτές είναι:

  - Ικανοποιησιμότητα (Satisfiability)

    Μια έννοια είναι ικανοποιήσιμη αν υπάρχει ένα μη κενό σύνολο από άτομα της KB τα οποία ανήκουν σε αυτή. Πιο αυστηρά, μια έννοια C είναι ικανοποιήσιμη, δεδομένου ενός TBox T, αν υπάρχει ένα μοντέλο I του T τέτοιο ώστε $C^I \neq \emptyset$. Σε αυτή την περίπτωση, η διερμηνεία I ονομάζεται και μοντέλο της C. Η ικανοποιησιμότητα μιας έννοιας C μπορεί να αναχθεί σε εύρεση της σχέσης υπαγωγής (βλ. αμέσως μετά): $\bot \sqsubseteq C$.

  - Υπαγωγή (Subsumption)

    Μια έννοια C υπάγεται σε μια έννοια D αν όλα τα άτομα που ανήκουν στη C ανήκουν και στη D, αλλά όχι το αντίστροφο. Πιο αυστηρά, μια έννοια C υπάγεται μιας έννοιας D αν, δεδομένου ενός TBox T, υπάρχει ένα μοντέλο I για το οποίο ισχύει $C^I \subseteq D^I$. Αν η υπαγωγή των εννοιών C και D δηλώνεται ρητά στο T ή μπορεί να εξαχθεί από αυτό με συλλογιστική, τότε γράφουμε:

    $T \models C \sqsubseteq D$.

- Ισοδυναμία (Equivalence)

  Μια έννοια C είναι ισοδύναμη με μια έννοια D αν όλα τα άτομα που ανήκουν στη C ανήκουν και στη D και αντίστροφα. Πιο αυστηρά, μια έννοια C είναι ισοδύναμη με μια έννοια D αν, δεδομένου ενός TBox *T*, υπάρχει ένα μοντέλο *I* για το οποίο ισχύει $C^I \equiv D^I$. Βάσει των όσων έχουμε πει μέχρι τώρα, η εύρεση σχέσης ισοδυναμίας μπορεί να αναχθεί σε εύρεση σχέσεων υπαγωγής. Αν η ισοδυναμία των εννοιών C και D δηλώνεται ρητά στο *T* ή μπορεί να εξαχθεί από αυτό με συλλογιστική, τότε γράφουμε: $T \models C \equiv D$.

- Ασυμβατότητα (Disjointness)

  Μια έννοια C είναι ασύμβατη με μια έννοια D αν δεν υπάρχουν άτομα που να ανήκουν και στις δύο μαζί. Πιο αυστηρά, μια έννοια C είναι ασύμβατη με μια έννοια D αν, δεδομένου ενός TBox *T*, υπάρχει ένα μοντέλο *I* για το οποίο ισχύει $C^I \cap D^I = \emptyset$. Επομένως, είναι φανερό ότι η εύρεση σχέσης ασυμβατότητας μπορεί να αναχθεί σε εύρεση της σχέσης υπαγωγής:

  $(C \sqcap D) \sqsubseteq \bot$.

Έως τώρα, είδαμε ότι οι επαγωγικές διαδικασίες του TBox μπορούν να αναχθούν όλες σε εύρεση σχέσεων υπαγωγής. Όμως, ένα ενδιαφέρον στοιχείο της Περιγραφικής Λογικής είναι ότι όλες μπορούν επίσης να αναχθούν σε εύρεση ικανοποιησιμότητας, δεδομένου ότι επιτρέπεται σύνθετη άρνηση (*C*). Υπό αυτές τις συνθήκες, δεδομένου ενός TBox *T*, ισχύουν:

- Η έννοια C υπάγεται στην έννοια D αν $C \sqcap \neg D$ μη ικανοποιήσιμη

- Οι έννοιες C και D είναι ισοδύναμες αν $C \sqcap \neg D$ και $\neg C \sqcap D$ μη ικανοποιήσιμες

- Οι έννοιες C και D είναι ασύμβατες αν $C \sqcap D$ μη ικανοποιήσιμη

- ABox επαγωγικές διαδικασίες

  Πρόκειται για επαγωγικές διαδικασίες που σχετίζονται με τις δηλώσεις της KB. Αυτές είναι:

  - Συνέπεια (Consistency)

    Ένα ABox είναι συνεπές αν, δεδομένου ενός TBox, δε διαθέτει αξιώματα που προκαλούν αντιφάσεις (contradictions). Πιο αυστηρά,

ένα ABox A είναι συνεπές αν, δεδομένου ενός TBox T, υπάρχει ένα μοντέλο I που ικανοποιεί όλα τα αξιώματα του A.

- Συνεπαγωγή (Entailment)

  Ένα ABox A συνεπάγεται ένα αξίωμα αν κάθε διερμηνεία I που ικανοποιεί το A, ικανοποιεί και το αξίωμα αυτό. Στην περίπτωση αυτή, για παράδειγμα σε μια δήλωση έννοιας a : C, γράφουμε $A \models$ a : C.

Κατ' αντιστοιχία με τις αναγωγές στο TBox, η συνεπαγωγή σε ένα ABox μπορεί να αναχθεί σε συνέπεια του ABox ως εξής:

- $A \models$ a : C αν $A \cup$ {a: ¬C} μη συνεπές

Από το τελευταίο, αντιλαμβανόμαστε ότι όλες οι επαγωγικές διαδικασίες του TBox μπορούν να αναχθούν εύκολα σε συνέπεια ABox. Για παράδειγμα, δυο σύνθετες έννοιες C και D είναι ασύμβατες αν, δεδομένου ενός TBox T και ενός υποτιθέμενου[14] ατόμου a, για κάθε διερμηνεία I του ABox A, το $A \cup$ {a : C⊓D} είναι μη συνεπές. Την ιδέα αυτή, δηλαδή την αναγωγή όλων των επαγωγικών διαδικασιών σε εύρεση συνέπειας ABox, χρησιμοποιεί ο ευρύτατα διαδεδομένος Tableau αλγόριθμος που θα δούμε στην Παράγραφο 2.2.1.2.

2.2.1.1 Η μέθοδος της Προσυμπλήρωσης

Η βασική ιδέα της μεθόδου της προσυμπλήρωσης (precompletion) είναι η εξαγωγή όλης της γνώσης που προκύπτει από τις δηλώσεις ρόλων του ABox σε μορφή δηλώσεων εννοιών. Με άλλα λόγια, σκοπός της μεθόδου είναι η κατασκευή ενός, ισοδύναμου με το αρχικό, «ελεύθερου από ρόλους» (role-free) ABox.

Με τη διαδικασία απαλοιφής των δηλώσεων ρόλων, κάθε άτομο της KB απομονώνεται από τα υπόλοιπα, καθότι δε συνδέεται πλέον (μέσω κάποιου ρόλου) με κανένα άλλο. Οι δηλώσεις ρόλων σε μια KB αυξάνουν την πολυπλοκότητα της γνώσης και, κατά συνέπεια, δυσκολεύουν το έργο της ανάλυσής της. Στην περίπτωση που δεν εξάγουμε εξαρχής όλη την πληροφορία που προκύπτει από τις δηλώσεις ρόλων, χρειαζόμαστε πιο πολύπλοκους (και ίσως λιγότερο αποδοτικούς)

[14] Υποτιθέμενο με την έννοια ότι δεν υπάρχει εξαρχής στο ABox της KB, αλλά εισάγεται προσωρινά προκειμένου να αποδειχθεί το ζητούμενο.

αλγόριθμους συλλογιστικής ανάλυσης προκειμένου να χειριστούμε και τις σχέσεις των ατόμων. Αυτό το πρόβλημα ήταν που γέννησε την ιδέα της προσυμπλήρωσης.

Για να γίνει πιο κατανοητό, δίνουμε μια εκδοχή του αλγορίθμου σε γνώση εκφραστικότητας *SHF* [BCG+03]. Ο αλγόριθμος βασίζεται στην εφαρμογή κανόνων (rules) στα αξιώματα της KB έτσι ώστε να απαλειφθούν, εφόσον δε θα χρειάζονται μετά το πέρας της διαδικασίας, οι δηλώσεις ρόλων του ABox. Παρακάτω ισχύουν οι συμβολισμοί που ακολουθήσαμε μέχρι τώρα για τις σύνθετες έννοιες και τους ατομικούς ρόλους. Οι κανόνες για εκφραστικότητα *SHF* είναι:

- $A \rightarrow \{a : C_2\} \cup A$ αν $a : C_1$ περιέχεται στο ABox A, $C_1 \sqsubseteq C_2$ περιέχεται στο TBox και $a : C_2$ δεν περιέχεται στο A. Ο κανόνας αυτός είναι γενικός.
- $A \rightarrow \{a : D\} \cup A$ αν $a : C_1 \cup C_2$ περιέχεται στο A, $D = C_1$ ή $D = C_2$ και καμία από τις δηλώσεις $a : C_1$, $a : C_2$ δεν περιέχεται στο A. Αυτός ο κανόνας χειρίζεται την ένωση εννοιών.
- $A \rightarrow \{b : C\} \cup A$ αν $a : \exists R.C$ περιέχεται στο A, $(a , b) : R_1$ υπάρχει στο A, R_1 απλός ρόλος αντικειμένου, $R \sim_A R_1$ και $b : C$ δεν περιέχεται στο A. Αυτός ο κανόνας χειρίζεται την πλήρη υπαρξιακή ποσοτικοποίηση.
- $A \rightarrow \{b : \forall R_2.C\} \cup A$ αν $a : \forall R_3.C$ περιέχεται στο A, $(a , b) : R_1$ περιέχεται στο A, R_2 μεταβατικός ρόλος, $R_1 \sqsubseteq R_2 \sqsubseteq R_3$ περιέχεται στο TBox και $b : \forall R_2.C$ δεν περιέχεται στο A. Ο κανόνας αυτός χειρίζεται περιορισμούς τιμών για μεταβατικούς ρόλους.
- $A \rightarrow \{a : C_1, a : C_2\} \cup A$ αν $a : C_1 \cap C_2$ περιέχεται στο A και κανένα από τα $a : C_1$, $a : C_2$ δεν περιέχεται στο A. Αυτός ο κανόνας χειρίζεται την τομή εννοιών.
- $A \rightarrow \{b : C\} \cup A$ αν $a : \forall R.C$ περιέχεται στο A, $(a , b) : R_1$ περιέχεται στο A, R_1 απλός ρόλος αντικειμένου, $R_2 \sqsubseteq R$ περιέχεται στο TBox, $R_1 \sim_A R_2$ και $b : C$ δεν περιέχεται στο A. Ο κανόνας αυτός χειρίζεται τους περιορισμούς τιμών.
- $A \rightarrow \{b : C\} \cup A$ αν $a : \forall R.C$ περιέχεται στο A, $(a , b) : R_1$ περιέχεται στο A, R_1 απλός ρόλος αντικειμένου, $R_1 \sqsubseteq R$ περιέχεται στο TBox και $b : C$ δεν περιέχεται στο A. Ο κανόνας αυτός χειρίζεται τους περιορισμούς τιμών.

Εισάγαμε το σύμβολο $\sim_A$ προκειμένου να εκφράσουμε με συντομία μια ιδιαίτερη σχέση μεταξύ δύο ατομικών ρόλων. Το $R_1 \sim_A R_2$ δηλώνει ότι, αν $(x , y) : R_1$ και $(x , z) :$

R_2, τότε y ταυτόσημο του z. Με άλλα λόγια, πρόκειται για το ίδιο άτομο με διαφορετικό όμως όνομα. Όπου στα προηγούμενα χρησιμοποιούμε τον συγκεκριμένο συμβολισμό, εννοούμε ακριβώς αυτό.

Η λογική ορθότητα των κανόνων που παραθέσαμε είναι προφανής. Θα προσπαθήσουμε να εξηγήσουμε με απλά λόγια έναν από αυτούς, τον κανόνα που χειρίζεται περιορισμούς τιμών για μεταβατικούς ρόλους. Υποθέτουμε ότι έχουμε μια KB και ισχύουν τα εξής:

R_2 ατομικός μεταβατικός ρόλος

(a , b) : R_1

$R_1 \sqsubseteq R_2$ και $R_2 \sqsubseteq R_3$

a : $\forall R_3.C$

Βάσει της σημασιολογίας που έχουμε ορίσει, η τελευταία σχέση σημαίνει ότι το άτομο a ανήκει στην έννοια που περιλαμβάνει όλα εκείνα τα άτομα της KB που συνδέονται μέσω του ρόλου R_3 μόνο με άτομα που ανήκουν στην έννοια C. Επομένως, το άτομο a συνδέεται μέσω του R_3 μόνο με άτομο της έννοιας C. Επειδή όμως το άτομο a συνδέεται με το b μέσω του R_1 και ισχύει ότι $R_1 \sqsubseteq R_2$, αυτό σημαίνει ότι το άτομο a συνδέεται με το b μέσω και του ρόλου R_2. Όμως, ο ρόλος R_2 είναι μεταβατικός πράγμα που σημαίνει ότι, αν ισχύουν όλα τα προηγούμενα και υποθέσουμε ότι το b συνδέεται μέσω του R_2 με ένα άτομο c, τότε και το a συνδέεται μέσω του R_2 με το c. Φτάσαμε λοιπόν σε ένα σημείο που για να διατηρείται η συνέπεια της KB πρέπει να ισχύει b : $\forall R_2.C$, δηλαδή το b να συνδέεται μέσω του R_2 μόνο με άτομο της έννοιας C. Διαφορετικά θα προέκυπτε αντίφαση, αφού το άτομο a θα συνδεόταν μέσω του R_3 ($R_2 \sqsubseteq R_3$) με άτομο που δεν ανήκει στην έννοια C.

Στο [TH02] αποδείχθηκε ότι, όποια κι αν είναι οι σειρά με την οποία εφαρμόζονται οι κανόνες, ο αλγόριθμος θα σταματήσει σε πεπερασμένο χρόνο έχοντας υπολογίσει την πλήρη πληροφορία. Ωστόσο, μια προσεκτική στρατηγική στην σειρά εφαρμογής των κανόνων μπορεί να βελτιώσει το συνολικό χρόνο της διαδικασίας.

Όταν ο αλγόριθμος σταματήσει, τα αξιώματα δηλώσεων ρόλων του ABox είναι περιττά και μπορούν να απαλειφθούν, καθότι η πληροφορία που συνεπάγονται

είναι πλέον ρητά δηλωμένη στη ΚΒ με τη μορφή αξιωμάτων δηλώσεων εννοιών. Έτσι, για παράδειγμα, οι τύποι κάθε ατόμου μπορούν να εξαχθούν εύκολα με συνδυασμό των ήδη υπολογισμένων δηλώσεων εννοιών για το συγκεκριμένο άτομο και μιας TBox επαγωγικής διαδικασίας η οποία θα αφορά στην εύρεση σχέσεων υπαγωγής για τις έννοιες που εμφανίζονται σε αυτές τις δηλώσεις. Υπό αυτήν την έννοια, η προσυμπλήρωση μας απαλλάσσει από όλες τις ABox επαγωγικές διαδικασίες. Ο λόγος για τον οποίο δεν επικράτησε είναι επειδή χαρακτηρίζεται από εκθετική πολυπλοκότητα O(exp(n)) εξαιτίας των μη ντετερμινιστικών κανόνων που διαθέτει (π.χ. κανόνας που χειρίζεται την ένωση εννοιών).

2.2.1.2 Tableau Αλγόριθμος

Η σημερινή μορφή των αλγορίθμων Tableau που χρησιμοποιούνται για συλλογιστική ανάλυση γνώσης είναι αποτέλεσμα μακρόχρονης θεωρητικής έρευνας και πρακτικών εφαρμογών[15]. Οι πρώτες απόπειρες μελέτης επαγωγικών διαδικασιών στην Περιγραφική Λογική έκαναν την εμφάνισή τους στα μέσα της δεκαετίας του 1980. Τότε ήταν που εμφανίστηκαν και τα πρώτα προβλήματα πολυπλοκότητας και μη αποφασισιμότητας (undecidability). Ήδη, έχει αποδειχθεί ότι η εκφραστικότητα μιας DL γλώσσας αναπαράστασης είναι αντιστρόφως ανάλογη της ευκολίας της συλλογιστικής ανάλυσης (reasoning) της γνώσης που μοντελοποιεί. Με άλλα λόγια, όσο περισσότερο εκφραστική είναι η γλώσσα αναπαράστασης, τόσο δυσκολότερη, έως και αδύνατη, είναι η πλήρης (sound & complete) αλγοριθμική ανάλυση της γνώσης που μοντελοποιείται.

Η εργασία που παρουσίασε τον πρώτο ολοκληρωμένο Tableau αλγόριθμο για ανάλυση γνώσης μοντελοποιημένης με Περιγραφική Λογική, ήταν αυτή των Schmidt-Schauß και Smolka το 1991. Αφορούσε σε ένα αλγόριθμο εύρεσης σχέσεων υπαγωγής (subsumption algorithm) για γλώσσα αναπαράστασης με εκφραστικότητα *ALC*. Η προσέγγιση που εισήγαγε, πέρα από το ότι είναι αρκετά ενδιαφέρουσα,

---

[15] Η θεωρητική αυτή έρευνα εντάσσεται στις γενικότερες προσπάθειες θεμελίωσης διαδικασιών απόδειξης (proof procedures) στα πλαίσια της μαθηματικής Λογικής, ενώ οι πρακτικές εφαρμογές εστιάζονται στην υλοποίηση Συστημάτων Βάσης Γνώσης (Knowledge Base Systems - KBSs) με δυνατότητες χειρισμού γλωσσών επαρκούς εκφραστικότητας και ταυτόχρονα υψηλή απόδοση συλλογιστικής ανάλυσης.

υιοθετήθηκε από την πλειονότητα των μετέπειτα προσπαθειών ανάπτυξης παρόμοιων αλγορίθμων και, ως εκ τούτου, αξίζει να την περιγράψουμε. Για το σκοπό αυτό θα χρησιμοποιήσουμε ένα απλό παράδειγμα:

Έστω C, D σύνθετες έννοιες και T η καθολική έννοια. Θέλουμε να βρούμε αν το αξίωμα C⊑D αληθεύει, δεδομένου ενός TBox *T* και ενός ABox *A*. Βάσει των όσων αναλύσαμε στα προηγούμενα, το αξίωμα C⊑D ισοδυναμεί[16] με το αξίωμα C⊓¬D $\sqsubseteq \bot$, όπου $\bot$ η κενή έννοια. Ο αλγόριθμος μετασχηματίζει κάθε αξίωμα του *T* κατά τον τρόπο που μόλις περιγράψαμε, φέρνοντάς το επίσης σε κανονική μορφή άρνησης (Negation Normal Form - NNF). Η NNF είναι εκείνη η μορφή των αξιωμάτων στην οποία η σύνθετη άρνηση (¬) εμφανίζεται μόνο μπροστά από ονόματα εννοιών. Αφού γίνει αυτός ο μετασχηματισμός, ο αλγόριθμος προχωράει στην εφαρμογή κανόνων (rules) στα αξιώματα της KB.

Κεντρική ιδέα είναι η κατασκευή μιας διερμηνείας *I* για την οποία το σύνολο $(C \sqcap \neg D)^I$ δε θα είναι κενό. Στην περίπτωση που ο αλγόριθμος κατασκευάσει μια τέτοια διερμηνεία, είναι προφανές ότι η αρχική σχέση C⊑D δεν ισχύει. Αυτή η προσέγγιση, δηλαδή η προσπάθεια να αποδειχθεί ότι ισχύει το αντίθετο από αυτό που καλούμαστε να αποδείξουμε, αποτελεί το βασικό χαρακτηριστικό μιας Tableau διαδικασίας [BCG+03]. Κατά την εφαρμογή των κανόνων, οι δηλώσεις του ABox αντιμετωπίζονται ως περιορισμοί (constraints) οι οποίοι πρέπει να διαδοθούν (propagate)[BS01]. Η διαδικασία σταματά όταν προκύψει αντίφαση (contradiction)[17] και έχουν «δοκιμαστεί» όλες οι πιθανές διερμηνείες *I* ή όταν δεν έχει προκύψει κάτι αντιφατικό και οι κανόνες που περιγράφουμε στη συνέχεια δε μπορούν να εφαρμοστούν περαιτέρω. Διευκρινίζουμε ότι ο αλγόριθμος μπορεί να «παράξει»

---

[16] Με απλά λόγια, αυτό σημαίνει ότι αν αληθεύει ένα από τα δύο αξιώματα, τότε αληθεύει και το άλλο. Ομοίως, αν ένα από τα δύο είναι ψευδές, τότε είναι ψευδές και το άλλο.

[17] Η μόνη περίπτωση αντίφασης σε εκφραστικότητα ALC (και η πιο συνηθισμένη γενικότερα) είναι η ταυτόχρονη ύπαρξη, στο ABox της KB, των δηλώσεων a : A και a : ¬A. Όσο η εκφραστικότητα της γλώσσας αυξάνεται, ορίζονται και άλλου είδους αντιφάσεις, όπως π.χ. η αντίφαση που προκύπτει από μέγιστο περιορισμό αριθμού ≤n R και τις δηλώσεις $(a , b_i) : R$ με $b_i \neq b_j$ για $i,j \in \{0,1,2,\ldots,m\}$ και $i \neq j$ και $m>n$.

άτομα, τα οποία στην ουσία είναι εικονικά, και να τα τοποθετήσει προσωρινά στο ABox προκειμένου να καταλήξει σε ένα μοντέλο *I*. Τα νέα αυτά άτομα αναφέρονται στη βιβλιογραφία ως ανώνυμα (anonymous individuals).

Για την περιγραφή των κανόνων ακολουθούμε τους μέχρι τώρα γνωστούς συμβολισμούς για σύνθετες έννοιες και ατομικούς ρόλους. Η μόνη διαφορά είναι ότι εδώ εισάγουμε τα σύμβολα x, y, y_i (i=1,2,…), z, z_i (i=1,2,…) για την αναπαράσταση ατόμων, επειδή ακριβώς αυτά μπορεί είτε να ανήκουν εξαρχής στο ABox είτε να έχουν παραχθεί σε κάποιο βήμα του Tableau αλγορίθμου (ανώνυμα άτομα). Έχουμε:

- Κανόνας τομής εννοιών

  $A' \rightarrow \{x : C_1 , x : C_2\} \cup A$ αν $x : (C_1 \sqcap C_2)$ περιέχεται στο *A*, αλλά κανένα από τα $x : C_1$, $x : C_2$ δεν περιέχεται στο *A*.

- Κανόνας ένωσης εννοιών(μη ντετερμινιστικός)

  $A' \rightarrow \{x : C_1 \} \cup A$ αν $x : C_1 \, C_2$ και $x : C_1$ δεν περιέχεται στο *A* ή $A'' \rightarrow \{x : C_2\} \cup A$ αν $x : C_1 \, C_2$ και $x : C_2$ δεν περιέχεται στο *A*. Ο κανόνας αυτός δημιουργεί κλαδιά.

- Κανόνας περιορισμού τιμής

  $A' \rightarrow \{y : C\} \cup A$ αν $x : \forall R.C$ περιέχεται στο *A*, $(x , y) : R$ περιέχεται στο *A* και $y : C$ δεν περιέχεται στο *A*.

- Κανόνας υπαρξιακής ποσοτικοποίησης

  $A' \rightarrow \{y : C , (x , y) : R\} \cup A$ αν $x : \exists R.C$ περιέχεται στο *A*, ενώ $z : C$ και $(x , z) : R$ δεν περιέχονται στο *A*. Το y εδώ είναι παραγόμενο άτομο.

Οι παραπάνω τέσσερεις κανόνες εφαρμόζονται με συγκεκριμένη σειρά. Η γενική αρχή συνοψίζεται στο εξής: Ο κανόνας υπαρξιακής ποσοτικοποίησης, επειδή ακριβώς δημιουργεί νέο ανώνυμο άτομο, εφαρμόζεται αφότου έχουν εφαρμοστεί όλοι οι υπόλοιποι κανόνες στα υπάρχοντα άτομα του ABox και εφόσον δεν παραβιάζεται το κριτήριο της φραγής που θα αναλύσουμε στη συνέχεια.

Αυτή η διαδικασία μετασχηματισμού του ABox ονομάζεται συμπλήρωση (completion) και το ABox A' στο οποίο καταλήγει, εφόσον δεν παρουσιαστεί αντίφαση, ονομάζεται συμπληρωμένο ABox (completed ABox) ή ψευδομοντέλο

(pseudomodel) του αρχικού ABox. Λόγω των μη ντετερμινιστικών κανόνων (π.χ. κανόνας ένωσης εννοιών), μπορεί προφανώς να έχουμε περισσότερα του ενός συμπληρωμένα ABoxes για το ίδιο αρχικό (και συνεπές) ABox. Κατ' αντιστοιχία, ένα ABox καλείται ασυνεπές αν και μόνο αν δεν μπορεί να συμπληρωθεί, δηλαδή δε διαθέτει κανένα ψευδομοντέλο. Προς αποφυγή παρεξηγήσεων, τονίζουμε εδώ ότι το αρχικό ABox δεν είναι ισοδύναμο με το ψευδομοντέλο του και, θα μπορούσαμε να πούμε, ότι το συμπληρωμένο ABox αποτελεί μια «εκδοχή» του αρχικού η οποία δεν εμφανίζει αντιφάσεις.

Κάθε ένα από τα πιθανά ψευδομοντέλα ενός συνεπούς ABox αντιστοιχεί σε διαφορετικά «κλαδιά» (που επιλέγονται κατά τη συμπλήρωση) ή, με άλλα λόγια, σε διαφορετική «διαδρομή» στο δέντρο του Tableau αλγορίθμου. Ας γίνουμε όμως πιο συγκεκριμένοι επιστρέφοντας στο παράδειγμά μας:

Έστω C = (∃R.A ⊓ ∃R.B) και D = (∃R.(A ⊓ B)). Βάσει της διαδικασίας που περιγράψαμε και υποθέτοντας ότι το ABox δεν περιλαμβάνει δηλώσεις ούτε για τις ατομικές έννοιες A και B ούτε για το ρόλο R, τα βήματα που θα ακολουθήσει ο Tableau αλγόριθμος είναι:

1. Μετασχηματισμός του αξιώματος (∃R.A ⊓ ∃R.B) ⊑ (∃R.(A ⊓ B)) στο αξίωμα (∃R.A ⊓ ∃R.B) ⊓ ¬ (∃R.(A ⊓ B)) ⊑ ⊥
2. Μετασχηματισμός του νέου αξιώματος σε κανονική μορφή άρνησης:

   (∃R.A ⊓ ∃R.B) ⊓ (∀R.(¬A ⊔ ¬B)) ⊑ ⊥
3. Παραγωγή νέου ατόμου, έστω b, για το οποίο ισχύει η δήλωση:

   b : (∃R.A ⊓ ∃R.B) ⊓ (∀R.(¬A ⊔ ¬B))
4. Εφαρμογή κανόνα τομής εννοιών

   Προσθήκη, στο *A'*, των δηλώσεων:

   b : ∃R.A

   b : ∃R.B

   b : ∀R.(¬A ⊔ ¬B)
5. Εφαρμογή κανόνα υπαρξιακής ποσοτικοποίησης

   Παραγωγή νέων ατόμων, έστω c και d, και προσθήκη, στο *A'*, των δηλώσεων:

   c : A και (b ,c) : R και

d : B και (b ,d) : R

6. Εφαρμογή κανόνα περιορισμού τιμής

   Προσθήκη, στο A', των δηλώσεων:

   c : (¬A ⊔ ¬B)

   d : (¬A ⊔ ¬B)

7. Εφαρμογή κανόνα ένωσης εννοιών

   Προσθήκη, στο A', των δηλώσεων:

   c : ¬B

   d : ¬A

Το 7ο βήμα είναι και το τελευταίο, αφού ο αλγόριθμος κατασκεύασε μια διερμηνεία I που ικανοποιεί το αξίωμα b : (∃R.A Π ∃R.B)Π(∀R.(¬A ⊔ ¬B)) και για την οποία ισχύουν τα εξής:

Δ^I = {b, c, d}

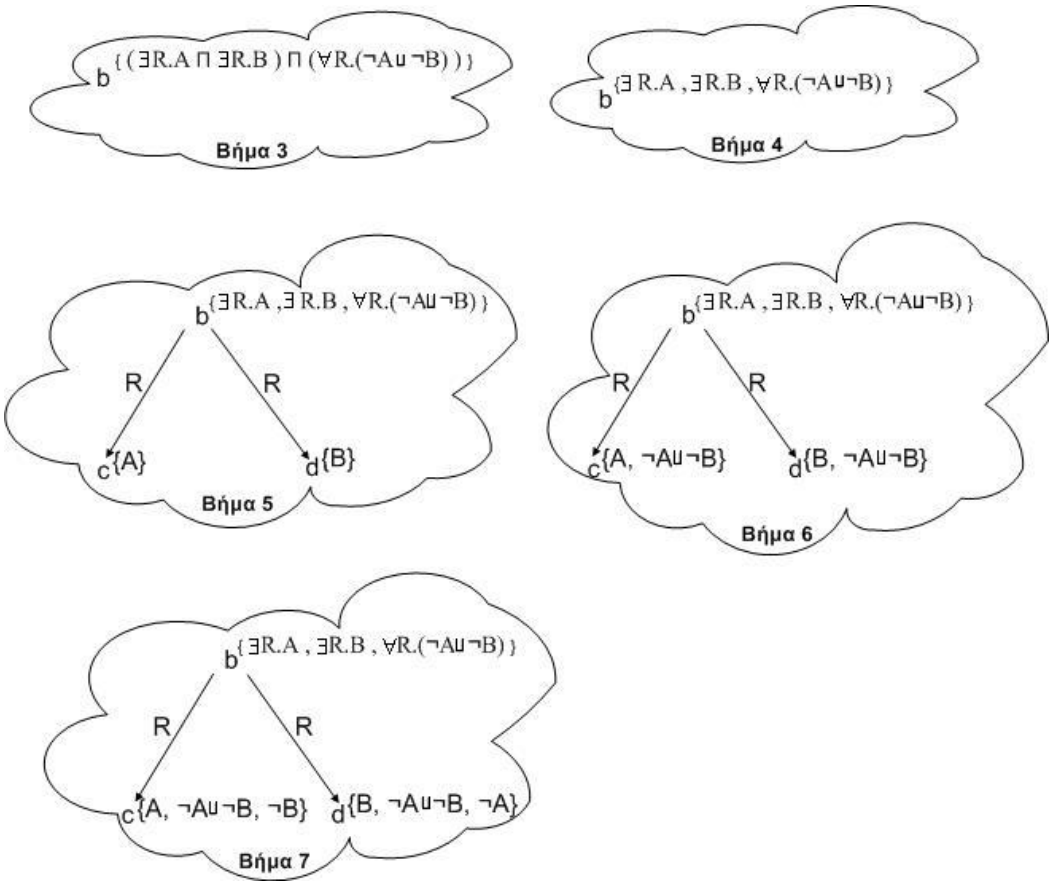
R^I = {(b , c), (b , d)}

A^I = {c}

B^I = {d}

Καταλήγουμε, λοιπόν, στο ότι το σύνολο ((∃R.A Π ∃R.B) Π (∀R.(¬A ⊔ ¬B)))^I δεν είναι κενό και άρα η σχέση C⊑D για την οποία κληθήκαμε να αποφανθούμε δεν ισχύει. Τα παραπάνω βήματα φαίνονται επίσης και στο Σχήμα 2.1. Το δέντρο που σταδιακά σχηματίζεται αντιστοιχεί σε μια γραφική αναπαράσταση αυτού που αποκαλέσαμε πριν δέντρο του Tableau αλγορίθμου. Κάθε κόμβος είναι ένα άτομο του συμπληρωμένου ABox A' και συνοδεύεται από ένα σύνολο εννοιών[18] στις οποίες «οφείλει» να ανήκει βάσει της συγκεκριμένης συμπλήρωσης. Οι ακμές του δέντρου αντιστοιχούν σε ρόλους που συνδέουν τα άτομα μεταξύ τους.

[18] Ο αυστηρός ορισμός αυτών των συνόλων θα δοθεί παρακάτω. Εδώ σημειώνουμε μόνο ότι ένα τέτοιο σύνολο μπορεί σε μια διαφορετική συμπλήρωση του A να αλλάξει.



Σχήμα 2.1: Τα βήματα του Tableau αλγορίθμου

Τονίζουμε ότι ο κανόνας που εφαρμόστηκε στο 7ο βήμα για την ένωση εννοιών είναι μη ντετερμινιστικός και δημιουργεί δύο «κλαδιά», όπως φαίνεται και στο Σχήμα 2.1. Για συντομία, προτιμήσαμε να δώσουμε μόνο τις δηλώσεις που δεν προκάλεσαν αντιφάσεις και που οδήγησαν επιτυχώς στην κατασκευή του μοντέλου *I*. Πράγματι, αν στο 7ο βήμα επιλέγαμε τη δήλωση c : ¬A, τότε θα προέκυπτε αντίφαση με τη δήλωση c : A που είχαμε ήδη προσθέσει στο βήμα 5. Ομοίως, αν για το άτομο d επιλέγαμε, στο 7ο βήμα, τη δήλωση d : ¬B, αυτή θα προκαλούσε αντίφαση με τη δήλωση d : B του 5ου βήματος. Γενικεύοντας, σε παρόμοιες με αυτή περιπτώσεις, δηλαδή όταν προκύψει αντίφαση σε «κλαδί» που εξαρτάται από μη ντετερμινιστικό κανόνα, ο αλγόριθμος επιστρέφει πίσω (backtracking) και διαλέγει άλλο «κλαδί». Προφανώς, αν η αντίφαση που προκύψει δεν εξαρτάται από μη ντετερμινιστικό κανόνα, δηλαδή ο αλγόριθμος δεν έχει περιθώρια «δοκιμής εναλλακτικών διερμηνειών», η διαδικασία σταματάει.

Ωστόσο, ο τερματισμός του αλγορίθμου πρέπει να επιτυγχάνεται και στην περίπτωση που δεν εμφανιστεί κάτι αντιφατικό. Αυτό γίνεται με την εισαγωγή μιας

τεχνικής γνωστής ως φραγή ατόμων (blocking of individuals). Για να κατανοήσουμε τη συγκεκριμένη τεχνική πρέπει πρώτα να εισάγουμε τις εξής έννοιες:

- Σειρά ατόμων (ordering of individuals)

  Συμβολίζεται με < και δηλώνει τη σειρά με την οποία τα άτομα έχουν εισαχθεί στο ABox A κατά τη διαδικασία της συμπλήρωσης. Προφανώς, αν x ανώνυμο άτομο και z άτομο που υπήρχε εξαρχής στο A, τότε z < x.

- Σύνολο εννοιών $\sigma(A_c , x)$

  Πρόκειται για το σύνολο των εννοιών στις οποίες «οφείλει» να ανήκει ένα άτομο x (ανώνυμο ή μη) στο τρέχον βήμα της διαδικασίας συμπλήρωσης. Τα σύνολα στα οποία αναφερθήκαμε προηγουμένως, και που φαίνονται στο Σχήμα 2.1, ανήκουν σε αυτήν την κατηγορία και αντιστοιχούν στο τελικό βήμα της συμπλήρωσης. Ο αυστηρός ορισμός των συνόλων σ είναι:

  $\sigma(A_c , x) = \{ T \} \cup \{C \mid a : C \in A_c\}$, όπου A_c το Abox A στο τρέχον βήμα του αλγορίθμου.

Έστω τώρα x, y δύο άτομα του A. Βάσει των όσων είπαμε στα προηγούμενα, ορίζουμε:

- Αν x < y και $\sigma(A , x) \supseteq \sigma(A , y)$ τότε το y φράσσεται από το x.

Η σημασία της φραγής ατόμων αρχίζει πλέον να γίνεται εμφανής. Αν ο Tableau αλγόριθμος φτάσει σε ένα σημείο που δεν έχει προκύψει αντίφαση και έχουν εφαρμοστεί όλοι οι κανόνες στις δηλώσεις που ορίζει το σύνολο $\sigma(A_c , x)$, τότε δεν χρειάζεται να τους εφαρμόσει και στις αντίστοιχες δηλώσεις του συνόλου $\sigma(A_c , y)$. Αντιθέτως, στο βήμα εκείνο πρέπει να σταματήσει, μιας και η εφαρμογή των κανόνων στο σύνολο των δηλώσεων του «κόμβου» y δεν μπορεί σε καμία περίπτωση να οδηγήσει σε ασυνέπεια του ABox. Αν ίσχυε το αντίθετο, τότε η ασυνέπεια αυτή θα είχε ήδη προκύψει με την εφαρμογή των κανόνων στο υπερσύνολο των δηλώσεων του x. Κατ' αυτόν τον τρόπο, παύεται η άσκοπη δημιουργία ανώνυμων ατόμων και ο αλγόριθμος τερματίζει, αποφεύγοντας την επ' άπειρον επανάληψη της ίδιας διαδικασίας.

Είναι προφανές ότι, αν η εκφραστικότητα της γλώσσας διευρυνθεί πέρα από τα όρια της *ALC*, οι κανόνες που περιγράψαμε στα προηγούμενα δεν αρκούν για την

πλήρη συλλογιστική ανάλυση και πρέπει σε αυτούς να προστεθούν νέοι που θα χειρίζονται τους επιπλέον κατασκευαστές[19]. Στη συνέχεια δίνουμε μόνο ένα απλό παράδειγμα επέκτασης του αλγορίθμου για την περίπτωση που στην *ALC* προσθέσουμε περιορισμούς αριθμού (*N*). Όπως αντιλαμβανόμαστε, η περιγραφή ενός πλήρους Tableau αλγόριθμου για εκφραστικότητα $SHOIN^{(D)}$ είναι αρκετά περίπλοκη και ξεπερνά τους σκοπούς αυτής της ενότητας. Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να ανατρέξει στα [HS05] και [HKS06].

Για εκφραστικότητα *ALCN*, οι κανόνες που προστίθενται στους τέσσερεις προηγούμενους είναι:

- Κανόνας ελάχιστου περιορισμού αριθμού

  $A' \rightarrow \{ (x , y_k) : R_1 \mid k=0,1,...,n \} \cup \{ y_i \neq y_j \mid i,j \in 0,1,\ldots n , i \neq j \} \cup A$ αν

  - $x : \exists_{\geq n} R_1$ υπάρχει στο *A*
  - δεν υπάρχουν στο *A* y_k με $k = 0,1,\ldots,n$ τέτοια ώστε, για $y_i \neq y_j$ με $i,j \in 0,1,\ldots n$ και $i \neq j$, να ισχύουν τα $(x , y_k) : R_2$ με $R_2 \sqsubseteq R_1$

  Όπως είναι φανερό, τα άτομα y_k (k=0,1,...,n) που προστίθενται στο *A* είναι ανώνυμα.

- Κανόνας μέγιστου περιορισμού αριθμού

  $A' \rightarrow A[y_i/y_j]$ αν

  - $y : \exists_{\leq n} R_1$ υπάρχει στο *A*
  - $(x , y_k) : R_2$ υπάρχουν στο *A* με k=0,1,...,m και m>n και $R_2 \sqsubseteq R_1$
  - για $y_i, y_j \in \{ y_1,\ldots, y_m \}$ και $i \neq j$, το $y_i \neq y_j$ δεν υπάρχει στο *A*

  Ο κανόνας αυτός είναι συγχωνευτικός (merge rule), με την έννοια ότι αντικαθιστά κάθε y_i με το y_j. Όπως προκύπτει από τα παραπάνω, υπάρχουν πολλοί διαφορετικοί τρόποι γι' αυτήν την αντικατάσταση και, επομένως, ο κανόνας αυτός είναι μη ντετερμινιστικός, δημιουργεί δηλαδή «κλαδιά».

---

[19] Ενδεχομένως, πέρα από νέους κανόνες, να χρειάζεται να οριστούν και νέα κριτήρια για τη φραγή των ατόμων. Κάτι τέτοιο απαιτείται στην περίπτωση των αντίστροφων ρόλων [HS02].

Τελειώνοντας, τονίζουμε ότι η πολυπλοκότητα του Tableau αλγορίθμου είναι στη χειρότερη περίπτωση εκθετική O(exp(n)) ως προς το μέγεθος n του αρχικού ABox *A*. Ωστόσο, ο τερματισμός του απαιτεί μόνο την εύρεση ενός «κλαδιού» που δεν εμφανίζει αντιφάσεις. Αν σε αυτό προσθέσουμε επίσης και το γεγονός ότι το μήκος αυτών των «κλαδιών» είναι γραμμικό (linear) σε σχέση με το μέγεθος n, αντιλαμβανόμαστε εύκολα το λόγο για τον οποίο ο Tableau αλγόριθμος αποτελεί την έως τώρα επικρατέστερη μέθοδο συλλογιστικής ανάλυσης στην Περιγραφική Λογική.

2.2.2 Τεχνικές συνδυασμού συλλογιστικής ανάλυσης με DBMS

Στο σημείο αυτό και πριν προχωρήσουμε στην ανάλυση επιμέρους τεχνικών, τονίζουμε μια ουσιαστική διαφορά μεταξύ των DL βάσεων γνώσης και των γνωστών σχεσιακών βάσεων δεδομένων (relational DBs). Η διαφορά αυτή καταδεικνύει με τον καλύτερο τρόπο τη σημασία και τις προοπτικές μιας προσπάθειας συνδυασμού συλλογιστικών διαδικασιών με DBMS και, συνήθως, αναφέρεται στη βιβλιογραφία με τη φράση «Ανοιχτός κόσμος εναντίον Κλειστού Κόσμου» (Open world vs Closed world).

Πιο συγκεκριμένα, στις KBs που βασίζονται σε DL, όταν κάτι δεν υπάρχει ρητά (explicitly) με τη μορφή αξιώματος, δε θεωρούμε απαραίτητα ότι δεν ισχύει, όπως στην περίπτωση των σχεσιακών DBs, αλλά μπορεί να εξαχθεί από τα υπάρχοντα αξιώματα μέσω μιας αλγοριθμικής διαδικασίας. Στη συνέχεια, δίνουμε ένα απλό παράδειγμα για καλύτερη κατανόηση.

Έστω μια «κλασσική» σχεσιακή βάση δεδομένων στην οποία αποθηκεύουμε τα στοιχεία των πελατών μιας επιχείρησης και έστω ότι μας απασχολεί η εύρεση των σταθερών τηλεφώνων καθενός από αυτούς. Θεωρούμε ένα SQL ερώτημα που μας επιστρέφει το περιεχόμενο του πεδίου «Αριθμός Σταθερού Τηλεφώνου» του πίνακα «Σταθερό Τηλέφωνο» για ένα άτομο με όνομα Χ. Αν δεν υπάρχει εγγραφή για το άτομο Χ, τότε το ερώτημα δε θα επιστρέψει τίποτα και εμείς θα καταλήξουμε στο ότι τηλέφωνο του συγκεκριμένου ατόμου δεν υπάρχει στη βάση δεδομένων. Από μια άλλη οπτική γωνία, θα μπορούσαμε να πούμε, ότι το ερώτημα «Υπάρχει τηλέφωνο καταχωρημένο για το άτομο Χ στη ΒΔ;» απαντάει «ΟΧΙ», αφού δεν επιστρέφεται τίποτα.

Ας φανταστούμε τώρα την περίπτωση όπου οι πληροφορίες για τους πελάτες της ίδιας επιχείρησης είναι οργανωμένες σε μια ΚΒ. Έστω ότι στο TBox της ΚΒ υπάρχει (ανάμεσα σε άλλα) το αξίωμα:

Σπίτι $\sqcap \exists_{\geq 2}$ Διαθέτει_Σταθερό_Τηλέφωνο.Σταθερό_Τηλέφωνο $\sqsubseteq \bot$ και στο ABox υπάρχουν οι δηλώσεις:

Σ_1 : Σπίτι $\sqcap \exists$ Διαθέτει_Σταθερό_Τηλέφωνο.Σταθερό_Τηλέφωνο

(Χ , Σ_1) : Έχει_Σπίτι

(Υ , Σ_1) : Έχει_Σπίτι, όπου Υ όνομα ατόμου

(Υ , Τ) : Έχει_Σταθερό_Τηλέφωνο, όπου Τ ο αριθμός σταθερού τηλεφώνου

Από τη συλλογιστική ανάλυση της παραπάνω γνώσης μπορούμε να αποφανθούμε εύκολα ότι το τηλέφωνο του ατόμου Χ είναι το Τ, παρότι δε δηλώνεται ρητά στην αρχική πληροφορία που μας έχει δοθεί (closed world). Επομένως, εδώ το ερώτημα «Υπάρχει τηλέφωνο καταχωρημένο για το άτομο Χ στη ΒΔ;» απαντάει «ΝΑΙ» και μάλιστα αυτό είναι το Τ (open world).

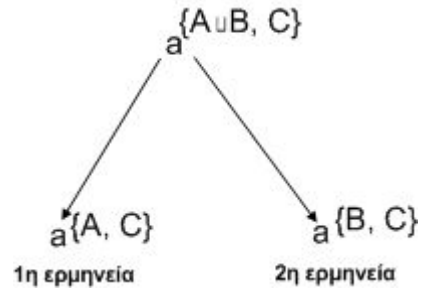
Στην Ενότητα αυτή περιγράφουμε ενδεικτικά δύο ενδιαφέρουσες μεθόδους συνδυασμού συλλογιστικής ανάλυσης με DBMS. Η συνεισφορά τους μπορεί να αξιολογηθεί από δύο διαφορετικές οπτικές γωνίες. Από τη μία, διευρύνουν τις συλλογιστικές διαδικασίες έτσι ώστε αυτές να μπορούν να εκτελούνται και με μηχανισμούς δευτερεύουσας μνήμης και, από την άλλη, εξοπλίζουν τα υπάρχοντα Συστήματα Διαχείρισης Βάσεων Δεδομένων με δυνατότητες απάντησης ερωτημάτων «ανοιχτού κόσμου». Τη μέθοδο των ψευδομοντέλων (Παράγραφος 2.2.2.1) εκμεταλλευτήκαμε κι εμείς στο DBRS, όχι μόνο επειδή προσφέρει στο σύστημα μια σχετική αυτονομία σχετικά με ερωτήματα «ανοικτού κόσμου», αλλά επειδή διευκολύνει επίσης, μέσω της συνεργατικής λειτουργίας DBMS και Reasoner, τη διαδικασία συλλογιστικής ανάλυσης από τον δεύτερο.

2.2.2.1 Ψευδομοντέλα

Ο όρος ψευδομοντέλο (pseudomodel) αναφέρθηκε ήδη στην Παράγραφο 2.2.1.2. όπου περιγράψαμε τον Tableau αλγόριθμο. Πιο συγκεκριμένα, ορίσαμε ως ψευδομοντέλο ενός συνεπούς ABox A το μετασχηματισμένο ABox A' που προκύπτει

από την εφαρμογή των κανόνων του Tableau αλγορίθμου. Διευκρινίσαμε ότι το ψευδομοντέλο *A'* δεν είναι ισοδύναμο με το αρχικό ABox *A*, αλλά αποτελεί μια «εκδοχή» του η οποία δε φέρει αντιφάσεις. Ο όρος «εκδοχή» χρησιμοποιήθηκε επειδή ακριβώς παραπέμπει στη δυνατότητα να δώσουμε σε ένα συνεπές ABox περισσότερες από μία ερμηνείες *I*, οι οποίες είναι διαφορετικές μεταξύ τους και προκύπτουν από τους μη ντετερμινιστικούς κανόνες του Tableau αλγορίθμου. Στην πραγματικότητα, το ψευδομοντέλο του *A* απαρτίζεται από τα επιμέρους ψευδομοντέλα όλων των ατόμων που αυτό διαθέτει, συμπεριλαμβανομένων και των ψευδομοντέλων των ανώνυμων ατόμων που παρήχθησαν από τον Tableau αλγόριθμο. Ας γίνουμε όμως πιο συγκεκριμένοι δίνοντας ένα απλό παράδειγμα:

Έστω μια έννοια $C \equiv A \sqcup B$ με A, B ατομικές έννοιες και ένα άτομο a για το οποίο ισχύει a : C. Η εφαρμογή του μη ντετερμινιστικού κανόνα ένωσης εννοιών οδηγεί σε δύο διαφορετικές ερμηνείες-κλαδιά για το άτομο a που φαίνονται στο Σχήμα 2.2:



Σχήμα 2.2: Δύο εναλλακτικές ερμηνείες για το άτομο a

Η κάθε μια από αυτές αποτελεί ένα ψευδομοντέλο του ατόμου a (individual pseudomodel) το οποίο, με απλά λόγια, προκύπτει από την τυχαία επιλογή της ατομικής έννοιας (A ή B) στην οποία ανήκει το άτομο a, δεδομένου ότι ανήκει εξαρχής στη σύνθετη έννοια $C \equiv A \sqcup B$.

Κατ' αντιστοιχία με το ψευδομοντέλο του ατόμου a, ορίζουμε και το ψευδομοντέλο της έννοιας C (concept pseudomodel). Στην περίπτωση που ο αλγόριθμος επιλέξει την πρώτη από τις παραπάνω ερμηνείες, τότε το ψευδομοντέλο της C είναι η ατομική έννοια A, ενώ στη δεύτερη περίπτωση είναι η ατομική έννοια B.

Όσο η εκφραστικότητα της γλώσσας αναπαράστασης αυξάνεται, τόσο στα ψευδομοντέλα των ατόμων και των εννοιών της KB προστίθενται νέα στοιχεία. Το DBRS χειρίζεται ψευδομοντέλα για εκφραστικότητα *SHN* των οποίων ο αυστηρός ορισμός είναι:

- Ψευδομοντέλο ατόμου (Individual pseudomodel)

  Δεδομένου ενός συμπληρωμένου ABox A', το ψευδομοντέλο ενός ατόμου a (ανώνυμου ή μη) συμβολίζεται με $M = < M^{A}, M^{\neg A}, M^{\exists}, M^{\forall} >$, όπου:

  - $M^{A} = \{A \mid a : A \in A'\}$

    Το σύνολο όλων εκείνων των εννοιών A (σύνθετων ή ατομικών) στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης.

  - $M^{\neg A} = \{A \mid a : \neg A \in A'\}$

    Το σύνολο όλων εκείνων των εννοιών A (σύνθετων ή ατομικών) στις οποίες δεν ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης.

  - $M^{\exists} = \{\exists R.C \mid a : \exists R.C \in A'\} \cup \{\exists_{\geq n} R.C \mid a : \exists_{\geq n} R.C \in A'\}$

    Το σύνολο όλων εκείνων των εννοιών της μορφής $\exists R.C$ ή $\exists_{\geq n} R.C$ στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης. Εδώ με R συμβολίζουμε ατομικό ρόλο και με C σύνθετη ή ατομική έννοια.

  - $M^{\forall} = \{\forall R.C \mid a : \forall R.C \in A'\} \cup \{\exists_{\leq n} R.C \mid a : \exists_{\leq n} R.C \in A'\}$

    Το σύνολο όλων εκείνων των εννοιών της μορφής $\forall R.C$ ή $\exists_{\leq n} R.C$ στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης. Ισχύουν οι ίδιοι συμβολισμοί με προηγουμένως. Στο σύνολο αυτό ανήκουν και οι έννοιες της μορφής $\exists F.C$, όπου F λειτουργικός ρόλος. Οι έννοιες αυτές γράφονται ισοδύναμα ως $\exists_{\leq 1} R.C$, με R = F.

- Ψευδομοντέλο έννοιας (Concept pseudomodel)

  Δεδομένου ενός συμπληρωμένου ABox A' και ενός ατόμου για το οποίο υπάρχει στο αρχικό ABox A (ή προκύπτει σε κάποιο βήμα του Tableau αλγορίθμου) η δήλωση $a : D$[20], το ψευδομοντέλο της έννοιας D συμβολίζεται με $M = < M^{A}, M^{\neg A}, M^{\exists}, M^{\forall} >$, όπου:

[20] Μπορεί για μια έννοια να μην υπάρχουν δηλώσεις στο αρχικό ABox, ούτε να προκύπτουν τέτοιες μετά από την εφαρμογή των κανόνων του Tableau αλγορίθμου. Σε αυτήν την περίπτωση, το άτομο a που αναφέρουμε στον ορισμό του ψευδομοντέλου της έννοιας D είναι παραγόμενο άτομο για το οποίο ο αλγόριθμος θεωρεί ότι ισχύει $a : D$.

- $M^{A} = \{A \mid a : A \in A'\}$

  Το σύνολο όλων εκείνων των εννοιών A (σύνθετων ή ατομικών) στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης.

- $M^{\neg A} = \{A \mid a : \neg A \in A'\}$

  Το σύνολο όλων εκείνων των εννοιών A (σύνθετων ή ατομικών) στις οποίες δεν ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης.

- $M^{\exists} = \{\exists R.C \mid a : \exists R.C \in A'\} \cup \{\exists_{\geq n} R.C \mid a : \exists_{\geq n} R.C \in A'\}$

  Το σύνολο όλων εκείνων των εννοιών της μορφής $\exists R.C$ ή $\exists_{\geq n} R.C$ στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης. Εδώ με R συμβολίζουμε ατομικό ρόλο και με C σύνθετη ή ατομική έννοια.

- $M^{\forall} = \{\forall R.C \mid a : \forall R.C \in A'\} \cup \{\exists_{\leq n} R.C \mid a : \exists_{\leq n} R.C \in A'\}$

  Το σύνολο όλων εκείνων των εννοιών της μορφής $\forall R.C$ ή $\exists_{\leq n} R.C$ στις οποίες ανήκει το a βάσει της συγκεκριμένης συμπλήρωσης. Ισχύουν οι ίδιοι συμβολισμοί με προηγουμένως. Στο σύνολο αυτό ανήκουν και οι έννοιες της μορφής $\exists F.C$, όπου F λειτουργικός ρόλος. Οι έννοιες αυτές γράφονται ισοδύναμα ως $\exists_{\leq 1} R.C$, με R = F.

Στο [HMT01] προτάθηκε μια μέθοδος χρήσης των ψευδομοντέλων για την υπό προϋποθέσεις απάντηση ερωτημάτων «ανοιχτού κόσμου» χωρίς την ανάγκη εκτέλεσης του «ακριβού» Tableau αλγορίθμου. Η μέθοδος αυτή είναι γνωστή ως Έλεγχος Συγχώνευσης (Mergable Test - MT) και μπορεί να εφαρμοστεί σε ερωτήματα ιεραρχίας (subsumption) και τύπων ατόμου (individual types). Στο DBRS υλοποιήσαμε, και για τους δύο προηγούμενους τύπους ερωτημάτων, τον «επίπεδο» Έλεγχο Συγχώνευσης (flat Mergable Test) , τον οποίο και περιγράφουμε στη συνέχεια.

Έστω δύο σύνθετες έννοιες C, D για τις οποίες θέλουμε να ελέγξουμε αν ισχύει το αξίωμα $C \sqsubseteq D$. Θεωρούμε τα ψευδομοντέλα των εννοιών C και $\neg D$, όπως τα ορίσαμε προηγουμένως, τα οποία και συμβολίζουμε με M_1 και M_2 αντιστοίχως. Ο «επίπεδος» Έλεγχος Συγχώνευσης περιλαμβάνει τους παρακάτω ελέγχους:

$M_1^A \cap M_2^{\neg A} = \emptyset$

$M_2^A \cap M_1^{\neg A} = \emptyset$

$M_1^{\exists} \cap M_2^{\forall} = \emptyset$

$M_2^{\exists} \cap M_1^{\forall} = \emptyset$

Αν και οι τέσσερεις είναι αληθείς, τότε το αξίωμα C⊑D δεν ισχύει. Αυτό συμβαίνει γιατί, αν a και b είναι δύο τυχαία άτομα για τα οποία ισχύουν a : C και b : ¬D, τότε οι παραπάνω έλεγχοι «διαβεβαιώνουν» (σε περίπτωση που είναι όλοι αληθείς) ότι τα a, b δε γίνεται στη συγκεκριμένη συμπλήρωση να ανήκουν σε ασύμβατες μεταξύ τους έννοιες. Κατά συνέπεια, αφού δεν προκύπτει αντίφαση, μπορούμε να θεωρήσουμε ότι το a είναι ταυτόσημο του b και να καταλήξουμε στο αξίωμα $\perp \sqsubseteq C \sqcap \neg D$, χωρίς τη χρήση του Tableau αλγορίθμου.

Έστω τώρα ότι θέλουμε να ελέγξουμε αν ισχύει το αξίωμα a : C με χρήση του «επίπεδου» Ελέγχου Συγχώνευσης. Κατ' αντιστοιχία με πριν, θεωρούμε τα ψευδομοντέλα του ατόμου a και της έννοιας ¬C, τα οποία και συμβολίζουμε με M_1 και M_2 αντιστοίχως. Εφαρμόζοντας τους τέσσερεις κανόνες, μπορούμε (σε περίπτωση που είναι όλοι αληθείς) να αποφανθούμε ότι το αξίωμα a : C δεν ισχύει. Η εξήγηση είναι ακριβώς η ίδια με παραπάνω, μόνο που εδώ το ένα από τα δύο άτομα δεν είναι τυχαίο αλλά δοσμένο (a).

Το βασικό πλεονέκτημα του «επίπεδου» Ελέγχου Συγχώνευσης έναντι του Tableau αλγόριθμου είναι η δυνατότητά του να απαντά σε ερωτήματα «ανοιχτού κόσμου» χωρίς την χρήση μη ντετερμινιστικών κανόνων. Όπως έγινε φανερό από τα προηγούμενα, ο Έλεγχος Συγχώνευσης βασίζεται κάθε φορά σε μια μόνο συμπλήρωση-κλαδί και γι' αυτόν ακριβώς το λόγο έχει μικρότερο «κόστος» από τον Tableau, αλλά δεν είναι πλήρης (incomplete algorithm). Πιο συγκεκριμένα, αν ένας τουλάχιστον από τους τέσσερεις ελέγχους δεν είναι αληθής (δηλαδή τα δύο σύνολα έχουν κοινά στοιχεία), τότε ο αλγόριθμος απαντάει FALSE, χωρίς όμως να μπορούμε να το δεχτούμε. Για να πάρουμε τη σωστή απάντηση, πρέπει, στην περίπτωση αυτή, το ερώτημα να υποβληθεί σε Tableau διαδικασία, έτσι ώστε να «ελεγχθούν» όλες οι δυνατές συμπληρώσεις-κλαδιά.

Επιπλέον, ένα βασικό χαρακτηριστικό του «επίπεδου» Ελέγχου Συγχώνευσης (και ο λόγος για τον οποίο τον εντάξαμε εδώ) είναι το γεγονός ότι μπορεί να συνδυαστεί με σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων. Μια τέτοια προσέγγιση εισήχθη αρχικά στο [CHW04] και υλοποιήθηκε, για μεγαλύτερη όμως εκφραστικότητα, στο DBRS. Αποθηκεύοντας, ενδεχομένως κωδικοποιημένα, τα εκάστοτε ψευδομοντέλα ατόμων και εννοιών σε δευτερεύουσα μνήμη, έχουμε τη δυνατότητα να εκτελέσουμε τον «επίπεδο» Έλεγχο Συγχώνευσης με απλά SQL ερωτήματα. Με αυτόν τον τρόπο επιτυγχάνουμε τα εξής:

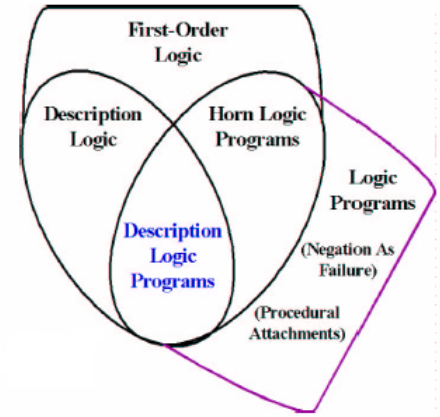
- Δυνατότητα μόνιμης αποθήκευσης (και άρα επαναχρησιμοποίησης) περισσοτέρων του ενός ψευδομοντέλων για ένα άτομο ή μια έννοια[21]. Με απλά λόγια, το DBMS «κρατάει» το «ξεδιπλωμένο» (από τον Tableau αλγόριθμο) ABox και σταδιακά, όσο αυξάνονται τα αποθηκευμένα ψευδομοντέλα, αυξάνονται και οι πιθανότητες απάντησης σε ερωτήματα «ανοιχτού κόσμου» χωρίς τη χρήση Tableau.
- Διευκόλυνση της Tableau διαδικασίας, καθώς σε αυτήν υποβάλλονται μόνο εκείνες οι περιπτώσεις για τις οποίες ο «επίπεδος» Έλεγχος Συγχώνευσης δεν κατάφερε να απαντήσει. Εδώ, το DBMS λειτουργεί ως «φίλτρο».

2.2.2.2 Αντιστοίχηση της Περιγραφικής Λογικής με το Λογικό Προγραμματισμό. Ο φορμαλισμός DLP

Μια αρκετά πρόσφατη ιδέα στις προσπάθειες αναπαράστασης των μεταδεδομένων, με απώτερο στόχο την εύκολη ανάλυσή τους από υπολογιστές, είναι ο συνδυασμός των λογικών φορμαλισμών DL (Description Logics) και LP (Logic Programs), γνωστός και ως Description Logic Programs - DLP. Ο συνδυασμός αυτός έχει νόημα μόνο στο κοινό εκείνο τμήμα των DL και LP (βλ. Σχήμα 2.3) και προϋποθέτει τη δημιουργία ενός μηχανισμού αντιστοίχησης (mapping), έτσι ώστε να

[21] Συνήθως, νέα ψευδομοντέλα προκύπτουν μόνο για άτομα και αυτό όταν ο Tableau αλγόριθμος αλλάξει «κλαδί» μέσω της διαδικασίας που περιγράψαμε στην Παράγραφο 2.2.1.2. Η αλλαγή αυτή μπορεί, σε μια βάση γνώσης (KB), να συμβεί κατά την αποτίμηση ερωτημάτων τύπου ατόμου (individual type).

μπορεί το σύστημα που τον χρησιμοποιεί εσωτερικά να «μεταπηδά» από τον ένα τρόπο αναπαράστασης στον άλλο.



Σχήμα 2.3: Γραφική αναπαράσταση εκφραστικότητας λογικών φορμαλισμών

Οι λόγοι που γέννησαν τον DLP είναι πολλοί. Προφανώς, η επιλογή των δύο φορμαλισμών δεν είναι καθόλου τυχαία. Ο πρώτος (DL) αποτελεί το θεωρητικό υπόβαθρο της γλώσσας OWL που θα δούμε στη συνέχεια, ενώ ο δεύτερος (LP) παρέχει αποδοτικούς επαγωγικούς αλγόριθμους[22] και μπορεί να συνδυαστεί με τη χρήση επαγωγικών Συστημάτων Διαχείρισης Βάσεων Δεδομένων (deductive DBMS). Τα τελευταία εισάγουν μια σειρά από καινοτομίες σε σχέση με τα «κλασσικά» DBMS. Χρησιμοποιούν, για παράδειγμα, γλώσσες Λογικής, όπως η Datalog, τόσο για την αναπαράσταση των δεδομένων όσο και για την υποβολή ερωτημάτων.

Ο DLP καλύπτει πλήρως το RDF Schema καθώς και ένα μικρό μέρος της OWL (OWL Lite και General Concept Inclusion - GCI). Για το μηχανισμό αντιστοίχησης έχουν μέχρι στιγμής προταθεί δύο τρόποι. Δε σκοπεύουμε να τους περιγράψουμε εξαντλητικά παρά μόνο να τονίσουμε τη διαφορετική τους «φιλοσοφία» μέσω παραδειγμάτων. Για περισσότερες λεπτομέρειες βλ. στα [GHVD03] και [WLS03].

- Direct Mapping (D.M)

  Κάθε ορισμός κλάσης ή ιδιότητας του DL αντιστοιχίζεται σε ένα κανόνα (rule) του LP και κάθε αρχικοποίηση κλάσης ή ιδιότητας του DL σε ένα

[22] Ενδεικτικά αναφέρουμε ότι, στο τμήμα definite (χωρίς άρνηση) των LP, η πολυπλοκότητα των επαγωγικών αλγορίθμων είναι O(n). Απ' την άλλη, η DL παρουσιάζει γενικά εκθετική πολυπλοκότητα O(exp(n)) όσον αφορά στους αλγορίθμους επαγωγής.

γεγονός (fact) του LP. Άρα, με C, D έννοιες και Q, P ιδιότητες έχουμε:

- Το $C \sqsubseteq D$ ισοδυναμεί με τον κανόνα: $D(x) \leftarrow C(x)$

- Το $T \sqsubseteq \forall P.C$ ισοδυναμεί με τον κανόνα: $C(y) \leftarrow P(x, y)$

- Το a : D ισοδυναμεί με τον κανόνα (χωρίς σώμα): D(a)

- Το $P \equiv Q$ ισοδυναμεί με τους κανόνες: $Q(x, y) \leftarrow P(x, y)$ και $P(x, y) \leftarrow Q(x, y)$

- Meta Mapping (M.M.)

  Η παραπάνω αντιστοίχηση γίνεται σε ένα μετα-επίπεδο, πράγμα που βοηθά στην υπέρβαση των δυσκολιών που θέτει ο D.M. και που θα αναλύσουμε στη συνέχεια. Εδώ έχουμε:

  - Το $C \sqsubseteq D$ ισοδυναμεί με τη δήλωση: isSub(C, D)

  - Το $T \sqsubseteq \forall P.C$ ισοδυναμεί με τη δήλωση: rhsAllValuesFrom(T, P, C)

  - Το a : D ισοδυναμεί με τη δήλωση: type("a", "D")

  - Το $P \equiv Q$ ισοδυναμεί με τις δηλώσεις: isSub(P, Q) και isSub(Q, P)

Η σύγκριση των D.M. και M.M. έγινε στο [WLS03] με χρήση του επαγωγικού συστήματος CORAL. Η γλώσσα που χρησιμοποιήθηκε για τον ορισμό των δεδομένων και την υποβολή ερωτημάτων ήταν η Datalog. Ως συμπεράσματα της συγκεκριμένης έρευνας παραθέτουμε κομβικά τα εξής:

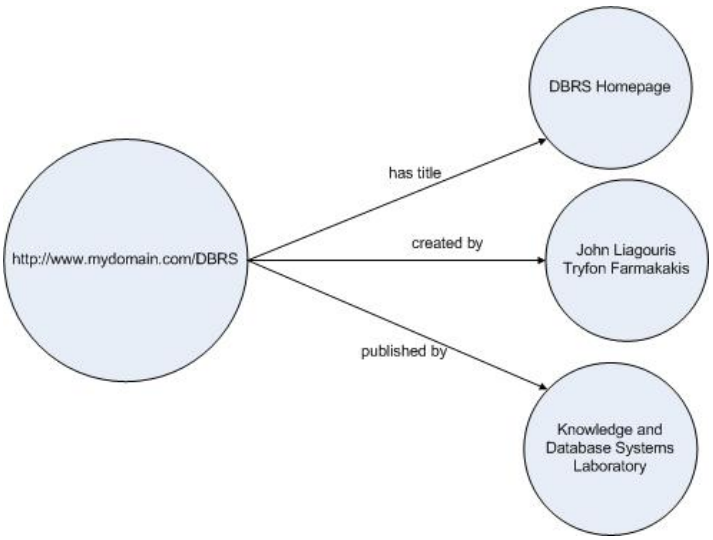
- Στην περίπτωση του M.M. μπορούμε να αναφερόμαστε στην οντολογία έχοντας μόνο μια βασική γνώση γι' αυτήν. Δε χρειάζονται τα ακριβή ονόματα των κλάσεων προκειμένου να έχουμε πρόσβαση στους κανόνες, όπως στον D.M., πράγμα που ισοδυναμεί με περισσότερη ευελιξία στην αναπαράσταση και ευκολία στη συλλογιστική ανάλυση.
- Χρησιμοποιώντας το μηχανισμό αντιστοίχησης M.M., εξασφαλίζουμε ένα σταθερό σύνολο κανόνων (rules) ανεξαρτήτως του μεγέθους της οντολογίας. Μόνο ο αριθμός των γεγονότων (facts) αυξάνεται ανάλογα με τον αριθμό των κλάσεων/ιδιοτήτων και όχι οι κανόνες. Κάτι τέτοιο δεν ισχύει στην περίπτωση του D.M., αφού το σύνολο των κανόνων που απαιτούνται για την περιγραφή της πληροφορίας αυξάνεται γραμμικά σε σχέση με το μέγεθός της.

- Ο χρόνος απάντησης σε ερωτήματα σχετικά με δηλώσεις κλάσεων και σχέσεις υπαγωγής είναι ανάλογος με τον αριθμό των ατόμων και για τις δύο προσεγγίσεις. Ωστόσο, ο Μ.Μ. αναμένεται να έχει καλύτερες επιδόσεις σε συστήματα με μηχανισμούς ευρετηρίων δευτερεύουσας μνήμης (second storage indexing).

2.3 Η γλώσσα RDF/S

Η γλώσσα RDF (Resource Description Framework) [RDF] σχεδιάστηκε αρχικά για τον τυπικό προσδιορισμό μεταδεδομένων σχετικών με τον παγκόσμιο ιστό, αλλά στην πραγματικότητα μπορεί να αναπαραστήσει οποιαδήποτε δεδομένα είτε αυτά είναι μεταδεδομένα είτε όχι. Η δομική μονάδα της γλώσσας είναι η τριάδα Υποκείμενο - Κατηγόρημα - Αντικείμενο (Subject -Predicate - Object) που ονομάζεται δήλωση (statement). Η RDF αναπαριστάται ως κατευθυνόμενος γράφος (directed graph) με δύο κόμβους (nodes) που αντιστοιχούν σε Υποκείμενο και Αντικείμενο και μια ακμή (edge) που ξεκινάει από τον κόμβο του Υποκειμένου και καταλήγει σ' αυτόν του Αντικειμένου. Η ακμή του κατηγορήματος δηλώνει την ύπαρξη μιας σχέσης μεταξύ Υποκειμένου και Αντικειμένου. Τέτοιες σχέσεις μπορεί να υπάρχουν και μεταξύ κόμβων διαφορετικών τριάδων. Για παράδειγμα, το Υποκείμενο μιας δήλωσης μπορεί να είναι Αντικείμενο σε μια άλλη δήλωση μέσω μιας σχέσης κ.ο.κ.

Ένα παράδειγμα περιγραφής διαδικτυακού πόρου με RDF γλώσσα δίνεται στο Σχήμα 2.4.



Σχήμα 2.4: Παράδειγμα διαδικτυακού πόρου που περιγράφεται με RDF

Πέρα από το γραφικό μέρος, η γλώσσα επεκτείνεται με ένα μοντέλο γνωστό ως RDF Schema (RDFS) που επιτρέπει στους χρήστες να ορίσουν ένα συγκεκριμένο λεξιλόγιο (vocabulary) για τα εκάστοτε RDF δεδομένα και μέσω αυτού να τα προσδιορίσουν αυστηρά, να δηλώσουν σχέσεις και να σχηματίσουν, με άλλα λόγια, τη σημασιολογία τους. Στην ουσία, το RDFS αφορά σε ένα σύνολο όρων που μπορούν να χρησιμοποιηθούν για να δομήσουν δεδομένα και να θέσουν περιορισμούς σ' αυτά. Τέτοιοι όροι είναι, για παράδειγμα, οι rdfs:Class, rdfs:subClassOf, rdfs:Property, rdfs:subPropertyOf, rdfs:domain, rdfs:range κ.α.

Η σύνταξη που προτάθηκε, προκειμένου η πληροφορία να μπορεί να γραφτεί και να «διαβαστεί» από υπολογιστές, θυμίζει κατά πολύ την αντίστοιχη μιας άλλης τεχνολογίας, της XML. Η ομοιότητα αυτή δεν είναι καθόλου τυχαία, αφού έτσι εξασφαλίζεται η συμβατότητα που επικαλεστήκαμε στο τέλος της Ενότητας 2.1.

Η περιγραφή του προηγούμενου παραδείγματος διαδικτυακού πόρου σε σύνταξη συμβατή με αυτή της XML γλώσσας δίνεται στη συνέχεια.

```
<?xml version="1.0"?>
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description rdf:about="http://www.mydomain.com/DBRS">
            <dc:title>DBRS Homepage</dc:title>
            <dc:creator>John Liagouris</dc:creator>
            <dc:creator>Tryfon Farmakakis</dc:creator>
            <dc:publisher>Knowledge and Database Systems Laboratory
      </dc:publisher>
      </rdf:Description>
      </rdf:RDF>
```

2.4 Η γλώσσα OWL

Οι έρευνες που οδήγησαν στην κατασκευή της γλώσσας OWL (Web Ontology Language) [OWL]προήλθαν από μια αδιαμφισβήτητη διαπίστωση. Το μοντέλο RDFS έχει περιορισμένη εκφραστικότητα. Ομάδες εργασίας του World Wide Web Consortium (W3C) αναγνώρισαν μια σειρά από περιπτώσεις όπου το RDFS δεν επιδείκνυε επάρκεια και έπρεπε να επεκταθεί. Αυτές είναι:

- Λογικός συνδυασμός κλάσεων (Boolean combination of classes)

  Δε μπορούμε να ορίσουμε κλάσεις ως συνδυασμό ήδη υπαρχόντων κλάσεων. (π.χ. Άνθρωπος ≡ Άντρας ⊔ Γυναίκα)
- Ασυμβατότητα κλάσεων (Disjointness of classes)

  Δε μπορούμε να πούμε ότι οι κλάσεις Άντρας και Γυναίκα είναι ασύμβατες.
- Ειδικά χαρακτηριστικά ιδιοτήτων (special characteristics of properties)

  Δε μπορούμε να δηλώσουμε μεταβατικές ιδιότητες (π.χ. Μεγαλύτερος_από), αντίστροφες ιδιότητες (π.χ. Είναι_μέρος_του και Έχει_μέρος) και λειτουργικές ιδιότητες (π.χ. Έχει_μητέρα).
- Τοπικό εύρος ιδιοτήτων (Local scope of properties)

  Δε μπορούμε να πούμε ότι το εύρος (range) της ιδιότητας Έχει_παιδί είναι Άτομο, όταν αναφέρεται σε ανθρώπους, ή Ελέφαντας, όταν αναφέρεται σε ελέφαντες.
- Περιορισμοί μεγέθους συνόλων (Cardinality restrictions)

  Δε μπορούμε να εκφράσουμε περιορισμούς όσον αφορά στον αριθμό (μέγιστο ή ελάχιστο) των διακριτών ατόμων που μπορούν να συσχετιστούν μέσω μιας ιδιότητας (π.χ. ένα κατάστημα μπορεί να έχει το πολύ 30 υπαλλήλους).
- Ταυτότητα/Διαφορετικότητα ατόμων (Equality/Inequality of individuals)

  Δε μπορούμε να πούμε ότι δύο άτομα με διαφορετικά ονόματα είναι τα ίδια ή να δηλώσουμε ρητά ότι είναι διαφορετικά.

Η υπέρβαση των παραπάνω μειονεκτημάτων και η ταυτόχρονη προσπάθεια για εξασφάλιση αποδοτικής συλλογιστικής ανάλυσης οδήγησαν σε μια πιο «δυνατή» γλώσσα κατασκευής οντολογιών. Αρχικά και για να «κληροδοτηθούν» τα πλεονεκτήματα του RDF/S, προτάθηκαν οι DAML-ONT και OIL. Από αυτές προέκυψε η DAML+OIL [DAML] που με τη σειρά της αποτέλεσε τη βάση για την υλοποίηση της OWL.

Η OWL χρησιμοποιεί σε μεγάλο βαθμό το RDF/S και μπορούμε να πούμε ότι κατά κάποιο τρόπο το επεκτείνει. Αυτό γιατί η OWL, αν και έχει XML-like σύνταξη όπως ο «πρόγονός» της, αυτή είναι σαφώς βελτιωμένη και συνοδεύεται από ένα γραφικό μέρος που υιοθετεί συμβάσεις της UML (Unified Modeling Language) [UML] γλώσσας. Παρέχει ένα σύνολο κατασκευαστών (constructors) όπως τομή

(conjuction), ένωση (disjunction), άρνηση (negation), υπαρξιακή ποσοτικοποίηση (existential quantification), περιορισμούς αριθμών (cardinality restrictions) κ.α. Η παραπομπή στην ορολογία της Περιγραφικής Λογικής είναι προφανής, γι' αυτό και στη βιβλιογραφία η OWL χαρακτηρίζεται ως DL-based. Διαιρείται σε τρεις γλώσσες:

- OWL Full: Ο όρος αυτός αναφέρεται σ' ολόκληρη τη γλώσσα. Μπορεί να θεωρηθεί ως επέκταση του RDFS και του OWL DL μοντέλου που περιγράφουμε στη συνέχεια. Καλύπτει τις αδυναμίες που παραθέσαμε προηγουμένως και ταυτόχρονα επιτρέπει meta-modeling, δηλαδή δηλώσεις σχετικά με το λεξιλόγιο της γλώσσας που απαγορεύονται στα DL. Κάθε RDFS κείμενο είναι ένα έγκυρο (legal) OWL Full κείμενο και επομένως μπορεί να ιδωθεί και να αναλυθεί ως τέτοιο.
- OWL DL: Είναι το μέρος της OWL που θα μας απασχολήσει κυρίως μιας και, σε αντίθεση με την OWL Full, είναι αποφασίσιμο (decidable in finite time) και προσφέρει πλήρη (sound & complete) συλλογιστική ανάλυση. Είναι το τμήμα της OWL Full που αντιστοιχεί στην εκφραστικότητα $SHOIN^{(D)}$ της Περιγραφικής Λογικής και, επομένως, κάθε έγκυρο OWL DL κείμενο είναι ένα έγκυρο OWL Full κείμενο. Πρόσφατα, προτάθηκε μια επέκταση της OWL DL, η OWL 1.1 [OWL1.1], που αντιστοιχεί σε εκφραστικότητα $SROIQ^{(D)}$. Προκειμένου η OWL Full να περιοριστεί στο τμήμα $SHOIN^{(D)}$ της DL, επιβάλλουμε τους εξής περιορισμούς:
    - Κατάτμηση Λεξιλογίου (Vocabulary partitioning)

      Κάθε πόρος του διαδικτύου (web resource) μπορεί να είναι μόνο ένα από τα παρακάτω: κλάση, άτομο, τύπος δεδομένων, ιδιότητα τύπου δεδομένων ή ιδιότητα ατόμου.
    - Ρητή Διατύπωση (Explicit typing)

      Το τι από τα παραπάνω είναι ένας πόρος πρέπει να δηλώνεται ρητά.
    - Διαχωρισμός Ιδιοτήτων (Property separation)

      Μια ιδιότητα τύπου δεδομένων δε μπορεί να είναι ταυτόχρονα και ιδιότητα ατόμου (τα δύο σύνολα είναι ασύμβατα).

- Απαγόρευση περιορισμών μεγέθους συνόλων σε μεταβατικές ιδιότητες (No transitive cardinality restrictions)

  Δε μπορούμε να ορίσουμε περιορισμούς μεγέθους σε σύνολα ατόμων που αντιστοιχούν σε μεταβατικές ιδιότητες.

- Περιορισμός ανώνυμων κλάσεων (Restricted anonymous classes)

  Κλάσεις δίχως όνομα επιτρέπονται μόνο ως πεδίο και εύρος ιδιοτήτων.

- OWL Lite: Πρόκειται για το τμήμα της OWL με τη χαμηλότερη εκφραστικότητα και κατά συνέπεια την ευκολότερη συλλογιστική ανάλυση. Μπορεί να θεωρηθεί ως επέκταση ενός υποσυνόλου του RDFS. Αντιστοιχεί στο τμήμα $SHIF^{(D)}$ της Περιγραφικής Λογικής και επιτρέπει ορισμούς κλάσεων, χαρακτηριστικά ιδιοτήτων, δηλώσεις ιεραρχίας και απλούς περιορισμούς (constraints) με τιμές μηδέν ή ένα. Αποτελεί υποσύνολο της OWL DL και, επομένως, κάθε OWL Lite κείμενο είναι ένα έγκυρο OWL DL κείμενο.

3

Σχετικές Εργασίες

Στο κεφάλαιο αυτό εισάγουμε τον αναγνώστη στην αναγκαιότητα αποθήκευσης και διαχείρισης των οντολογιών με χρήση τεχνολογίας Βάσεων Δεδομένων. Μελετάμε τα επικρατέστερα σχήματα βάσης δεδομένων (database schemas) που έχουν μέχρι στιγμής προταθεί για την αποθήκευση σημασιολογικής πληροφορίας, καθώς και μια σειρά από τεχνικές που τα συνοδεύουν (Forward/Backward chaining, Labeling Schemes). Το δεύτερο μέρος του κεφαλαίου αναφέρεται στα δημοφιλέστερα συστήματα που, με παραλλαγές ή όχι, υλοποιούν μοντέλα και χρησιμοποιούν τεχνικές που περιγράφονται στην Ενότητα 3.1. Επιχειρούμε μια συστηματοποιημένη παρουσίαση των χαρακτηριστικών κάθε συστήματος με απώτερο στόχο την αιτιολόγηση της προσέγγισης που εμείς υιοθετήσαμε κατά τη διάρκεια κατασκευής του DBRS. Συνεπώς, η μελέτη αυτού του κεφαλαίου θεωρείται απαραίτητη για την αξιολόγηση των όσων ακολουθούν στα επόμενα κεφάλαια.

3.1 Σχήματα Βάσης Δεδομένων για αποθήκευση σημασιολογικής πληροφορίας

Στο προηγούμενο κεφάλαιο, αναφερθήκαμε στα μεταδεδομένα και στα μοντέλα ορισμού τους. Παρουσιάσαμε, στην ουσία, τα «εργαλεία» που χρησιμοποιήθηκαν προκειμένου να επιλυθεί το θεμελιακό πρόβλημα που εισήγαγε το όραμα του ΣΙ και που είναι ο τυπικός προσδιορισμός της μετα-πληροφορίας. Πέρα όμως από αυτό, προκύπτουν και μια σειρά από άλλα ζητήματα. Όταν αναφερόμαστε σε πραγματικές εφαρμογές (real-world applications), οι οντολογίες τείνουν να αυξάνουν σε μέγεθος και πολυπλοκότητα, με αποτέλεσμα να μην μπορούν να «φορτωθούν» πλήρως στη μνήμη (λόγω όγκου), αλλά και όταν αυτό είναι δυνατό να μην μπορούν να ελεγχθούν ως προς τη συνέπειά τους (λόγω πολυπλοκότητας που συνεπάγεται αυξημένο κόστος σε κατανάλωση μνήμης).[23] Δεδομένης μάλιστα και της ανάγκης για υποβολή ερωτήσεων στα δεδομένα και γρήγορης αποτίμησης αυτών, η υλοποίηση κατάλληλων μηχανισμών αποθήκευσης και ανάκτησης πληροφορίας από δευτερεύουσα μνήμη κρίνεται απαραίτητη. Στην κατεύθυνση αυτή, δε μπορούσαν παρά να χρησιμοποιηθούν τα αποτελέσματα των πολυετών ερευνών πάνω στη Θεωρία Βάσεων Δεδομένων, με τα Σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων (Relational Database Management Systems - RDBMSs) να αποτελούν σήμερα έναν από τους πιο δημοφιλείς τρόπους οργάνωσης και αποθήκευσης σημασιολογικών σχημάτων.

Η χρήση DBMS, αν και λύνει το πρόβλημα της οργάνωσης μεγάλου όγκου σημασιολογικής πληροφορίας, αυτομάτως γεννά μια σειρά από άλλες προκλήσεις. Οι σημαντικότερες από αυτές αφορούν αφενός στην σχεσιακή αναπαράσταση των σημασιολογικών σχέσεων, αφετέρου στο συνδυασμό των DBMSs με τις συλλογιστικές διαδικασίες που μέχρι στιγμής επιτελούνται στην Κύρια Μνήμη. Όπως έχουμε τονίσει, η αποθήκευση απλώς της σημασιολογικής πληροφορίας δεν είναι αρκετή.

[23] Μια εναλλακτική λύση γι' αυτό θα ήταν η κατάτμηση (partition) της οντολογίας με τρόπο που θα εξασφάλιζε την δυνατότητα μερικής ανάλυσης και σύνθεσης των αποτελεσμάτων στο τέλος. Κάτι τέτοιο, αν και αποτελεί αντικείμενο εντατικής έρευνας, βρίσκεται ακόμα σε εμβρυακό στάδιο. Μια ενδιαφέρουσα προσπάθεια μπορεί να βρεθεί στο [FKM+06].

Πρέπει η τελευταία να υποβάλλεται σε συλλογιστική ανάλυση έτσι ώστε να ελέγχεται η συνέπειά της, αλλά και να εξάγονται οι υπονοούμενες σχέσεις που ενδεχομένως να υπάρχουν. Αυτό, σε περίπτωση οντολογιών μικρού εκφραστικού εύρους, μπορεί να γίνει με χρήση απλών μεθόδων (ενδεχομένως σε SQL) που θα «λαμβάνουν υπόψη τους» τη σημασιολογία, όπως αυτή έχει κωδικοποιηθεί ή αντιστοιχηθεί (mapped) στο σχήμα της βάσης («ελαφριά» ανάλυση - light-weight reasoning). Σε πολύπλοκες όμως OWL οντολογίες και εφόσον επιθυμούμε την καθολική διαχείρισή τους («βαριά» ανάλυση - heavy-weight reasoning), η χρήση μηχανών συλλογιστικής ανάλυσης (Reasoners), και επομένως η συνεργασία του DBMS με αυτούς, είναι αναγκαία. Δύο τεχνικές συνδυασμού «βαριάς» ανάλυσης με DBMS δώσαμε στο προηγούμενο κεφάλαιο (ψευδομοντέλα, DLP). Στην Ενότητα 3.1.2 θα εστιάσουμε στην περίπτωση της «ελαφριάς» συλλογιστικής ανάλυσης, μιας και πολλές τέτοιες διεργασίες επιτελούνται από το DBRS τόσο κατά τη φόρτωση όσο και κατά την αποτίμηση ερωτημάτων.

Τα δημοφιλέστερα μοντέλα που υιοθετούνται, συνήθως με παραλλαγές, για την οργάνωση RDF/S και OWL πληροφορίας σε σχεσιακούς πίνακες (relational tables) είναι:

- Schema Oblivious

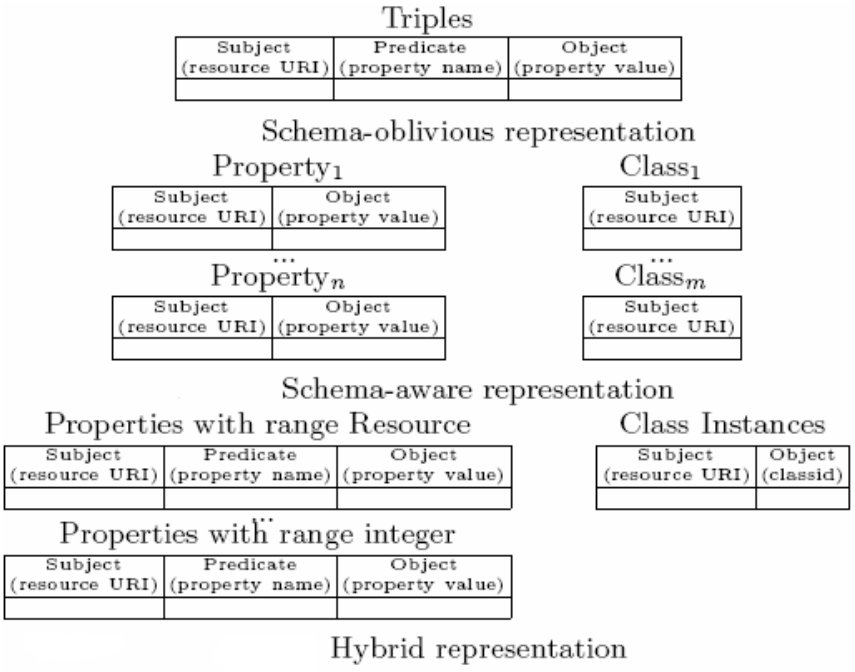
  Η πληροφορία εδώ είναι αποθηκευμένη σε ένα πίνακα με τρεις στήλες. Κάθε γραμμή-τριάδα του πίνακα ισοδυναμεί με μια δήλωση των μοντέλων RDF/S και OWL, ενώ οι στήλες αντιστοιχούν στα Υποκείμενο, Κατηγόρημα και Αντικείμενο. Για την αναπαράσταση των πόρων χρησιμοποιούνται είτε URIs είτε IDs. Το μοντέλο αυτό είναι γνωστό από τη βιβλιογραφία και ως Generic ή Vertical.

- Schema Aware

  Στην περίπτωση αυτή χρησιμοποιείται ένας πίνακας για κάθε ιδιότητα ή κλάση. Οι πίνακες που αντιστοιχούν στις ιδιότητες έχουν δύο στήλες (Υποκείμενο, Αντικείμενο), ενώ οι πίνακες των κλάσεων έχουν μια στήλη (URI του πόρου). Η σχέση υπαγωγής μεταξύ κλάσεων ή ιδιοτήτων άλλοτε δηλώνεται ρητά (ISA) και άλλοτε όχι (NOISA). Το μοντέλο αυτό είναι γνωστό και ως Specific.

- Hybrid model

  Πρόκειται για μοντέλο που συνδυάζει στοιχεία των δύο προηγουμένων. Τα υπάρχοντα συστήματα που υλοποιούν υβριδικό σχήμα βάσης εμφανίζουν διάφορες παραλλαγές. Ένα παράδειγμα αρκετά δημοφιλές είναι αυτό που φαίνεται στο Σχήμα 3.1. Εδώ, χρησιμοποιείται ένας πίνακας για όλες τις κλάσεις και ένας για όλες τις ιδιότητες ατόμων. Ο πίνακας των κλάσεων αποτελείται από δύο στήλες (subject URI, classid) και αντιστοιχίζει URIs ατόμων με αναγνωριστικά κλάσεων. Ο πίνακας των ιδιοτήτων αντικειμένου αποτελείται από τρεις στήλες (subject URI, property name, object URI) και συσχετίζει ζεύγη ατόμων μέσω συγκεκριμένων ιδιοτήτων. Οι ιδιότητες τύπου δεδομένων ομαδοποιούνται βάσει του εύρους (range) και σε κάθε τέτοια ομάδα αντιστοιχεί ένας πίνακας με τρεις στήλες (subject URI, property name, object value). Στο παράδειγμά μας φαίνεται μόνο ο πίνακας για τις ιδιότητες με εύρος ακέραιου αριθμού (integer). Αναλόγως, θα μπορούσε να υπάρχει και ένας πίνακας για τις ιδιότητες με εύρος αλφαριθμητικού (string) κ.ο.κ.



Σχήμα 3.1: Τα δημοφιλέστερα μοντέλα αποθήκευσης σημασιολογικών σχημάτων

Ιδανικό μοντέλο δεν υπάρχει, με την έννοια ότι η καταλληλότητα ενός τέτοιου προκύπτει από τα λειτουργικά γνωρίσματα της συγκεκριμένης εφαρμογής (semantic web application). Κάθε φορά, η επιλογή ενός μοντέλου έπεται της αξιολόγησης μιας σειράς αλληλοσυνδεόμενων χαρακτηριστικών. Αυτά είναι:

- Η εκφραστικότητα των σημασιολογικών σχέσεων που χρειάζεται να αποθηκευτούν
- Οι επιδόσεις του συστήματος τόσο κατά τη φόρτωση της πληροφορίας όσο και κατά τη διεξαγωγή ερωτημάτων
- Το μέγεθος της βάσης που προκύπτει (storage space)
- Η ευελιξία που πρέπει να παρέχεται σχετικά με την ανανέωση των δεδομένων

Για την εκφραστικότητα των σχέσεων που πρέπει να αποθηκευτούν, τα πράγματα είναι προφανή. Όλες οι κατηγορίες μοντέλων μπορούν, με χρήση κατάλληλων τεχνικών, να αποθηκεύσουν γνώση οποιασδήποτε εκφραστικότητας, έστω και αν αυτό γίνεται με μη αποδοτικό τρόπο. Ως χαρακτηριστικό παράδειγμα, δίνουμε την περίπτωση αποθήκευσης δηλώσεων ρόλων αντικειμένου και ρόλων τύπου δεδομένων σε ένα μόνο πίνακα με τρεις στήλες (Schema Oblivious). Η αρχική δυσκολία αποθήκευσης URIs και ακεραίων ή αριθμών κινητής υποδιαστολής σε ένα κοινό πεδίο επιλύεται με την υιοθέτηση τύπου string για το πεδίο αυτό και την μετατροπή (casting) της σταθεράς (literal) στον εκάστοτε τύπο (integer, float, κ.λπ.), όταν αυτό απαιτείται.

Η επίδοση των μοντέλων είναι συνάρτηση πολλών παραγόντων με βασικούς το χρόνο πρόσβασης στα δεδομένα (access time) και τον τρόπο με τον οποίο γίνεται η ανάλυση των σημασιολογικών σχέσεων (inference time). Γενικώς, ο χρόνος πρόσβασης στα δεδομένα μπορεί να θεωρηθεί ανάλογος του μεγέθους των σχεσιακών πινάκων[24]. Επομένως, για μεγάλο όγκο πληροφορίας, ένα Schema Oblivious μοντέλο παρουσιάζει εν γένει μεγαλύτερο χρόνο πρόσβασης έναντι ενός Schema Aware ή Hybrid μοντέλου. Αυτό συμβαίνει γιατί η αποθήκευση όλης της πληροφορίας στην πρώτη περίπτωση γίνεται σε ένα μόνο πίνακα, ενώ στις άλλες ακολουθείται μια κανονικοποιημένη προσέγγιση (περισσότεροι πίνακες με μικρότερο μέγεθος). Επίσης και σε συνδυασμό με το προηγούμενο, η ανάσυρση πληροφορίας στο Schema Oblivous μοντέλο απαιτεί πολλά περισσότερα merge joins, τα οποία ως γνωστόν είναι λιγότερο αποδοτικά από τις συνενώσεις μεταξύ διαφορετικών σχεσιακών

[24] Παραβλέπουμε την επίδραση της χρήσης ευρετηρίων στο γενικό χρόνο πρόσβασης, καθότι αναφερόμαστε σε επίπεδο σχεδίασης των μοντέλων και όχι σε επιμέρους υλοποιήσεις και τεχνολογίες.

πινάκων. Μια ενδιαφέρουσα συγκριτική μελέτη των επιδόσεων των τριών αυτών μοντέλων, που αποδεικνύει και τους παραπάνω ισχυρισμούς μας, δίνεται στο [TCK05].

Όσον αφορά στην ανάλυση των σημασιολογικών σχέσεων, είτε αυτή πρόκειται για «ελαφριά» είτε για «βαριά» συλλογιστική, υπάρχουν δύο τεχνικές με δυνατότητα συνδυασμού τους. Η πρώτη είναι να προϋπολογίσουμε τις υπονοούμενες σχέσεις στο στάδιο της αποθήκευσης (at compile time - backward chaining), ενώ η δεύτερη να τις υπολογίζουμε κατά τη λειτουργία του συστήματος όταν αυτό ζητηθεί (on demand - forward chaining).

Προφανώς, η μέθοδος του προϋπολογισμού απαιτεί περισσότερο αποθηκευτικό χώρο (storage overhead). Ωστόσο, οι επαγόμενες σχέσεις υπολογίζονται όλες μια μόνο φορά στην αρχή και διατηρούνται καθ' όλη τη διάρκεια λειτουργίας του συστήματος. Έτσι, σε συνδυασμό με το ότι αποφεύγεται ο πιθανός επαναϋπολογισμός του μεταβατικού κλεισίματος (transitive closure -TC)[25] για κάθε νέο ερώτημα, το backward chaining καθίσταται ιδανικό για εξαγωγή σχέσεων που ζητούνται συχνά και που η εύρεσή τους είναι χρονοβόρα.

Σε αντιδιαστολή με τα προηγούμενα, το forward chaining προσφέρει περισσότερη ευελιξία στο σύστημα, με τις νέες σχέσεις να υπολογίζονται όταν χρειάζεται χωρίς απαραίτητα να κρατούνται μονίμως στη βάση. Επίσης, σε αυτή την περίπτωση, διευκολύνεται η ενημέρωση των δεδομένων (data update), αφού για τον υπολογισμό των νέων υπονοούμενων σχέσεων το σύστημα δεν απαιτεί επανεκκίνηση, όπως στο backward chaining.

Τα σχήματα βάσης που ανήκουν στην κατηγορία Schema Oblivious υιοθετούν συνήθως την πρώτη προσέγγιση, σε αντίθεση με τα σχήματα των δύο άλλων κατηγοριών (Schema Aware και Hybrid) που υιοθετούν τη δεύτερη. Αυτός είναι και ο βασικός λόγος που τα μεν μπορούν να παρουσιάζουν καλύτερες επιδόσεις στην αποτίμηση ερωτημάτων (οι υπονοούμενες σχέσεις έχουν υπολογιστεί κατά την αρχικοποίηση), με τα δε να διακρίνονται για τους μικρούς χρόνους φόρτωσης (η συλλογιστική ανάλυση γίνεται κατά τη φάση των ερωτημάτων). Βέβαια, υπάρχουν

[25] Η έννοια του μεταβατικού κλεισίματος και η αναπαράσταση του σε σχεσιακές ΒΔ θα μας απασχολήσει ιδιαίτερα στην ενότητα 3.1.1

και συστήματα, όπως θα δούμε στην επόμενη ενότητα, που εκμεταλλεύονται και τις δύο μεθόδους με έναν συνδυαστικό τρόπο.

Μια επιπλέον τεχνική που συνδυάζεται συχνά με backward-chaining και η οποία επηρεάζει τον χρόνο απόκρισης, αλλά και το μέγεθος της βάσης που προκύπτει, είναι οι υλοποιημένες όψεις (materialized views – MatViews). Οι MatViews χρησιμοποιούνται κυρίως για την μόνιμη αποθήκευση της πληροφορίας που προκύπτει από συλλογιστική ανάλυση και ως εκ τούτου αυξάνουν το μέγεθος της βάσης. Μοντέλα που περιέχουν υλοποιημένες όψεις εμφανίζουν, για ευνόητους λόγους, μειωμένο χρόνο απόκρισης και εντάσσονται συνήθως στην κατηγορία Schema Aware.

Μια επίσης πολύ συνηθισμένη περίπτωση που επηρεάζει το χρόνο απόκρισης και το μέγεθος της βάσης με έναν αντιστρόφως ανάλογο τρόπο είναι η αναπαράσταση των πόρων. Όπως ειπώθηκε και προηγουμένως, μπορούν να χρησιμοποιηθούν είτε URIs είτε IDs. Με τα τελευταία κερδίζουμε σε χώρο (τα IDs είναι συνήθως integers), αλλά χάνουμε σε απόδοση, γιατί στο τέλος κάθε ερωτήματος πρέπει να προσθέσουμε ένα επιπλέον ερώτημα συνένωσης (join) για να βρούμε το URI του ζητούμενου πόρου.

Όσον αφορά στην ανανέωση των δεδομένων, τα Schema Oblivious μοντέλα προσφέρουν ιδιαίτερη ευκολία στον διαχειριστή (administrator), καθώς η πρόσθεση ή αφαίρεση μιας κλάσης ή ιδιότητας επιτυγχάνεται με την πρόσθεση ή αφαίρεση των αντίστοιχων τριάδων/εγγραφών στον πίνακα. Από την άλλη, στα Schema Aware, μια τέτοια αλλαγή θα απαιτούσε την εισαγωγή ή διαγραφή ενός πίνακα κάτι το οποίο μπορεί να έχει πολύ μεγαλύτερο κόστος, ιδιαίτερα αν διαθέτει πεδία που αποτελούν ξένα κλειδιά (foreign keys - FK) σε άλλους πίνακες. Σε περίπτωση που επιθυμούμε την αφαίρεση ενός ατόμου, η διαδικασία είναι ίδια και για τα δύο μοντέλα και ισοδυναμεί με τη διαγραφή των αντίστοιχων εγγραφών.

Το σχήμα της βάσης που υλοποιήσαμε στο DBRS ανήκει στην κατηγορία Hybrid. Η βασική ιδέα που ακολουθήσαμε είναι να εκμεταλλευτούμε χαρακτηριστικά και των δύο κατηγοριών (Schema Aware, Schema Oblivious) συνδυάζοντάς τα με τρόπο που θα αύξανε τις επιδόσεις του DBRS τόσο στη φόρτωση (loading phase) όσο και στην αποτίμηση ερωτημάτων (querying phase). Πρόκειται για μια «πολιτική» που έχουν ακολουθήσει και άλλα συστήματα που συναντήσαμε στη βιβλιογραφία και σε πολλές περιπτώσεις αποδείχθηκε επιτυχημένη. Προσπεράσαμε τη «μονολιθική»

προσέγγιση του Schema Oblivious μοντέλου, με τον ένα μόνο πίνακα, όχι μόνο επειδή θέλαμε την καλύτερη δυνατή κανονικοποίηση (normalization) του σχήματος της βάσης, αλλά και επειδή μας ενδιέφερε ένα σχεσιακό σχήμα που θα «πλησίαζε» το μοντέλο οργάνωσης μιας $SHOIN^{(D)}$ οντολογίας. Με απλά λόγια, επιλέξαμε ένα σχήμα που διαχωρίζει τη σημασιολογία (κλάσεις, ρόλοι) από την πληροφορία που θέλουμε εξαρχής να διαχειριστούμε (URIs πόρων). Οι επιλογές μας αυτές βασίστηκαν σε μεγάλο βαθμό και στην εργασία [TCK05].

3.1.1 Τεχνικές αναπαράστασης ιεραρχικής πληροφορίας σε σχεσιακές Βάσεις Δεδομένων

Η αναπαράσταση ιεραρχικής πληροφορίας σε μια σχεσιακή βάση δεδομένων και η αποδοτική διαχείρισή της από το DBMS είναι θέματα που έχουν απασχολήσει έντονα την επιστημονική κοινότητα. Η ιδιαιτερότητά τους έγκειται στην προσπάθεια αντιμετώπισης του προβλήματος υπολογισμού του μεταβατικού κλεισίματος (transitive closure - TC) που είναι απαραίτητο για την πλήρη (complete) αποτίμηση ενός ερωτήματος ιεραρχίας. Αν μάλιστα αναλογιστούμε ότι το μεγαλύτερο μέρος των TBox αξιωμάτων μιας οντολογίας είναι, στην ουσία, σχέσεις ιεραρχίας μεταξύ κλάσεων ή ρόλων, τότε αντιλαμβανόμαστε, αυτομάτως, το λόγο για τον οποίο το πρόβλημα αυτό τίθεται στο επίκεντρο του ενδιαφέροντος.

Το κόστος υπολογισμού του TC εξαρτάται από τον τρόπο με τον οποίο αναπαρίσταται η ιεραρχική δομή, η οποία μπορεί (αφαιρετικά) να έχει είτε τη μορφή δέντρου είτε τη μορφή κατευθυνόμενου ακυκλικού γράφου (Directed Acyclic Graph - DAG), από το μέγεθος και από την «πυκνότητά» της. Το μέγεθος της ιεραρχίας αναφέρεται στο πλήθος των οντοτήτων-κόμβων που υπάρχουν σε αυτήν, ενώ η πυκνότητά της στον αριθμό των ιεραρχικών σχέσεων (σχέσεις υπαγωγής) μεταξύ αυτών των οντοτήτων.

Οι προσεγγίσεις που έχουν προταθεί και που παρουσιάζουμε εδώ εστιάζονται σε πολύ μεγάλες ιεραρχίες, η διαχείριση των οποίων δεν μπορεί να γίνει αποκλειστικά στην κύρια μνήμη του υπολογιστή. Αυτός είναι και ο λόγος για τον οποίο η έρευνα προσανατολίστηκε εξαρχής στην χρήση τεχνολογίας Βάσεων Δεδομένων. Κάποια σύγχρονα αντικειμενο-σχεσιακά Συστήματα Διαχείρισης Βάσεων Δεδομένων, (object-relational DBMSs) προσφέρουν ενσωματωμένες εφαρμογές για την αναπαράσταση και διαχείριση ιεραρχικής πληροφορίας παραδείγματα των οποίων θα δούμε στη

συνέχεια (Ιεραρχία σχεσιακών πινάκων, Αναδρομικές συνενώσεις). Η προσέγγιση, όμως, που θα μας απασχολήσει εκτενώς σε αυτήν την ενότητα είναι τα λεγόμενα Σχήματα Ετικετών (Labeling Schemes). Τα Σχήματα Ετικετών αποτελούν «έξυπνους» τρόπους κωδικοποίησης ιεραρχικών σχέσεων, μέσω ετικετών (labels) που δίνονται (assigned) σε κάθε μια από τις εμπλεκόμενες στην συνολική ιεραρχία οντότητες. Κατασκευάζονται με συγκεκριμένες αλγοριθμικές διαδικασίες και η αποτίμηση ερωτημάτων σε αυτά μπορεί να γίνει με απλά SQL ερωτήματα.

Παρακάτω δίνουμε τις πιο δημοφιλείς μεθόδους αναπαράστασης ιεραρχικών σχέσεων σε σχεσιακές ΒΔ:

Πίνακες ακμών (NOISA)

Αποτελεί μια από τις πιο απλές τεχνικές αναπαράστασης ιεραρχικής πληροφορίας σε ΒΔ. Ένας πίνακας ακμών έχει συνήθως δύο στήλες για την αποθήκευση αναγνωριστικών (IDs). Κάθε εγγραφή του πίνακα αποτελεί, στην ουσία, μια ακμή του ιδεατού γράφου της ιεραρχίας, όντας ένα ζεύγος από αναγνωριστικά. Το ένα από αυτά τα αναγνωριστικά ανήκει σε μια οντότητα της ιεραρχίας, ενώ το άλλο σε μια υπο-οντότητά της (ή υπερ-οντότητά της). Η μέθοδος αυτή, αν και αρκετά απλή στην κατανόηση, παρουσιάζει δυσκολίες ως προς τον υπολογισμό του TC και γι' αυτό το λόγο δεν την προτιμήσαμε στην ανάπτυξη του DBRS.

Ιεραρχία σχεσιακών πινάκων (ISA)

Η δυνατότητα ορισμού ενός πίνακα ως απογόνου ενός ή περισσοτέρων άλλων πινάκων ενσωματώθηκε στο πρότυπο SQL99 και υποστηρίζεται πλέον από πολλά DBMSs (Oracle, PostgreSQL). Η υλοποίηση θυμίζει την κληρονομικότητα όπως την γνωρίζουμε από τον αντικειμενοστρεφή προγραμματισμό. Δίνουμε ένα παράδειγμα: Έστω ότι έχουμε δύο πίνακες Α και Β με σχέση ιεραρχίας «Β υποπίνακας του Α». Το SQL ερώτημα «select * from A» θα μας επιστρέψει, σε αυτήν την περίπτωση, και τις επιπλέον εγγραφές που είναι αποθηκευμένες στον Β και δεν βρίσκονται στον Α.

Σημειώνουμε εδώ ότι η δυνατότητα αυτή δεν θα μπορούσε να χρησιμοποιηθεί στην δική μας περίπτωση, καθώς για να αναπαραστήσουμε, παραδείγματος χάριν, την ιεραρχία των κλάσεων μιας οντολογίας, θα έπρεπε να κατασκευαστούν τόσοι σχεσιακοί πίνακες όσες και οι κλάσεις. Προφανώς, κάτι τέτοιο δεν είναι καθόλου

ρεαλιστικό, επειδή ο αριθμός των κλάσεων σε μια πραγματική οντολογία μπορεί να κυμαίνεται από μερικές δεκάδες έως και μερικά εκατομμύρια.

Αναδρομικές συνενώσεις (joins) σε σχεσιακούς πίνακες

Πρόκειται για μια εφαρμογή που συναντάται, μέχρι στιγμής, μόνο στην πλατφόρμα Oracle. Ο λόγος για την οικογένεια μεθόδων CONNECT_BY [ORC], η οποία μπορεί να χρησιμοποιηθεί για τον υπολογισμό του μεταβατικού κλεισίματος μιας ιεραρχίας που είναι αποθηκευμένη με κατάλληλο τρόπο σε ένα σχεσιακό πίνακα. Όπως αναφέρθηκε και προηγουμένως, κάθε ιεραρχία μπορεί να αναπαρασταθεί με τη βοήθεια ενός γράφου οι ακμές του οποίου δηλώνουν την ύπαρξη σχέσεων υπαγωγής. Προκειμένου λοιπόν να εφαρμοστούν οι μέθοδοι CONNECT BY από την Oracle, η ιεραρχία πρέπει να αποθηκευτεί σε μορφή Πίνακα Ακμών (NOISA). Η μέθοδος λειτουργεί εκτελώντας αναδρομικές συνενώσεις μεταξύ εγγραφών του συγκεκριμένου πίνακα (merge-joins).

Ο βασικός λόγος για τον οποίο δεν εκμεταλλευτήκαμε αυτό το χαρακτηριστικό της Oracle είναι επειδή θέλαμε να επιτύχουμε όσο το δυνατόν μεγαλύτερη ανεξαρτησία από συγκεκριμένα DBMSs. Ως γνωστό, η Oracle είναι ένα εμπορικό σύστημα κλειστού κώδικα πράγμα που σημαίνει ότι, σε καμία περίπτωση, δεν πληρεί τις βασικές απαιτήσεις του DBRS τις οποίες και αναλύουμε στα επόμενα κεφάλαια.

Σχήματα Ετικετών

Η ιδέα πίσω από ένα Σχήμα Ετικετών (Labeling Scheme) είναι η ανάθεση μιας ετικέτας σε κάθε κόμβο του (ιδεατού) γράφου της ιεραρχίας. Με τον όρο ετικέτα αναφερόμαστε σε ένα σύνολο στοιχείων, συνήθως αλφαριθμητικών ή αριθμών, που δίνονται βάσει μιας αλγοριθμικής διαδικασίας και επιτρέπουν τη συμπίεση και αποθήκευση του TC. Παρακάτω δίνουμε τρεις από τις πιο διαδεδομένες κατηγορίες Σχημάτων Ετικετών:

- Σχήματα Bit-Vector
  Η ετικέτα ενός κόμβου αναπαριστάται από ένα διάνυσμα n bits, όπου n είναι ο συνολικός αριθμός των κόμβων του γράφου. Αρχικά, κάθε κόμβος έχει μια ετικέτα γεμάτη από n-1 false bits και ένα true bit σε μια θέση που τον προσδιορίζει μοναδικά μεταξύ των υπολοίπων. Στο πλήρες σχήμα, κάθε

κόμβος κληρονομεί τα bits που χαρακτηρίζουν τους προγόνους (ή απογόνους) του βάσει μιας κωδικοποίησης από πάνω προς τα κάτω (ή από κάτω προς τα πάνω).

Ο αλγόριθμος αυτός προτάθηκε αρχικά από τον N. Wirth [W88] και εξελίχθηκε στη συνέχεια από διάφορους, συμπιέζοντας το μέγεθος των ετικετών, αδυνατώντας όμως να βελτιώσουν την πολυπλοκότητα απάντησης σε ερωτήματα ιεραρχίας σε επίπεδα καλύτερα από O(n). Επιπλέον, το μέγεθος των παραγόμενων ετικετών εξαρτάται άμεσα από το μέγεθος (και τη μορφολογία σε κάποιες εκδοχές σχημάτων) της ιεραρχίας, καθιστώντας τα Bit-Vector σχήματα ακατάλληλα για υλοποίηση σε βάση δεδομένων και πολύ περισσότερο στην περίπτωση που έχουμε αυξητικές ενημερώσεις. Σε αυτή την περίπτωση, το σχήμα ετικετών θα έπρεπε να κατασκευάζεται εξ' αρχής σε κάθε προσθήκη νέου κόμβου.

- Σχήματα Προθέματος (Prefix Schemes)
  Τα σχήματα αυτά, κωδικοποιούν απ' ευθείας τον πατέρα ενός κόμβου σε ένα δέντρο ως πρόθεμα της ετικέτας τους, χρησιμοποιώντας για παράδειγμα μια διάσχιση κατά βάθος. Έτσι οι ετικέτες ενός δέντρου μπορούν να υπολογιστούν σε χρόνο γραμμικό ως προς τον αριθμό των κόμβων του. Έπειτα ο έλεγχος για το αν ένας κόμβος είναι πρόγονος ενός άλλου, γίνεται σε πρακτικά σταθερό χρόνο, ελέγχοντας απλά αν ένα αλφαριθμητικό είναι πρόθεμα του άλλου. Σε αυτό το σχήμα, το μέγεθος της ετικέτας κάθε κόμβου σε bytes είναι ίσο με το επίπεδο του δέντρου όπου βρίσκεται ο κόμβος και, επομένως, το μέγιστο μέγεθος ετικέτας σε bytes εξαρτάται αποκλειστικά από το βάθος του δέντρου.

  Το βασικό πλεονέκτημα των σχημάτων προθέματος είναι η δυναμικότητά τους στον χειρισμό αυξητικών ενημερώσεων. Εφόσον η σειρά μεταξύ των απογόνων δεν έχει σημασία, μπορεί κανείς να προσθέτει καινούριους κόμβους-παιδιά στα δεξιά των υπαρχόντων κόμβων, χωρίς να χρειαστεί να κατασκευάσει ξανά τις ετικέτες τους. Δυστυχώς όμως, η αποτίμηση ερωτημάτων σε ετικέτες μεταβλητών μεγεθών εξαρτάται από μεθόδους χειρισμού αλφαριθμητικών (ειδικά για συμπιεσμένα σχήματα προθέματος), μειώνοντας έτσι τις δυνατότητες βελτιστοποίησης των ερωτημάτων από τους βελτιστοποιητές SQL του DBMS, δεδομένου ότι το κόστος αποτίμησης μεθόδων, ορισμένων από το χρήστη, είναι άγνωστο σε αυτούς. Τέλος, τα

σχήματα προθέματος παράγουν διπλές ετικέτες στην περίπτωση των DAGs, μπορεί δηλαδή να παραχθεί η ίδια ετικέτα παραπάνω από μια φορά, καθιστώντας τα ακατάλληλα για την περίπτωση αυτή και κατά προέκταση για το DBRS.

- Σχήματα Διαστημάτων (Interval Schemes)
  Σε αυτό το σχήμα η ετικέτα ενός κόμβου σε ένα δέντρο αποτελείται από ένα διάστημα (*start*,*end*) τέτοιο ώστε αυτό να περιέχεται στο διάστημα της ετικέτας του πατέρα του. Έτσι, ο έλεγχος για το αν ένας κόμβος είναι απόγονος ενός άλλου μπορεί να γίνει με δύο απλές συγκρίσεις μεταξύ των στοιχείων *start* και *end* των ετικετών τους. Για το αρχικό σχήμα που προτάθηκε από τον Dietz [D82], οι ετικέτες των κόμβων μπορούν να υπολογιστούν σε γραμμικό χρόνο ως προς το μέγεθος του δέντρου, τα ερωτήματα υπαγωγής αποτιμώνται σε σταθερό χρόνο, ο χώρος που απαιτείται για το σύνολο των ετικετών είναι O(n) και το μέγεθος των ετικετών σε bits είναι ακριβώς 2logn.

  Μια παραλλαγή για γράφους προτάθηκε στο [ABJ89] και βασίζεται στην κατασκευή ενός συνδετικού δέντρου (spanning tree). Εδώ, οι ακμές του γράφου χωρίζονται σε αυτές που ανήκουν και σε εκείνες που δεν ανήκουν στο συνδετικό δέντρο. Προτείνεται ένα υβριδικό σχήμα στο οποίο οι ακμές του συνδετικού δέντρου εκμεταλλεύονται πλήρως το σχήμα διαστημάτων, ενώ οι ακμές που δεν ανήκουν σε αυτό, απαιτούν τη διάδοση της ετικέτας του κόμβου-πηγή προς τα πάνω, δηλαδή προς τον κόμβο-τέλος και τους πρόγονούς του. Συγκρινόμενο με το σχήμα του Dietz και άλλες παραλλαγές αυτού, το σχήμα των Agrawal et al. παρουσιάζει τα εξής πλεονεκτήματα:

  i) Απαιτεί μικρότερο μέγεθος ευρετηρίων, καθώς χρειαζόμαστε ένα μόνον b-tree στην δεύτερη τιμή της ετικέτας, σε αντίθεση με το σχήμα του Dietz, όπου χρειάζονται δύο ευρετήρια, ένα για κάθε τιμή της ετικέτας.
  ii) Παρουσιάζει πιο αποδοτική αποτίμηση μέσω standard SQL, καθώς οι βασικές συνθήκες που χαρακτηρίζουν τις δομικές σχέσεις μεταξύ των κόμβων είναι απλούστερες (αντίθετα με τις ετικέτες στο σχήμα πχ. των Li και Moon [LM2001] που περιλαμβάνουν αριθμητικούς υπολογισμούς σε όλα τα ερωτήματα)
  iii) Δίνει δυνατότητες συμπίεσης στη περίπτωση των γράφων, όπου τα

διαστήματα που προέρχονται από ακμές εκτός του συνδετικού δέντρου μπορούν είτε να απορροφηθούν από διαστήματα στα οποία περιέχονται, είτε να ενοποιηθούν με διαστήματα με τα οποία είναι γειτονικά.

Για όλους τους λόγους που προαναφέρθηκαν, η παρούσα έκδοση του DBRS υλοποιεί το σχήμα ετικετών των Agrawal, Borgida και Jagadish για την αναπαράσταση των ιεραρχιών των κλάσεων, των ρόλων αντικειμένου, των ρόλων τύπου δεδομένων, αλλά και για την κωδικοποίηση του TC των δηλώσεων μεταβατικών ρόλων. Η διαδικασία κατασκευής βασίστηκε στους αλγόριθμους που περιέχονται στο [ABJ89]. Για τη μετατροπή των ερωτημάτων ιεραρχίας στα αντίστοιχα SQL ερωτήματα βασιστήκαμε στο [CPST03].

Σημειώνουμε, εδώ, ότι η σχετική αδυναμία που επιδεικνύει το συγκεκριμένο σχήμα ετικετών έναντι άλλων, όσον αφορά στην ανανέωσή του, δεν μας απασχόλησε ιδιαίτερα αφενός γιατί η παρούσα έκδοση του DBRS δεν ασχολείται με ανανεώσεις (updates), αφετέρου (και πιο σημαντικό) γιατί, σε μια πραγματική οντολογία, οι κλάσεις και οι ρόλοι τείνουν να ανανεώνονται σπάνια. Από την άλλη, τα άτομα συνηθίζεται να αλλάζουν σχετικά συχνά, πράγμα που σημαίνει ότι, σε μια τέτοια περίπτωση, η ιεραρχία όσων εξ' αυτών συνδέονται μέσω ενός μεταβατικού ρόλου μπορεί να μεταβάλλεται και να απαιτεί ανανέωση. Μια άλλη περίπτωση, όπου απαιτείται ανανέωση της αντίστοιχης ιεραρχίας, είναι η εμφάνιση νέων (υπονοούμενων) μεταβατικών δηλώσεων, μετά από συλλογιστική ανάλυση. Ένα σύγχρονο σύστημα διαχείρισης οντολογιών οφείλει να χειρίζεται αποδοτικά και τις δύο αυτές περιπτώσεις. Εμείς αντιμετωπίζουμε μόνο τη δεύτερη (με δυνατότητες επέκτασης και στην πρώτη), χρησιμοποιώντας ένα μηχανισμό ανακατασκευής της ιεραρχίας των ατόμων που μετέχουν σε μεταβατικές δηλώσεις, χωρίς να χρειάζεται να «διαβαστεί» εξαρχής το αρχείο της οντολογίας. Περισσότερες λεπτομέρειες θα δώσουμε στα επόμενα κεφάλαια.

3.1.2 Ελαφριά συλλογιστική ανάλυση

Όπως προαναφέρθηκε, ο όρος «ελαφριά» συλλογιστική ανάλυση αναφέρεται σε όλες εκείνες τις διεργασίες που εκμεταλλεύονται είτε την κωδικοποίηση σε επιμέρους πεδία πινάκων είτε το σχεσιακό σχήμα της ΒΔ, στην οποία έχουμε αποθηκεύσει τη γνώση, για να εξάγουν υπονοούμενη πληροφορία. Συνήθως, πρόκειται για απλές, ως

προς την πολυπλοκότητά τους, αλγοριθμικές διαδικασίες που συνδυάζονται με τον «ακριβό» Tableau αλγόριθμο. Στην ενότητα αυτή δεν επιδιώκουμε μια αυστηρή κατηγοριοποίηση των «ελαφριών» διεργασιών, αλλά επειδή ακριβώς το DBRS επιτελεί τέτοιου είδους λειτουργίες τόσο στην φόρτωση όσο και στην αποτίμηση ερωτημάτων, επιθυμούμε να αποσαφηνίσουμε τον όρο με τη χρήση δυο απλών παραδειγμάτων:

Διάδοση Χαρακτηριστικών Ρόλων

Έστω ότι το σχήμα της ΒΔ που χρησιμοποιούμε για την αποθήκευση μιας οντολογίας ανήκει στην κατηγορία Schema Aware. Αυτό σημαίνει, βάσει των όσων έχουμε πει μέχρι στιγμής, ότι για κάθε ρόλο της οντολογίας θα έχουμε ένα ξεχωριστό σχεσιακό πίνακα. Σε κάθε ένα τέτοιο πίνακα, μαζί με άλλα δεδομένα, θα αποθηκεύονται και τα χαρακτηριστικά του ρόλου (role characteristics). Αυτά, στην περίπτωση της $SHOIN^{(D)}$εκφραστικότητας, είναι:

Χαρακτηριστικό
Απλός Ρόλος Αντικειμένου
Μεταβατικός Ρόλος Αντικειμένου (Transitive)
Αντίστροφος Ρόλος Αντικειμένου (Inverse)
Λειτουργικός Ρόλος (Functional)
Συμμετρικός Ρόλος Αντικειμένου (Symmetric)
Ρόλος Τύπου Δεδομένων (Datatype role)

Πίνακας 3.1: Χαρακτηριστικά ρόλων σε εκφραστικότητα $SHOIN^{(D)}$

Έστω τώρα ότι η ιεραρχία των ρόλων δηλώνεται ρητά με σχέσεις υποπινάκων (ISA). Αν λοιπόν στο αρχικό αρχείο της οντολογίας έχουμε ένα λειτουργικό ρόλο ο οποίος έχει υπορόλους, αυτό σημαίνει ότι όλοι του οι υπορόλοι θα πρέπει να είναι επίσης λειτουργικοί, άσχετα με το αν έχουν δηλωθεί ρητά ή όχι στην αρχική οντολογία ως τέτοιοι. Σε περίπτωση που δεν έχουν δηλωθεί, η σχετική πληροφορία μπορεί να εξαχθεί είτε με χρήση reasoner (και άρα tableau αλγόριθμου) είτε με χρήση μιας απλής «ελαφριάς» διεργασίας που θα εκμεταλλεύεται αφενός τα

χαρακτηριστικά των ρόλων, αφετέρου την ιεραρχία τους, όπως αυτή δηλώνεται από το σχήμα της ΒΔ.

Εύρεση Δηλώσεων Ρόλων Αντικειμένου

Το παράδειγμα αυτό αποτελεί μια περίπτωση «ελαφριάς» ανάλυσης περισσότερο περίπλοκη από την προηγούμενη. Έστω ότι το σχήμα της ΒΔ που χρησιμοποιείται για την αποθήκευση της οντολογίας ανήκει στην κατηγορία Hybrid. Ένας πίνακας Α χρησιμοποιείται για την αποθήκευση όλων των ρόλων της οντολογίας και ένας ακόμα (Β) για την αποθήκευση των δηλώσεών τους (role assertions). Θεωρούμε ότι ο πίνακας Α χρησιμοποιεί κάποιο σχήμα ετικετών για την αναπαράσταση της ιεραρχίας των ρόλων και θέλουμε να βρούμε, για ένα ρόλο, όσο περισσότερες δηλώσεις μπορούμε χωρίς να χρειαστεί να ανατρέξουμε στον tableau αλγόριθμο. Σε μια τέτοια περίπτωση μπορούμε στις υπάρχουσες (στον πίνακα Β) δηλώσεις του ρόλου που μας ενδιαφέρει να προσθέσουμε τις εξής:

i) Δηλώσεις των ισοδύναμων ρόλων του δεδομένου ρόλου.
ii) Δηλώσεις των υπορόλων του δεδομένου ρόλου.
iii) Δηλώσεις των αντιστρόφων ρόλων του ρόλου και των ισοδύναμων αυτών (επιστρέφονται ανεστραμμένες).
iv) Δηλώσεις των αντιστρόφων ρόλων των υπορόλων του δεδομένου ρόλου (επιστρέφονται ανεστραμμένες).

Στα παραπάνω μπορούν να προστεθούν και όσες δηλώσεις προκύπτουν από μεταβατικότητα, σε περίπτωση βέβαια που ο ρόλος είναι μεταβατικός και η ιεραρχία των δηλώσεών του αναπαρίσταται στη βάση με κάποιον από τους τρόπους που αναλύσαμε στην Παράγραφο 3.1.1.

Προφανώς, η διαδικασία που περιγράψαμε εκμεταλλεύεται το σχήμα της ΒΔ για να εξάγει υπονοούμενη πληροφορία, χωρίς όμως να είναι πλήρης (complete). Για να έχουμε πλήρη πληροφορία, πρέπει σε κάθε περίπτωση να ανατρέχουμε στον tableau αλγόριθμο. Αυτό δεν είναι πάντοτε επιθυμητό και, όπως θα δούμε στα επόμενα κεφάλαια, ένας από τους βασικούς μας στόχους κατά την ανάπτυξη του DBRS ήταν να εξασφαλίσουμε (και μέσω τέτοιου είδους «ελαφριών» διαδικασιών) μια σχετική «αυτονομία» της ΒΔ όσον αφορά σε ερωτήματα «ανοιχτού κόσμου».

Στον Πίνακα 3.2 συνοψίζονται τα χαρακτηριστικά των τριών δημοφιλέστερων μοντέλων οργάνωσης σημασιολογικών σχημάτων, βάσει των όσων αναλύσαμε σε ολόκληρη την Ενότητα 2.1.

	Schema Oblivious	**Schema Aware**	**Hybrid Models**
Εξαγωγή Υπονοούμενων Σχέσεων	backward/forward (συνήθως backward)	backward/forward (συνήθως forward)	backward/forward (συνήθως forward)
Αναπαράσταση Ιεραρχίας	NOISA	ISA/NOISA	NOISA Labeling Schemes
Χρήση Υλοποιημένων Όψεων	-	Συνηθίζεται σε συνδυασμό με backward chaining	-
Αναπαράσταση Πόρων	IDs/URIs	URIs	IDs/URIs
Διαγραφή/Εισαγωγή Κλάσεων ή Ρόλων	Διαγραφή/Εισαγωγή Εγγραφών	Διαγραφή/ Εισαγωγή Πίνακα	Διαγραφή/Εισαγωγή Εγγραφών
Μέγεθος Βάσης - Αριθμός Πινάκων	Εξαρτάται από τον τρόπο αναπαράστασης των πόρων (IDs/URIs) και από τη χρήση backward chaining Ένας βασικός πίνακας τριάδων	Εξαρτάται από τη χρήση MatViews Ο αριθμός των πινάκων εξαρτάται από το πλήθος των κλάσεων και των ρόλων	Εξαρτάται από τον τρόπο αναπαράστασης των πόρων (IDs/URIs) και από τη χρήση backward chaining

Πίνακας 3.2: Συγκεντρωτικά χαρακτηριστικά τριών μοντέλων οργάνωσης σημασιολογικών σχημάτων

3.2 Συστήματα Διαχείρισης Οντολογιών

Καθένα από τα συστήματα που περιγράφουμε εμφανίζει μια σειρά από ιδιαιτερότητες τόσο στο μηχανισμό αποθήκευσης που υλοποιεί εσωτερικά όσο και στις επιμέρους «τεχνικές» που υιοθετεί προκειμένου να επιτύχει τους σκοπούς για τους οποίους κατασκευάστηκε. Ορισμένα θεωρούνται ήδη ξεπερασμένα ή τουλάχιστον μη εξελίξιμα. Ωστόσο, κυρίως μέσω των «ατελειών» τους, τροφοδότησαν μια σειρά από μεταγενέστερες έρευνες και, επομένως, η αναφορά σε αυτά συνθέτει μια πολύ καλή εικόνα της πορείας αυτών των ερευνών.

Η παρουσίαση που ακολουθεί ομαδοποιεί τα συστήματα ανάλογα με το εύρος της πληροφορίας που μπορούν να διαχειριστούν. Αρχικά παραθέτουμε εκείνα που περιορίζονται στο RDF/S μοντέλο (Ενότητες 2.2.1, 2.2.2, 2.2.3) και στη συνέχεια επεκτεινόμαστε σε υλοποιήσεις που υποστηρίζουν μέρος της OWL. Στόχος μας είναι η ανάδειξη του τρόπου με τον οποίο γίνεται:

- Η φόρτωση των δεδομένων στο μηχανισμό αποθήκευσης
- Η οργάνωση των δεδομένων
- Η εξαγωγή των υπονοούμενων σχέσεων
- Η υποβολή ερωτημάτων και η εξαγωγή των αποτελεσμάτων

έτσι ώστε να φανούν στην συνέχεια οι βασικές ομοιότητες και κυρίως οι διαφορές με το DBRS.

Σημειώνουμε, εδώ, ότι στο τέλος της ενότητας γίνεται αναφορά σε δύο πρωτότυπες πλατφόρμες που, αν και δεν χρησιμοποιούν DBMS, παρουσιάζουν ιδιαίτερο ενδιαφέρον και περισσότερο αποτελούν γενικές (generic) αρχιτεκτονικές συστημάτων παρά συγκεκριμένα συστήματα.

3.2.1 Η πλατφόρμα RDFSuite

Η πλατφόρμα RDFSuite [ACK+00] αναπτύχθηκε με σκοπό την αποδοτική αποθήκευση και διαχείριση μεγάλου όγκου RDFS δεδομένων. Αποτελείται από τέσσερα βασικά μέρη:

- Έναν RDF αναλυτή (Validating RDF Parser - VRP)
- Ένα μηχανισμό φόρτωσης της RDF πληροφορίας (RDF Description Loader) στη βάση δεδομένων
- Ένα Object Relational DBMS (ORDBMS), το PostgreSQL
- Ένα μηχανισμό υποβολής και βελτιστοποίησης ερωτημάτων που χρησιμοποιεί τη γλώσσα RQL [KAC+02] (RDF Query Language)

Το σύστημα είναι «γραμμένο» σε γλώσσα προγραμματισμού Java (με εξαίρεση τον μεταφραστή της RQL που είναι σε C++) και η επικοινωνία μεταξύ των διακριτών μονάδων του γίνεται μέσω κατάλληλα διαμορφωμένων διαπροσωπιών (APIs - Aplication Program Interfaces). Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει σχεδόν όλο το RDF/S μοντέλο, έχοντας μόνο ένα βασικό περιορισμό (constraint) προς αποφυγή σημασιολογικών ασυνεπειών: Το πεδίο και εύρος των ιδιοτήτων πρέπει πάντα να είναι ορισμένο (defined) και μοναδικό (unique).

Σ' αυτόν προστίθεται και ένας επιπλέον περιορισμός συντακτικής φύσης. Οι ορισμοί των εννοιών και των ιδιοτήτων πρέπει να είναι πλήρεις (complete). Με άλλα λόγια, οι δηλώσεις superclass και superproperty πρέπει να συνοδεύουν τους ορισμούς των εννοιών και ιδιοτήτων αντίστοιχα, ενώ το πεδίο και το εύρος πρέπει να δίνεται μαζί με τον ορισμό της ιδιότητας.

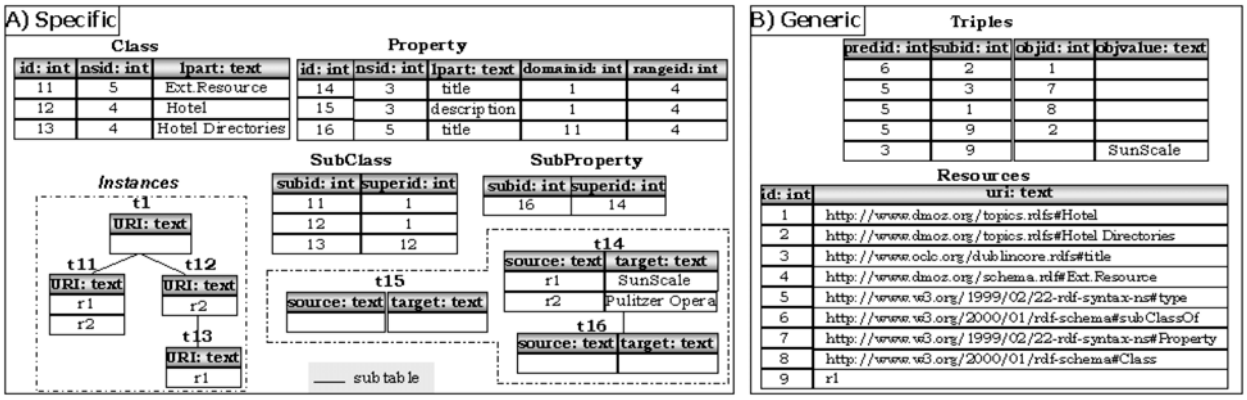
Ο αναλυτής VRP ελέγχει την εγκυρότητα των προς αποθήκευση δεδομένων, αλλά και του ίδιου του σχήματος, βάσει των περιορισμών του RDF/S μοντέλου και αυτών που επισημάναμε πιο πάνω. Παρέχει ιδιαίτερη ευελιξία στους χρήστες, δεδομένου ότι η γραμματική του μπορεί εύκολα να επεκταθεί και να συμπεριλάβει νέα χαρακτηριστικά. Η λειτουργία του βασίζεται σε:

- Έναν αλγόριθμο που μεταφράζει την ορισμένη με «XMLοειδή» σύνταξη πληροφορία στο γνωστό μοντέλο τριάδων (triple-based model) της RDF.
- Μια εσωτερική αναπαράσταση αυτού του μοντέλου σε Java έτσι ώστε να διαχωρίζεται το σχήμα (schema) από τα ίδια τα δεδομένα (data information). Αυτός ο διαχωρισμός διακρίνεται και στο σχήμα της βάσης και είναι η ουσιαστικότερη καινοτομία του συστήματος.

Ο Description Loader μηχανισμός επιβαρύνεται με την αποθήκευση των δεδομένων στο PostgreSQL. Έχει δύο χαρακτηριστικές λειτουργίες:

- Ελέγχει τη συνέπεια του σχήματος αναφορικά με την πληροφορία που έχει αποθηκευτεί στη βάση. Για παράδειγμα, αν μια ιδιότητα έχει ήδη «φορτωθεί», ελέγχει αν τα domain και range της είναι ίδια με αυτά που έχουν αποθηκευτεί στο PostgreSQL.) Έτσι, εγκρίνει τα RDF μεταδεδομένα βάσει του σχήματος (και όχι των namespaces που θα δούμε παρακάτω) μειώνοντας τις απαιτήσεις του συστήματος σε κύρια μνήμη (main memory).
- Φορτώνει τα δεδομένα στη βάση. Αυτό γίνεται με μια σειρά μεθόδων (methods κατά το αντικειμενοστρεφές πρότυπο) που υλοποιούνται ως μέλη των κλάσεων του VRP. Πιο συγκεκριμένα, για κάθε κλάση του VRP δημιουργείται μια μέθοδος που φορτώνει την πληροφορία στο ORDBMS.

Το RDFSuite, ενώ μπορεί να χρησιμοποιήσει και Schema Oblivious μοντέλο, υιοθετεί ένα ιδιόμορφο σχήμα βάσης που εκμεταλλεύεται χαρακτηριστικά κυρίως του Schema Aware, αλλά και του Hybrid μοντέλου. Επιλέχθηκε ύστερα από μια εκτενή σύγκριση με τη Generic προσέγγιση. Υπάρχει ένας πίνακας για κάθε κλάση και ένας για κάθε ιδιότητα. Οι πίνακες των κλάσεων αποτελούνται από μία στήλη (URI του πόρου), ενώ οι πίνακες των ιδιοτήτων από δύο (URIs των source και target πόρων). Η πληροφορία σχετικά με το σχήμα (schema) του RDF/S μοντέλου αποθηκεύεται σε τέσσερεις βασικούς πίνακες: Class, Property, SubClass και Subproperty (NOISA). Για εξοικονόμηση χώρου, χρησιμοποιείται ένας πίνακας NameSpace ο οποίος αποθηκεύει τα namespaces του RDF σχήματος που φορτώνεται στο ORDBMS. Επιπλέον, ένας πίνακας Type περιέχει τους built-in Types (rdf:Property, rdfs:-Class κ.λπ.), τους Containers (rdf:Bag, rdf:Seq κ.λπ.) και τους Literal Types (π.χ. string, integer κ.λπ.). Όλα τα παραπάνω φαίνονται καλύτερα στο Σχήμα 3.2.



Σχήμα 3.2: Τα εναλλακτικά σχήματα της βάσης δεδομένων του RDFSuite

Για την υποβολή και βελτιστοποίηση ερωτημάτων πρέπει, προφανώς, η γλώσσα που χρησιμοποιεί το RDFSuite (RQL) να μπορεί να μεταφραστεί σε SQL. Αυτό επιτυγχάνει ο RQL Interpreter που συνίσταται από τρία βασικά μέρη:

- Έναν αναλυτή (parser) που εξετάζει τη σύνταξη των ερωτημάτων (queries)
- Έναν κατασκευαστή γράφων (graph constructor) που συλλέγει τη σημασιολογία των ερωτημάτων
- Μία μηχανή αποτίμησης (evaluation) η οποία έχει πρόσβαση στη βάση δεδομένων

Η βελτιστοποίηση (optimization) των ερωτημάτων γίνεται με τη βοήθεια του PostgreSQL, καθώς βασικό χαρακτηριστικό του RQL Interpreter είναι να «ωθεί» όσο περισσότερο γίνεται την αποτίμηση στην SQL3 μηχανή. Εκμεταλλεύεται έτσι τους αποδοτικούς αλγορίθμους βελτιστοποίησης του ORDBMS τους οποίους και συνδυάζει με την ευελιξία της σύνταξης της RQL. Για την τελευταία δε θα δώσουμε περισσότερες πληροφορίες. Τονίζουμε απλώς εδώ ότι την προσέγγιση αυτή ακολουθήσαμε κι εμείς στο DBRS όσον αφορά στην βελτιστοποίηση των ερωτημάτων.

3.2.2 Το σύστημα 3Store

Το σύστημα 3Store [HG03] κατασκευάστηκε με σκοπό την αποθήκευση και διαχείριση πολύ μεγάλων οντολογιών της τάξης των δεκάδων εκατομμυρίων δηλώσεων (RDF Statements). Προσανατολίζεται στην αλληλεπίδραση με το χρήστη μέσω διαδικτύου. Έχει ιδιαίτερα απλή δομή και διαιρείται στα εξής μέρη:

- Ένα μηχανισμό φόρτωσης της πληροφορίας στη βάση δεδομένων
- Ένα DBMS (MySQL)
- Ένα μηχανισμό υποβολής ερωτημάτων (RDQL Engine) και εξαγωγής των αποτελεσμάτων

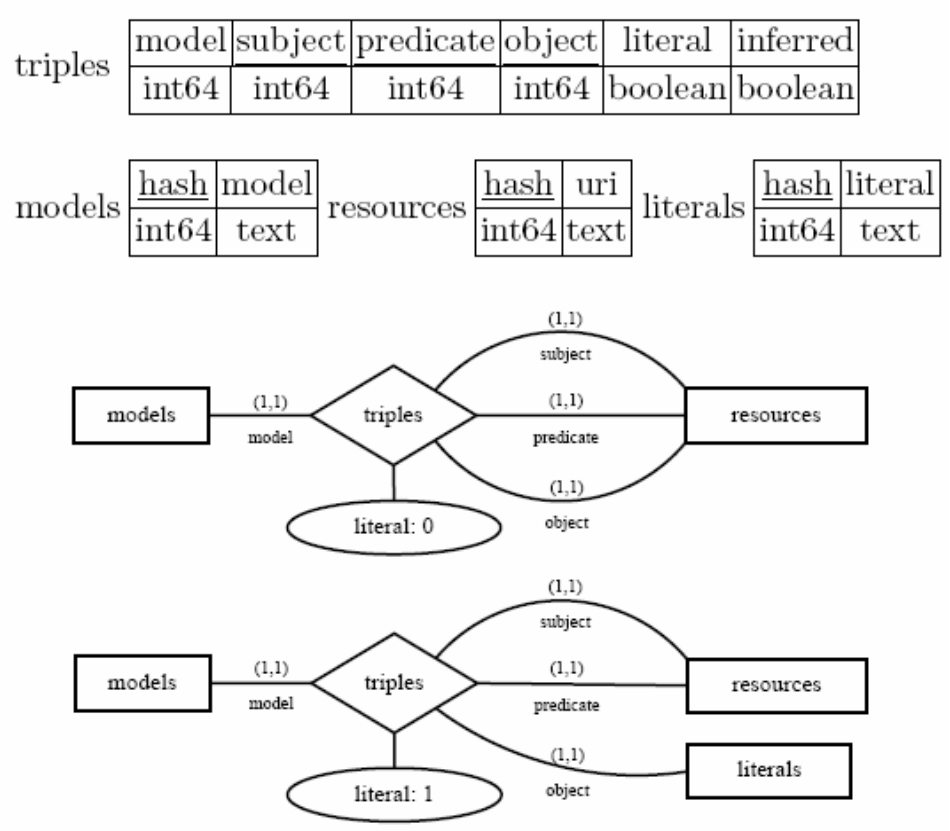
Το σύστημα είναι γραμμένο εξολοκλήρου σε γλώσσα προγραμματισμού C. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο. Η αλληλεπίδραση του συστήματος με τον χρήστη γίνεται μέσω μιας διαπροσωπίας, του OKBC, που είναι συμβατό με το HTTP.

Ο μηχανισμός φόρτωσης των δεδομένων στη βάση χρησιμοποιεί ένα αναλυτή (Raptor) που είναι ήδη ενσωματωμένος σε ένα άλλο σύστημα, το RedLand [Redland]. Ο αναλυτής αυτός διαβάζει το RDF/S αρχείο που βρίσκεται σε XML μορφή και κατασκευάζει ένα μοντέλο τριάδων που μπορεί να αποθηκευτεί εύκολα με χρήση SQL μεθόδων.

Το σχήμα της βάσης που υιοθετήθηκε ανήκει στην κατηγορία Oblivious, έχοντας όμως κάποιες μικρές παραλλαγές. Η πληροφορία εισάγεται σε έναν πίνακα τριάδων με έξι στήλες. Η πρώτη αποθηκεύει το αναγνωριστικό της οντολογίας και οι τρεις επόμενες περιλαμβάνουν τα IDs των Υποκείμενο, Κατηγόρημα και Αντικείμενο. Υπάρχουν ακόμα άλλες δύο στήλες: Μία που δηλώνει αν το αντικείμενο είναι literal (σταθερά) και μια που δηλώνει αν η συγκεκριμένη τριάδα δηλώθηκε ρητά ή προέκυψε από επαγωγή. Τρεις επιπλέον πίνακες χρησιμεύουν για την αντιστοίχιση των IDs με τα αντίστοιχα ονόματα (text). Έχουμε έναν πίνακα models που αντιστοιχίζει IDs με ονόματα οντολογιών (URLs), έναν πίνακα resources που αντιστοιχίζει IDs με URIs πόρων (άτομα/κλάσεις/ιδιότητες) και έναν πίνακα literals που αντιστοιχίζει IDs με σταθερές (literals). Όλα αυτά, μαζί με το ER διάγραμμα, φαίνονται στο Σχήμα 3.3.

Για την αποδοτική εύρεση των URLs/URIs χρησιμοποιείται μια 64-bit integer hush function. Σε περίπτωση που κάποιος πόρος έχει το ίδιο όνομα με ένα αλφαριθμητικό, το πεδίο literal (boolean) του πίνακα triples προσδιορίζει τη «φύση» του δεδομένου. Έτσι αποφεύγονται πιθανές συγκρούσεις (collisions).

Τα ερωτήματα υποβάλλονται από το χρήστη σε RDQL [RDQL] γλώσσα. Ο μηχανισμός ερωτημάτων τα μετατρέπει σε SQL μεθόδους αναγνωρίσιμες από το DBMS. Η διαδικασία αυτή είναι αρκετά απλή καθότι η RDQL και SQL έχουν παρόμοια σύνταξη.



Σχήμα 3.3: Το σχήμα της βάσης δεδομένων του 3Store

Ιδιαίτερο ενδιαφέρον παρουσιάζει η διαδικασία επαγωγής πάνω στα RDF/S δεδομένα. Οι μέθοδοι backward chaining και forward chaining αναφέρονται εδώ ως Eager Production και Lazy Production αντίστοιχα. Το 3Store υιοθετεί ένα υβριδικό τρόπο επαγωγής. Άλλοτε χρησιμοποιεί την Eager Production (όπου η διαδικασία επαγωγής είναι χρονοβόρα και τα αποτελέσματα χρησιμοποιούνται συχνά) και άλλοτε την Lazy Production (όπου η επαγωγή είναι εύκολη ενώ τα αποτελέσματα χρησιμοποιούνται σπάνια).

3.2.3 Το σύστημα RStar

Το σύστημα RStar [MSP+04] κατασκευάστηκε για την αποδοτική αποθήκευση και διαχείριση πολύ μεγάλων οντολογιών (RDF data sets). Προσανατολίζεται για εμπορικές εφαρμογές στις οποίες η αλληλεπίδραση με το χρήστη θα γίνεται μέσω διαδικτύου (client-server architecture). Αποτελείται από τα εξής μέρη:

- Ένα μηχανισμό φόρτωσης της πληροφορίας στη βάση (Data Loader)
- Ένα RDBMS (IBM DB2)
- Ένα μηχανισμό υποβολής ερωτημάτων και εξαγωγής αποτελεσμάτων σε XML μορφή (Query Engine)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η επικοινωνία μεταξύ των διαφορετικών τμημάτων του γίνεται με κατάλληλα διαμορφωμένα APIs. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο.

Οι δημιουργοί του RStar επικέντρωσαν το ενδιαφέρον τους στο πρόβλημα αποδοτικής διαχείρισης μεγάλου όγκου RDF γράφων. Τα αποτελέσματα των ερευνών τους οδήγησαν στην υλοποίηση του παρακάτω μηχανισμού φόρτωσης:

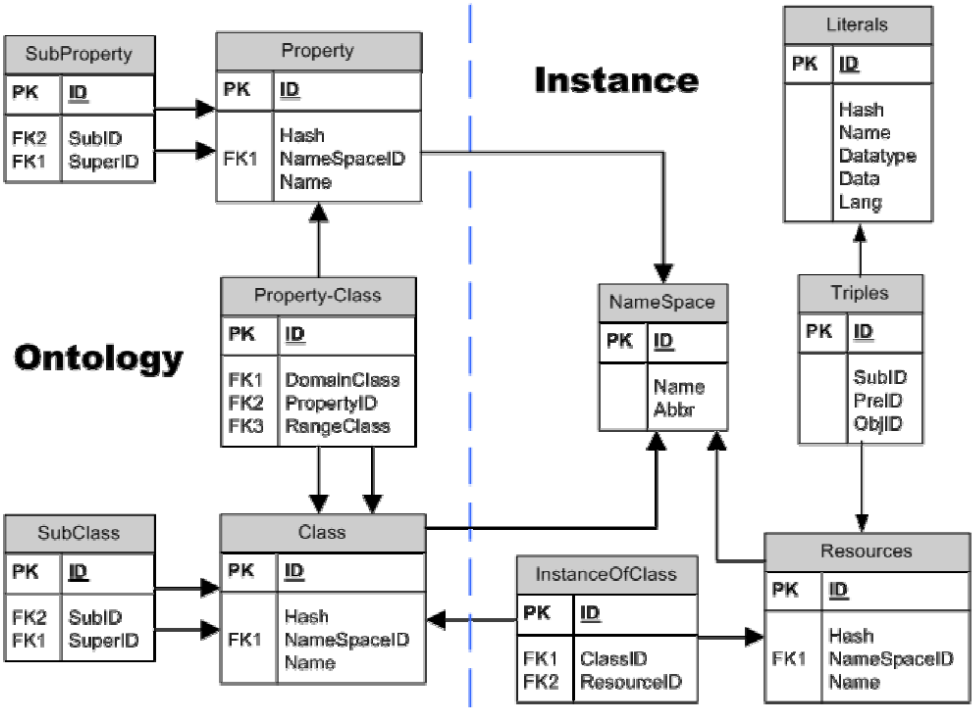
Ο Data Loader «διαβάζει» τα RDFS αρχεία (σε μορφή XML) και μεταφέρει στη βάση τις αντίστοιχες τριάδες μαζί με αυτές που υπονοούνται (backward chaining). Ο αναλυτής που χρησιμοποιεί είναι βασισμένος στο ευρύτατα διαδεδομένο API της XML (SAX). Ιδιαίτερη έμφαση δόθηκε στην επιτάχυνση του επαγωγικού μηχανισμού (inferencing). Γι' αυτό το λόγο, χρησιμοποιείται ένας προσωρινός πίνακας όπου αποθηκεύονται όλες οι τριάδες (ώστε να είναι εύκολη η πρόσβαση σε αυτές), ενώ οι κανόνες του αλγόριθμου εφαρμόζονται με συγκεκριμένη σειρά για την μεγιστοποίηση της απόδοσης.

Το σχήμα της βάσης του συστήματος ανήκει στην κατηγορία Hybrid και έχει κατασκευαστεί σε IBM DB2. Εμφανίζει αρκετές καινοτομίες, με βασικό του χαρακτηριστικό ότι διαχωρίζει την πληροφορία που αφορά στη δομή της οντολογίας από τα στιγμιότυπα (instances) των κλάσεων και των ιδιοτήτων. Αυτός ο διαχωρισμός έχει ως αποτέλεσμα την καλύτερη αποτίμηση ερωτημάτων, πράγμα που αποδεικνύουν και τα πειραματικά δεδομένα, και τον ακολουθήσαμε και εμείς κατά την σχεδίαση της ΒΔ του DBRS. Καθένας από τους πίνακες της βάσης χρησιμοποιεί «δικά του» αναγνωριστικά (IDs) ως πρωτεύοντα κλειδιά. Το τελευταίο, σε συνδυασμό με τη χρήση hash function αυξάνει κατά πολύ την απόδοση του συστήματος.

Υπάρχουν συνολικά 10 πίνακες που φαίνονται στο Σχήμα 3.4. Η πληροφορία σχετικά με το πεδίο και εύρος μιας ιδιότητας αποθηκεύεται στον πίνακα Property-

Class. Οι δηλώσεις των κλάσεων υπάρχουν στον πίνακα InstanceClass, ενώ των ιδιοτήτων στον πίνακα Triples.

Τα ερωτήματα υποβάλλονται σε γλώσσα RSQL [MSP+04] που έχει παρόμοια σύνταξη με την SQL και κατασκευάστηκε στα πλαίσια του RStar. Η Query Engine επιφορτίζεται με τη διαδικασία μετάφρασής τους σε SQL. Όπως και στο RDFSuite, η «πολιτική» που ακολουθείται βασίζεται κατά κύριο λόγο στο DBMS, «σπρώχνοντας» τη αποτίμηση των ερωτημάτων σε αυτό.



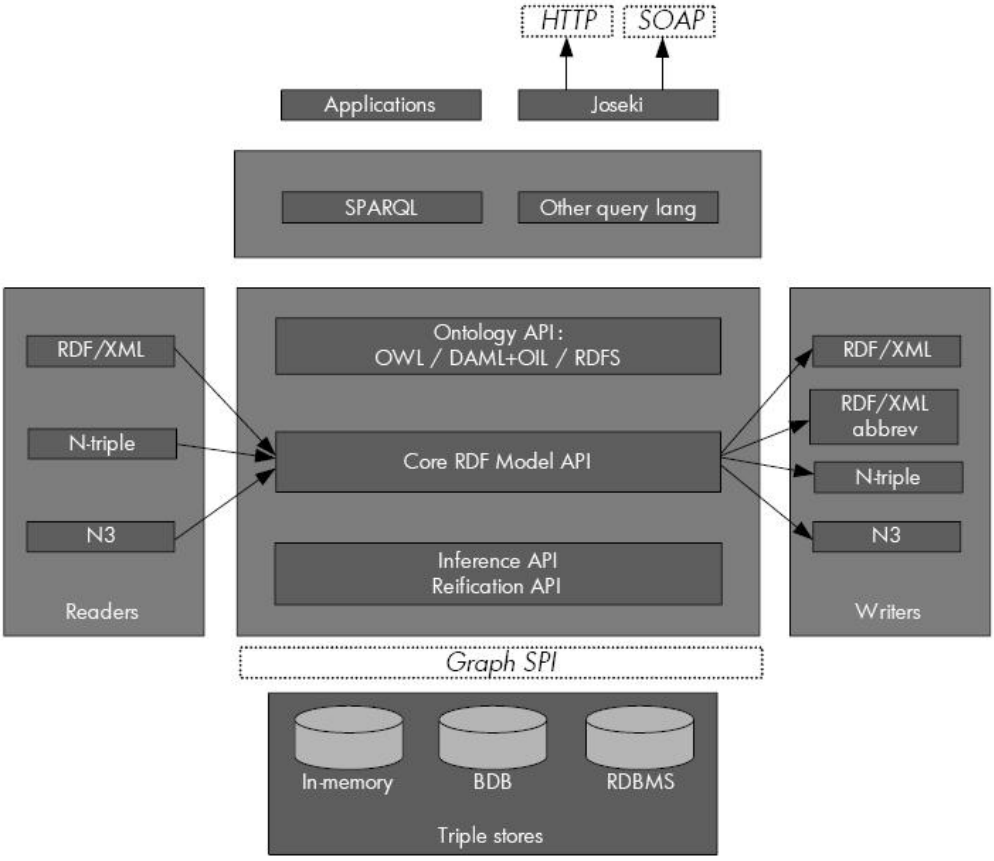
Σχήμα 3.4: Το σχήμα της βάσης δεδομένων του RStar

3.2.4 Η πλατφόρμα Jena

Η πλατφόρμα Jena [CDD+04] κατασκευάστηκε για την αποθήκευση και διαχείριση πολύ μεγάλων οντολογιών, της τάξης των δεκάδων εκατομμυρίων RDF δηλώσεων. Στην ουσία, είναι μια ανοιχτού κώδικα εργαλειοθήκη προγραμματισμού κατάλληλη για εμπορικές εφαρμογές που δεν απαιτούν γλώσσες αναπαράστασης υψηλής εκφραστικότητας (π.χ. OWL-DL). Πρόκειται για ένα από τα πιο γρήγορα εξελισσόμενα συστήματα. Η τελευταία έκδοση του Jena, σύμφωνη με τις ισχύουσες προδιαγραφές της RDF, είναι το Jena2 [WSKR03]. Αποτελείται από τα εξής βασικά μέρη:

- Ένα Αφηρημένο Μοντέλο (Abstract Model) που μεταφράζει υψηλού επιπέδου λειτουργίες των εξωτερικών εφαρμογών σε χαμηλού επιπέδου λειτουργίες πάνω σε RDF γράφους
- Ένα σύστημα διαχείρισης βάσεων δεδομένων (DBMS)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η επικοινωνία μεταξύ των εσωτερικών του τμημάτων γίνεται μέσω κατάλληλα διαμορφωμένων APIs. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο και τμήμα της OWL (περιορισμοί αριθμού).



Σχήμα 3.5: Η αρχιτεκτονική του συστήματος Jena

Το αφηρημένο μοντέλο του Jena2 αποτελεί τον πυρήνα του συστήματος ή αλλιώς το κεντρικό σημείο διεπαφής (Graph Interface). Χρησιμοποιείται για την υλοποίηση γράφων είτε βασισμένων στη μνήμη είτε σε συστήματα βάσεων δεδομένων. Στην ουσία πρόκειται για ένα πλούσιο API που περιλαμβάνει διάφορα εργαλεία: Ένα RDF/XML parser, μια μηχανή υποβολής ερωτήσεων (σε RDQL), I/O μονάδες για N3 και N-triple, RDF/XML εξαγωγείς και οδηγούς (drivers) για τη σύνδεση με τις πιο δημοφιλείς σχεσιακές βάσεις δεδομένων. Επίσης, παρέχει ένα επιπλέον API

(Ontology API) για την υποστήριξη RDF/S και OWL. Μέχρι στιγμής, έχουν παρουσιαστεί διάφορες υλοποιήσεις του Graph interface, η ανάλυσή των οποίων δε θα μας απασχολήσει εδώ. Περισσότερες πληροφορίες υπάρχουν στο [CDD+03].

Το σχήμα της βάσης που υιοθετείται ανήκει στην κατηγορία Oblivious, έχοντας όμως αρκετές παραλλαγές. Το Jena2, προκειμένου να βελτιώσει την απόδοσή του σε χρόνο, κάνει παραχωρήσεις από άποψη χώρου. Χρησιμοποιεί έναν πίνακα δηλώσεων όπου αποθηκεύει τις RDF δηλώσεις. Για να ξεχωρίζουν οι πόροι από τις σταθερές, οι τιμές των στηλών κωδικοποιούνται με ένα πρόθεμα που δείχνει το είδος τους. Για τις σταθερές που ξεπερνούν ένα συγκεκριμένο «κατώφλι» μεγέθους χρησιμοποιείται ξεχωριστός πίνακας. Ομοίως και για τα μεγάλα URIs. Αποθηκεύοντας πληροφορίες κατ' ευθείαν στον πίνακα των δηλώσεων, είναι δυνατόν να εκτελεσθούν πολλές λειτουργίες find (βλ. παρακάτω) χωρίς τη χρήση συνενώσεων (joins). Το βασικό μειονέκτημα αυτής της προσέγγισης είναι ότι απαιτεί πολύ περισσότερο χώρο, καθώς η ίδια τιμή (literal ή URI) αποθηκεύεται επανειλημμένα.

Το τελευταίο πρόβλημα αντιμετωπίζεται με διάφορους τρόπους. Συμπίεση και αποθήκευση σε ξεχωριστό πίνακα των κοινών προθεμάτων που παρατηρούνται στα URIs, μοναδική αποθήκευση των μεγάλων URIs και literals στους πίνακες που αναφέρθηκαν προηγουμένως και υποστήριξη πινάκων ιδιοτήτων.

Οι πίνακες ιδιοτήτων (Property Table) του Jena2 είναι ξεχωριστοί πίνακες όπου αποθηκεύονται όλα τα ζευγάρια τιμών-αντικειμένων μιας συγκεκριμένης ιδιότητας. Με άλλα λόγια, ένας πίνακας ιδιοτήτων αποθηκεύει όλα τα στιγμιότυπα μιας σχέσης στο γράφο και, κατά συνέπεια, αυτή η ιδιότητα δεν εμφανίζεται σε κανέναν άλλο πίνακα του γράφου. Μάλιστα, για ιδιότητες που έχουν μέγιστο περιορισμό αριθμού ίσο με ένα, αλλά και για πλειότιμες ιδιότητες, υπάρχει η δυνατότητα αποδοτικής ομαδοποίησής τους σε έναν πίνακα. Η ομαδοποίηση τιμών μπορεί να βελτιώσει κατά πολύ την απόδοση, ιδιαίτερα σε περιπτώσεις ομάδων τιμών που προσπελαύνονται συχνά μαζί. Επιπλέον, επιτυγχάνεται μια μικρή εξοικονόμηση χώρου που οφείλεται σε δύο λόγους. Αφενός τα URIs των ιδιοτήτων δεν αποθηκεύονται στον πίνακα ιδιοτήτων, αφετέρου στους ομαδοποιημένους πίνακες ιδιοτήτων το αντικείμενο αποθηκεύεται μοναδικά.

Εκτός από τους πίνακες ιδιοτήτων, το Jena2 χρησιμοποιεί και τους πίνακες ιδιοτήτων-κλάσεων (Property-Class Tables). Αυτοί είναι ειδικοί πίνακες όπου

αποθηκεύονται όλα τα στιγμιότυπα μιας κλάσης και οι ιδιότητες που έχουν ως πεδίο (domain) αυτή την κλάση. Με τη χρήση των Property-Class Tables, το Jena2 υλοποιεί τη λειτουργία reification της RDF [RDF]. Έτσι, αποφεύγει να αποθηκεύει και τις τέσσερις δηλώσεις που απαιτεί η reification και εξοικονομεί πολύτιμο χρόνο.

Σε ένα αφαιρετικό πλαίσιο, το υποσύστημα αποθήκευσης του Jena2 χρειάζεται να υλοποιεί μόνον τρεις λειτουργίες:

- Λειτουργία add (για την αποθήκευση μιας RDF δήλωσης στη βάση)
- Λειτουργία delete (για την απομάκρυνση μιας RDF δήλωσης από τη βάση)
- Λειτουργία find (αναζήτηση). Η λειτουργία find ανασύρει όλες τις δηλώσεις που συμφωνούν με ένα πρότυπο της μορφής <S,P,O> (< Υποκείμενο, Κατηγόρημα, Αντικείμενο >) όπου καθένα από τα S, P, O είναι είτε μια σταθερά είτε αδιάφορου τύπου (don't-care).

Για την εξαγωγή των υπονοούμενων σχέσεων, χρησιμοποιούνται εσωτερικοί Reasoners που λειτουργούν αποκλειστικά στη μνήμη. Η τεχνική που ακολουθείται είναι η forward chaining. Επίσης, παρέχεται η δυνατότητα συνδυασμού της πλατφόρμας με εξωτερικούς Reasoners όπως οι Pellet και Racer [RACER].

Τα ερωτήματα του χρήστη υποβάλλονται σε γλώσσα RDQL [RDQL], ενώ έχει επίσης κατασκευαστεί και ένας SPARQL [SPARQL] εξυπηρετητής γνωστός με το όνομα Joseki [Joseki]. Η Query Engine τα μετατρέπει σε ένα σύνολο λειτουργιών find στο οποίο επιτρέπονται μεταβλητές στα τριαδικά πρότυπα αναζήτησης. Οι μεταβλητές αυτές είναι που καθιστούν δυνατές τις συνενώσεις (joins) μεταξύ των προτύπων.

Όπως φάνηκε και από τα προηγούμενα, το Jena ακολουθεί την «γραφοθεωρητική» προσέγγιση, χειρίζεται δηλαδή την οντολογία σαν ένα γράφο. Όλες οι λειτουργίες και τα APIs του συστήματος είναι βασισμένα σε αυτή τη «φιλοσοφία» και μπορούν έτσι να εκμεταλλεύονται αλγόριθμους της θεωρίας γραφημάτων. Εμείς, κατά την ανάλυση των προδιαγραφών του DBRS, θεωρήσαμε καλύτερο να χειριζόμαστε την οντολογία ως ένα σύνολο αξιωμάτων και όχι ως γράφο. Γι' αυτόν ακριβώς το λόγο, για την αναπαράσταση και διαχείριση της

σημασιολογικής πληροφορίας, υιοθετήσαμε το OWL API [OWLAPI] και όχι το Jena API.

3.2.5 Το σύστημα KAON

Το σύστημα KAON [VOSM03] σχεδιάστηκε με στόχο την υλοποίηση ενός ολοκληρωμένου συστήματος διαχείρισης Σημασιολογικού Ιστού (Semantic Web Management System), δηλαδή ενός περιβάλλοντος ανάπτυξης εφαρμογών οντολογιών. Προσανατολίζεται σε λειτουργία πελάτη-εξυπηρετητή (client-server), όπου η αλληλεπίδραση με το χρήστη μπορεί να γίνει μέσω του DIG Interface[26]. Αποτελείται από τα παρακάτω μέρη:

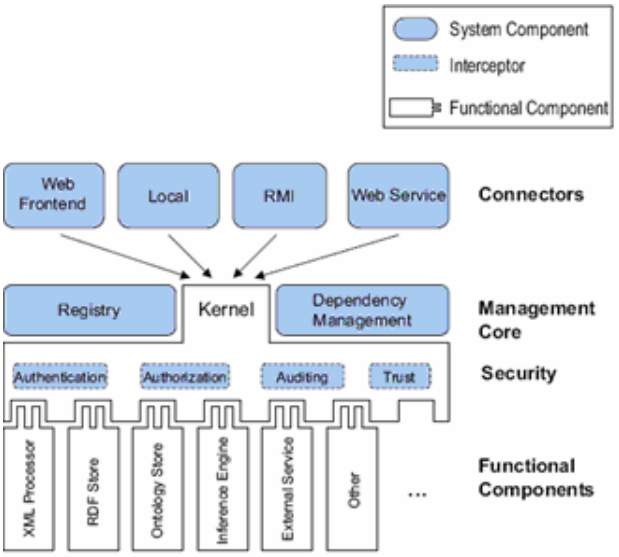
- Σύνδεσμοι
- Πυρήνας Διαχείρισης
- Μονάδα Ασφάλειας
- Λειτουργικά Εξαρτήματα (Database, Reasoner κ.λπ.)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η επικοινωνία των εσωτερικών του τμημάτων γίνεται με χρήση κατάλληλων APIs υπό την εποπτεία των Registry και Dependency Management modules. Η πληροφορία που μπορεί να αποθηκευτεί ανήκει στο RDF/S μοντέλο, ενώ υποστηρίζεται και διαχείριση OWL-Lite πληροφορίας μέσω του OWL API.

Οι σύνδεσμοι (Connectors) του KAON εξασφαλίζουν τη δυνατότητα επικοινωνίας μέσω πρωτοκόλλων απομακρυσμένης σύνδεσης, όπως τα Java RMI [RMI] και DIG. Συνεργάζονται με τον πυρήνα (Kernel) ο οποίος ευθύνεται για την ανεύρεση, κατανομή και φόρτωση των εφαρμογών που τελικά θα εξυπηρετήσουν κάποιο αίτημα και έχει υλοποιηθεί με τη χρήση του Java Management Extensions [JMX]. Η μονάδα Ασφάλειας (Security) αποτελείται από διάφορους ανιχνευτές

[26] Πρότυπη διαπροσωπεία μεταφοράς σημασιολογικής πληροφορίας σε XML μορφή. Είναι συμβατή με το HTTP αλλά εμφανίζει δυο βασικά μειονεκτήματα: Είναι, συνήθως, πιο αργό από τα συγκεκριμένα για κάθε σύστημα APIs και έχει όρια ως προς το είδος της πληροφορίας που μπορεί να χειριστεί.

υπεύθυνους για τα δικαιώματα πρόσβασης των πελατών στα εξαρτήματα και τις υπηρεσίες τους.

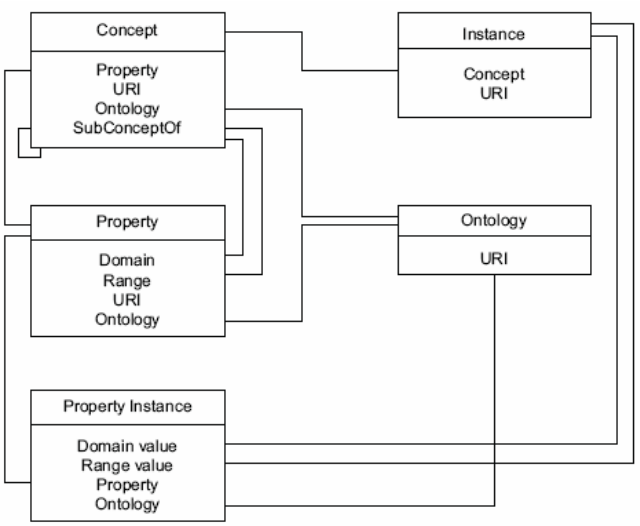


Σχήμα 3.6 Η αρχιτεκτονική του συστήματος KAON

Η φόρτωση της πληροφορίας γίνεται μέσω ενός XML αναλυτή που συνδέεται στο σύστημα ως λειτουργικό εξάρτημα. Διαβάζει το αρχικό αρχείο της οντολογίας σε μορφή XML και μεταφέρει τα δεδομένα στη βάση. Το KAON προσφέρει δύο είδη μηχανισμών αποθήκευσης :

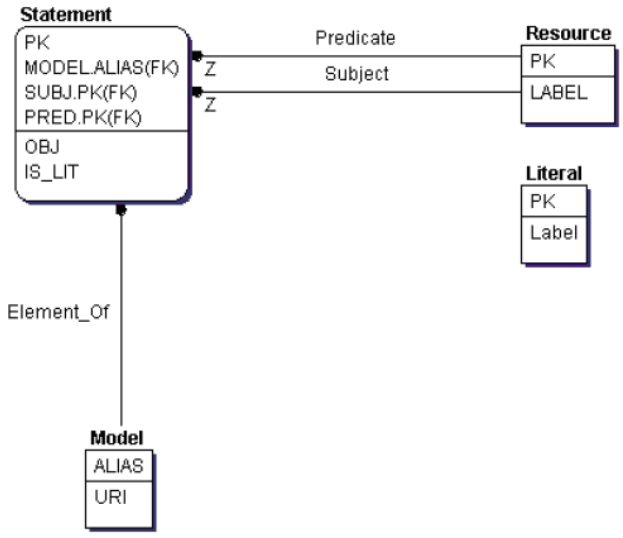
- RDF Store
- Ontology Store

Το πρώτο χρησιμοποιείται για την μόνιμη αποθήκευση και διαχείριση των RDF δεδομένων. Υιοθετεί το μοντέλο Schema Oblivious που φαίνεται στο Σχήμα 3.7. Υπάρχουν συνολικά 4 πίνακες: Ένας πίνακας Statements για την αποθήκευση των δηλώσεων, ένας πίνακας Resource για την αντιστοίχιση URIs με IDs, ένας πίνακας Model που αποθηκεύει τα URLs των οντολογιών και ένας πίνακας Literals για την ξεχωριστή αποθήκευση των σταθερών. Η επικοινωνία μεταξύ χρήστη και DBMS γίνεται μέσω του JBoss Application Server [JBoss].



Σχήμα 3.7: Το σχήμα της βάσης δεδομένων του RDF Store

Το δεύτερο σύστημα αποθήκευσης (Ontology Store) χρησιμοποιείται για την αποδοτική διαχείριση της πληροφορίας σε ένα μετά-επίπεδο. Σε συνδυασμό με την μηχανή επαγωγής (Inference Engine), προσφέρει καθολικό έλεγχο της οντολογίας που μεταφράζεται όχι μόνο στην εξαγωγή των υπονοούμενων σχέσεων, αλλά και στον έλεγχο της συνέπειας (consistency). Υιοθετεί το Schema Aware μοντέλο με αρκετές παραλλαγές που φαίνονται στο Σχήμα 3.8. Εδώ, υπάρχει ένας πίνακας για κάθε κλάση και ένας για κάθε ιδιότητα. Οι ισχυρισμοί αποθηκεύονται σε ξεχωριστούς πίνακες (Property Instance, Class Instance).



Σχήμα 3.8: Το σχήμα της βάσης δεδομένων του Ontology Store

Ως σύστημα συλλογιστικής ανάλυσης μπορεί να χρησιμοποιηθεί το FaCT [FaCT], ενώ ήδη βρίσκεται υπό κατασκευή ένας εσωτερικός Reasoner που αντιστοιχεί στο τμήμα της OWL-DL (KAON OWL-DL). Επίσης, έχει υλοποιηθεί και ένας μηχανισμός

επαγωγής (Bubo) που κάνει χρήση των Description Horn Logics (βλέπε Παράγραφο 2.2.2.2) και βασίζεται σε μια μηχανή Datalog.

Σε περίπτωση που το σύστημα λειτουργεί ως αυτόνομη εφαρμογή (standalone application) και όχι μέσω HTTP, οι ερωτήσεις μπορούν να υποβληθούν σε γλώσσα KAON Query Language.

3.2.6 Το σύστημα DLDB

Το DLDB (Description Logics DataBase) [PH03] υλοποιήθηκε με σκοπό την αποθήκευση σχετικά μικρού μεγέθους σημασιολογικής πληροφορίας της τάξης των εκατοντάδων κλάσεων και ιδιοτήτων. Αποτελείται από τα εξής μέρη:

- Ένα μηχανισμό φόρτωσης της πληροφορίας στη βάση δεδομένων
- Ένα DBMS (Microsoft Access)
- Ένα σύστημα συλλογιστικής ανάλυσης (FACT – Fast Classification Terminologies)
- Ένα μηχανισμό υποβολής ερωτημάτων

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA με εξαίρεση τον Reasoner που είναι γραμμένος σε Lisp. Η επικοινωνία μεταξύ των διαφορετικών τμημάτων γίνεται μέσω κατάλληλα διαμορφωμένων APIs. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο εκτός από τα rdf:domain και rdf:range. Επιπλέον, εκμεταλλευόμενο τη χρήση του FACT, το DLDB προσφέρει τη δυνατότητα επαγωγής σε πληροφορία που ανήκει στο τμήμα της OWL – DL και αφορά στην εξαγωγή της ιεραρχίας των κλάσεων (class hierarchy).

Το DLDB σχεδιάστηκε με στόχο την αποθήκευση πολλών οντολογιών. Έτσι, το σχήμα της βάσης που υλοποιείται σε Microsoft Access, αν και ανήκει στην κατηγορία Specific/MatView, εμφανίζει βασικές ιδιαιτερότητες σε σχέση με την περιγραφή που δώσαμε στην προηγούμενη ενότητα. Χρησιμοποιείται ένας πίνακας για κάθε κλάση και ένας για κάθε ιδιότητα. Οι πίνακες των κλάσεων αποτελούνται από δύο στήλες, όπου στην πρώτη αποθηκεύονται τα IDs των πόρων και στη δεύτερη τα IDs των οντολογιών. Κατ' αντιστοιχία, οι πίνακες των ιδιοτήτων αποτελούνται από τρεις στήλες: μία για το ID του Υποκειμένου, μια για το ID του Αντικειμένου και μία για

το ID της οντολογίας. Όπως έγινε φανερό, δε χρησιμοποιούνται απευθείας τα URIs, αλλά γίνεται χρήση IDs. Αυτό αφ' ενός εξοικονομεί χώρο, αφ' ετέρου έχει μια επιπλέον επιβάρυνση στο χρόνο απόκρισης. Για να αντιμετωπιστεί αυτή η χρονική καθυστέρηση, έχουν υλοποιηθεί ευρετήρια (indices) και ένας πίνακας Hash που «κρατάει» ζεύγη URI-ID που βρέθηκαν κατά τη φόρτωση και είναι πολύ πιθανό να ξαναχρησιμοποιηθούν. Η αντιστοίχιση του ονόματος της κάθε οντολογίας με το αναγνωριστικό της γίνεται σε ένα πίνακα ONTOLOGIES(URL, ID).

Η πλήρης πληροφορία για την ιεραρχία των κλάσεων της οντολογίας, όπως αυτή λαμβάνεται από τον FACT, αποθηκεύεται σε έναν πίνακα HIERARCHY με δύο στήλες: μια για το όνομα της κλάσης (ClassName) και μια για το ID της κλάσης-πατέρας. Προφανώς υπάρχουν τόσοι τέτοιοι πίνακες όσες και οι αποθηκευμένες οντολογίες.

Η καινοτομία του DLDB έγκειται ακριβώς στη χρήση του FaCT και αυτό γιατί αποτέλεσε ιστορικά μια από τις πρώτες προσπάθειες να καταδειχθούν εμπράκτως τα πλεονεκτήματα της σύνδεσης μεταξύ Reasoner και DBMS. Αυτή η προσέγγιση υιοθετήθηκε μεταγενέστερα και από άλλα συστήματα, όπως θα δούμε παρακάτω. Στο DLDB υλοποιήθηκε ως εξής :

Για τη φόρτωση της αρχικής οντολογίας, χρησιμοποιήθηκε ένας αναλυτής (DAML) που μετατρέπει το αρχείο της οντολογίας σε ένα αντικείμενο (object - κατά την έννοια που δίνεται στον αντικειμενοστρεφή προγραμματισμό). Στη συνέχεια, το αντικείμενο αυτό μεταφράζεται σε ένα έγκυρο SHIQ αρχείο έτσι ώστε να μπορεί να «διαβαστεί» από τον FaCT. Το αρχείο αυτό είναι, προφανώς, σε XML μορφή. Ο Reasoner το αναλύει και, αφού εξάγει την πληροφορία για την ιεραρχία των κλάσεων, το ενημερώνει με τις νέες υπονοούμενες (implicit) σχέσεις. Έπειτα λαμβάνει χώρα η αντίστροφη διαδικασία: Το XML αρχείο γίνεται αντικείμενο και στέλνεται στη βάση. Τότε αναλαμβάνει το DBMS (MS Access). Κατασκευάζει πίνακες και όψεις (views), για κάθε κλάση και ιδιότητα, που στην αρχή περιέχουν μόνο την πληροφορία που δηλώνεται ρητά. Η διαδικασία κατασκευής ολοκληρώνεται με την επέκταση των όψεων βάσει της πλήρους ιεραρχίας: Κάθε όψη που αντιστοιχεί σε κλάση με υπο-κλάσεις συμπληρώνεται με τα άτομα (instances) όλων των «απογόνων» της.

Ο μηχανισμός ερωτήσεων δέχεται είσοδο από τον χρήστη και επιστρέφει έξοδο σε αυτόν. Κατά τη διαδικασία αποτίμησης, δεν υπάρχει καμία επικοινωνία με τον FaCT. Τα ερωτήματα υποβάλλονται σε KIF-like[27] μορφή και μεταφράζονται σε SQL.

3.2.7 Το σύστημα Instance Store

Το σύστημα Instance Store (IS) [HLTB04] κατασκευάστηκε με σκοπό την αποθήκευση και διαχείριση πολύ μεγάλων οντολογιών, όπως αυτές που συναντούμε σε πραγματικές εφαρμογές (real world applications). Συνδυάζει DBMS και Reasoner, με έναν πιο «εξελιγμένο» τρόπο από αυτόν που περιγράψαμε στο DLDB. Προσφέρει ιδιαίτερη ευελιξία όσον αφορά τόσο στο DBMS που ο διαχειριστής του θα επιλέξει, όσο και στο Reasoner. Αποτελείται από τα εξής μέρη :

- Ένα μηχανισμό φόρτωσης της οντολογίας στη βάση δεδομένων
- Ένα σύστημα διαχείρισης Βάσεων Δεδομένων
- Ένα σύστημα συλλογιστικής ανάλυσης
- Ένα μηχανισμό υποβολής ερωτημάτων

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Για την ενδοεπικοινωνία μεταξύ των επιμέρους τμημάτων χρησιμοποιούνται κατάλληλα διαμορφωμένα APIs, ενώ η σύνδεση με τον Reasoner γίνεται μέσω του DIG Interface. Αυτό αποτελεί και ένα από τα βασικά χαρακτηριστικά του συστήματος. Χάρη στο DIG το IS αποκτά «αυτονομία» ως προς τον Reasoner που θα χρησιμοποιηθεί και πράγματι, στις έως τώρα εφαρμογές του, έχουν επιλεγεί οι FaCT (Lisp) , Racer (JAVA) και FaCT++ (C++). Την ανεξαρτησία από συγκεκριμένα συστήματα συλλογιστικής ανάλυσης επιδιώξαμε κι εμείς κατά την ανάπτυξη του DBRS, δημιουργώντας ένα wrapper για κάθε Reasoner.

Η πληροφορία που μπορεί να αποθηκευτεί καθορίζεται από τον σκοπό για τον οποίο το IS αρχικά κατασκευάστηκε: Αποδοτική συλλογιστική ανάλυση σε πολύ

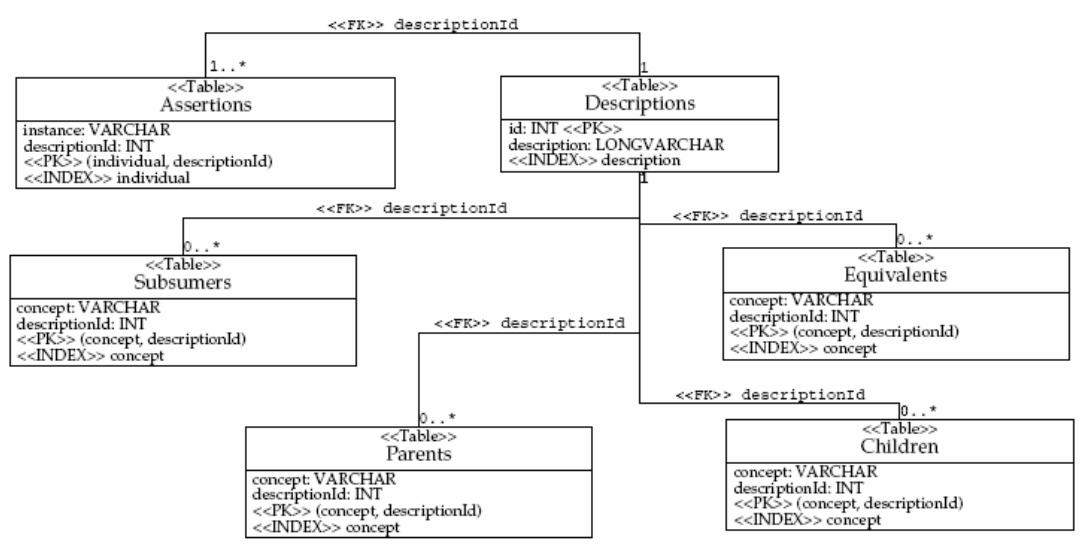
---

[27] KIF (Knowledge Interchange Format): πρότυπο που δημιουργήθηκε για την αναπαράσταση Λογικής Πρώτης Τάξης (FOL). Έχει αυστηρή σύνταξη έτσι ώστε να είναι επεξεργάσιμο από υπολογιστές.

μεγάλο αριθμό ατόμων (individuals) της τάξης των εκατομμυρίων και επιλεκτική εξαγωγή τους (retrieval) βάσει της σημασιολογίας.

Έτσι, στη βάση δεν αποθηκεύεται καθόλου η γνώση σχετικά με τις ιδιότητες της οντολογίας (role assertions). Τα άτομα θεωρούνται ανεξάρτητα μεταξύ τους και αντιστοιχίζονται μόνο στις κλάσεις (OWL Descriptions) που ανήκουν. Υπό αυτήν την έννοια, μπορούμε να πούμε ότι το IS αναφέρεται μόνο σε role-free οντολογίες και κατά συνέπεια δεν καλύπτει ούτε το RDF/S ούτε το OWL μοντέλο, αλλά μόνο ένα τμήμα τους.

Το σχήμα της βάσης (Hybrid) είναι ιδιαίτερα απλό και για την υλοποίησή του μπορούν να χρησιμοποιηθούν διάφορα DBMSs. Μέχρι στιγμής, έχουν επιλεγεί τα MySQL, Oracle και Hypersonic, ενώ η πρόσβαση σε αυτά έγινε μέσω JDBC ή Hibernate. Υπάρχουν συνολικά έξι πίνακες: Ένας πίνακας Assertions που αντιστοιχίζει άτομα με κλάσεις, ένας πίνακας Descriptions που περιλαμβάνει όλες τις κλάσεις της οντολογίας και τέσσερεις επιπλέον πίνακες που αποθηκεύουν τη σχετική με την ιεραρχία πληροφορία. Τα παραπάνω, μαζί με τα συγκεκριμένα πεδία του κάθε πίνακα, φαίνονται στο Σχήμα 3.9.



Σχήμα 3.9: Το σχήμα της βάσης δεδομένων του Instance Store

Το IS μπορεί να αποθηκεύσει ταυτόχρονα πολλές οντολογίες σε καθεμία απ' τις οποίες αντιστοιχεί ένα τέτοιο σχήμα, προσέγγιση που υιοθετήθηκε και στο DBRS.

Όσον αφορά στη σύνδεση μεταξύ DBMS και συστήματος συλλογιστικής ανάλυσης, το IS ξεπερνά το DLDB με την έννοια ότι βάση και Reasoner συνεργάζονται όχι μόνο για τη απόκτηση της ιεραρχίας κατά την αρχικοποίηση του

συστήματος (at compile time), αλλά και για την απάντηση των ερωτημάτων που δε μπορούν να αποτιμηθούν μόνο με χρήση SQL ερωτημάτων στην υπάρχουσα πληροφορία (at run-time), αλλά απαιτούν συλλογιστική.

Τα ερωτήματα που υποστηρίζονται είναι δύο ειδών : Retrieve και Individual Types. Στα πρώτα ζητούνται άτομα που ανήκουν σε μια αυθαίρετη κλάση που δίνεται από τον χρήστη, ενώ στα δεύτερα ζητούνται τύποι ατόμων, δηλαδή οι κλάσεις στις οποίες ανήκουν. Λόγω της φύσης τους, δεν απαιτούν κάποια ιδιαίτερη γλώσσα ερωτημάτων. Για την ανάσυρση (retrieval), η λειτουργία του συστήματος παρουσιάζει εξαιρετικό ενδιαφέρον:

Αρχικά, το αρχείο της οντολογίας διαβάζεται και εισάγεται στον Reasoner για την εξαγωγή της ιεραρχίας. Κατά τη διάρκεια αυτής της ενέργειας, δημιουργείται και η σύνδεσή του με το DBMS και κατασκευάζονται οι πίνακες. Το δεύτερο βήμα περιλαμβάνει την απευθείας (από το αρχείο) φόρτωση των (explicit) δηλώσεων κλάσεων στη βάση μαζί, όμως, με τις επιπλέον πληροφορίες για την ιεραρχία. Η μετέπειτα διαδικασία εξαρτάται από τις κλάσεις που υπάρχουν στα ερωτήματα που υποβάλλονται. Αν πρόκειται για κλάση που έχει ήδη αποθηκευτεί στη βάση, το IS δε «ρωτάει» καθόλου το Reasoner, αλλά (έχοντας την πλήρη ιεραρχία) απαντάει εύκολα. Απ' την άλλη, στην περίπτωση μιας αυθαίρετης εννοιακής έκφρασης «ζητάει» από το reasoner να την ιεραρχήσει. Αν μετά από αυτό προκύψει ότι υπάρχει ισοδύναμη κλάση ήδη στη βάση, τότε το ερώτημα απαντιέται εύκολα. Αν όχι, υπάρχουν δύο ενδεχόμενα:

- Η εννοιακή έκφραση να ισοδυναμεί με την τομή (conjcuction) των «γονέων» της
- Η εννοιακή έκφραση να μην είναι ισοδύναμη με την τομή των «γονέων» της.

Στην πρώτη περίπτωση το IS απλώς επιστρέφει την τομή των «γονέων» της έκφρασης μαζί με τα άτομα που ανήκουν στα «παιδιά» της. Στη δεύτερη, υπολογίζει ποια από τα άτομα των «γονέων» δεν ανήκουν στα «παιδιά» και μετά «ζητάει» από τον Reasoner να προσδιορίσει απ' το σύνολο των πιθανών ατόμων αυτά που όντως ανήκουν στη συγκεκριμένη έκφραση. Μόλις λάβει τα αποτελέσματα, επιστρέφει μαζί με αυτά και τα άτομα όλων των «παιδιών» της.

Στο σημείο αυτό θα θέλαμε να σημειώσουμε ότι το IS αποτέλεσε και το πρώτο σύστημα που έθεσε με αξιώσεις την ιδέα μιας επιλεκτικής επικοινωνίας της ΒΔ με τον Reasoner. Της αρχικής αυτής προσπάθειας ακολούθησαν πολλές άλλες, όπως το σύστημα LAS που περιγράφεται αμέσως μετά.

3.2.8 Το σύστημα LAS

Το σύστημα LAS [C05] αποτελεί μια από τις πιο πρόσφατα δημοσιευμένες προσπάθειες διαχείρισης οντολογιών μεγάλου όγκου και υψηλής εκφραστικότητας. Επιτυγχάνει την «έξυπνη» συνεργασία με μια μηχανή συλλογιστικής ανάλυσης το οποίο αφ' ενός επεκτείνει με μηχανισμό δευτερεύουσας μνήμης, αφ' ετέρου διευκολύνει στην εκτέλεση συγκεκριμένων επαγωγικών καθηκόντων. Αποτελείται από τρία βασικά μέρη:

- Ένα περιβάλλον διαπροσωπείας (User Interface) για την επικοινωνία με το χρήστη, την εισαγωγή δεδομένων, την υποβολή ερωτημάτων και την εξαγωγή των αποτελεσμάτων
- Ένα DBMS (Oracle)
- Ένα σύστημα συλλογιστικής ανάλυσης (Racer)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η επικοινωνία μεταξύ των επιμέρους τμημάτων γίνεται μέσω κατάλληλα διαμορφωμένων APIs που θα περιγράψουμε στη συνέχεια. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει τμήμα της OWL-DL (*SHR*+).

Το User Interface επιφορτίζεται με τις διαδικασίες αρχικοποίησης του συστήματος και υποβολής ερωτημάτων. Προσφέρει τη δυνατότητα ορισμού λογαριασμών χρηστών (user accounts) για τους οποίους κατασκευάζει ξεχωριστό σχήμα βάσης. Η προς εισαγωγή οντολογία δίνεται ως OWL file ή Racer file έτσι ώστε να μπορεί να αναγνωριστεί από τον αναλυτή (parser) του Reasoner. Κατά την εισαγωγή, κατασκευάζεται το σχήμα της βάσης και η σύνδεσή της με τον Racer. Στη συνέχεια, το DBMS και ο Reasoner επικοινωνούν με σκοπό την φόρτωση στη βάση των αρχικών δεδομένων. Αυτό γίνεται μέσω του LASRacer Interface που είναι TCP-based. Περισσότερες λεπτομέρειες για το είδος της πληροφορίας που ανταλλάσσεται θα δοθούν παρακάτω.

Το σχήμα της βάσης του LAS είναι αρκετά διαφορετικό απ' ότι έχουμε δει μέχρι τώρα. Αυτό συμβαίνει, γιατί η βασική καινοτομία του συστήματος δεν είναι η χρησιμοποίηση του Racer για την εξαγωγή χρήσιμης πληροφορίας (όπως έχουμε δει και σε άλλα συστήματα), αλλά κυρίως η επιτάχυνση της συλλογιστικής διαδικασίας με τη χρήση αποθηκευμένων στη βάση ψευδομοντέλων (βλέπε Ενότητα 2.2.2.1).

Η βάση δεδομένων (Hybrid μοντέλο) αποθηκεύεται σε Oracle και η πρόσβαση σε αυτή γίνεται μέσω του JDBC. Αποτελείται από δεκαεννιά συνολικά πίνακες από τους οποίους οι τέσσερεις κατασκευάζονται προσωρινά κάθε φορά που πρέπει να αποτιμηθούν συγκεκριμένα ερωτήματα με πολύπλοκες εννοιακές εκφράσεις. Μπορούμε να τους κατηγοριοποιήσουμε σε:

- Πίνακες που αποθηκεύουν το TBox

  Αυτοί είναι οι Individual, Description, RoleType, TransitiveRole Rsynonyms, SymmetryRole, DesParent, DesAncestors, TmpDesparent, TmpDesAncestors, RAncestors, RParents, TmpRAncestors.
- Πίνακες που αποθηκεύουν το ABox

  Αυτοί είναι οι InAssertion και RoleAssertion
- Πίνακες που αποθηκεύουν τα ψευδομοντέλα

  Αυτοί είναι οι IndividualPseudoModel, DescriptionPseudoModel, TmpDesPM

Κατά την αρχικοποίηση, το LAS φορτώνει στη βάση όλες τις κλάσεις, τις ιδιότητες και τα άτομα της οντολογίας καλώντας αντίστοιχες μεθόδους του Reasoner. Όσον αφορά στην ιεραρχία, «ζητά» από τον Racer μόνο την πληροφορία που σχετίζεται με τους άμεσους «προγόνους» και «απογόνους» κάθε κλάσης και ιδιότητας. Από αυτή εξάγει (μέσω μεθόδων της Oracle) την πλήρη ιεραρχία. Επίσης, βασιζόμενο στα ιδιαίτερα χαρακτηριστικά του Racer, αποθηκεύει ένα ψευδομοντέλο για κάθε άτομο (Individual Pseudo Model) και ένα για κάθε κλάση (Description Pseudo Model) στους αντίστοιχους πίνακες. Τα ψευδομοντέλα αυτά ο Racer τα κατασκευάζει κατά την έναρξη της λειτουργίας του, αμέσως μόλις φορτωθεί η οντολογία. Τα ιδιαίτερα χαρακτηριστικά των ρόλων (role characteristics) τα «παίρνει» με κλήση κατάλληλων μεθόδων επίσης από τον Reasoner.

Η συνεργασία DMBS και Racer εξαρτάται κάθε φορά από το είδος των ερωτημάτων. Επειδή η «φιλοσοφία» του συστήματος μοιάζει πολύ με αυτή του DBRS, αναλύουμε όλες τις περιπτώσεις. Τονίζουμε εξαρχής ότι όλα τα ερωτήματα υποβάλλονται μέσω GUI χωρίς τη χρήση κάποιας συγκεκριμένης γλώσσας, αλλά με comboboxes όπου ο χρήστης ορίζει το ερώτημα και τις παραμέτρους του. Μεταβιβάζονται πρώτα στην Oracle και όπου κριθεί απαραίτητο καλείται ο Reasoner:

- Ερωτήματα σχετικά με το TBox
    - Query Description Ancestors

      Ζητούνται οι «πρόγονοι» μιας εννοιακής έκφρασης. Το LAS μεταβιβάζει την πολύπλοκη έννοια στο Racer για να λάβει το ψευδομοντέλο της. Στη βάση ελέγχεται αν αυτό είναι συγχωνεύσιμο (mergable) με τα ψευδομοντέλα των συμπληρωμάτων των ατομικών εννοιών από τις οποίες συντίθεται η έκφραση και επιστρέφει τους πιθανούς προγόνους στον Reasoner για την τελική απάντηση.
    - Query Description Descendants

      Ζητούνται οι «απόγονοι» μιας εννοιακής έκφρασης. Ακολουθείται η ίδια διαδικασία με πριν μόνο που το mergable test γίνεται με τα ψευδομοντέλα των ατομικών κλάσεων και όχι των συμπληρωμάτων τους.
    - Query Description Synonyms

      Ζητούνται οι συνώνυμες κλάσεις μιας έκφρασης. Συνδυάζονται οι δύο παραπάνω ενέργειες.
    - Query Description Parents

      Ζητούνται οι άμεσοι «πρόγονοι» μιας έκφρασης. Εδώ το LAS, λόγω έλλειψης κατάλληλων μεθόδων στον Reasoner, βασίζεται αποκλειστικά στον Tableau αλγόριθμο.
    - Query Description Children

      Ζητούνται οι άμεσοι «απόγονοι» μιας έκφρασης. Για τον ίδιο λόγο με πάνω, το LAS απαντάει μέσω του Tableau αλγορίθμου.

- Ερωτήματα σχετικά με το ABox
    - Individual Types

      Ζητούνται οι κλάσεις στις οποίες ανήκει ένα άτομο. Το LAS ελέγχει αν η πληροφορία που υπάρχει στη βάση για τους τύπους του ατόμου a είναι πλήρης και σε περίπτωση που ισχύει κάτι τέτοιο απαντάει μέσω SQL. Διαφορετικά, «ζητά» από το Racer να υπολογίσει το αποτέλεσμα και φορτώνει στη βάση την επιπλέον πληροφορία. Προφανώς, μετά από αυτό, η σχετική πληροφορία είναι πλέον πλήρης, γεγονός που κωδικοποιείται από το LAS σε ένα συγκεκριμένο (flag) πεδίο με σκοπό να μην χρειαστεί καθόλου τον Racer σε επόμενο ερώτημα.
    - Individual Direct Types

      Ζητούνται οι άμεσοι τύποι ενός ατόμου. Το LAS ελέγχει αν η πληροφορία που υπάρχει στη βάση είναι πλήρης και σε περίπτωση που ισχύει κάτι τέτοιο επιστρέφεται το αποτέλεσμα μέσω SQL ερωτήματος. Στην αντίθετη περίπτωση καλεί τον Racer να αποτιμήσει το ερώτημα και, εισάγοντας τη νέα πληροφορία στη βάση, ενημερώνει ταυτόχρονα το πεδίο πληρότητας της πληροφορίας.
    - Query Retreive

      Ζητούνται τα άτομα που ανήκουν σε μια συγκεκριμένη κλάση. Υπάρχουν δύο ενδεχόμενα:

        - Η κλάση είναι γνωστή (named), υπάρχει δηλαδή ήδη στη βάση

          Στην πρώτη περίπτωση, το LAS ελέγχει το πεδίο πληρότητας του πίνακα Description. Αν είναι true (υπάρχει η πλήρης πληροφορία στη βάση), απαντάει με SQL. Αν όχι, πρέπει να εκτελέσει το mergable test μεταξύ των ψευδομοντέλων του ατόμου και της κλάσης και στη συνέχεια να μεταβιβάσει τα πιθανά άτομα για έλεγχο στο Reasoner. Μόλις ο Racer επιστρέψει το αποτέλεσμα, αυτό αποθηκεύεται στη βάση και ενημερώνεται το πεδίο πληρότητας (true).
        - Η κλάση είναι αυθαίρετη που σημαίνει ότι συνίσταται από άλλες υπάρχουσες έννοιες

Στην περίπτωση που έχουμε αυθαίρετη έκφραση, το LAS τη μεταβιβάζει στο Racer για να πάρει το ψευδομοντέλο της και μετά εκτελεί το mergable test στη βάση δεδομένων. Στη συνέχεια, όπως και πριν, επιστρέφει τα πιθανά άτομα στον Reasoner για την τελική απάντηση. Η διαφορά με προηγουμένως είναι ότι εδώ το τελικό αποτέλεσμα δεν αποθηκεύεται στη βάση. Κάτι τέτοιο γίνεται μόνο για τις ατομικές κλάσεις.

- Query Individual Fillers

  Ζητούνται τα άτομα με τα οποία συνδέεται ένα δοσμένο άτομο μέσω ενός συγκεκριμένου ρόλου. Το LAS ελέγχει αρχικά το πεδίο του πίνακα RoleAssertion που δηλώνει την πληρότητα της πληροφορίας που υπάρχει στη βάση. Αν είναι true (υπάρχει η πλήρης πληροφορία στη βάση), απαντάει μέσω SQL. Διαφορετικά, πρέπει να «ζητήσει» από τον Racer να υπολογίσει το αποτέλεσμα και να ενημερώσει με τα νέα δεδομένα τη βάση.

- Query Direct Predecessors

  Ζητούνται, δεδομένου ενός ατόμου a, τα άτομα b.(b,a):R ,όπου R συγκεκριμένος ρόλος. Η διαδικασία είναι παρόμοια με παραπάνω.

- Query Individual Filled Roles

  Ζητούνται τα ζεύγη ατόμων που σχετίζονται μεταξύ τους μέσω ενός δοσμένου ρόλου. Η διαδικασία είναι παρόμοια με παραπάνω.

Διευκρινίζουμε ότι για τα τρία τελευταία ήδη ερωτημάτων, το LAS μπορεί να απαντήσει και χωρίς τον Racer, υπολογίζοντας τις υπονοούμενες σχέσεις στη βάση με χρήση της πληροφορίας για τους μεταβατικούς και συμμετρικούς ρόλους.

Συνοψίζοντας, στο LAS, η βάση δεδομένων όχι μόνο επεκτείνει το Reasoner με τον οποίο συνεργάζεται, αλλά τον διευκολύνει και στη διαδικασία συλλογιστικής ανάλυσης λειτουργώντας ως «φίλτρο». Κάθε φορά, στέλνει στον Racer τα πιο «πιθανά» άτομα/κλάσεις, μειώνοντας έτσι κατά πολύ τον όγκο των δεδομένων που εκείνος θα έλεγχε αν δεν υπήρχε το LAS. Τα πειραματικά αποτελέσματα είναι αρκετά ενθαρρυντικά, καθότι δείχνουν εμφανή βελτίωση της απόδοσης του Reasoner.

3.2.9 Η πλατφόρμα OWLIM

Η πλατφόρμα OWLIM [OWLIM] κατασκευάστηκε με σκοπό την αποθήκευση και επεξεργασία πολύ μεγάλου όγκου σημασιολογικών σχημάτων της τάξης των δεκάδων εκατομμυρίων δηλώσεων. Εκμεταλλεύεται λογικούς φορμαλισμούς αναπαράστασης της γνώσης και διατίθεται σε δύο εκδόσεις: SwiftOWLIM και BigOWLIM. Στην πρώτη, οι διαδικασίες επαγωγής και επεξεργασίας ερωτημάτων επιτελούνται στη μνήμη (memory-based), ενώ η δεύτερη διαχειρίζεται δυαδικά αρχεία (binary files). Η πρόσβαση στο σύστημα γίνεται μέσω του πρωτοκόλλου RMI (Remote Access Invocation). Ανεξάρτητα από την έκδοση στην οποία αναφερόμαστε, μπορούμε να διακρίνουμε δύο βασικά μέρη:

- Ένα μηχανισμό αποθήκευσης βασισμένο σε αρχεία (files)
- Ένα εσωτερικό μηχανισμό συλλογιστικής ανάλυσης (TRREE Engine)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA και η ενδοεπικοινωνία μεταξύ των τμημάτων του γίνεται με κατάλληλα διαμορφωμένα APIs. Η πληροφορία την οποία μπορεί να διαχειριστεί καλύπτει όλο το RDF/S και OWL-Lite μοντέλο, με εξαίρεση τους περιορισμούς συνόλων για τιμές μεγαλύτερες του ένα.

Η έκδοση SwiftOWLIM μπορεί να ενσωματωθεί ως πακέτο SAIL (Storage and Inference Layer) στην πλατφόρμα Sesame που θα περιγράψουμε αμέσως μετά. Συνεπώς, έχει τη δυνατότητα να εκμεταλλευτεί όλα τα χαρακτηριστικά που η τελευταία προσφέρει, όπως RDF parsers, Query Engine και περιβάλλοντα αλληλεπίδρασης με το χρήστη μέσω διαδικτύου.

Η μόνιμη αποθήκευση των δεδομένων γίνεται σε ένα σύνολο αρχείων με τη μορφή N-triples. Τα αρχεία αυτά χρησιμοποιούνται μόνο ως back-up σε περίπτωση τερματισμού της λειτουργίας του συστήματος, πράγμα που σημαίνει ότι το SwiftOWLIM δεν τα χρησιμοποιεί καθόλου κατά τις διαδικασίες επαγωγής ή αποτίμησης ερωτημάτων. Οι διαδικασίες αυτές γίνονται αποκλειστικά στη μνήμη.

Για την εξαγωγή των υπονοούμενων σχέσεων χρησιμοποιείται η TRREE (Triple Rule Entailment) Engine, οι δυνατότητες της οποίας διαφέρουν ανάλογα με την έκδοση που χρησιμοποιείται. Βασικό χαρακτηριστικό αυτού του μηχανισμού είναι

ότι δεν υποστηρίζει ελέγχους συνέπειας (consistency) της οντολογίας. Υιοθετεί την forward chaining προσέγγιση και επεξεργάζεται RDF δηλώσεις τις οποίες αναπαριστά μέσω των λογικών φορμαλισμών DLP και Horst. Ο φορμαλισμός OWL-Horst ξεπερνά τον DLP που περιγράψαμε σε προηγούμενη ενότητα και δεν θα τον αναλύσουμε εδώ. Σημειώνουμε μόνο ότι εμφανίζει βελτιωμένη απόδοση συλλογιστικής ανάλυσης και επεκτείνει το RDF/S για καλύτερο χειρισμό των σταθερών (literals). Για περισσότερες πληροφορίες βλ. [OWLIM].

Στην περίπτωση του BigOWL, ο μηχανισμός αποθήκευσης είναι οργανωμένος σε binary files. Η υλοποίηση αυτή αποσκοπεί, κατά τους κατασκευαστές, στη διευκόλυνση της αρχικοποίησης του συστήματος και κατά συνέπεια στην βελτίωση της απόδοσής του. Κάθε φορά που το BigOWLIM ξεκινά, δε χρειάζεται να «ξαναδιαβάσει» και να επεξεργαστεί την οντολογία από την αρχή.

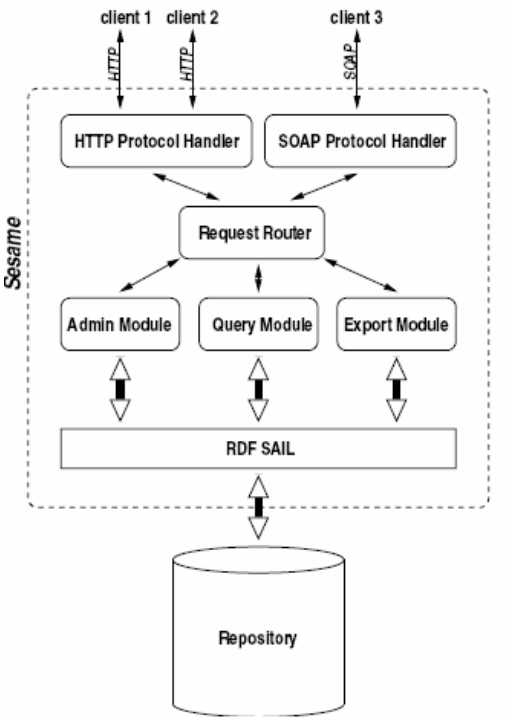
Μια βασική διαφορά σε σχέση με την προηγούμενη έκδοση (SwiftOWLIM), είναι η χρήση των αρχείων αποθήκευσης κατά τις διαδικασίες επαγωγής και αποτίμησης ερωτημάτων. Το BigOWLIM δε φορτώνει όλη την πληροφορία στη μνήμη, αλλά μόνο εκείνη που χρειάζεται για να εκτελέσει μια συγκεκριμένη λειτουργία. Έτσι, και σε συνδυασμό με το ότι «κρατάει» στατιστικά δεδομένα για τα ερωτήματα που υποβάλλονται, έχει τη δυνατότητα να διαχειριστεί πολύ μεγαλύτερο όγκο πληροφορίας με ικανοποιητική απόδοση.

3.2.10 Η πλατφόρμα Sesame

Το Sesame [Sesame] αποτελεί μια αρχιτεκτονική συστήματος αποθήκευσης σημασιολογικών σχημάτων με κύριο γνώρισμά της την ανεξαρτησία των δομικών της στοιχείων (π.χ. μηχανισμός υποβολής ερωτημάτων) από το σχήμα της βάσης και το DBMS βάσει του οποίου υλοποιήθηκε. Σχεδιάστηκε για την διαχείριση μεγάλου όγκου πληροφορίας και αποτελείται από τα εξής μέρη:

- Ένα περιβάλλον διαπροσωπείας (SAIL) με το DBMS
- Ένα μηχανισμό διαχείρισης (εισαγωγή/διαγραφή) της πληροφορίας του συστήματος (Admin Module)
- Ένα μηχανισμό υποβολής ερωτημάτων (RQL Module)
- Ένα μηχανισμό εξαγωγής της πληροφορίας (Export Module)

Στα παραπάνω σκοπίμως δεν συμπεριλάβαμε το DBMS για τον λόγο που αναφέραμε πριν. Το Sesame είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο με δυνατότητα εύκολης επέκτασης σε μέρος της OWL. Για την εξαγωγή υπονοούμενων σχέσεων υιοθετεί το backward chaining.



Σχήμα 3.10: Η αρχιτεκτονική του συστήματος Sesame

Το SAIL (Storage And Interface Layer) είναι ο πυρήνας του συστήματος. Αφ' ενός, επιφορτίζεται με τη μετάφραση των RDF/S ερωτημάτων σε κλήσεις του συστήματος διαχείρισης της βάσης (SQL) και αντιστρόφως. Αφ' ετέρου, μεταφράζει όλες τις μεθόδους των επιμέρους τμημάτων (modules) σε ενέργειες αναγνωρίσιμες από το DBMS. Αποτελεί, με άλλα λόγια, ένα ευέλικτο δίαυλο επικοινωνίας ολόκληρου του συστήματος με τη Βάση Δεδομένων και είναι αυτό που του χαρίζει την πολύτιμη ανεξαρτησία από τις συγκεκριμένες επιλογές του διαχειριστή.

Το Admin Module προσφέρει τη δυνατότητα αυξητικής (incremental) εισαγωγής RDF/S δεδομένων στη βάση, καθώς και ολικής διαγραφής της όταν αυτό απαιτείται. Με τη χρήση ενός αναλυτή (ARP), διαβάζει το RDF/S αρχείο που βρίσκεται σε XML μορφή και φορτώνει, μέσω του SAIL, την πληροφορία στο ειδικό σχήμα της βάσης που ο διαχειριστής έχει επιλέξει.

Το RQL Module μεταφράζει τα εκφρασμένα με RQL γλώσσα ερωτήματα σε δενδρικά μοντέλα συμβατά με το SAIL, χωρίς όμως να περιορίζεται σε αυτό. Πέρα από τον απαραίτητο αναλυτή για την αρχική μετατροπή, περιλαμβάνει και ένα μηχανισμό βελτιστοποίησης ερωτημάτων, έτσι ώστε αυτά να «φτάνουν» στο SAIL έτοιμα για μεταβίβαση στο DBMS. Αν και αυτό φαίνεται περιττό, καθότι η βελτιστοποίηση γίνεται εύκολα από το DBMS, ωστόσο, έτσι ενισχύεται η ανεξαρτησία του συστήματος.

Το Export Module είναι ιδιαίτερα απλό. Επιτελεί τη λειτουργία εξαγωγής της RDF/S πληροφορίας σε XML μορφή έτσι ώστε να είναι αναγνώσιμη από κειμενογράφους (editors).

Το Sesame υποστηρίζει την επικοινωνία με τον «εξωτερικό κόσμο» βάσει πολλαπλών πρωτοκόλλων όπως HTTP, RMI και SOAP. Προσφέρει τους αντίστοιχους χειριστές (handlers) μαζί με ένα μηχανισμό δρομολόγησης των διαφόρων αιτήσεων (client requests). Όλα αυτά φαίνονται συνοπτικά στο Σχήμα 3.10.

Στο σημείο αυτό, αξίζει μια αναφορά στα ιδιαίτερα χαρακτηριστικά του SAIL σε σχέση με άλλα RDF APIs:

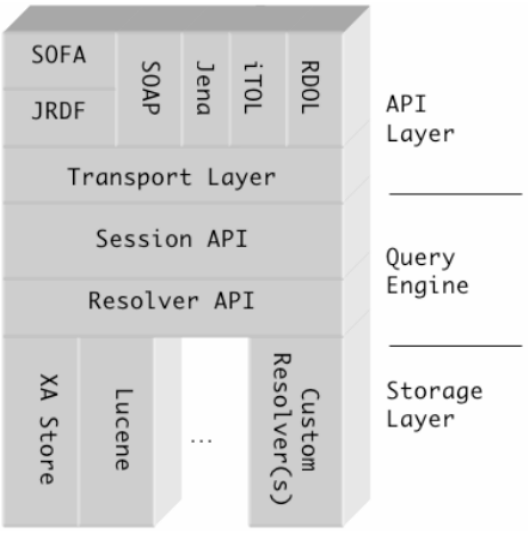
- Το SAIL έχει πλήρη εικόνα του σχήματος της βάσης. Έτσι, επαγωγικές διεργασίες που αφορούν είτε στην υπαγωγή (subsumption) κλάσεων/ιδιοτήτων είτε στο πεδίο ή εύρος ιδιοτήτων γίνονται εσωτερικά. Αυτό σημαίνει ότι ο χρήστης μπορεί ευθέως να ρωτήσει τέτοιου είδους πληροφορία σε αντίθεση με άλλα APIs (π.χ. Jena Toolkit και Redland Application Framework) που αυτό δεν υποστηρίζεται.
- Για τον ίδιο λόγο με παραπάνω (γνώση του σχήματος της βάσης), το SAIL μπορεί να χρησιμοποιηθεί για την εξασφάλιση της συνέπειάς της (consistency). Αυτό σημαίνει ότι «παράνομες» εγγραφές ή διαγραφές είναι δυνατόν να αποκλείονται από το Interface που σε αυτή την περίπτωση λειτουργεί ως «φίλτρο».
- Το SAIL προσφέρει την δυνατότητα πολλών επιπέδων κατά τη χρήση του. Με απλά λόγια, μπορεί να τοποθετηθεί το ένα πάνω στο άλλο, λειτουργώντας έτσι ως ένας «εικονικός μηχανισμός» προσωρινής αποθήκευσης (cache) πληροφορίας που ζητείται συχνά ή είναι δύσκολο να υπολογιστεί (π.χ. υπαγωγή).

3.2.11 Η πλατφόρμα Kowari

Η πλατφόρμα Kowari [Kowari] κατασκευάστηκε με σκοπό την αποθήκευση και διαχείριση πολύ μεγάλου όγκου σημασιολογικής πληροφορίας της τάξης των εκατοντάδων εκατομμυρίων RDF δηλώσεων. Η «φιλοσοφία» του μοιάζει πολύ με αυτή του Sesame. Βασικό του γνώρισμα είναι η χρήση ενός εσωτερικού (native) συστήματος αποθήκευσης των RDF δεδομένων (XA Statement Store) με χρήση AVL trees, ενώ ταυτόχρονα προσφέρει και τη δυνατότητα σύνδεσης με βάση δεδομένων λειτουργώντας ως εξυπηρετητής (server). Αποτελείται από τα εξής μέρη :

- Ένα περιβάλλον διαπροσωπίας (Resolver API) μεταξύ των δομικών στοιχείων του συστήματος και του συστήματος αποθήκευσης (data store) είτε αυτό είναι εσωτερικό (XA Store) είτε εξωτερικό (database)
- Ένα σύστημα αποθήκευσης δεδομένων στο σκληρό δίσκο (XA Statement Store)
- Ένα μηχανισμό υποβολής ερωτημάτων (Query Engine)

Το σύστημα είναι γραμμένο σε γλώσσα προγραμματισμού JAVA. Η επικοινωνία μεταξύ των τμημάτων του γίνεται με χρήση κατάλληλα διαμορφωμένων APIs. Η πληροφορία που μπορεί να αποθηκευτεί καλύπτει όλο το RDF/S μοντέλο και μέρος της OWL (OWL Lite και περιορισμοί αριθμού). Για την εξαγωγή υπονοούμενων σχέσεων υιοθετεί την τεχνική forward chaining.



Σχήμα 3.11: Η αρχιτεκτονική του συστήματος Kowari

Το Resolver API είναι το βασικό τμήμα της πλατφόρμας. Μπορούμε να το θεωρήσουμε ως ένα «πολύ-εργαλείο» που επιτελεί όλες τις ενέργειες που έχουν σχέση με το σύστημα αποθήκευσης. Επίσης, μεταφράζει και τα RDF/S ερωτήματα σε μοντέλα (graph implementations) που μπορεί να χειρίζονται οι Resolvers. Οι τελευταίοι διακρίνονται σε εσωτερικούς (internal) και εξωτερικούς (external). Παράδειγμα εσωτερικού Resolver είναι το XA Statement Store, ήδη ενσωματωμένο στο Kowari. Από την άλλη, μια Βάση Δεδομένων μπορεί να συνδεθεί (ενδεχομένως μέσω κάποιου πρωτοκόλλου) με το σύστημα ως εξωτερικός Resolver.

Η διαπροσωπία έχει σχεδιαστεί με τέτοιο τρόπο ώστε να επιτρέπει κατασκευή μοντέλων συμβατών με σχεδόν οποιοδήποτε σύστημα αποθήκευσης. Διαιρείται σε τρία υπο-τμήματα: Resolver Interface, ResolverFactory Interface και Resolution Interface. Το πρώτο είναι υπεύθυνο για τη διαχείριση των συναλλαγών (transactions) και τη μετατροπή (modification) των δεδομένων. Το δεύτερο χειρίζεται τις διαδικασίες αρχικοποίησης και εκκαθάρισης, ενώ δημιουργεί και τα στιγμιότυπα (instances) του Resolver. Το τρίτο αφορά στην επιστροφή των δεδομένων από τον Resolver.

Ιδιαίτερο ενδιαφέρον παρουσιάζει το σύστημα αποθήκευσης XA Statement Store. Η ουσιαστική του διαφορά σε σχέση με τις σχεσιακές Βάσεις Δεδομένων είναι ότι χειρίζεται τα κατηγορήματα (predicates) ως δεδομένα και όχι ως σχέσεις (relationships). Αποθηκεύει τις RDF δηλώσεις με τη μορφή «μονάδων» αποτελούμενων από Υποκείμενο, Κατηγόρημα, Αντικείμενο και μετα-κόμβους. Οι μετα-κόμβοι δηλώνουν την οντολογία στην οποία ανήκει η δήλωση και παρέχονται από το Resolver API. Οι κόμβοι (nodes) που συνθέτουν μια δήλωση αποθηκεύονται ως 64-bit ακέραιοι σε μια δομή που ονομάζεται Node Pool. Τα URIs, Literals και XML datatypes στα οποία αντιστοιχούν, αποθηκεύονται σε αρχεία (files). Μια άλλη δομή, ονόματι String Pool, χρησιμοποιεί AVL δέντρα για την κατασκευή ευρετηρίων στα αρχεία που περιέχουν τα RDF/S δεδομένα. Τα AVL trees επιλέχθηκαν επειδή ακριβώς προσφέρουν γρήγορη και απλή εύρεση. Τα υπόλοιπα ευρετήρια του XA Statement Store μπορούν να θεωρηθούν ως υβρίδια AVL/B*-trees, καθότι παρέχουν τα πλεονεκτήματα και των δύο δενδρικών δομών. Ένα επιπλέον χαρακτηριστικό του συστήματος είναι η χρήση παράλληλων ευρετηρίων που έχει ως στόχο τον περιορισμό τους σε μικρά μεγέθη, έτσι ώστε να μπορούν εύκολα να φορτωθούν στη μνήμη. Περισσότερες λεπτομέρειες για το μηχανισμό ευρετηρίων δίνονται στο [Kowari].

Η Query Engine (QE) του Kowari αναλύει (parses), βελτιστοποιεί (optimizes) και προωθεί ερωτήματα γραμμένα σε iTQL [Kowari], μια γλώσσα που δεν αποτελεί πρότυπη RDF γλώσσα ερωτημάτων και κατασκευάστηκε παράλληλα με το σύστημα. Η περιγραφή της ξεπερνά τους σκοπούς αυτής της εργασίας. Σημειώνουμε, απλώς, εδώ ότι η QE επιτρέπει την υιοθέτηση πολλών γλωσσών ερωτημάτων (μεταξύ αυτών και της RDQL) μέσω APIs.

Η σύνδεση με την πλατφόρμα μπορεί να γίνει με πολλούς τρόπους. Τα εξωτερικά APIs που υποστηρίζονται είναι: SOFA (Simple Ontology Framework API) , Java RDF, Jena και SOAP.

Στον Πίνακα 3.3 συνοψίζονται τα χαρακτηριστικά όλων των συστημάτων που αναλύσαμε. Η πρώτη γραμμή αναφέρεται στο είδος της πληροφορίας που μπορούν τα συστήματα να αποθηκεύσουν. Ωστόσο, ορισμένες υλοποιήσεις, επειδή ακριβώς συνεργάζονται με Reasoner, μπορούν, έστω και εν μέρει, να επεξεργαστούν πληροφορία μεγαλύτερου εύρους από αυτήν που αποθηκεύουν, καθότι αυτή διέρχεται πρώτα από το σύστημα συλλογιστικής ανάλυσης. Παραδείγματα τέτοιων περιπτώσεων αποτελούν τα DLDB και Instance Store.

Όσον αφορά στην τεχνική εξαγωγής υπονοούμενων σχέσεων, συμβολίζουμε με f.c. την forward chaining και με b.c. την backward chaining.

	RDFSuite	3Store	RStar	Jena2	KAON	DLDB	Instance Store	LAS	OWLIM	Sesame	Kowari
Εκφραστικότητα που αποθηκεύεται	RDF/S	RDF/S	RDF/S	RDF/S Υποσύνολο OWL (cardinalities)	RDF/S OWL-Lite	RDF/S (εκτός domain & range)	role-free υποσύνολο RDF/S και OWL	SHR+	RDF/S OWL-Lite	RDF/S	RDF/S OWL-Lite (full cardinality constraints)
Σχήμα ΒΔ ή Μηχανισμός Αποθήκευσης	Specific Hybrid	Generic	Hybrid	Generic	Generic Hybrid	Specific	Hybrid	Hybrid	File System	File System Generic Specific Hybrid	File System
Χρήση MatViews	-	-	-	-	-	NAI	-	-	-	-	-
Τεχνική εξαγωγής υπονοούμενων σχέσεων	f.c.	f.c. b.c.	b.c.	f.c	f.c.	b.c.	f.c. b.c.	f.c. b.c.	f.c.	b.c.	f.c.
Γλώσσα Ερωτημάτων	RQL	RDQL	RSQL	RDQL SPARQL	KAON QL	KIF-like μορφή	-	-	RQL	RQL	iTQL RDQL
Reasoner	-	-	-	internal Reasoners Pellet, Racer	FaCT Bubo	FaCT	FaCT Racer FaCT++	Racer	TRREE	-	-

Πίνακας 3.3: Συγκεντρωτικά χαρακτηριστικά έντεκα συστημάτων διαχείρισης οντολογιών

4

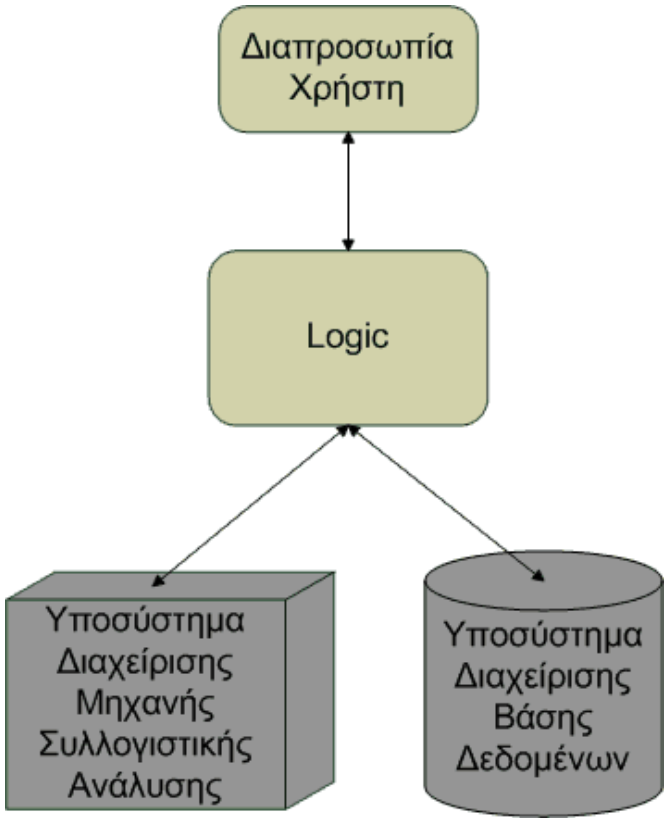
Ανάλυση Συστήματος

Στο κεφάλαιο αυτό αναλύουμε τις απαιτήσεις που ικανοποιεί το DBRS. Η μεθοδολογία παρουσίασης έχει ως βασικό άξονα το διαχωρισμό του συστήματος σε διακριτά υποσυστήματα (Ενότητα 4.1), την συνοπτική περιγραφή των λειτουργιών που αυτά επιτελούν (Ενότητα 4.2) και την ανάδειξη του τρόπου επικοινωνίας μεταξύ τους. Ως υποσύστημα ορίζουμε μια «οικογένεια» από εφαρμογές που είτε επιτελούν παρόμοιες λειτουργίες (και επομένως μπορούν να ομαδοποιηθούν) είτε συμμετέχουν στην εκτέλεση ενός αυτοτελούς καθήκοντος, όπως για παράδειγμα στη φόρτωση του TBox της οντολογίας. Ένα υποσύστημα μπορεί να διαιρείται σε επιμέρους υποσυστήματα (υποσύνολα εφαρμογών) τα οποία, με τη σειρά τους, μπορεί επίσης να διαιρούνται σε άλλα κ.ο.κ. Στην Ενότητα 4.3, δίνουμε το διάγραμμα Οντοτήτων-Συσχετίσεων της βάσης δεδομένων που χρησιμοποιεί το DBRS και το οποίο σχεδιάστηκε με σκοπό την πλήρη και αποδοτική αποθήκευση σημασιολογικών σχημάτων εκφραστικότητας $SHOIN^{(D)}$.

4.1 Αρχιτεκτονική - Διαχωρισμός υποσυστημάτων

Το DBRS αποτελείται από τρία βασικά υποσυστήματα. Το υποσύστημα του χρήστη, το υποσύστημα διαχείρισης της βάσης δεδομένων και το υποσύστημα διαχείρισης της μηχανής συλλογιστικής ανάλυσης. Το πρώτο από αυτά διαιρείται σε

τέσσερα υποσυστήματα, ένα εκ των οποίων (το υποσύστημα φόρτωσης της οντολογίας) διαιρείται σε δύο επιπλέον. Αφαιρετικά, θα μπορούσαμε να πούμε ότι το DBRS διαθέτει μια τριεπίπεδη (three-layered) αρχιτεκτονική η οποία φαίνεται στο Σχήμα 4.1. Το πρώτο επίπεδο αφορά στη γραφική διαπροσωπία με την οποία ο χρήστης έρχεται σε επαφή με το σύστημα, ενώ το δεύτερο περιλαμβάνει όλα εκείνα τα υποσυστήματα του DBRS που αποτελούν το επίπεδο Λογικής (Logic) και μεσολαβούν μεταξύ του πρώτου και του τρίτου επιπέδου, δηλαδή μεταξύ της διαπροσωπίας του χρήστη και των υποσυστημάτων διαχείρισης της βάσης δεδομένων και της μηχανής συλλογιστικής ανάλυσης. Το επίπεδο Λογικής του DBRS αποτελεί στην ουσία και τον πυρήνα (core) του συστήματος.



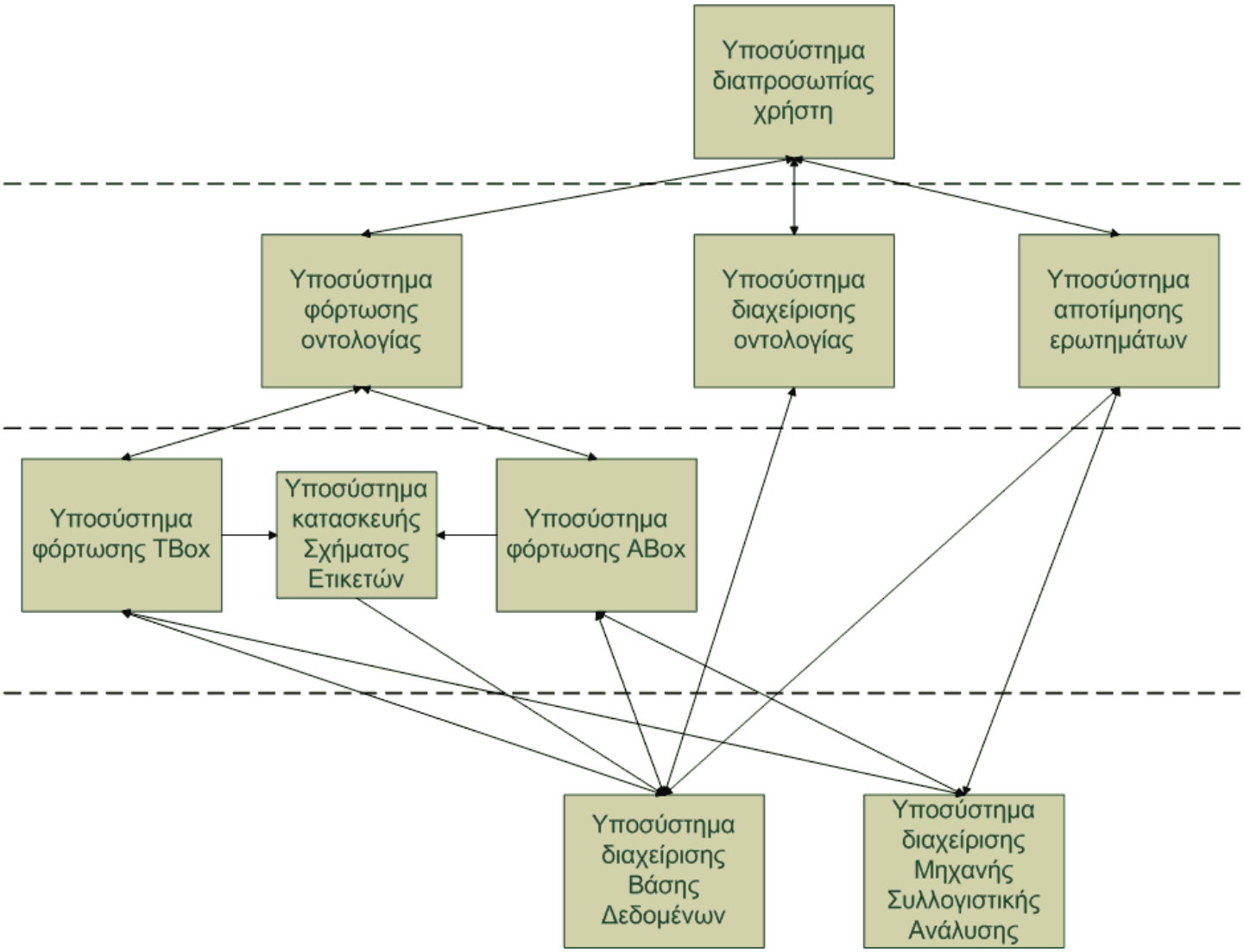
Σχήμα 4.1: Τριεπίπεδη Αρχιτεκτονική του συστήματος DBRS

Πιο συγκεκριμένα και όπως φαίνεται στο Σχήμα 4.2, η δομή του DBRS είναι:

1. Υποσύστημα γραφικής διαπροσωπίας χρήστη
2. Υποσύστημα φόρτωσης οντολογίας
   2.1. Υποσύστημα φόρτωσης TBox
   2.2. Υποσύστημα φόρτωσης ABox
3. Υποσύστημα κατασκευής σχήματος ετικετών
4. Υποσύστημα διαχείρισης οντολογίας
5. Υποσύστημα αποτίμησης ερωτημάτων
6. Υποσύστημα διαχείρισης Βάσης Δεδομένων
7. Υποσύστημα διαχείρισης Μηχανής Συλλογιστικής Ανάλυσης

Η κατεύθυνση των βελών δηλώνει ροή δεδομένων είτε από το ένα επίπεδο στο άλλο (για το Σχήμα 4.1), είτε από το ένα υποσύστημα στο άλλο (για το Σχήμα 4.2). Η αντιστοιχία μεταξύ των δύο σχημάτων έχει ως εξής:

Το Υποσύστημα γραφικής διαπροσωπίας χρήστη του Σχήματος 4.2 αντιστοιχεί στη Διαπροσωπία Χρήστη, δηλαδή στο πρώτο επίπεδο του Σχήματος 4.1. Τα υπόλοιπα Υποσυστήματα του Σχήματος 4.2 αντιστοιχούν στο δεύτερο επίπεδο (Logic) του Σχήματος 4.1, με εξαίρεση τα Υποσυστήματα διαχείρισης Βάσης Δεδομένων και Μηχανής Συλλογιστικής Ανάλυσης που αντιστοιχούν στο τρίτο και κατώτερο επίπεδο. Για την ακρίβεια, ως Υποσύστημα διαχείρισης Μηχανής Συλλογιστικής Ανάλυσης (στο Σχήμα 4.2), εννοούμε την ίδια τη Μηχανή Συλλογιστικής Ανάλυσης (Reasoner), μαζί με την εφαρμογή διασύνδεσής της (Reasoner Wrapper). Το ίδιο ακριβώς ισχύει και για το Υποσύστημα διαχείρισης Βάσης Δεδομένων του Σχήματος 4.2, το οποίο περιλαμβάνει το DBMS και την εφαρμογή διασύνδεσής του (DBMS Wrapper) με τα λοιπά υποσυστήματα του DBRS.



Σχήμα 4.2: Αρχιτεκτονική του συστήματος DBRS

4.2 Περιγραφή υποσυστημάτων

Στην ενότητα αυτή περιγράφεται συνοπτικά η λειτουργία κάθε υποσυστήματος που αναφέρθηκε προηγουμένως. Οι εφαρμογές που υλοποιούν αυτές τις λειτουργίες θα παρουσιαστούν λεπτομερώς στο επόμενο κεφάλαιο.

4.2.1 Υποσύστημα γραφικής διαπροσωπίας χρήστη

Το Υποσύστημα γραφικής διαπροσωπίας χρήστη αποτελεί το περιβάλλον με το οποίο ο χρήστης έρχεται σε επαφή με το σύστημα. Μέσω αυτού, του δίνεται η πρόσβαση στις λειτουργίες που προσφέρει το DBRS σχετικά με τη φόρτωση και διαχείριση οντολογιών, αλλά και με τη διεξαγωγή ερωτημάτων.

Όσον αφορά στη φόρτωση οντολογιών, παρέχονται στο χρήστη οι εξής δυνατότητες:

- Να ορίσει το Web URL του αρχείου της οντολογίας που θέλει να φορτώσει ή να αναζητήσει το αρχείο στο σύστημα του (browse).
- Να ορίσει τις παραμέτρους φόρτωσης της οντολογίας. Αυτές είναι ανεξάρτητες μεταξύ τους και μπορούν να επιλεγούν σε οποιοδήποτε συνδυασμό. Επιγραμματικά είναι οι εξής:
    - Υπολογισμός πλήρους πληροφορίας περί ασυμβατότητας κλάσεων
    - Υπολογισμός πλήρους πληροφορίας περί ασυμβατότητας ρόλων
    - Υπολογισμός πλήρους πληροφορίας περί διαφορετικών ατόμων
    - Υπολογισμός πλήρους πληροφορίας περί συμπληρωματικών κλάσεων
    - Αποθήκευση ψευδομοντέλων κλάσεων
    - Αποθήκευση ψευδομοντέλων ατόμων
    - Κατασκευή και αποθήκευση του διευρυμένου TBox

Όσον αφορά στη διεξαγωγή ερωτημάτων, προσφέρονται τα εξής:

- Ένα πεδίο όπου ο χρήστης συντάσσει το ερώτημα σε ψευδογλώσσα.
- Ένα πεδίο όπου αναγράφονται τα επεξηγηματικά μηνύματα του συστήματος, καθώς και το αποτέλεσμα αποτίμησης του ερωτήματος που τέθηκε από τον χρήστη.
- Ένα κουμπί με το οποίο, όταν επιλεγεί από το χρήστη, το σύστημα μπαίνει σε «οκνηρή» λειτουργία (lazy mode) και απαντάει στα ερωτήματα αποκλειστικά μέσω του Reasoner. Σε αυτή την περίπτωση, η επιπλέον πληροφορία που προκύπτει (από την απάντηση του Reasoner) αποθηκεύεται στη βάση δεδομένων (log λειτουργία).
- Ένα κουμπί βοήθειας (help), με το οποίο ο χρήστης μπορεί να δει ποια είναι η σωστή σύνταξη των ερωτημάτων στη ψευδογλώσσα που έχει υλοποιηθεί.

4.2.2 Υποσύστημα φόρτωσης οντολογίας

Το Υποσύστημα φόρτωσης οντολογίας περιλαμβάνει τις λειτουργίες που επιτελούνται κατά τη φόρτωση της οντολογίας στη βάση δεδομένων του DBRS, παραμετροποιημένες με βάση τις επιλογές του χρήστη. Διαιρείται στο Υποσύστημα φόρτωσης TBox και στο Υποσύστημα φόρτωσης ABox που περιγράφουμε στη συνέχεια.

4.2.2.1 Υποσύστημα φόρτωσης TBox

Το Υποσύστημα φόρτωσης TBox περιλαμβάνει τις λειτουργίες που σχετίζονται με τη φόρτωση του TBox της οντολογίας. Αυτές είναι:

- Αποθήκευση Namespaces οντολογίας
- Αποθήκευση κλάσεων και κωδικοποίηση της ιεραρχίας τους μέσω κλήσης του Υποσυστήματος κατασκευής σχήματος ετικετών
- Αποθήκευση πληροφορίας περί ασυμβατότητας κλάσεων
- Αποθήκευση ρόλων και κωδικοποίηση της ιεραρχίας τους μέσω κλήσης του Υποσυστήματος κατασκευής σχήματος ετικετών
- Αποθήκευση πληροφορίας περί αντίστροφων ρόλων
- Αποθήκευση πληροφορίας περί χαρακτηριστικών ρόλων
- Αποθήκευση πληροφορίας σχετικά με το πεδίο (domain) και το εύρος (range) των ρόλων
- Αποθήκευση πληροφορίας περί ασυμβατότητας ρόλων
- Αποθήκευση ορισμών κλάσεων
- Κατασκευή και αποθήκευση του διευρυμένου TBox
- Αποθήκευση ψευδομοντέλων κλάσεων

4.2.2.2 Υποσύστημα φόρτωσης ABox

Το Υποσύστημα φόρτωσης ABox περιλαμβάνει τις λειτουργίες που σχετίζονται με τη φόρτωση του ABox της οντολογίας. Αυτές είναι:

- Αποθήκευση ατόμων
- Αποθήκευση πληροφορίας περί ταυτότητας ατόμων
- Αποθήκευση πληροφορίας περί διαφορετικών ατόμων
- Αποθήκευση δηλώσεων κλάσεων
- Αποθήκευση δηλώσεων ρόλων αντικειμένου και ρόλων τύπου δεδομένων
- Αποθήκευση δηλώσεων μεταβατικών ρόλων και κωδικοποίησή τους μέσω κλήσης του Υποσυστήματος κατασκευής σχήματος ετικετών
- Αποθήκευση ψευδομοντέλων ατόμων

4.2.3 Υποσύστημα κατασκευής Σχήματος Ετικετών

Το Υποσύστημα κατασκευής Σχήματος Ετικετών αναλαμβάνει την κωδικοποίηση (μέσω ετικετών) των σχέσεων ιεραρχίας της οντολογίας και την αποθήκευσή τους στη βάση δεδομένων. Οι σχέσεις ιεραρχίας μπορεί να αναφέρονται σε κλάσεις, ρόλους, αλλά και δηλώσεις μεταβατικών ρόλων.

4.2.4 Υποσύστημα διαχείρισης οντολογίας

Το Υποσύστημα διαχείρισης οντολογίας περιλαμβάνει όλες τις λειτουργίες που σχετίζονται με τη διαχείριση των οντολογιών οι οποίες είναι αποθηκευμένες σε βάση δεδομένων. Στην παρούσα έκδοση, οι δυνατότητες που προσφέρονται στο χρήστη είναι:

- Να δει τα ονόματα των υπαρχουσών βάσεων δεδομένων και των ιδιοκτητών τους.
- Να εκκαθαρίσει μια βάση από τα δεδομένα της, με την προϋπόθεση βέβαια ότι έχει τα απαραίτητα δικαιώματα.
- Να ενημερώσει το σχήμα ετικετών για τα άτομα των δηλώσεων κάποιου μεταβατικού ρόλου.

4.2.5 Υποσύστημα αποτίμησης ερωτημάτων

Το Υποσύστημα αποτίμησης ερωτημάτων αναλαμβάνει την αναγνώριση των ερωτημάτων που υποβάλλονται από τον χρήστη μέσω του GUI, τον έλεγχο των παραμέτρων τους και την επιλογή της στρατηγικής αποτίμησής τους. Μπορεί να

λειτουργεί είτε σε «πρόθυμη», όπου χρησιμοποιούνται από κοινού DBMS και Reasoner, είτε σε «οκνηρή» λειτουργία, όπου απαντάει μόνο ο Reasoner. Τα ερωτήματα που υποστηρίζει αντιστοιχούν σε γνώση εκφραστικότητας $SHOIN^{(D)}$[28] και αφορούν σε:

- Ισοδυναμία κλάσεων (Equivalent classes).
- Συμπληρωματικότητα κλάσεων (Complement classes).
- Υπαγωγή κλάσεων (Subclasses).
- Ασυμβατότητα κλάσεων (Disjoint classes).
- Ισοδυναμία ρόλων (Equivalent roles).
- Υπαγωγή ρόλων (Subroles).
- Ασυμβατότητα ρόλων (Disjoint roles).
- Αντίστροφους ρόλους (Inverse roles).
- Συμμετρικότητα ρόλου (Symmetric role).
- Μεταβατικότητα ρόλου (Transistive role).
- Λειτουργικότητα ρόλου αντικειμένου (Functional object role).
- Λειτουργικότητα ρόλου τύπου δεδομένων (Functional datatype role).
- Λειτουργικότητα και αντιστροφή ρόλου (Inverse Functional role).
- Είδος ρόλου (αντικειμένου ή τύπου δεδομένων).
- Τύπους ατόμου - Δηλώσεις κλάσης (Individual types - Class assertions).
- Ταυτότητα ατόμων (Same individuals).
- Διαφορετικότητα ατόμων (Different individuals).
- Δηλώσεις ρόλων (Role assertions).
- Πεδίο ρόλου (Domain).
- Εύρος ρόλου (Range).

[28] Με εξαίρεση το ερώτημα σχετικά με ασυμβατότητα ρόλων. Όπως αναφέραμε στο Κεφάλαιο 2, η ασυμβατότητα ρόλων (*R*) δεν περιλαμβάνεται στο τμήμα *SHOIN(D)* της Περιγραφικής Λογικής.

Σε αυτά έρχονται να προστεθούν και τρία επιπλέον ερωτήματα που ονομάζονται «εποπτικά» και αφορούν στην εύρεση όλων εκείνων των σχετικών με τη δοσμένη από τον χρήστη οντότητα (κλάση/ρόλο/άτομο) αξιωμάτων. Περισσότερες λεπτομέρειες γι' αυτά δίνονται στο Κεφάλαιο 5. Εδώ τα αναφέρουμε επιγραμματικά:

- Εποπτικό ερώτημα σχετικά με κλάση.
- Εποπτικό ερώτημα σχετικά με ρόλο.
- Εποπτικό ερώτημα σχετικά με άτομο.

Σημειώνουμε ότι, σε όλα τα παραπάνω, όπου αναφερόμαστε σε κλάση, εννοούμε γνωστή (named) ή αυθαίρετη (arbitrary) κλάση που έχει δοθεί από τον χρήστη του DBRS κατά την υποβολή του ερωτήματος.

4.2.6 Υποσύστημα διαχείρισης Βάσης Δεδομένων

Το Υποσύστημα διαχείρισης Βάσης Δεδομένων είναι υπεύθυνο για την επικοινωνία των υποσυστημάτων του DBRS με τη βάση δεδομένων και ως εκ τούτου αναλαμβάνει την εκτέλεση όλων εκείνων των λειτουργιών που σχετίζονται με εισαγωγή, ενημέρωση και εξαγωγή πληροφορίας από αυτήν. Επίσης, διεκπεραιώνει όλες τις αιτήσεις που αφορούν είτε σε δημιουργία νέας βάσης είτε σε διαχείριση υπάρχουσας, αλλά και τις αιτήσεις για εγγραφή νέου χρήστη στο DBMS.

Η βάση δεδομένων του DBRS είναι μια σχεσιακή αναπαράσταση της βάσης γνώσης του Reasoner και υπό αυτήν την έννοια μπορεί να θεωρηθεί ως μια «σχεσιακή βάση γνώσης». Στην ουσία, το DBRS, με το Υποσύστημα διαχείρισης Βάσης Δεδομένων να έχει προεξέχοντα ρόλο σε αυτό, δημιουργεί μια αμφίδρομη επικοινωνία ανάμεσα στη βάση γνώσης του Reasoner (που βρίσκεται αποκλειστικά στην κύρια μνήμη) και στη βάση δεδομένων που βρίσκεται στη δευτερεύουσα μνήμη, δίνοντας τη δυνατότητα στο χρήστη να συνδέεται ταυτόχρονα και με τις δύο.

4.2.7 Υποσύστημα διαχείρισης Μηχανής Συλλογιστικής Ανάλυσης

Το Υποσύστημα διαχείρισης Μηχανής Συλλογιστικής Ανάλυσης είναι υπεύθυνο για τη επικοινωνία των υποσυστημάτων του DBRS με τον Reasoner. Στην ουσία, αναλαμβάνει την εκτέλεση όλων εκείνων των λειτουργιών που σχετίζονται με εξαγωγή πληροφορίας από τη βάση γνώσης (στην κύρια μνήμη) του Reasoner.

4.3 Μοντέλο Οντοτήτων Συσχετίσεων

Στο Σχήμα 4.3 δίνεται το μοντέλο Οντοτήτων Συσχετίσεων (Entity Relationship Diagram) της βάσης δεδομένων που χρησιμοποιεί το DBRS και το οποίο αποτελεί μια σχεσιακή αναπαράσταση γνώσης με εκφραστικότητα $SHOIN^{(D)}$.

Το μοντέλο διαθέτει τέσσερα κύρια σύνολα οντοτήτων που αναπαριστούν τα namespaces, τις κλάσεις (classes), τα άτομα (individuals) και τους ρόλους (roles) της οντολογίας. Όλα τα υπόλοιπα σύνολα οντοτήτων είναι αδύνατα και καθορίζονται από τις σχέσεις τους με τα τέσσερα κύρια. Συνολικά έχουμε έντεκα σύνολα οντοτήτων:

- Namespace

  Αναπαριστά τα namespaces που αποθηκεύονται στη βάση δεδομένων. Έχει δύο ιδιότητες: id_namespace (Πρωτεύον κλειδί) και namespace.

- Κλάση

  Αναπαριστά τις κλάσεις που αποθηκεύονται στη βάση δεδομένων είτε αυτές είναι γνωστές (named) είτε ανώνυμες (anonymous)[29]. Έχει δώδεκα ιδιότητες από τις οποίες οι τρεις (id_father_class, c_label_pre, c_label_post, DAG Ετικέτα Κλάσης) αφορούν στο σχήμα ετικετών και οι πέντε (c_assertion_complete, c_disj_complete, c_compl_complete, unique_pseudomodel, unique_complement_pseudomodel) στην πληρότητα της πληροφορίας που είναι αποθηκευμένη στη βάση. Η πλειότιμη ιδιότητα DAG Ετικέτα Κλάσης αποτελείται από τις επιμέρους ιδιότητες c_label_pre_dag και c_label_post_dag. Οι υπόλοιπες ιδιότητες είναι: id_concept (Πρωτεύον κλειδί), concept_name, concept_definition, expanded_concept_definition.

  Οι ανώνυμες κλάσεις δεν τοποθετούνται (μέσω ετικετών) στη συνολική ιεραρχία των κλάσεων της οντολογίας, αλλά απλώς αποθηκεύονται στο πεδίο

[29] Οι μόνες ανώνυμες κλάσεις που επιτρέπονται σε εκφραστικότητα *SHOIN(D)* και που αποθηκεύονται στη βάση δεδομένων του DBRS είναι το ρητά δηλωμένο πεδίο (domain) και εύρος (range) ρόλων της οντολογίας.

concept_definition, έχοντας ως ιδιότητα concept_name το χαρακτηριστικό «anonymous».

- Ρόλος

  Αναπαριστά τους ρόλους που αποθηκεύονται στη βάση δεδομένων. Έχει οκτώ ιδιότητες από τις οποίες οι τέσσερεις (id_father_role, r_label_pre, r_label_post, DAG Ετικέτα Ρόλου) αφορούν στο σχήμα ετικετών και οι δύο (r_assertion_complete, r_disj_complete) στην πληρότητα της πληροφορίας που είναι αποθηκευμένη στη βάση. Η πλειότιμη ιδιότητα DAG Ετικέτα Ρόλου αποτελείται από τις επιμέρους ιδιότητες r_label_pre_dag και r_label_post_dag. Οι υπόλοιπες ιδιότητες είναι: id_role (Πρωτεύον κλειδί), role_name και role_characteristics.

- Σταθερά

  Αναπαριστά τις σταθερές της οντολογίας που αποθηκεύονται στη βάση δεδομένων. Έχει δύο ιδιότητες (Πρωτεύον Κλειδί) από τις οποίες η μια (value) δηλώνει την τιμή της σταθεράς και η άλλη (type) τον XML τύπο της.

- Άτομο

  Αναπαριστά τα άτομα της οντολογίας που αποθηκεύονται στη βάση δεδομένων. Έχει πέντε ιδιότητες από τις οποίες οι τρεις (i_type_complete, i_diff_complete, unique_pseudomodel) αφορούν στην πληρότητα της πληροφορίας που είναι αποθηκευμένη στη βάση. Οι υπόλοιπες είναι: id_individual (Πρωτεύον κλειδί) και individual_name.

- Ψευδομοντέλο κλάσης

  Αδύνατο σύνολο οντοτήτων που αναπαριστά τα ψευδομοντέλα των κλάσεων που αποθηκεύονται στη βάση δεδομένων. Προσδιοριστικό του σύνολο είναι η Κλάση με το οποίο συνδέεται μέσω του συνόλου σχέσεων «έχει ψευδομοντέλο». Συνδέεται με τα σύνολα Κλάση και Ρόλος μέσω των συνόλων σχέσεων «συμμετέχει σε M^{A}», «συμμετέχει σε $M^{\neg A}$», « συμμετέχει σε $M^{\exists}$ » και «συμμετέχει σε $M^{\forall}$ ». Επίσης, έχει δύο ιδιότητες, τις id_concept_pseudomodel και complement.

- Ψευδομοντέλο ατόμου

  Αδύνατο σύνολο οντοτήτων που αναπαριστά τα ψευδομοντέλα των ατόμων που αποθηκεύονται στη βάση δεδομένων. Προσδιοριστικό του σύνολο είναι

το Άτομο με το οποίο συνδέεται μέσω του συνόλου σχέσεων «έχει ψευδομοντέλο». Συνδέεται με τα σύνολα Κλάση και Ρόλος μέσω των συνόλων σχέσεων «συμμετέχει σε M^{A}», «συμμετέχει σε $M^{\neg A}$», « συμμετέχει σε $M^{\exists}$ » και «συμμετέχει σε $M^{\forall}$ ». Επίσης, έχει μια ιδιότητα, την id_individual_pseudomodel.

Πέρα από τα βασικά σύνολα οντοτήτων που αναφέραμε, υπάρχουν και τρία ακόμα, τα οποία όμως δεν αφορούν στην αποθήκευση $SHOIN^{(D)}$ πληροφορίας, αλλά στην κωδικοποιημένη αποθήκευση ερωτημάτων που υποβάλλονται στο DBRS. Είναι τα εξής:

- Ερώτημα

  Σύνολο οντοτήτων που αναπαριστά όλα εκείνα τα ερωτήματα που αποθηκεύονται κωδικοποιημένα στη βάση δεδομένων του DBRS. Διαιρούνται σε ερωτήματα δήλωσης και δυαδικά ερωτήματα. Κάθε ερώτημα έχει ένα μοναδικό id_query (Πρωτεύον Κλειδί).

- Παράμετρος Ερωτήματος

  Σύνολο οντοτήτων που αναπαριστά τις παραμέτρους ενός ερωτήματος που αποθηκεύονται κωδικοποιημένα στη βάση δεδομένων του DBRS. Αυτές διαιρούνται σε Κλάση, Άτομο, Ρόλο και Σταθερά (Literal).

- Ερώτημα Δήλωσης

  Σύνολο οντοτήτων που αναπαριστά όλα εκείνα τα ερωτήματα που διαθέτουν τρία ορίσματα και που αποθηκεύονται κωδικοποιημένα στη βάση δεδομένων του DBRS, προκειμένου να αποφευχθεί η χρονοβόρα επαναποτίμηση αυτών ή ισοδύναμών τους.

  Διαθέτει τρεις ιδιότητες που αποτελούν και το πρωτεύον κλειδί, τις id_arg1, id_arg2 και id_arg3.

- Δυαδικό Ερώτημα

  Σύνολο οντοτήτων που αναπαριστά όλα εκείνα τα ερωτήματα που διαθέτουν δύο ορίσματα και που αποθηκεύονται κωδικοποιημένα στη βάση δεδομένων του DBRS, προκειμένου να αποφευχθεί η χρονοβόρα επαναποτίμηση αυτών

ή ισοδύναμών τους. Διαθέτει τρεις ιδιότητες που αποτελούν και το πρωτεύον κλειδί, τις id_arg1, id_arg2 και id_query.

Σε αντίθεση με την περίπτωση των ατόμων, παρατηρούμε ότι στο Σχήμα 4.3 δεν ορίζεται σύνολο οντοτήτων ή σχέσεων που να περιγράφει την πληροφορία περί ισοδυναμίας κλάσεων και ρόλων. Αυτό συμβαίνει, γιατί έχει γίνει η παραδοχή ότι οι ισοδύναμες κλάσεις ή ρόλοι έχουν κοινές τιμές για τα ζεύγη (c_label_pre, c_label_post) και (r_label_pre, r_label_post) αντιστοίχως και, επομένως, αναγνωρίζονται μέσω αυτών. Προφανώς, στα άτομα της οντολογίας δεν ήταν δυνατόν να εφαρμοσθεί κάτι αντίστοιχο, αφού γι' αυτά δεν ορίζεται ιεραρχία.

Ακολουθεί η περιγραφή του συνόλου συσχετίσεων που φαίνονται στο Σχήμα 4.3. Για την γραφική αναπαράσταση των ίδιων, αλλά και της συμμετοχής των συνόλων οντοτήτων σε αυτές, ακολουθήσαμε τους συμβολισμούς που δίνονται στο [AKS97].

- έχει πεδίο

  Ένας ρόλος επιτρέπεται να έχει δηλωμένη από καμία έως μια κλάση στο πεδίο του (domain)[30]. Μια κλάση μπορεί να είναι πεδίο κανενός ή πολλών ρόλων.

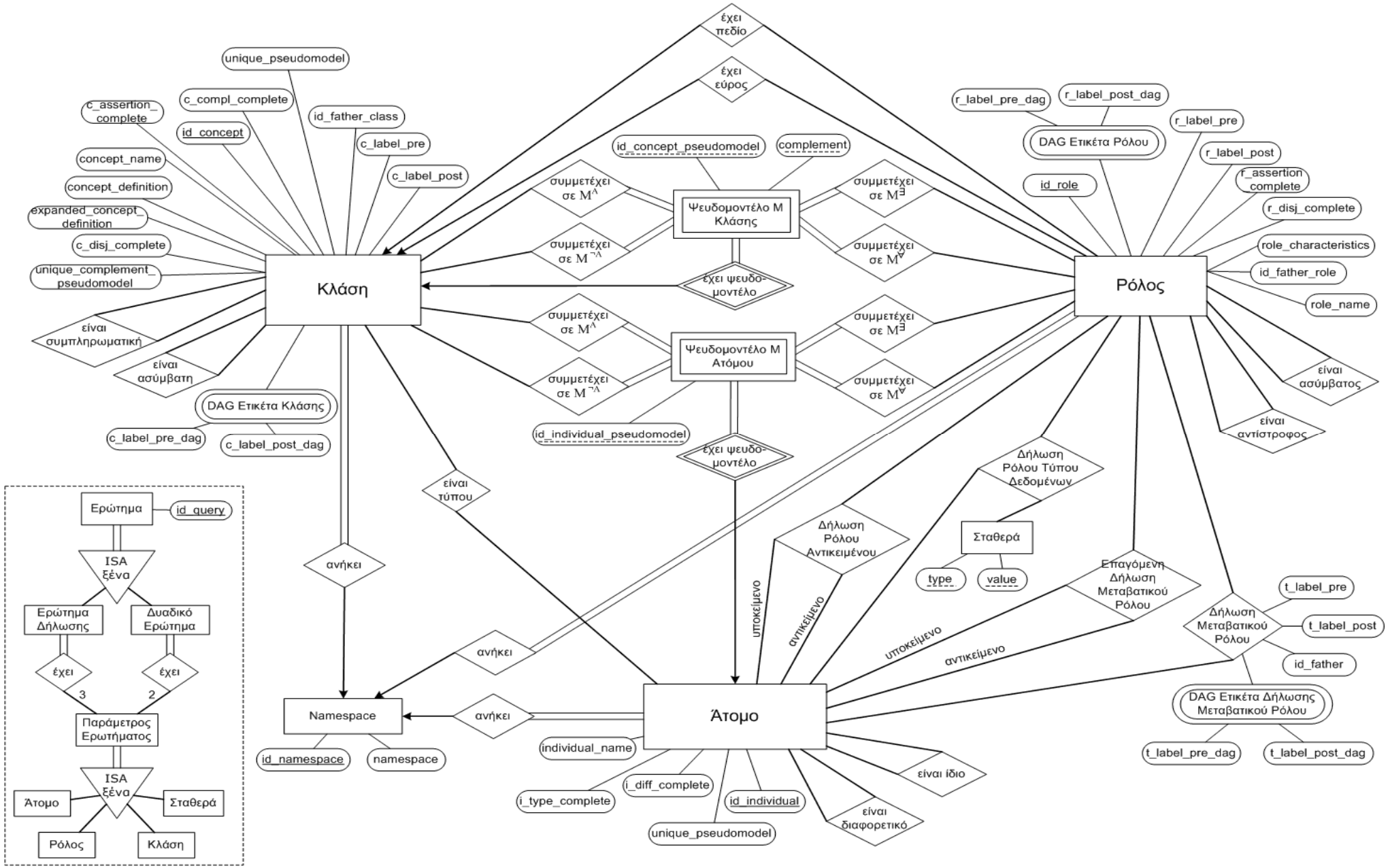
- έχει εύρος

  Ένας ρόλος επιτρέπεται να έχει δηλωμένη από καμία έως μια κλάση στο εύρος του (range). Μια κλάση μπορεί να είναι εύρος κανενός ή πολλών ρόλων.

- ανήκει

  Μια κλάση ή ένας ρόλος ή ένα άτομο ανήκει ακριβώς σε ένα namespace. Ένα namespace μπορεί να αντιστοιχεί σε μία ή περισσότερες κλάσεις (ή ρόλους ή άτομα) της οντολογίας.

[30] Στην πραγματικότητα ένας ρόλος μπορεί να έχει πολλές κλάσεις δηλωμένες ως πεδίο (ή εύρος). Στην περίπτωση αυτή, το συνολικό πεδίο (ή εύρος) του ρόλου προκύπτει από την τομή τους (intersection). Το DBRS επιτελεί μια τέτοιου είδους διαδικασία «ελαφριάς» συλλογιστικής ανάλυσης, έτσι ώστε να περιορίσει τη συμμετοχή κάθε ρόλου στη σχέση «πεδίο» (ή «εύρος») στο «το πολύ ένα».



Σχήμα 4.3: Μοντέλο Οντοτήτων Συσχετίσεων της βάσης δεδομένων του DBRS

- είναι τύπου

  Ένα άτομο μπορεί να έχει ως τύπο από καμία έως πολλές κλάσεις. Σε μια κλάση μπορεί να ανήκουν από κανένα έως πολλά άτομα.

- Δήλωση Ρόλου Αντικειμένου

  Ένας ρόλος αντικειμένου μπορεί να έχει από καμία έως πολλές δηλώσεις, καθεμία από τις οποίες διαθέτει ένα άτομο ως υποκείμενο και ένα άτομο ως αντικείμενο. Ένα άτομο μπορεί να μην ανήκει σε καμία ή να ανήκει σε μία ή πολλές δηλώσεις ρόλων αντικειμένου.

- Δήλωση Ρόλου τύπου δεδομένων

  Ένας ρόλος τύπου δεδομένων μπορεί να έχει από καμία έως πολλές δηλώσεις, καθεμία από τις οποίες διαθέτει ένα άτομο ως υποκείμενο και μια σταθερά ως αντικείμενο. Ένα άτομο ή μια σταθερά μπορεί να μην ανήκουν σε καμία ή να ανήκουν σε μία ή πολλές δηλώσεις ρόλων αντικειμένου.

- Δήλωση Μεταβατικού Ρόλου

  Ένας μεταβατικός ρόλος μπορεί να διαθέτει από καμία έως πολλές δηλώσεις. Ένα άτομο μπορεί να μην ανήκει σε καμία ή να ανήκει σε πολλές δηλώσεις μεταβατικών ρόλων. Κάθε δήλωση μεταβατικού ρόλου προσδιορίζεται από τις ιδιότητες t_label_pre, t_label_post, id_father και DAG ετικέτα Δήλωσης μεταβατικού ρόλου, οι οποίες σχετίζονται με το σχήμα ετικετών. Η τελευταία είναι πολλαπλών τιμών και διαιρείται στις t_label_pre_dag και t_label_post_dag.

- Επαγόμενη Δήλωση Μεταβατικού Ρόλου

  Ένας μεταβατικός ρόλος μπορεί να διαθέτει από καμία έως πολλές επαγόμενες δηλώσεις, δηλώσεις δηλαδή που προκύπτουν κατά την αποτίμηση ερωτημάτων και δεν έχουν τοποθετηθεί στο σχήμα ετικετών των δηλώσεων μεταβατικών ρόλων που κατασκεύασε το DBRS. Ένα άτομο μπορεί να μην ανήκει σε καμία ή να ανήκει σε μία ή πολλές επαγόμενες δηλώσεις μεταβατικών ρόλων. Κάθε επαγόμενη δήλωση μεταβατικού ρόλου διαθέτει ένα άτομο ως υποκείμενο και ένα άτομο ως αντικείμενο.

- είναι ασύμβατη

  Μια έννοια δεν είναι ασύμβατη με καμία άλλη ή είναι ασύμβατη με μία ή με πολλές άλλες έννοιες.

- είναι ασύμβατος

  Ένας ρόλος δεν είναι ασύμβατος με κανέναν άλλο ή είναι ασύμβατος με έναν ή με πολλούς άλλους ρόλους.

- είναι συμπληρωματική

  Μια έννοια δεν είναι συμπληρωματική καμίας άλλης ή είναι συμπληρωματική μίας ή πολλών άλλων εννοιών.

- είναι αντίστροφος

  Ένας ρόλος δεν είναι αντίστροφος με κανέναν άλλο ή είναι αντίστροφος με έναν ή με πολλούς άλλους ρόλους.

- είναι διαφορετικό

  Ένα άτομο δεν είναι διαφορετικό με κανένα άλλο ή είναι διαφορετικό με ένα ή με πολλά άλλα άτομα.

- είναι ίδιο

  Ένα άτομο δεν είναι ίδιο με κανένα άλλο ή είναι ίδιο με ένα ή με πολλά άλλα άτομα.

- έχει ψευδομοντέλο

  Προσδιορίζει τα αδύνατα σύνολα οντοτήτων «Ψευδομοντέλο Κλάσης» και «Ψευδομοντέλο Ατόμου». Μια κλάση ή ένα άτομο έχει από κανένα έως πολλά ψευδομοντέλα, ενώ ένα ψευδομοντέλο ανήκει ακριβώς σε μια κλάση ή ένα άτομο.

- συμμετέχει σε M^{A}

  Μια κλάση μπορεί να περιέχεται στην πλειάδα M^{A} ενός ή πολλών ψευδομοντέλων ατόμου ή κλάσης. Η πλειάδα M^{A} ενός ψευδομοντέλου μπορεί να περιέχει από καμία έως πολλές κλάσεις.

- συμμετέχει σε $M^{\neg A}$

  Μια κλάση μπορεί να περιέχεται στην πλειάδα $M^{\neg A}$ ενός ή πολλών ψευδομοντέλων ατόμου ή κλάσης. Η πλειάδα $M^{\neg A}$ ενός ψευδομοντέλου μπορεί να περιέχει από καμία έως πολλές κλάσεις.

- συμμετέχει σε $M^{\exists}$

  Ένας ρόλος μπορεί να περιέχεται στην πλειάδα $M^{\exists}$ ενός ή πολλών ψευδομοντέλων ατόμου ή κλάσης. Η πλειάδα $M^{\exists}$ ενός ψευδομοντέλου μπορεί να περιέχει από κανέναν έως πολλούς ρόλους.

- συμμετέχει σε $M^{\forall}$

  Ένας ρόλος μπορεί να περιέχεται στην πλειάδα $M^{\forall}$ ενός ή πολλών ψευδομοντέλων ατόμου ή κλάσης. Η πλειάδα $M^{\forall}$ ενός ψευδομοντέλου μπορεί να περιέχει από κανέναν έως πολλούς ρόλους.

5

Σχεδίαση Συστήματος

Στο κεφάλαιο αυτό περιγράφονται οι εφαρμογές κάθε υποσυστήματος του DBRS και, όπου απαιτείται, παρατίθενται διαγράμματα ροής για την καλύτερη κατανόηση των επιμέρους διαδικασιών που εμπλέκονται σε αυτές. Η δομή της παρουσίασης είναι παρόμοια με αυτή του Κεφαλαίου 4, μόνο που εδώ αναφερόμαστε στο επίπεδο των εφαρμογών. Στο τέλος του Κεφαλαίου περιγράφεται αναλυτικά το σχεσιακό σχήμα της βάσης δεδομένων που υλοποιήθηκε στο DBRS.

5.1 Υποσύστημα γραφικής διαπροσωπίας χρήστη

Στο υποσύστημα γραφικής διαπροσωπίας χρήστη ανήκουν όλες οι εφαρμογές του DBRS στις οποίες έχει πρόσβαση ο χρήστης διαμέσου της γραφικής διαπροσωπίας (GUI) του συστήματος.

5.1.1 Εφαρμογή επιλογής του OWL αρχείου της οντολογίας

Η εφαρμογή αυτή είναι υπεύθυνη για την επιλογή του OWL αρχείου της οντολογίας που επιθυμεί να φορτώσει ο χρήστης. Του δίνει τη δυνατότητα είτε να αναζητήσει το OWL αρχείο στο σύστημά του (browse), είτε να ορίσει το URL της τοποθεσίας του στον παγκόσμιο ιστό.

5.1.2 Εφαρμογή παραμετροποίησης διαδικασίας φόρτωσης

Η εφαρμογή δίνει στο χρήστη τη δυνατότητα να παραμετροποιήσει τη διαδικασία φόρτωσης της οντολογίας που έχει επιλέξει. Οι παρακάτω παράμετροι φόρτωσης μπορούν να επιλεχθούν σε οποιονδήποτε συνδυασμό:

- Υπολογισμός πλήρους πληροφορίας περί ασυμβατότητας κλάσεων (class disjointness)

  Δεδομένου ότι η διαδικασία εξαγωγής της πλήρους πληροφορίας περί ασυμβατότητας κλάσεων είναι αρκετά χρονοβόρα (ειδικά για μεγάλα TBoxes), ο χρήστης μπορεί να επιλέξει αν θέλει κάτι τέτοιο να γίνει κατά την αρχικοποίηση του συστήματος. Διαφορετικά φορτώνεται και αποθηκεύεται μόνο η ρητά δηλωμένη (explicit) πληροφορία, ενώ η επιπλέον (υπονοούμενη) πληροφορία προστίθεται αυξητικά (incrementally) κατά τη λειτουργία του συστήματος και τη διεξαγωγή ερωτημάτων.
- Υπολογισμός πλήρους πληροφορίας περί ασυμβατότητας ρόλων (role disjointness)

  Δεδομένου ότι η διαδικασία εξαγωγής της πλήρους πληροφορίας περί ασυμβατότητας ρόλων είναι αρκετά χρονοβόρα (ειδικά για μεγάλα RBoxes), ο χρήστης μπορεί να επιλέξει αν θέλει κάτι τέτοιο να γίνει κατά την αρχικοποίηση του συστήματος. Διαφορετικά φορτώνεται και αποθηκεύεται μόνο η ρητά δηλωμένη (explicit) πληροφορία, ενώ η επιπλέον (υπονοούμενη) πληροφορία προστίθεται αυξητικά (incrementally) κατά τη λειτουργία του συστήματος και τη διεξαγωγή ερωτημάτων.
- Υπολογισμός πλήρους πληροφορίας περί διαφορετικότητας ατόμων (individual difference)

  Δεδομένου ότι η διαδικασία εξαγωγής της πλήρους πληροφορίας περί διαφορετικών ατόμων είναι αρκετά χρονοβόρα (ειδικά για μεγάλα ABoxes), ο χρήστης μπορεί να επιλέξει αν θέλει κάτι τέτοιο να γίνει κατά την αρχικοποίηση του συστήματος. Διαφορετικά φορτώνεται και αποθηκεύεται μόνο η ρητά δηλωμένη (explicit) πληροφορία, ενώ η επιπλέον (υπονοούμενη) πληροφορία προστίθεται αυξητικά (incrementally) κατά τη λειτουργία του συστήματος και τη διεξαγωγή ερωτημάτων.

- Υπολογισμός πλήρους πληροφορίας περί συμπληρωματικών κλάσεων (complement classes)

  Δεδομένου ότι η διαδικασία εξαγωγής της πλήρους πληροφορίας περί συμπληρωματικών κλάσεων είναι αρκετά χρονοβόρα, ο χρήστης μπορεί να επιλέξει αν θέλει κάτι τέτοιο να γίνει κατά την αρχικοποίηση του συστήματος. Διαφορετικά, αποθηκεύεται μόνο η πληροφορία που μπορεί να εξαχθεί εύκολα από τους ορισμούς των κλάσεων[31], ενώ η επιπλέον υπονοούμενη πληροφορία προστίθεται αυξητικά (incrementally) κατά τη λειτουργία του συστήματος και την αποτίμηση ερωτημάτων.

- Υπολογισμός ψευδομοντέλων κλάσεων (class pseudomodels)

  Κατά τον έλεγχο της συνέπειας της οντολογίας από τον Reasoner, κατασκευάζεται ένα ψευδομοντέλο για κάθε έννοια. Ωστόσο, η διαδικασία αποθήκευσής τους στη βάση δεδομένων του DBRS δεν απαιτεί την κατασκευή τους (και κατά συνέπεια την επιπλέον επιβάρυνση του συστήματος), αλλά μόνο την επιλεκτική τους λήψη από το δέντρο του Tableau αλγορίθμου. Ο πλήρης υπολογισμός και η αποθήκευση όλων των ψευδομοντέλων για κάθε έννοια μπορεί να καθυστερήσει το σύστημα σε περίπτωση πολύ μεγάλων TBoxes. Έτσι, επιτρέπεται στο χρήστη να επιλέξει αν θα γίνεται κατά την αρχικοποίηση ή σταδιακά κατά την αποτίμηση ερωτημάτων.

- Υπολογισμός ψευδομοντέλων ατόμων (individual pseudomodels)

  Ομοίως με την περίπτωση των ψευδομοντέλων των εννοιών, το σύστημα επιτρέπει στο χρήστη να επιλέξει αν η αποθήκευση των ψευδομοντέλων των ατόμων θα γίνεται κατά την αρχικοποίηση ή σταδιακά κατά την αποτίμηση ερωτημάτων (on-the-fly). Σημειώνουμε εδώ, βάσει των όσων είπαμε στο Κεφάλαιο 2, ότι, κατά την αποτίμηση ερωτημάτων τύπου ατόμου (individual type), μπορεί να προκύψουν νέα ψευδομοντέλα για ένα άτομο της οντολογίας. Τα επιπλέον αυτά ψευδομοντέλα αποθηκεύονται από το DBRS στη βάση δεδομένων του είτε έχει επιλεγεί η φόρτωση ενός ψευδομοντέλου από την αρχή είτε όχι. Με άλλα λόγια, στην περίπτωση των ψευδομοντέλων

[31] Αναφερόμαστε σε περιπτώσεις αξιωμάτων ορισμού (ισοδυναμίας) της μορφής A ≡ Not(B), όπου εύκολα μέσω μιας διαδικασίας «ελαφριάς» συλλογιστικής ανάλυσης μπορούμε να αποφανθούμε ότι η κλάση A είναι συμπληρωματική της B.

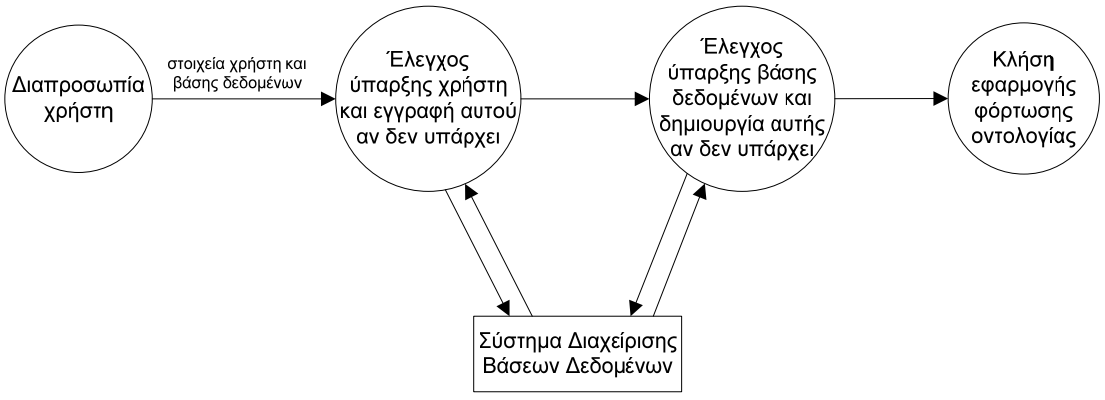
ατόμων, γίνεται πρόσθεση νέας πληροφορίας κατά τη λειτουργία του συστήματος, ακόμα και αν ο χρήστης έχει από την αρχή επιλέξει να φορτώσει ένα ψευδομοντέλο για κάθε άτομο.

- Κατασκευή του διευρυμένου TBox (expanded TBox)

  Η διαδικασία κατασκευής του διευρυμένου TBox έχει στην χειρότερη περίπτωση εκθετική πολυπλοκότητα και, επομένως, μπορεί να καθυστερήσει αρκετά τη διαδικασία φόρτωσης του συστήματος. Γι' αυτό τον λόγο, επιτρέπεται στο χρήστη να επιλέξει αν επιθυμεί να γίνει κατά την αρχικοποίηση η κατασκευή του διευρυμένου TBox.

5.1.3 Εφαρμογή εγγραφής νέου χρήστη και επιλογής βάσης δεδομένων

Η εφαρμογή αυτή δίνει τη δυνατότητα στο χρήστη να επιλέξει τη βάση δεδομένων στην οποία επιθυμεί να αποθηκεύσει την οντολογία (ή απλώς να επιλέξει τη βάση δεδομένων με την οποία επιθυμεί να συνδεθεί, αν αυτή περιέχει ήδη την οντολογία) και επίσης αναλαμβάνει και την εγγραφή (registration) του τελευταίου. Κάθε χρήστης μπορεί να αποθηκεύσει μια οντολογία σε κάποια βάση δεδομένων που του ανήκει και μπορεί να δημιουργήσει και να κατέχει περισσότερες από μια βάσεις δεδομένων. Αν ο χρήστης χρησιμοποιεί πρώτη φορά το σύστημα, τότε το όνομα (username) και ο κωδικός πρόσβασης (password) που θα δώσει για την κατασκευή της νέας βάσης θα είναι αυτά που θα τον ταυτοποιούν από εδώ και στο εξής ανάμεσα στους λοιπούς χρήστες του DBRS.

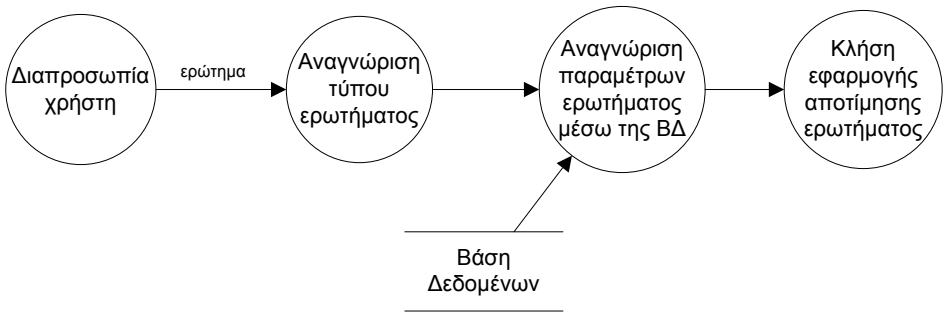


Σχήμα 5.1: Διάγραμμα ροής δεδομένων για την εφαρμογή εγγραφής νέου χρήστη και επιλογής βάσης δεδομένων για αποθήκευση οντολογίας

Έπειτα από την επιλογή της βάσης δεδομένων και την εισαγωγή των στοιχείων πρόσβασης από τον χρήστη, καλείται η «Εφαρμογή φόρτωσης οντολογίας» (Παράγραφος 5.2.1).

5.1.4 Εφαρμογή υποβολής ερωτημάτων

Η εφαρμογή αυτή δίνει στο χρήστη τη δυνατότητα να συντάξει και να υποβάλει ερωτήματα στο σύστημα και πιο συγκεκριμένα στη βάση γνώσης[32] με την οποία είναι συνδεδεμένος. Επειδή τα ερωτήματα έχουν αυστηρή σύνταξη, η οποία ακολουθεί την ψευδογλώσσα που περιγράφεται στην Παράγραφο 6.4.2, η εφαρμογή παρέχει βοήθεια στον χρήστη μέσω προβολής παραδειγμάτων. Επίσης του δίνεται η δυνατότητα να επιλέξει μεταξύ δύο μεθόδων αποτίμησης, «οκνηρής» και «πρόθυμης» (lazy and eager mode), η λειτουργία των οποίων περιγράφεται στην Παράγραφο 5.5.1



Σχήμα 5.2: Διάγραμμα ροής δεδομένων για την Εφαρμογή υποβολής ερωτημάτων

Έπειτα από την υποβολή του ερωτήματος, η εφαρμογή αναλαμβάνει την αναγνώριση του τύπου του ερωτήματος και την αναγνώριση των παραμέτρων αυτού. Η αναγνώριση του τύπου του ερωτήματος γίνεται βάσει του συντακτικού της ψευδογλώσσας. Η αναγνώριση των παραμέτρων, οι οποίες ταυτοποιούνται μεταξύ γνωστών κλάσεων (named concepts), ρόλων ή ατόμων, αυθαίρετων εννοιακών εκφράσεων και μεταβλητών, γίνεται μέσω της βάσης δεδομένων.

5.1.5 Εφαρμογή παρουσίασης αποτελέσματος

Η εφαρμογή αυτή δέχεται την απάντηση των ερωτημάτων που υποβάλει ο χρήστης από την εφαρμογή αποτίμησης ερωτημάτων (Παράγραφος 5.5.1) και την παρουσιάζει, μαζί με άλλες πληροφορίες (όπως π.χ. πλήθος αποτελεσμάτων) με γραφικό τρόπο ώστε να είναι εύκολα αναγνώσιμη. Επίσης προβάλει και μηνύματα

[32] Χρησιμοποιούμε τον όρο αυτό επειδή ο χρήστης δεν είναι συνδεδεμένος μόνο με τη βάση δεδομένων του DBRS, αλλά και με τη βάση γνώσης (στην κύρια μνήμη) του Reasoner.

σφάλματος του συστήματος, που οφείλονται κατά πάσα πιθανότητα είτε σε λάθος σύνταξη του ερωτήματος είτε σε εισαγωγή μη ικανοποιήσιμων αυθαίρετων κλάσεων.

5.1.6 Εφαρμογή παρουσίασης των βάσεων δεδομένων και των ιδιοκτητών τους

Η εφαρμογή προβάλει αρχικά στον χρήστη διαλόγους εισαγωγής των στοιχείων του, έπειτα καλεί την «Εφαρμογή επισκόπησης των βάσεων δεδομένων και των ιδιοκτητών τους» (Παράγραφος 5.4.1) και παρουσιάζει σε κατάλληλο παράθυρο τις πληροφορίες που λαμβάνει από αυτήν.

5.1.7 Εφαρμογή επιλογής βάσης δεδομένων για εκκαθάριση

Η εφαρμογή προβάλει αρχικά στον χρήστη διαλόγους για την εισαγωγή του ονόματος της βάσης δεδομένων που επιθυμεί να εκκαθαρίσει (η οποία πρέπει να του ανήκει βεβαίως) και των στοιχείων του και έπειτα καλεί την «Εφαρμογή εκκαθάρισης υπάρχουσας βάσης δεδομένων που ανήκει στον χρήστη» (Παράγραφος 5.4.2).

5.1.8 Εφαρμογή επιλογής μεταβατικού ρόλου για την ενημέρωση του σχήματος ετικετών του

Η εφαρμογή προβάλει αρχικά στον χρήστη διαλόγους για την εισαγωγή του URI του ρόλου του οποίου το σχήμα ετικετών επιθυμεί να ενημερώσει, το όνομα της βάσης δεδομένων όπου βρίσκεται η οντολογία αυτού και τα προσωπικά στοιχεία του και έπειτα καλεί την «Εφαρμογή ενημέρωσης του σχήματος ετικετών συγκεκριμένου μεταβατικού ρόλου σε βάση δεδομένων που ανήκει στο χρήστη» (Παράγραφος 5.4.3).

5.2 Υποσύστημα φόρτωσης οντολογίας

Στο υποσύστημα φόρτωσης οντολογίας ανήκουν οι εφαρμογές που αναλαμβάνουν την φόρτωση της οντολογίας στη βάση δεδομένων.

5.2.1 Εφαρμογή φόρτωσης οντολογίας

Η εφαρμογή αναλαμβάνει τη φόρτωση της βάσης δεδομένων βάσει των στοιχείων που της διοχετεύονται από την «Εφαρμογή εγγραφής νέου χρήστη και επιλογής βάσης δεδομένων» (Παράγραφος 5.1.3), είτε την σύνδεση με αυτήν αν η βάση δεδομένων περιέχει ήδη την επιλεγμένη οντολογία.

Ακολουθεί η περιγραφή της διαδικασίας που ακολουθείται κατά την εκτέλεση της εφαρμογής. Η ροή των δεδομένων κατά την εκτέλεσή της περιγράφεται στο Σχήμα 5.3:

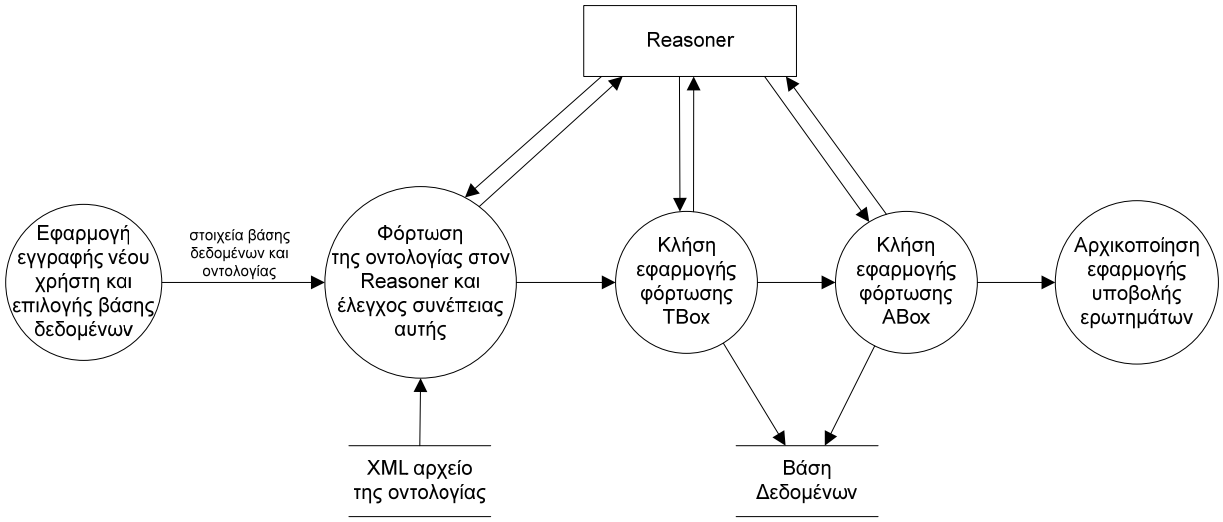
i) Αναλύεται το αρχικό αρχείο της οντολογίας και κατασκευάζεται το μοντέλο της (ontology model) στην κύρια μνήμη.

ii) Το μοντέλο της οντολογίας φορτώνεται στον Reasoner για να ελεγχθεί αν είναι συνεπές (consistent). Σε περίπτωση που αυτό ισχύει, η διαδικασία συνεχίζεται στο επόμενο βήμα. Διαφορετικά, η αρχικοποίηση σταματά και δίνεται επεξηγηματικό μήνυμα λάθους.

iii) Ελέγχεται αν υπάρχει ήδη η βάση δεδομένων την οποία επέλεξε ο χρήστης για να αποθηκεύσει την οντολογία. Αν δεν υπάρχει ο χρήστης ούτε η βάση, τότε καταχωρείται νέος χρήστης, δημιουργείται νέα βάση και η διαδικασία συνεχίζεται στο βήμα iv). Αν υπάρχει η βάση, ταυτοποιείται ο χρήστης και αν η βάση είναι άδεια η διαδικασία συνεχίζεται στο βήμα iv). Σε περίπτωση που η βάση δεν είναι άδεια, ελέγχεται μέσω URI[33] αν η οντολογία που είναι φορτωμένη σε αυτήν είναι η ίδια με αυτή που επιθυμεί να φορτώσει ο χρήστης. Αν δεν είναι, τότε επιστρέφεται επεξηγηματικό μήνυμα. Αν είναι τότε η διαδικασία φόρτωσης παρακάμπτεται και το σύστημα περνάει απ' ευθείας στο βήμα vi).

iv) Καλείται η εφαρμογή φόρτωσης του TBox, με παραμέτρους αυτές που έχουν οριστεί από το χρήστη. Η περιγραφή της γίνεται στην Παράγραφο 5.2.2.1

[33] Τονίζουμε ότι ο έλεγχος της ταυτότητας της οντολογίας που είναι φορτωμένη στη βάση δεδομένων γίνεται μέσω του URI της. Αν λοιπόν αυτό είναι μεν κοινό με το URI της οντολογίας που επιθυμεί να φορτώσει ο χρήστης, αλλά τα δεδομένα της υπάρχουσας οντολογίας που είναι αποθηκευμένη στη βάση δεδομένων έχουν υποστεί τροποποιήσεις, το σύστημα θα θεωρήσει ότι οι δυο οντολογίες είναι κοινές και θα προχωρήσει απλώς στο βήμα vi. Προφανώς, ένα τέτοιο σενάριο, για την δεδομένη έκδοση του DBRS, χαρακτηρίζεται ως παθολογικό, αφού το σύστημα δεν έχει αναπτυχθεί ώστε να υποστηρίζει ανανέωση ήδη φορτωμένων οντολογιών.



Σχήμα 5.3: Διάγραμμα ροής δεδομένων για την εφαρμογή φόρτωσης οντολογίας

v) Καλείται η εφαρμογή φόρτωσης του ABox, με παραμέτρους αυτές που έχουν οριστεί από το χρήστη. Η περιγραφή της γίνεται στην Παράγραφο 5.2.3.1

vi) Αρχικοποιείται η εφαρμογή αποτίμησης ερωτημάτων που περιγράφεται στην Παράγραφο 5.5.1

5.2.2 Υποσύστημα φόρτωσης TBox

Στο υποσύστημα φόρτωσης TBox ανήκουν οι εφαρμογές που αναλαμβάνουν την φόρτωση στη βάση δεδομένων του μέρους της πληροφορίας της οντολογίας που σχετίζεται με το TBox της.

5.2.2.1 Εφαρμογή φόρτωσης του TBox

Παρακάτω παρατίθενται τα βήματα της διαδικασίας φόρτωσης του TBox, της οποίας το διάγραμμα ροής δεδομένων φαίνεται στο Σχήμα 5.4.

i) Αποθηκεύεται το namespace της οντολογίας καθώς και όλων των οντολογιών που γίνονται imported στο αρχείο της. Επίσης, αποθηκεύονται και άλλα απαραίτητα namespaces, όπως το namespace των XML τύπων και το namespace των κλάσεων OWLThing και OWLNothing.

ii) Καλείται η «Εφαρμογή κατασκευής του σχήματος ετικετών για την ιεραρχία των κλάσεων της οντολογίας (βλέπε Παράγραφο 5.3.1) και κατασκευάζεται το

σχήμα ετικετών (labeling scheme) για την αναπαράσταση και αποθήκευση της ιεραρχίας των κλάσεων (classes' hierarchy) στη βάση δεδομένων.

iii) Αποθηκεύεται η πληροφορία περί ασυμβατότητας κλάσεων (class disjointness). Αναλόγως με το τι έχει επιλέξει ο χρήστης, αποθηκεύεται είτε η πλήρης πληροφορία για τις ασύμβατες κλάσεις, είτε η πληροφορία που δηλώνεται ρητά στο αρχείο της οντολογίας.

iv) Αποθηκεύεται η πληροφορία περί συμπληρωματικών κλάσεων (complement classes). Αναλόγως με το τι έχει επιλέξει ο χρήστης, αποθηκεύεται είτε η πλήρης πληροφορία, είτε η πληροφορία που μπορεί να εξαχθεί εύκολα από τους ορισμούς των κλάσεων μέσω μιας «ελαφριάς» συλλογιστικής ανάλυσης.

v) Καλείται η «Εφαρμογή κατασκευής του σχήματος ετικετών για την ιεραρχία των ρόλων της οντολογίας» (Παράγραφος 5.3.2) και κατασκευάζεται το σχήμα ετικετών για την αναπαράσταση και αποθήκευση της ιεραρχίας των ρόλων (roles' hierarchy) στη βάση δεδομένων.

vi) Αποθηκεύεται η πλήρης πληροφορία περί αντίστροφων ρόλων.

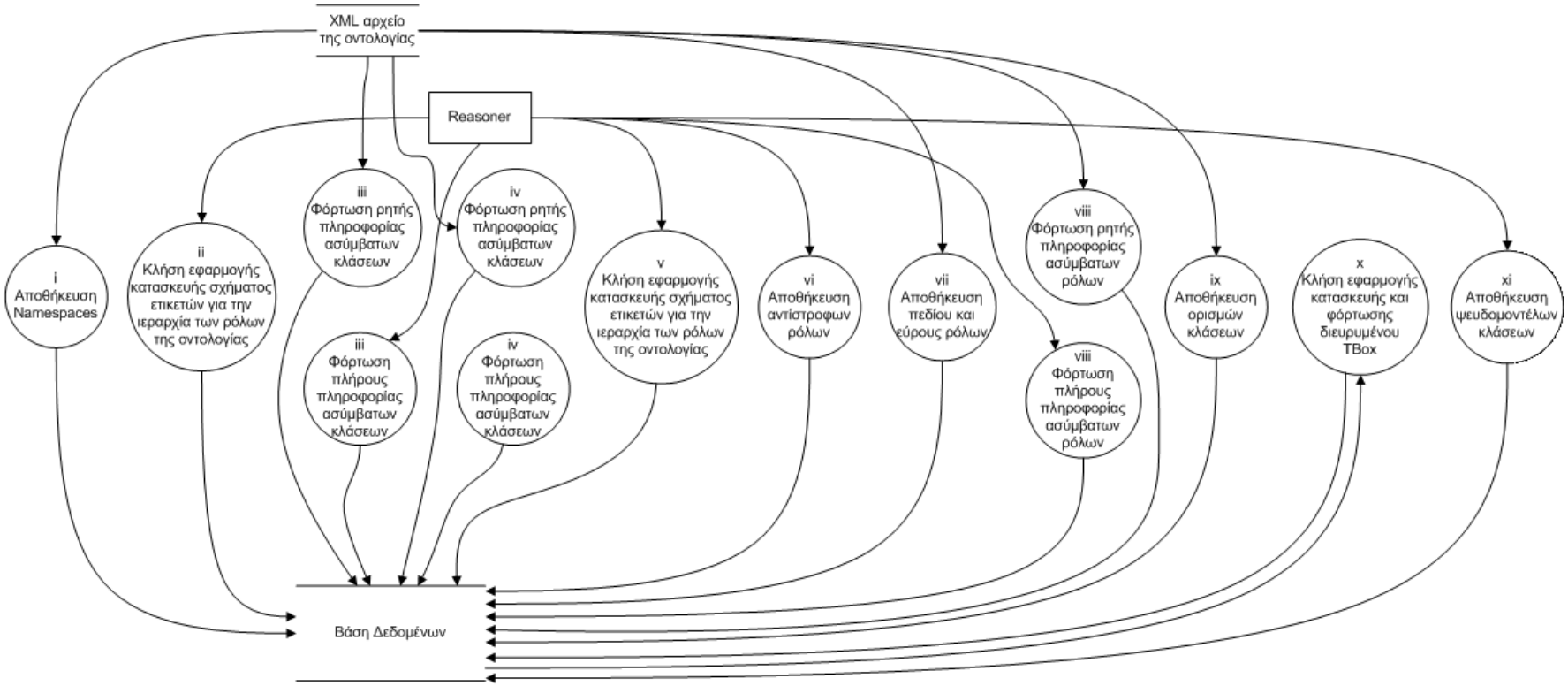
vii) Αποθηκεύεται η ρητά δηλωμένη πληροφορία σχετικά με το πεδίο (domain) και το εύρος (range) των ρόλων της οντολογίας.

viii) Αποθηκεύεται η πληροφορία περί ασυμβατότητας ρόλων (role disjointness). Αναλόγως με το τι έχει επιλέξει ο χρήστης, αποθηκεύεται είτε η πλήρης πληροφορία για τους ασύμβατους ρόλους, είτε η πληροφορία που δηλώνεται ρητά στο αρχείο της οντολογίας.

ix) Αποθηκεύονται οι ορισμοί των κλάσεων της οντολογίας (classes' definitions).

x) Αν έχει επιλεχθεί από το χρήστη, κατασκευάζεται και αποθηκεύεται το διευρυμένο TBox (expanded TBox) της οντολογίας.

xi) Αν έχει επιλεχθεί από το χρήστη, φορτώνεται ένα ψευδομοντέλο (class pseudomodel) για κάθε κλάση της οντολογίας.



Σχήμα 5.4: Διάγραμμα ροής δεδομένων για την εφαρμογή φόρτωσης του TBox

5.2.2.2 Εφαρμογή κατασκευής και φόρτωσης του διευρυμένου TBox

Η εφαρμογή αυτή αναλαμβάνει την κατασκευή του διευρυμένου TBox της οντολογίας και την αποθήκευσή του στη βάση δεδομένων. Λεπτομέρειες υλοποίησης του αλγορίθμου κατασκευής του διευρυμένου TBox δίνονται στην Παράγραφο 6.1.2

5.2.3 Υποσύστημα φόρτωσης ABox

Στο υποσύστημα φόρτωσης ABox ανήκουν οι εφαρμογές που αναλαμβάνουν την φόρτωση στη βάση δεδομένων του μέρους της πληροφορίας της οντολογίας που σχετίζεται με το ABox της.

5.2.3.1 Εφαρμογή φόρτωσης του ABox

Παρακάτω παρατίθενται τα βήματα της διαδικασίας φόρτωσης του ABox, της οποίας το διάγραμμα ροής δεδομένων φαίνεται στο Σχήμα 5.5.

i) Αποθηκεύονται στη βάση δεδομένων τα άτομα (individuals) της οντολογίας.

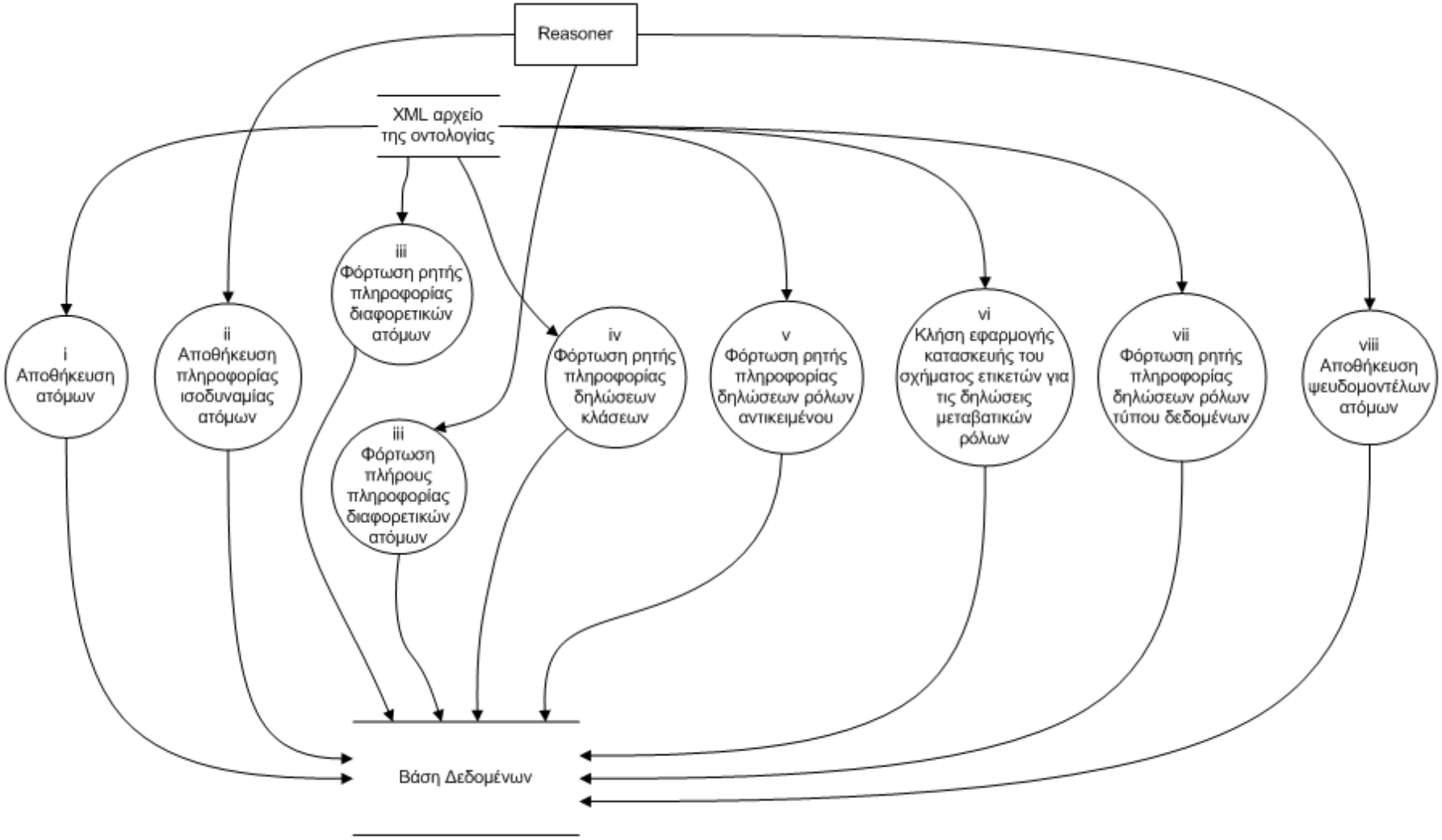
ii) Αποθηκεύεται η πλήρης πληροφορία περί ισοδυναμίας ατόμων (same individuals).

iii) Αποθηκεύεται η πληροφορία περί διαφορετικών ατόμων (different individuals). Αν έχει επιλεχθεί από το χρήστη, φορτώνεται η πλήρης πληροφορία, αλλιώς φορτώνεται μόνο η ρητά δηλωμένη στο αρχείο.

iv) Αποθηκεύεται η ρητά δηλωμένη πληροφορία σχετικά με τις δηλώσεις των κλάσεων της οντολογίας (classes assertions).

v) Αποθηκεύεται η ρητά δηλωμένη πληροφορία σχετικά με τις δηλώσεις των ρόλων αντικειμένου της οντολογίας (object role assertions).

vi) Καλείται επαναληπτικά η «Εφαρμογή κατασκευής του σχήματος ετικετών για τις δηλώσεις μεταβατικού ρόλου» (Παράγραφος 5.3.3). Η διαδικασία αυτή περιλαμβάνει την επιλογή ενός μεταβατικού ρόλου από κάθε σύνολο ισοδύναμων ή αντιστρόφων



Σχήμα 5.5: Διάγραμμα ροής δεδομένων για την εφαρμογή φόρτωσης του ABox

μεταβατικών ρόλων και την κατασκευή της ιεραρχίας μόνο αυτού. Για παράδειγμα, έστω οι μεταβατικοί ρόλοι R_1, R_2, R_3, R_4. Αν R_1 ισοδύναμος με R_2 και R_3 αντίστροφος του R_1, τότε σε αυτήν την περίπτωση θα κατασκευαστούν δυο ιεραρχίες. Μια για το σύνολο των δηλώσεων των ρόλων R_1, R_2, R_3 και μια για το ρόλο R_4.

vii) Αποθηκεύεται η ρητά δηλωμένη πληροφορία σχετικά με τις δηλώσεις των ρόλων τύπου δεδομένων της οντολογίας (datatype role assertions).

viii) Αν έχει επιλεχθεί από το χρήστη, αποθηκεύεται ένα ψευδομοντέλο (individual pseudomodel) για κάθε άτομο της οντολογίας.

5.3 Υποσύστημα κατασκευής Σχήματος Ετικετών

Στο υποσύστημα κατασκευής Σχήματος Ετικετών ανήκουν οι εφαρμογές που αναλαμβάνουν την κατασκευή και αποθήκευση των σχημάτων ετικετών για τις ιεραρχίες που απαντώνται στην οντολογία.

5.3.1 Εφαρμογή κατασκευής του Σχήματος Ετικετών για την ιεραρχία των κλάσεων της οντολογίας

Η εφαρμογή αυτή κατασκευάζει το σχήμα ετικετών για την ιεραρχία των κλάσεων της οντολογίας, εκτελώντας τον αλγόριθμο που προτάθηκε στο [ABJ89], με τον τρόπο που υλοποιήθηκε στο [CPST03].

5.3.2 Εφαρμογή κατασκευής του Σχήματος Ετικετών για την ιεραρχία των ρόλων της οντολογίας

Η εφαρμογή αυτή κατασκευάζει το σχήμα ετικετών για την ιεραρχία των ρόλων της οντολογίας, εκτελώντας τον αλγόριθμο που προτάθηκε στο [ABJ89], με τον τρόπο που υλοποιήθηκε στο [CPST03].

5.3.3 Εφαρμογή κατασκευής του Σχήματος Ετικετών για τις δηλώσεις μεταβατικού ρόλου

Εδώ, όπως έχει ήδη περιγραφεί, τα άτομα που συμμετέχουν σε δηλώσεις ενός μεταβατικού ρόλου σχηματίζουν έναν κατευθυνόμενο γράφο, που συμβατικά εδώ αποκαλούμε ιεραρχία ατόμων που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

Χρησιμοποιούμε τον όρο για να τονίσουμε το γεγονός ότι το DBRS χειρίζεται την περίπτωση αυτή κατ' αντιστοιχία με τις δύο προηγούμενες. Ομοίως λοιπόν με παραπάνω, η εφαρμογή αυτή κατασκευάζει το σχήμα ετικετών για την ιεραρχία των ατόμων της οντολογίας που συμμετέχουν σε δήλωση μεταβατικού ρόλου, εκτελώντας τον αλγόριθμο που προτάθηκε στο [ABJ89], με τον τρόπο που υλοποιήθηκε στο [CPST03]. Η εφαρμογή αναλαμβάνει την κατασκευή του σχήματος ετικετών για κάθε μεταβατικό ρόλο. Εξαίρεση αποτελεί η περίπτωση δύο ή περισσότερων ισοδύναμων ή αντίστροφων μεταβατικών ρόλων. Συγκεκριμένα για κάθε σύνολο ισοδύναμων και αντιστρόφων μεταβατικών ρόλων κατασκευάζεται ένα σχήμα ετικετών.

5.4 Υποσύστημα διαχείρισης οντολογίας

Στο υποσύστημα διαχείρισης οντολογίας ανήκουν οι εφαρμογές που αναλαμβάνουν εργασίες διαχείρισης των οντολογιών που βρίσκονται αποθηκευμένες στη βάση δεδομένων του DBRS.

5.4.1 Εφαρμογή επισκόπησης των βάσεων δεδομένων και των ιδιοκτητών τους

Η εφαρμογή αυτή δημιουργεί μια νέα σύνδεση με το DBMS και ανασύρει τα ονόματα των υπαρχόντων βάσεων δεδομένων και των ιδιοκτητών τους, εξαιρώντας τις βάσεις δεδομένων που ανήκουν στον διαχειριστή (superuser) του DBMS.

5.4.2 Εφαρμογή εκκαθάρισης υπάρχουσας βάσης δεδομένων που ανήκει στον χρήστη

Η εφαρμογή αυτή αναλαμβάνει την εκκαθάριση μιας βάσης δεδομένων από τα δεδομένα της[34].

[34] Χάρη στους περιορισμούς ακεραιότητας (cascade on update/delete) που έχουμε ορίσει στο σχεσιακό σχήμα της βάσης δεδομένων (βλέπε Παράγραφο 1.8.1), η εντολή που απαιτείται για την πλήρη εκκαθάριση της βάσης δεδομένων, έτσι ώστε να είναι έτοιμη για επαναχρησιμοποίηση, περιλαμβάνει μόνο τη διαγραφή των δεδομένων σχετικά με τα Namespaces.

5.4.3 Εφαρμογή ενημέρωσης του Σχήματος Ετικετών συγκεκριμένου μεταβατικού ρόλου σε βάση δεδομένων που ανήκει στο χρήστη

Η εφαρμογή αυτή αναλαμβάνει την ανακατασκευή του σχήματος ετικετών για τα άτομα των δηλώσεων ενός συγκεκριμένου μεταβατικού ρόλου. Η παρούσα εφαρμογή αποτελεί και τη μόνη που σχετίζεται με ενημέρωση σχήματος ετικετών, καθότι, όπως έχουμε ήδη τονίσει, η τρέχουσα έκδοση του DBRS δεν υποστηρίζει κάτι τέτοιο.

Η ανακατασκευή του σχήματος ετικετών προϋποθέτει μια προπαρασκευαστική διαδικασία η οποία πάντα πραγματοποιείται (χωρίς να το αντιλαμβάνεται ο χρήστης) κατά τη φόρτωση του ABox στη βάση δεδομένων. Η διαδικασία αυτή έχει ως αποτέλεσμα την κωδικοποιημένη αποθήκευση, σε αρχεία κειμένου (text files), των δηλώσεων ενός αριθμού επιλεγμένων μεταβατικών ρόλων, έτσι όπως αυτές ορίζονται στο OWL αρχείο. Οι συγκεκριμένοι ρόλοι είναι εκείνοι για τα άτομα των δηλώσεων των οποίων το DBRS επέλεξε, βάσει των όσων περιγράψαμε στην Παράγραφο 5.2.2.1, να κατασκευάσει σχήματα ετικετών. Για καθέναν από αυτούς δημιουργείται ένα ξεχωριστό αρχείο κειμένου.

Χάρη στην προπαρασκευαστική αυτή διαδικασία, η ανάλυση (parsing) του OWL αρχείου γίνεται μια μόνο φορά κατά την αρχική φόρτωση της οντολογίας και αυτό γιατί, με την κλήση της παρούσας εφαρμογής, το DBRS λαμβάνει πλέον όλη την απαραίτητη πληροφορία από το αντίστοιχο αρχείο κειμένου που έχει δημιουργηθεί από πριν. Έτσι, δεδομένου ότι το αρχείο αυτό είναι πολύ μικρότερο σε όγκο από το OWL αρχείο της οντολογίας, η διαδικασία ανακατασκευής επιταχύνεται σημαντικά.

Η εκτέλεση της εφαρμογής απαιτεί καταρχήν την ύπαρξη νέων δηλώσεων που έχουν προκύψει κατά την αποτίμηση ερωτημάτων είτε για τον συγκεκριμένο ρόλο που ορίζει ο χρήστης, είτε για κάποιον ισοδύναμό ή αντίστροφό του (οι νέες δηλώσεις αυτές αποθηκεύονται προσωρινά στη βάση δεδομένων. Βλέπε Παράγραφο 5.8.1). Αν κάτι τέτοιο δεν ισχύει, επιστρέφεται επεξηγηματικό μήνυμα και η διαδικασία σταματά. Σε αντίθετη περίπτωση, οι νέες δηλώσεις, στις οποίες άλλωστε οφείλεται και η ανάγκη ενημέρωσης του σχήματος ετικετών, προστίθενται στις δηλώσεις του αντίστοιχου αρχείου κειμένου, όπως αυτές έχουν αποθηκευτεί σε μορφή ζευγών (individual subject, individual object) κατά την φόρτωση του ABox. Στη συνέχεια, αποθηκεύονται σε προσωρινά στη βάση δεδομένων τα ζεύγη των

ατόμων και καλείται η εφαρμογή της Παραγράφου 5.3.3 για την ολοκλήρωση της διαδικασίας. Σημειώνουμε ότι οι νέες δηλώσεις, που βρίσκονται αποθηκευμένες προσωρινά όπως προαναφέρθηκε, διαγράφονται, αφού πρώτα έχουν προστεθεί στο αρχείο κειμένου με σκοπό να χρησιμοποιηθούν, κατά τον ίδιο τρόπο, σε επόμενη κλήση της εφαρμογής.

5.5 Υποσύστημα αποτίμησης ερωτημάτων

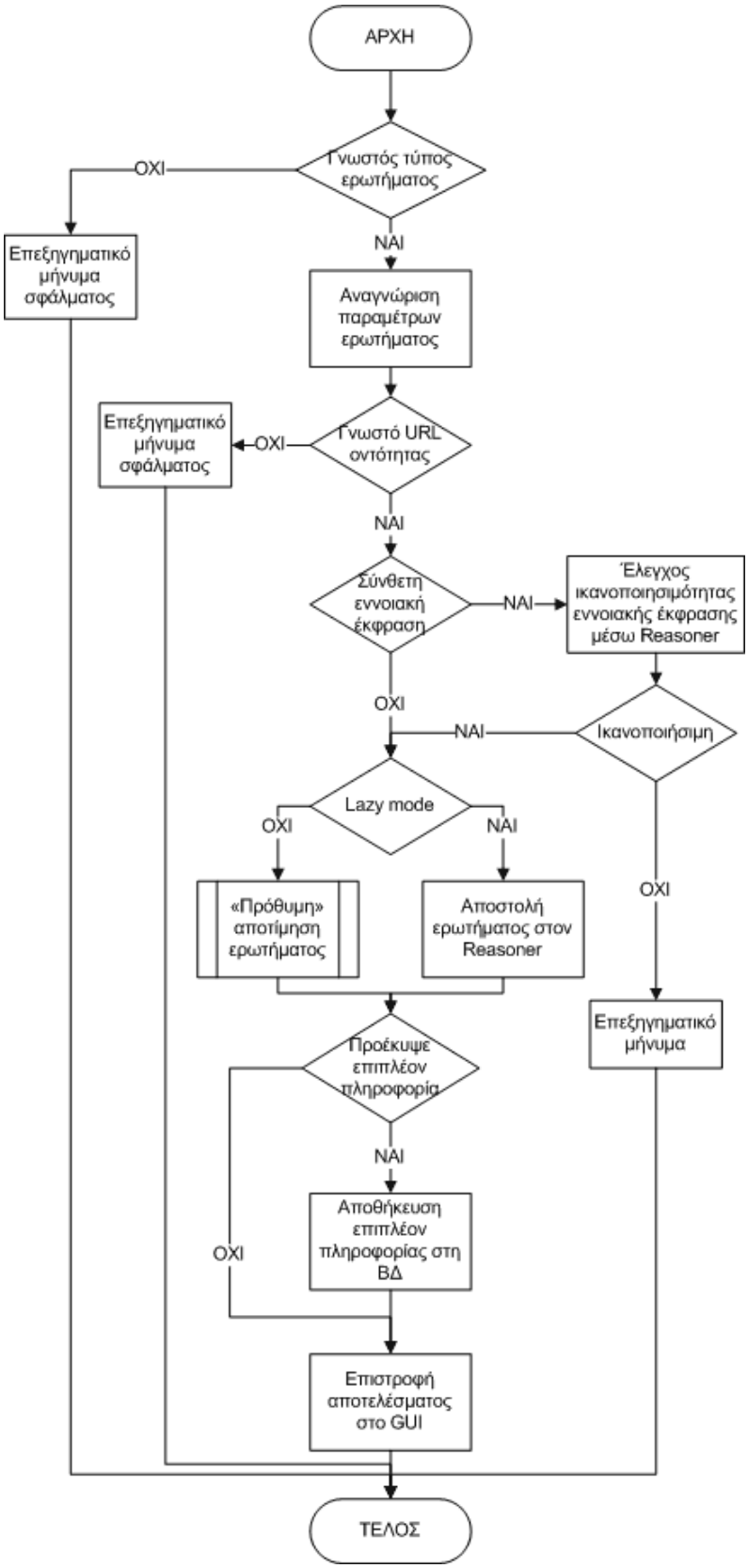
Στο υποσύστημα αποτίμησης ερωτημάτων ανήκουν οι εφαρμογές που αναλαμβάνουν τις εργασίες αποτίμησης των ερωτημάτων που υποβάλει ο χρήστης. Η διαδικασία αποτίμησης ενός οποιουδήποτε ερωτήματος, από την υποβολή του μέχρι την εμφάνισή των αποτελεσμάτων στο GUI, φαίνεται συνοπτικά στο Σχήμα 5.6.

5.5.1 Εφαρμογή αποτίμησης ερωτημάτων

Η εφαρμογή αυτή αναλαμβάνει την αποτίμηση των ερωτημάτων που υποβάλλει ο χρήστης μέσω της «Εφαρμογής υποβολής ερωτήματος» (Παράγραφος 5.1.3). Γενικά, κατά την αποτίμηση ενός ερωτήματος ακολουθούνται τα εξής τρία βήματα:

i) Αν υπάρχει αυθαίρετη κλάση μεταξύ των παραμέτρων, αυτή στέλνεται στον Reasoner για έλεγχο ικανοποιησιμότητας.

ii) Αποφασίζεται η τεχνική αποτίμησης του ερωτήματος.
   - Αν το σύστημα βρίσκεται σε «οκνηρή» λειτουργία, το ερώτημα απαντάται μόνο μέσω του Reasoner και η διαδικασία συνεχίζεται στο βήμα iii). Όπως έχει προαναφερθεί, στην περίπτωση αυτή το σύστημα λειτουργεί ως log.
   - Αν το σύστημα βρίσκεται σε «πρόθυμη» λειτουργία, ανάλογα με την πληρότητα της αποθηκευμένης στη βάση πληροφορίας, το ερώτημα απαντάται:
     - Μόνο από το DBMS
     - Μόνο από τον Reasoner
     - Από συνδυασμό DBMS και Reasoner



Σχήμα 5.6: Γενικό διαδικαστικό διάγραμμα ροής της πλήρους διαδικασίας υποβολής, αναγνώρισης και αποτίμησης ερωτήματος

iii) Αποθηκεύεται η επί πλέον πληροφορία, όπου αυτό απαιτείται. Ως επί πλέον πληροφορία εννοούμε δηλώσεις κλάσεων ή ρόλων που δεν έχουν αποθηκευτεί ρητά στη βάση δεδομένων και προέκυψαν από τη διαδικασία αποτίμησης του ερωτήματος.

Ανάλογα με τον τύπο του ερωτήματος και με την προϋπόθεση πάντα ότι το σύστημα βρίσκεται σε «πρόθυμη» λειτουργία, ακολουθείται μια συγκεκριμένη κάθε φορά στρατηγική αποτίμησης. Η στρατηγική αυτή εξαρτάται, πέρα από τον τύπο του ερωτήματος, από την πληρότητα της αποθηκευμένης πληροφορίας και από το αν το ερώτημα έχει ξανατεθεί. Πιο συγκεκριμένα, για ερωτήματα σχετικά με ασυμβατότητα κλάσεων, συμπληρωματικότητα κλάσεων, δηλώσεις κλάσεων, ασυμβατότητα ρόλων, δηλώσεις ρόλων, διαφορετικότητα ατόμων και τύπους ατόμων, ελέγχεται αν η πληροφορία που υπάρχει αποθηκευμένη για τις οντότητες (κλάση/ρόλο/άτομο) του ερωτήματος είναι πλήρης και σε αυτή την περίπτωση το ερώτημα απαντάται αποκλειστικά με χρήση της βάσης δεδομένων.

Η αποτίμηση μόνο με χρήση της βάσης δεδομένων γίνεται επίσης και στην περίπτωση που το ερώτημα έχει ξανατεθεί κάποια στιγμή στο παρελθόν και έχει αποθηκευθεί κωδικοποιημένο στη βάση δεδομένων. Στις υπόλοιπες περιπτώσεις και δεδομένου ότι αναφερόμαστε πάντα σε «πρόθυμη» λειτουργία, το ερώτημα απαντάται και με τη βοήθεια του Reasoner, αποθηκεύοντας, όπου αυτό απαιτείται, τη νέα πληροφορία που προκύπτει και ενημερώνοντας τα αποθηκευμένα στη βάση δεδομένων ερωτήματα και τα πεδία πληρότητας πληροφορίας.

Κατά τη διαδικασία αποτίμησης το DBRS χρησιμοποιεί διάφορες τεχνικές «ελαφριάς» συλλογιστικής ανάλυσης, εκμεταλλευόμενο κατά το μέγιστο δυνατό την, αποθηκευμένη στη βάση δεδομένων, πληροφορία. Ανάλυση των πιο σημαντικών από τις τεχνικές αυτές γίνεται στο Κεφάλαιο 6.

Εκτός από τις τεχνικές «ελαφριάς» συλλογιστικής ανάλυσης που προαναφέρθηκαν, το DBRS υλοποιεί και μια σημαντική τεχνική «βαριάς» συλλογιστικής ανάλυσης, την τεχνική των ψευδομοντέλων. Τα ψευδομοντέλα των κλάσεων (είτε γνωστών κλάσεων -named concepts- είτε αυθαίρετων) λαμβάνονται από τον Reasoner και αποθηκεύονται στη βάση δεδομένων για την εκτέλεση του «επίπεδου» Ελέγχου Συγχώνευσης (mergable test). Τα μεν ψευδομοντέλα των γνωστών κλάσεων αποθηκεύονται μόνιμα στη βάση δεδομένων, τα δε ψευδομοντέλα των αυθαίρετων κλάσεων αποθηκεύονται προσωρινά στη βάση δεδομένων, όσο

χρειάζεται για την εκτέλεση του mergable test. Φυσικά πριν την έναρξη οποιασδήποτε διαδικασίας αποτίμησης ερωτήματος που περιέχει στα ορίσματά του αυθαίρετη κλάση, αυτή στέλνεται στον Reasoner για έλεγχο ικανοποιησιμότητας.

Ακολουθεί η περιγραφή των επιμέρους εφαρμογών που απαρτίζουν την εφαρμογή αποτίμησης ερωτήματος. Στα παρακάτω, θεωρούμε πάντοτε την περίπτωση της «πρόθυμης» λειτουργίας, εκτός αν αναφέρεται ρητά το αντίθετο. Βάσει των όσων είπαμε προηγουμένως, στην «οκνηρή» λειτουργία απαντάει μόνο ο Reasoner και, επομένως, δεν υπάρχει κάτι σχετικά με αυτήν που μπορούμε να αναλύσουμε περισσότερο. Λεπτομέρειες υλοποίησης των αλγορίθμων αποτίμησης που χρησιμοποιούν οι παρακάτω εφαρμογές βρίσκονται στο Κεφάλαιο 6.

5.5.1.1 Εφαρμογή αποτίμησης ερωτήματος EquivalentClass

Η πληροφορία περί ισοδυναμίας των γνωστών κλάσεων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας. Συνεπώς, τα ερωτήματα αυτού του τύπου, στα ορίσματα των οποίων περιέχονται μόνο γνωστές κλάσεις, απαντώνται αποκλειστικά με χρήση της βάσης δεδομένων. Στην περίπτωση όμως που υπάρχουν αυθαίρετες εννοιακές εκφράσεις ως ορίσματα, επιστρατεύονται τεχνικές ψευδομοντέλων και, αν ο «επίπεδος» Έλεγχος Συγχώνευσης (mergable test) δεν απαντήσει, καλείται ο Reasoner. Εξαίρεση αποτελεί η περίπτωση που τα ορίσματα είναι μια αυθαίρετη εννοιακή έκφραση και μεταβλητή, οπότε το ερώτημα στέλνεται απ' ευθείας στον Reasoner.

5.5.1.2 Εφαρμογή αποτίμησης ερωτήματος ComplementOf

Στα πλαίσια της συγκεκριμένης εφαρμογής δεν είναι δεδομένο ότι η πληροφορία σχετικά με τη συμπληρωματικότητα κλάσεων που υπάρχει αποθηκευμένη στη βάση δεδομένων είναι πλήρης, καθώς αυτό εξαρτάται από τις επιλογές του χρήστη κατά την αρχική φόρτωση της οντολογίας. Για την αναπαράσταση αυτής της πληροφορίας, αλλά και για την βελτίωση της συμπεριφοράς του συστήματος κατά την αποτίμηση τέτοιου τύπου ερωτημάτων, χρησιμοποιούνται δύο βασικές τεχνικές. Πρώτον, αποθηκεύεται στη βάση δεδομένων αν η αποθηκευμένη πληροφορία σχετικά με τις ασύμβατες κάθε κλάσης της οντολογίας είναι πλήρης ή όχι και δεύτερον, αποθηκεύονται τα ερωτήματα ασυμβατότητας μεταξύ γνωστών κλάσεων που αποτιμώνται ως false από τον Reasoner. Λεπτομέρειες δίνονται στο Κεφάλαιο 6.

5.5.1.3 Εφαρμογή αποτίμησης ερωτήματος SubClassOf

Τα ερωτήματα αυτά, όταν τίθενται με ορίσματα γνωστές/-ή κλάσεις/-η της οντολογίας, αποτιμώνται αποκλειστικά με χρήση της βάσης δεδομένων, καθώς η ιεραρχία των κλάσεων κωδικοποιείται πλήρως, μέσω του σχήματος ετικετών, κατά την αρχική φόρτωση της οντολογίας. Όταν στα ορίσματα δεν περιέχεται μεταβλητή και περιέχονται μια ή περισσότερες αυθαίρετες εννοιακές εκφράσεις, τότε το ερώτημα επιχειρείται να αποτιμηθεί με χρήση του «Επίπεδου» Ελέγχου Συγχώνευσης (mergable test) και, αν αυτός δε μπορέσει να απαντήσει, καλείται ο Reasoner. Εξαίρεση αποτελεί η περίπτωση που τα ορίσματα είναι μια αυθαίρετη εννοιακή έκφραση και μεταβλητή, οπότε το ερώτημα στέλνεται απ' ευθείας στον Reasoner.

5.5.1.4 Εφαρμογή αποτίμησης ερωτήματος DisjointWith για κλάσεις

Η εφαρμογή αυτή είναι όμοια με την αντίστοιχη για τα ερωτήματα τύπου ComplementOf.

5.5.1.5 Εφαρμογή αποτίμησης ερωτήματος DisjointWith για ρόλους

Η εφαρμογή αυτή περιγράφεται είναι όμοια με την αντίστοιχη εφαρμογή για την περίπτωση των κλάσεων. Μόνη εξαίρεση αποτελούν οι αυθαίρετες εκφράσεις, καθώς δεν ορίζονται τέτοιες για ρόλους.

5.5.1.6 Εφαρμογή αποτίμησης ερωτήματος EquivalentProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Δεδομένου ότι η πληροφορία περί ισοδυναμίας ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.7 Εφαρμογή αποτίμησης ερωτήματος SubPropertyOf

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί ιεραρχίας ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.8 Εφαρμογή αποτίμησης ερωτήματος InverseOf

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί αντίστροφων ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.9 Εφαρμογή αποτίμησης ερωτήματος Symmetric

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί συμμετρικότητας ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.10 Εφαρμογή αποτίμησης ερωτήματος Transitive

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί μεταβατικότητας ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.11 Εφαρμογή αποτίμησης ερωτήματος FunctionalObjectProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί λειτουργικότητας ρόλων αντικειμένου αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.12 Εφαρμογή αποτίμησης ερωτήματος FunctionalDatatypeProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί λειτουργικότητας ρόλων τύπου δεδομένων αποθηκεύεται πλήρως

κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.13 Εφαρμογή αποτίμησης ερωτήματος InverseFunctionalProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί λειτουργικότητας και αντιστροφής ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.14 Εφαρμογή αποτίμησης ερωτήματος ObjectProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί χαρακτηριστικών ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.15 Εφαρμογή αποτίμησης ερωτήματος DatatypeProperty

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί χαρακτηριστικών ρόλων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.16 Εφαρμογή αποτίμησης ερωτήματος Type

Με την ολοκλήρωση της διαδικασίας φόρτωσης της οντολογίας και ανεξαρτήτως των επιλογών φόρτωσης του χρήστη, δεν μπορούμε ποτέ να είμαστε σίγουροι ότι η πληροφορία σχετικά με τις δηλώσεις κάποιας κλάσης ή τους τύπους κάποιου ατόμου είναι πλήρης. Είναι προφανές λοιπόν ότι και σε αυτή την περίπτωση χρειάζονται δυαδικά (boolean) πεδία κωδικοποίησης της πληρότητας της πληροφορίας. Μαζί με αυτά, είναι επίσης επιθυμητό να αποθηκεύουμε τα ερωτήματα που έχουν ως ορίσματα γνωστή κλάση και άτομο και που αποτιμώνται ως false από τον Reasoner, καθώς αυτή η διαδικασία αποτίμησης είναι συνήθως αρκετά απαιτητική. Επιπλέον,

και σε αυτόν τον τύπο ερωτήματος, όταν έχουμε ως όρισμα μεταβλητή, χρησιμοποιούνται τεχνικές ψευδομοντέλων.

5.5.1.17 Εφαρμογή αποτίμησης ερωτήματος SameAs

Η εφαρμογή αυτή περιλαμβάνει την κατασκευή και αποστολή του κατάλληλου ερωτήματος στη βάση δεδομένων του DBRS. Αυτό συμβαίνει γιατί, δεδομένου ότι η πληροφορία περί ταυτόσημων ατόμων αποθηκεύεται πλήρως κατά την αρχική φόρτωση της οντολογίας, η αποτίμηση αυτού του τύπου ερωτημάτων δε χρειάζεται την κλήση του Reasoner.

5.5.1.18 Εφαρμογή αποτίμησης ερωτήματος DifferentFrom

Η εφαρμογή αυτή είναι όμοια με την αντίστοιχη για τα ερωτήματα τύπου DisjointWith για ρόλους.

5.5.1.19 Εφαρμογή αποτίμησης ερωτήματος PropertyValue

Με την ολοκλήρωση της διαδικασίας φόρτωσης της οντολογίας και ανεξαρτήτως των επιλογών φόρτωσης του χρήστη, δεν μπορούμε ποτέ να είμαστε σίγουροι ότι η πληροφορία σχετικά με τις δηλώσεις ρόλων είναι πλήρης. Συνεπώς, και εδώ θα χρειαστούμε δυαδικά (boolean) πεδία κωδικοποίησης της πληρότητας της πληροφορίας. Επίσης, και σε αυτό τον τύπο ερωτημάτων είναι επιθυμητό να αποθηκεύσουμε κάποια ερωτήματα όταν τίθενται, ώστε να αποφύγουμε μελλοντικά την πιθανώς χρονοβόρα επαναποτίμησή τους. Κατά τη διαδικασία της επεξεργασίας του ερωτήματος και πριν την έναρξη της αποτίμησης, ιδιαίτερη προσοχή δίνεται στην αναγνώριση του ρόλου-ορίσματος, αν υπάρχει τέτοιος στην βάση, μεταξύ ρόλου αντικειμένου και ρόλου τύπου δεδομένων και στο χειρισμό πιθανών ασύμβατων ορισμάτων που δίνονται από τον χρήστη. Επίσης, ιδιαίτερη προσοχή δίνεται στο χειρισμό των μεταβατικών ρόλων αντικειμένου, καθώς οι δηλώσεις τους αποθηκεύονται με διαφορετικό τρόπο από τους υπόλοιπους με χρήση σχήματος ετικετών.

5.5.1.20 Εφαρμογή αποτίμησης ερωτήματος Domain

Το πεδίο (domain) C ενός ρόλου R μπορεί να εκφραστεί με το αξίωμα υπαγωγής $\exists_{\geq 1} R.\top \sqsubseteq C$ και, κατά συνέπεια, τα σχετικά ερωτήματα μπορούν να απαντηθούν από την εφαρμογή της Παραγράφου 5.5.1.3. Ωστόσο, στην παρούσα έκδοση του DBRS,

υλοποιήσαμε (για διευκόλυνση του χρήστη) μια ξεχωριστή εφαρμογή για τα ερωτήματα Domain την οποία και περιγράφουμε εδώ.

Η παρούσα εφαρμογή επιτρέπει τόσο την εύρεση των πεδίων ενός δοσμένου ρόλου, όσο και το αντίστροφο, δηλαδή την εύρεση ρόλων που έχουν ως πεδίο τη δοσμένη (γνωστή ή αυθαίρετη) κλάση. Επιπλέον, ελέγχει αν μια (δοσμένη) κλάση είναι πεδίο ενός (δοσμένου) ρόλου. Πιο συγκεκριμένα:

- Στην περίπτωση των ερωτημάτων πεδίου όπου δεν εμπλέκεται αυθαίρετη κλάση στα ορίσματα, το DBRS απαντά μόνο με χρήση της βάσης δεδομένων, αφού επιτελέσει την ανάλογη «ελαφριά» συλλογιστική ανάλυση.
- Σε κάθε άλλη περίπτωση, το ερώτημα στέλνεται στον Reasoner, χωρίς αποθήκευση επιπλέον πληροφορίας.

5.5.1.21 Εφαρμογή αποτίμησης ερωτήματος Range

Το εύρος (range) C ενός ρόλου R μπορεί να εκφραστεί με το αξίωμα υπαγωγής $T \sqsubseteq \forall R.C$ και, κατά συνέπεια, τα σχετικά ερωτήματα μπορούν να απαντηθούν από την εφαρμογή της Παραγράφου 5.5.1.3. Ωστόσο, στην παρούσα έκδοση του DBRS, υλοποιήσαμε (για διευκόλυνση του χρήστη) μια ξεχωριστή εφαρμογή για τα ερωτήματα Range την οποία και περιγράφουμε εδώ.

Η παρούσα εφαρμογή επιτρέπει τόσο την εύρεση των ευρών ενός δοσμένου ρόλου, όσο και το αντίστροφο, δηλαδή την εύρεση ρόλων που έχουν ως εύρος τη δοσμένη (γνωστή ή αυθαίρετη) κλάση. Επιπλέον, ελέγχει αν μια (δοσμένη) κλάση είναι εύρος ενός (δοσμένου) ρόλου. Πιο συγκεκριμένα:

- Στην περίπτωση των ερωτημάτων εύρους όπου δεν εμπλέκεται αυθαίρετη κλάση στα ορίσματα, το DBRS απαντά μόνο με χρήση της βάσης δεδομένων, αφού επιτελέσει την ανάλογη «ελαφριά» συλλογιστική ανάλυση.
- Σε κάθε άλλη περίπτωση (C αυθαίρετη κλάση), το ερώτημα στέλνεται στον Reasoner, χωρίς αποθήκευση επιπλέον πληροφορίας.

Οι εφαρμογές που αναλύσαμε μέχρι στιγμής αποτιμούν ερωτήματα που μπορούν να εκφραστούν με χρήση μιας τυπικής γλώσσας ερωτημάτων, όπως για παράδειγμα

της SPARQL. Ωστόσο, στην παρούσα έκδοση του DBRS, υλοποιήσαμε τρεις επιπλέον εφαρμογές που αποτιμούν ερωτήματα η σημασιολογία των οποίων δεν ορίζεται σε κάποια τυπική γλώσσα. Η αποτίμηση αυτών συνίσταται στην επιστροφή όλων εκείνων των αξιωμάτων στα οποία εμφανίζεται[35] μια οντότητα (κλάση/ρόλος/άτομο) και τα οποία αντιστοιχούν στη ρητά δηλωμένη γνώση[36] που είναι, μέχρι τη στιγμή που υποβάλλεται το ερώτημα, αποθηκευμένη στη βάση δεδομένων. Τα ερωτήματα αυτά τα ονομάσαμε «εποπτικά», επειδή ακριβώς δίνουν στο χρήστη μια «εικόνα» της οντότητας που τον ενδιαφέρει και έχουν αποδειχθεί ιδιαιτέρως χρήσιμα σε οντολογίες που μοντελοποιούν ιατρική γνώση.

5.5.1.22 Εφαρμογή αποτίμησης εποπτικού ερωτήματος ClassRelatedAxioms

Η εφαρμογή αυτή επιστρέφει, στη μορφή αξιωμάτων, τη ρητά δηλωμένη γνώση που είναι μέχρι εκείνη τη στιγμή αποθηκευμένη στη βάση δεδομένων του DBRS και που «σχετίζεται» με τη δοσμένη (γνωστή ή αυθαίρετη) κλάση. Εδώ, δε χρησιμοποιείται καθόλου ο Reasoner. Τα αξιώματα που επιστρέφονται είναι:

- Αξιώματα ισοδυναμίας που περιέχουν τη δοσμένη κλάση.
- Αξιώματα υπαγωγής που περιέχουν τη δοσμένη κλάση (εδώ συμπεριλαμβάνονται και αξιώματα που δηλώνουν Domain και Range κάποιου ρόλου).

---

[35] Λέμε εμφανίζεται, γιατί απλούστατα η δοσμένη από τον χρήστη οντότητα δεν είναι απαραίτητο να αποτελεί από μόνη της το ένα από τα δύο μέλη του αξιώματος, αλλά μπορεί για παράδειγμα να εμφανίζεται στο εσωτερικό μιας πολύπλοκης παράστασης ενός ή και των δύο μελών.

[36] Ως ρητά δηλωμένη γνώση που είναι αποθηκευμένη στη βάση δεδομένων, ορίζουμε αυτή που εξάγεται από τους αντίστοιχους για κάθε είδος αξιώματος πίνακες. Για παράδειγμα, η ρητά δηλωμένη γνώση για τις ασύμβατες κλάσεις μιας κλάσης της οντολογίας εξάγεται από τον πίνακα Concept Disjointness, χωρίς κάποια επιπλέον επεξεργασία. Βέβαια, κατά την αποτίμηση των συγκεκριμένων ερωτημάτων, το DBRS επιτελεί διαδικασίες «ελαφριάς» συλλογιστικής ανάλυσης, επιστρέφοντας ενδεχομένως στον χρήστη ένα σύνολο αξιωμάτων μεγαλύτερο από αυτό που αντιστοιχεί στην (όπως την ορίσαμε) ρητά δηλωμένη γνώση.

- Αξιώματα ασυμβατότητας που περιέχουν τη δοσμένη κλάση.
- Αξιώματα δηλώσεων που περιέχουν τη δοσμένη κλάση

5.5.1.23 Εφαρμογή αποτίμησης εποπτικού ερωτήματος PropertyRelatedAxioms

Η εφαρμογή αυτή επιστρέφει, στη μορφή αξιωμάτων, τη ρητά δηλωμένη γνώση που είναι μέχρι εκείνη τη στιγμή αποθηκευμένη στη βάση δεδομένων του DBRS και που «σχετίζεται» με το δοσμένο ρόλο. Εδώ, δε χρησιμοποιείται καθόλου ο Reasoner. Τα αξιώματα που επιστρέφονται είναι:

- Αξιώματα ισοδυναμίας που περιέχουν το δοσμένο ρόλο.
- Αξιώματα υπαγωγής που περιέχουν το δοσμένο ρόλο (εδώ συμπεριλαμβάνονται και αξιώματα που δηλώνουν Domain και Range του ρόλου αυτού).
- Αξιώματα ασυμβατότητας που περιέχουν το δοσμένο ρόλο.
- Αξιώματα δηλώσεων που περιέχουν το δοσμένο ρόλο.

5.5.1.24 Εφαρμογή αποτίμησης εποπτικού ερωτήματος IndividualRelatedAxioms

Η εφαρμογή αυτή επιστρέφει, στη μορφή αξιωμάτων, τη ρητά δηλωμένη γνώση που είναι μέχρι εκείνη τη στιγμή αποθηκευμένη στη βάση δεδομένων του DBRS και που «σχετίζεται» με το δοσμένο άτομο. Εδώ, δε χρησιμοποιείται καθόλου ο Reasoner. Τα αξιώματα που επιστρέφονται είναι:

- Αξιώματα ταυτότητας που περιέχουν το δοσμένο άτομο.
- Αξιώματα διαφορετικότητας που περιέχουν το δοσμένο άτομο.
- Αξιώματα δηλώσεων κλάσεων που περιέχουν το δοσμένο άτομο.
- Αξιώματα δηλώσεων ρόλων που περιέχουν το δοσμένο άτομο.

5.6 Υποσύστημα διαχείρισης Βάσης Δεδομένων

Το υποσύστημα διαχείρισης Βάσης Δεδομένων του DBRS περιλαμβάνει το Σύστημα Διαχείρισης Βάσεων Δεδομένων (DBMS) και την Εφαρμογή Διασύνδεσης (DBMS Wrapper) των λοιπών υποσυστημάτων με αυτό.

5.6.1 Εφαρμογή διασύνδεσης του DBRS με το DBMS

Η εφαρμογή διασύνδεσης χρησιμοποιεί το ευρύτατα διαδεδομένο JDBC API [JDBC]. Για λόγους ευελιξίας και επεκτασιμότητας, οι (υπο)εφαρμογές που αναφέρονται παρακάτω δεν εκμεταλλεύονται κανένα ειδικό χαρακτηριστικό του DBMS και μπορούν να λειτουργήσουν με οποιοδήποτε (αντικειμενο)σχεσιακό Σύστημα Διαχείρισης Βάσεων Δεδομένων (object relational DBMS).

5.6.1.1 Εφαρμογή δημιουργίας σύνδεσης με τη βάση δεδομένων

Η εφαρμογή αυτή αναλαμβάνει τη δημιουργία νέας σύνδεσης με κάποια υπάρχουσα βάση δεδομένων, χρησιμοποιώντας το όνομα της βάσης, το όνομα του χρήστη (username) και τον κωδικό πρόσβασης (password).

5.6.1.2 Εφαρμογή τερματισμού σύνδεσης με τη βάση δεδομένων

Η εφαρμογή αυτή αναλαμβάνει τον τερματισμό (termination) της τρέχουσας σύνδεσης με τη βάση δεδομένων.

5.6.1.3 Εφαρμογή ελέγχου ύπαρξης βάσης δεδομένων

Η εφαρμογή αυτή ελέγχει την ύπαρξη βάσης δεδομένων, δοσμένου του ονόματός της. Καλείται πριν από την «Εφαρμογή δημιουργίας βάσης δεδομένων» (Παράγραφος 5.6.2.7) για την αποφυγή σφαλμάτων. Δύο βάσεις δεδομένων που ανήκουν στο ίδιο σύστημα δεν μπορούν να έχουν κοινό όνομα.

5.6.1.4 Εφαρμογή ελέγχου ύπαρξης δεδομένων στη βάση δεδομένων

Η εφαρμογή αυτή ελέγχει αν η βάση δεδομένων με την οποία έχει γίνει η σύνδεση περιέχει δεδομένα.

5.6.1.5 Εφαρμογή ελέγχου ύπαρξης χρήστη του DBMS

Η εφαρμογή αυτή ελέγχει αν στο DBMS υπάρχει εγγεγραμμένος χρήστης (role) με το δοσμένο όνομα.

5.6.1.6 Εφαρμογή δημιουργίας χρήστη

Η εφαρμογή αυτή αναλαμβάνει την δημιουργία νέου χρήστη (role) στο DBMS. Στην ουσία, αποτελεί μια αίτηση στον διαχειριστή του DBMS (super user) ο οποίος

προβαίνει στη δημιουργία ενός χρήστη με το δοσμένο όνομα και με περιορισμένα δικαιώματα.

5.6.1.7 Εφαρμογή δημιουργίας βάσης δεδομένων

Η εφαρμογή αυτή αναλαμβάνει την δημιουργία νέας βάσης δεδομένων. Όσον αφορά σε αυτό, τονίζουμε ότι οι χρήστες του DBRS δεν έχουν τη δυνατότητα δημιουργίας νέας βάσης απευθείας στο όνομά τους. Στην ουσία, χωρίς να το αντιλαμβάνονται, γίνεται μια αίτηση στον διαχειριστή του DBMS (super user) ο οποίος με τη σειρά του δημιουργεί τη βάση δεδομένων και δίνει τα αντίστοιχα δικαιώματα σε αυτούς.

5.6.1.8 Εφαρμογή ανάλυσης του αρχείου κατασκευής του σχήματος της βάσης δεδομένων

Η εφαρμογή αυτή αναλαμβάνει την ανάγνωση, ανάλυση και εκτέλεση του script αρχείου κατασκευής του σχήματος της βάσης δεδομένων του DBRS. Μεταξύ άλλων, η ανάλυση του αρχείου περιλαμβάνει την αντιστοίχηση (granting) των δικαιωμάτων της δημιουργούμενης βάσης στον χρήστη, το όνομα του οποίου (username) δόθηκε κατά την έναρξη της διαδικασίας φόρτωσης της οντολογίας.

5.6.1.9 Εφαρμογή ανάλυσης του αρχείου κατασκευής των ευρετηρίων της βάσης δεδομένων

Η εφαρμογή αυτή αναλαμβάνει την ανάγνωση, ανάλυση και εκτέλεση του script αρχείου κατασκευής των ευρετηρίων της βάσης δεδομένων του DBRS. Το αρχείο των ευρετηρίων διαχωρίστηκε από το αντίστοιχο των σχεσιακών πινάκων της βάσης για διευκόλυνση των χρηστών. Πιο συγκεκριμένα, το πρώτο μπορεί να εκτελεστεί αυτούσιο στην πλειονότητα των υπαρχόντων DBMSs, καθότι ακολουθεί το πρότυπο SQL 99, ενώ το δεύτερο ενδεχομένως να παρουσιάσει προβλήματα, δεδομένου ότι κάποια ήδη ευρετηρίων (π.χ. hash) δεν υποστηρίζονται από όλα τα DBMSs. Έτσι, σε περίπτωση που οι χρήστες του DBRS επιλέξουν ένα τέτοιο DBMS, θα πρέπει να αφαιρέσουν (από το script αρχείο) όλα τα ευρετήρια που δεν υποστηρίζονται ή να ορίσουν άλλα στη θέση τους. Αντιλαμβανόμαστε, λοιπόν, ότι με τον διαχωρισμό που επιχειρήσαμε απαιτείται μόνο η τροποποίηση του αρχείου ευρετηρίων.

5.6.1.10 Εφαρμογές δημιουργίας προσωρινών πινάκων της βάσης δεδομένων

Σε αυτή την κατηγορία ανήκουν οι εφαρμογές που κατασκευάζουν τους προσωρινούς πίνακες της βάσης δεδομένων οι οποίοι απαιτούνται κατά τη διαδικασία φόρτωσης της οντολογίας (π.χ. "Concepts Temp Table").

5.6.1.11 Εφαρμογές διαγραφής προσωρινών πινάκων της βάσης δεδομένων

Σε αυτή την κατηγορία ανήκουν οι εφαρμογές που διαγράφουν τους προσωρινούς πίνακες που κατασκευάζουν οι εφαρμογές της Παραγράφου 5.6.2.10.

5.6.1.12 Εφαρμογές εισαγωγής δεδομένων στους πίνακες της βάσης δεδομένων

Οι εφαρμογές αυτής της κατηγορίας διαιρούνται σε:

- Εφαρμογές εισαγωγής δεδομένων στους προσωρινούς πίνακες (temporary tables) της βάσης δεδομένων
- Εφαρμογές εισαγωγής δεδομένων στους μόνιμους πίνακες της βάσης δεδομένων
- Εφαρμογές εισαγωγής δεδομένων στους πίνακες αποθήκευσης ερωτημάτων (query cachers) της βάσης δεδομένων

5.6.1.13 Εφαρμογές χειρισμού δεδομένων στους πίνακες της βάσης δεδομένων

Αυτές οι εφαρμογές αναλαμβάνουν την ανάσυρση (select), ενημέρωση (update) και διαγραφή (delete) πληροφορίας. Διαιρούνται σε:

- Εφαρμογές χειρισμού δεδομένων στους προσωρινούς πίνακες (temporary tables) της βάσης δεδομένων
- Εφαρμογές χειρισμού δεδομένων στους μόνιμους πίνακες της βάσης δεδομένων
- Εφαρμογές χειρισμού δεδομένων στους πίνακες αποθήκευσης ερωτημάτων (query cachers) της βάσης δεδομένων

5.6.1.14 Εφαρμογές ελαφριάς συλλογιστικής ανάλυσης

Βάσει των όσων έχουμε πει στο Κεφάλαιο 3, οι εφαρμογές αυτές εκμεταλλεύονται τη σημασιολογία που υποδηλώνεται είτε μέσω του σχήματος της βάσης δεδομένων είτε μέσω επιμέρους κωδικοποιήσεων σε συγκεκριμένα πεδία πινάκων (π.χ. χαρακτηριστικά ρόλων), με σκοπό την εύρεση υπονοούμενης πληροφορίας χωρίς την κλήση του Reasoner. Τονίζουμε ότι τέτοιες διαδικασίες «ελαφριάς» συλλογιστικής ανάλυσης λαμβάνουν χώρα κατά την εκτέλεση διαφόρων εφαρμογών του DBRS (στο εσωτερικό των αντίστοιχων μεθόδων) και θα ήταν υπερβολικό να αναλύσουμε εδώ όλες τις περιπτώσεις. Οι σημαντικότερες από αυτές περιγράφονται αναλυτικά στο Κεφάλαιο 6.

5.6.1.15 Εφαρμογές που χρησιμοποιούνται κατά την αποτίμηση ερωτημάτων

Οι εφαρμογές αυτής της κατηγορίας χρησιμοποιούνται (όταν απαιτείται ανάσυρση της αντίστοιχης πληροφορίας από τη βάση δεδομένων) από την «Εφαρμογή αποτίμησης ερωτημάτων» που περιγράφηκε στην Παράγραφο 5.5.1 Διαιρούνται σε:

- Εφαρμογές που χρησιμοποιούνται στην αποτίμηση TBox ερωτημάτων

  Αυτές σχετίζονται κυρίως με την εύρεση σχέσεων υπαγωγής και ισοδυναμίας, εκμεταλλευόμενες το σχήμα ετικετών, καθώς και με την εύρεση σχέσεων ασυμβατότητας και συμπληρωματικότητας. Στις παραπάνω, προστίθενται και οι εφαρμογές αναγνώρισης των χαρακτηριστικών των ρόλων.
- Εφαρμογές που χρησιμοποιούνται στην αποτίμηση ABox ερωτημάτων

  Αυτές σχετίζονται με την εύρεση δηλώσεων κλάσεων και ρόλων, καθώς και με την εύρεση σχέσεων ταυτότητας και διαφορετικότητας μεταξύ ατόμων.
- Εφαρμογές που χρησιμοποιούνται στην αποτίμηση εποπτικών ερωτημάτων

  Πρόκειται για τρεις εφαρμογές οι οποίες είναι:
  - Εφαρμογή που επιστρέφει αξιώματα που περιέχουν δοσμένη κλάση, είτε αυτή είναι αυθαίρετη είτε είναι γνωστή.
  - Εφαρμογή που επιστρέφει αξιώματα που περιέχουν δοσμένο ρόλο.
  - Εφαρμογή που επιστρέφει αξιώματα που περιέχουν δοσμένο άτομο.

- Εφαρμογές που εκτελούν τον «επίπεδο» Έλεγχο Συγχώνευσης ψευδομοντέλων

  Πρόκειται για δύο εφαρμογές που επιτελούν τις διαδικασίες που περιγράψαμε στην Παράγραφο 2.2.2.1. Αυτές είναι:

  - Εφαρμογή που εκτελεί τον έλεγχο συγχώνευσης μεταξύ των ψευδομοντέλων δύο κλάσεων.

    Η εφαρμογή αυτή εκτελεί ουσιαστικά ένα SQL ερώτημα συνένωσης (join) μεταξύ των δύο σχεσιακών πινάκων της βάσης δεδομένων όπου είναι αποθηκευμένα τα αντίστοιχα ψευδομοντέλα των κλάσεων[37].

  - Εφαρμογή που εκτελεί τον έλεγχο συγχώνευσης μεταξύ ψευδομοντέλων ατόμου και έννοιας

    Η εφαρμογή αυτή εκτελεί ουσιαστικά ένα SQL ερώτημα συνένωσης (join) μεταξύ των δύο σχεσιακών πινάκων της βάσης δεδομένων όπου είναι αποθηκευμένο το ζεύγος των ψευδομοντέλων του ατόμου[38] και της κλασης[7]. Επίσης, ελέγχει αν υπάρχουν περισσότερα του ενός ψευδομοντέλα για κάποιο άτομο της οντολογίας, προκειμένου να εκτελέσει την ίδια διαδικασία για καθένα από αυτά.

Οι αλγόριθμοι που εκτελούνται στα πλαίσια των σημαντικότερων από τις παραπάνω εφαρμογές περιγράφονται αναλυτικά στο Κεφάλαιο 6.

---

[37] Ή κάποιων ισοδύναμών τους. Ως γνωστό, οι ισοδύναμες κλάσεις διαθέτουν το ίδιο σύνολο ψευδομοντέλων και μάλιστα, για μια συμπλήρωση, έχουν πάντα το ίδιο ψευδομοντέλο. Το DBRS, για οικονομία χώρου, αποθηκεύει ένα ψευδομοντέλο για κάθε σύνολο ισοδύναμων κλάσεων.

[38] Ή κάποιων ταυτόσημών τους. Ως γνωστό, τα ταυτόσημα άτομα διαθέτουν το ίδιο σύνολο ψευδομοντέλων και μάλιστα, για μια συμπλήρωση, έχουν πάντα το ίδιο ψευδομοντέλο. Το DBRS, για οικονομία χώρου, αποθηκεύει ένα ψευδομοντέλο για κάθε σύνολο ταυτόσημων ατόμων.

5.7 Υποσύστημα διαχείρισης Μηχανής Συλλογιστικής Ανάλυσης

Το υποσύστημα διαχείρισης Συστήματος Συλλογιστικής Ανάλυσης του DBRS περιλαμβάνει τον Reasoner και την Εφαρμογή Διασύνδεσης (Reasoner Wrapper) των λοιπών υποσυστημάτων με αυτόν.

5.7.1 Εφαρμογή διασύνδεσης του DBRS με τον Reasoner

Η εφαρμογή διασύνδεσης χρησιμοποιεί (implements) το ευρύτατα διαδεδομένο OWL API [OWLAPI], ενώ οι (υπο)εφαρμογές που αναφέρονται παρακάτω έχουν αναπτυχθεί για να συνεργάζονται με τον Reasoner.

Τονίζουμε ότι, στην παρούσα έκδοση του DBRS, η εφαρμογή διασύνδεσης με τον Reasoner αποτελεί ουσιαστικά μια επέκταση (με τις εφαρμογές που παραθέτουμε στη συνέχεια) του ήδη υλοποιημένου wrapper που μπορεί να βρεθεί στο [Pellet].

5.7.1.1 Εφαρμογή μετατροπής οντοτήτων από OWL API τύπο σε ATerm τύπο

Η εφαρμογή αυτή αναλαμβάνει τη μετατροπή των οντοτήτων της οντολογίας από τύπους του OWL API στους οποίους είναι εκφρασμένες, σε τύπους της ATerm Library [ATerm] τους οποίους αναγνωρίζει εσωτερικά ο Reasoner. Πιο συγκεκριμένα, μετατρέπει αντικείμενα τύπου OWLClass, OWLProperty, OWLDataProperty, OWLIndividual και OWLTypedConstant σε αντικείμενα τύπου ATermAppl. Η ανάλυση και μετατροπή των οντοτήτων γίνεται με χρήση αναδρομικής τεχνικής.

5.7.1.2 Εφαρμογή μετατροπής αλφαριθμητικών σε ATerm τύπο

Η εφαρμογή αυτή αναλαμβάνει τη μετατροπή αλφαριθμητικών που αναπαριστούν αυθαίρετες εννοιακές εκφράσεις σε τύπους της ATerm Library, τους οποίους αναγνωρίζει εσωτερικά ο Reasoner. Είναι απαραίτητη για τον χειρισμό των αυθαίρετων εκφράσεων που εισάγει ο χρήστης στο σύστημα κατά τη διεξαγωγή ερωτημάτων. Η ανάλυση και μετατροπή των αλφαριθμητικών γίνεται με χρήση αναδρομικής τεχνικής.

5.7.1.3 *Εφαρμογή εύρεσης υποκειμένων που συμμετέχουν σε δήλωση ρόλου αντικειμένου με συγκεκριμένο άτομο ως αντικείμενο*

Η εφαρμογή αυτή δέχεται μια ατελή δήλωση ενός ρόλου τύπου OWLObjectProperty με ένα άτομο τύπου OWLIndividual ως αντικείμενο και επιστρέφει άτομα τύπου OWLIndividual που συμμετέχουν ως υποκείμενα σε αυτή τη δήλωση. Προφανώς, η πληροφορία αντλείται από τον Reasoner.

5.7.1.4 *Εφαρμογή εύρεσης υποκειμένων που συμμετέχουν σε δήλωση ρόλου τύπου δεδομένων με συγκεκριμένη σταθερά*

Η εφαρμογή αυτή δέχεται μια ατελή δήλωση ενός ρόλου τύπου OWLDataProperty με σταθερά τύπου OWLTypedConstant και επιστρέφει άτομα τύπου OWLIndividual που συμμετέχουν ως υποκείμενα σε αυτή τη δήλωση. Προφανώς, η πληροφορία αντλείται από τον Reasoner.

5.7.1.5 *Εφαρμογή ανάκτησης ψευδομοντέλου κλάσης*

Η εφαρμογή αυτή δέχεται μια κλάση τύπου ATermAppl και επιστρέφει ένα ψευδομοντέλο αυτής (ανακτώντας το από τον Reasoner) σε μορφή τέτοια που να μπορεί να το επεξεργαστεί και να το αποθηκεύσει το DBRS. Όπως αναφέραμε και στο Κεφάλαιο 2, τα ψευδομοντέλα κλάσεων που χειρίζεται το DBRS είναι εκφραστικότητας *SHN*. Επίσης, όσον αφορά στις πλειάδες $M^{\exists}$ και $M^{\forall}$ του ψευδομοντέλου κάθε κλάσης, το DBRS ακολουθεί μια «συντηρητική» προσέγγιση, αποθηκεύοντας μόνο το ρόλο R και όχι ολόκληρες τις κλάσεις της μορφής $\exists$R.C και $\forall$ R.C.

5.7.1.6 *Εφαρμογή ανάκτησης ψευδομοντέλου ατόμου*

Η εφαρμογή αυτή δέχεται ένα άτομο τύπου ATermAppl και επιστρέφει ένα ψευδομοντέλο αυτού (ανακτώντας το από τον Reasoner) σε μορφή τέτοια που να μπορεί να το επεξεργαστεί και να το αποθηκεύσει το DBRS. Όπως αναφέραμε και στο Κεφάλαιο 2, τα ψευδομοντέλα ατόμων που χειρίζεται το DBRS είναι εκφραστικότητας *SHN*. Επίσης, όσον αφορά στις πλειάδες $M^{\exists}$ και $M^{\forall}$ του ψευδομοντέλου κάθε ατόμου, το DBRS ακολουθεί μια «συντηρητική» προσέγγιση,

αποθηκεύοντας μόνο το ρόλο R και όχι ολόκληρες τις κλάσεις της μορφής $\exists$R.C και $\forall$ R.C.

5.7.1.7 Εφαρμογή ελέγχου μοναδικότητας ψευδομοντέλου ατόμου

Η εφαρμογή αυτή δέχεται το URI ενός ατόμου και ελέγχει αν το ψευδομοντέλο αυτού είναι μοναδικό, ανασύροντας την πληροφορία από τον Reasoner.

5.7.1.8 Εφαρμογή ελέγχου μοναδικότητας ψευδομοντέλου κλάσης

Η εφαρμογή αυτή δέχεται το URI μιας κλάσης και ελέγχει αν το ψευδομοντέλο αυτής είναι μοναδικό, ανασύροντας την πληροφορία από τον Reasoner.

5.7.1.9 Εφαρμογή επιλογής ατόμων που ανήκουν σε συγκεκριμένη κλάση

Η εφαρμογή αυτή δέχεται το URI μιας κλάσης μαζί με ένα σύνολο URIs ατόμων και επιστρέφει το υποσύνολο των ατόμων του αρχικού συνόλου που ανήκουν στην δεδομένη κλάση, ανασύροντας την πληροφορία από τον Reasoner.

5.7.1.10 Εφαρμογή επιλογής κλάσεων που είναι τύποι συγκεκριμένου ατόμου

Η εφαρμογή αυτή δέχεται το URI ενός ατόμου μαζί με ένα σύνολο URIs κλάσεων και επιστρέφει το υποσύνολο των κλάσεων του αρχικού συνόλου που είναι τύποι του δοσμένου ατόμου, ανασύροντας την πληροφορία από τον Reasoner.

5.8 Σχεσιακό σχήμα της βάσης δεδομένων του DBRS

Στο Σχήμα 5.7 φαίνεται το σχεσιακό σχήμα της βάσης δεδομένων που χρησιμοποιεί το DBRS, όπως αυτό προέκυψε από το μοντέλο Οντοτήτων Συσχετίσεων που παρατέθηκε στην Ενότητα 4.3. Ακολουθεί η αναλυτική περιγραφή του:

<u>Βασικοί Σχεσιακοί Πίνακες</u>

Περιέχουν τις βασικές οντότητες (κλάσεις/ρόλους/άτομα), καθώς και τα namespaces της οντολογίας. Αυτοί είναι:

- Πίνακας Namespaces

  Περιέχει τα URIs της οντολογίας, των οντολογιών που εισάγονται από αυτήν και άλλα απαραίτητα URIs, όπως αυτά των XML τύπων δεδομένων.

Πεδία

- namespace_id: Ακέραιος θετικός αριθμός που δίνεται αυτόματα από το DBMS, με χρήση κατάλληλου sequence. Είναι το πρωτεύον κλειδί του πίνακα. Η πρώτη εγγραφή στον πίνακα, με namespace_id=1, είναι πάντοτε το ψευδο-namepsace. Το ψευδο-namespace είναι ένα εικονικό namespace που χρησιμοποιεί εσωτερικά το DBRS για λόγους συνέπειας της βάσης δεδομένων.
- namespace: Αλφαριθμητικό. Περιέχει το URI του namespace. Δεν επιτρέπεται να είναι κενό (not null). Το ψευδο-namespace παίρνει την τιμή "pseudoNamespace".

- Πίνακας Concepts

  Περιέχει τη βασική πληροφορία σχετικά με τις κλάσεις τις οντολογίας.

  Πεδία

  - id_concept: Ακέραιος θετικός αριθμός που δίνεται αυτόματα από το DBMS, με χρήση κατάλληλου sequence. Είναι το πρωτεύον κλειδί του πίνακα.
  - concept_name: Αλφαριθμητικό. Είναι το τοπικό όνομα της κλάσης. Ανώνυμες κλάσεις που αποθηκεύονται λαμβάνουν το δεσμευμένο όνομα "anonymous". Δεν επιτρέπεται να είναι κενό (not null).
  - concept_definition: Αλφαριθμητικό. Είναι ο ορισμός της κλάσης όπως αυτός συναντάται στο αρχείο της οντολογίας. Επιτρέπεται να είναι κενό, καθώς μια κλάση μπορεί να είναι πρωτογενής (primitive), δηλαδή να μην διαθέτει ορισμό.
  - expanded_concept_definition: Αλφαριθμητικό. Είναι ο διευρυμένος ορισμός της κλάσης. Επιτρέπεται να είναι κενό για τον ίδιο ακριβώς λόγο με παραπάνω.
  - c_label_pre: Ακέραιος θετικός αριθμός. Είναι το πρώτο στοιχείο (index) της ετικέτας της κλάσης. Δεν επιτρέπεται να είναι κενό (not null). Οι ανώνυμες κλάσεις που αποθηκεύονται, μιας και δεν τοποθετούνται στην ιεραρχία των κλάσεων, λαμβάνουν στο πεδίο αυτό την τιμή 0.
  - c_label_post: Ακέραιος θετικός αριθμός. Είναι το δεύτερο στοιχείο (post) της ετικέτας της κλάσης. Δεν επιτρέπεται να είναι κενό (not null). Οι

ανώνυμες κλάσεις που αποθηκεύονται, μιας και δεν τοποθετούνται στην ιεραρχία των κλάσεων, λαμβάνουν στο πεδίο αυτό την τιμή 0.

- id_father_class: Ακέραιος αριθμός. Παίρνει την τιμή της ιδιότητας id_concept του πατέρα της κλάσης, βάσει του επιλεγμένου spanning tree κατά την κατασκευή του σχήματος των ετικετών. Δεν επιτρέπεται να είναι κενό (not null). Στην ρίζα του γράφου της ιεραρχίας (στην κλάση OWLThing) δίνεται η τιμή -1 στο πεδίο αυτό. Οι ανώνυμες κλάσεις που αποθηκεύονται, μιας και δεν τοποθετούνται στην ιεραρχία των κλάσεων, λαμβάνουν στο πεδίο αυτό την τιμή 0.
- c_assertion_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τις δηλώσεις της κλάσης είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).
- c_disj_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τις ασύμβατες κλάσεις της συγκεκριμένης κλάσης είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).
- c_compl_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τις συμπληρωματικές κλάσεις της συγκεκριμένης κλάσης είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).
- unique_pseudomodel: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η συγκεκριμένη κλάση δεν διαθέτει άλλο ψευδομοντέλο πέρα από αυτό που είναι αποθηκευμένο στη βάση. Δεν επιτρέπεται να είναι κενό (not null).
- unique_complement_pseudomodel: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η συμπληρωματική της συγκεκριμένης κλάσης δεν διαθέτει άλλο ψευδομοντέλο πέρα από αυτό που είναι αποθηκευμένο στη βάση. Δεν επιτρέπεται να είναι κενό (not null).
- namespace_id: Ακέραιος θετικός αριθμός. Είναι ξένο κλειδί από τη σχέση "Namespaces" και δείχνει σε ποια οντολογία ανήκει η κλάση. Δεν

επιτρέπεται να είναι κενό (not null). Για την ιδιότητα αυτή, οι ανώνυμες κλάσεις λαμβάνουν την τιμή 1 που αντιστοιχεί στο ψευδο-namespace.

Περιορισμοί μοναδικότητας (Υποψήφια κλειδιά)

- {concept_name , namespace_id}: Δυο κλάσεις που ανήκουν στην ίδια οντολογία δεν επιτρέπεται να έχουν το ίδιο τοπικό όνομα.

Παρατηρήσεις

- Οι ισοδύναμες κλάσεις τις οντολογίας έχουν ίδιες τιμές στα πεδία c_label_pre και c_label_post.

- Πίνακας Roles

Περιέχει τη βασική πληροφορία σχετικά με τους ρόλους της οντολογίας.

Πεδία

- id_role: Ακέραιος θετικός αριθμός που δίνεται αυτόματα από το DBMS, με χρήση κατάλληλου sequence. Είναι το πρωτεύον κλειδί της σχέσης.
- role_name: Αλφαριθμητικό. Είναι το τοπικό όνομα του ρόλου. Δεν επιτρέπεται να είναι κενό (not null). Στους ρόλους που συμβολίζουν τη ρίζα και το τέλος της ιεραρχίας των ρόλων, δίνονται τα δεσμευμένα ονόματα "_TOP_" και "not(_TOP_)" αντιστοίχως.
- r_label_pre: Ακέραιος θετικός αριθμός. Είναι το πρώτο στοιχείο (index) της ετικέτας του ρόλου. Δεν επιτρέπεται να είναι κενό (not null).
- r_label_post: Ακέραιος θετικός αριθμός. Είναι το δεύτερο στοιχείο (post) της ετικέτας του ρόλου. Δεν επιτρέπεται να είναι κενό (not null).
- id_father_role: Ακέραιος αριθμός. Παίρνει την τιμή της ιδιότητας id_role του πατέρα του ρόλου, βάσει του επιλεγμένου spanning tree κατά την κατασκευή του σχήματος των ετικετών. Δεν επιτρέπεται να είναι κενό (not null). Στην ρίζα του γράφου της ιεραρχίας (στον εικονικό ρόλο _TOP_ δηλαδή) δίνεται η τιμή -1.
- role_characteristics: Ακέραιος θετικός αριθμός. Καθορίζει τα χαρακτηριστικά του ρόλου βάσει της κωδικοποίησης που περιγράψαμε στην ενότητα 6.1.4. Δεν επιτρέπεται να είναι κενό (not null).
- r_assertion_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη

στη βάση σχετικά με τις δηλώσεις του ρόλου είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).

- r_disj_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τους ασύμβατους ρόλους του συγκεκριμένου ρόλου είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).
- namespace_id: Ακέραιος θετικός αριθμός. Είναι ξένο κλειδί από τη σχέση "Namespaces" και δείχνει σε ποια οντολογία ανήκει ο ρόλος. Δεν επιτρέπεται να είναι κενό (not null). Στους εικονικούς ρόλους "_TOP_" και "not(_TOP_)" δίνεται η τιμή 1. Ανήκουν δηλαδή στο ψευδο-namespace.

Περιορισμοί μοναδικότητας (Υποψήφια κλειδιά)

- {role_name , namespace_id}: Δυο ρόλοι που ανήκουν στην ίδια οντολογία δεν μπορούν να έχουν το ίδιο τοπικό όνομα.

Παρατηρήσεις

- Οι ισοδύναμοι ρόλοι τις οντολογίας έχουν ίδιες τιμές στα πεδία r_label_pre και r_label_post.

- Πίνακας Individuals

Περιέχει τη βασική πληροφορία σχετικά με τα άτομα της οντολογίας.

Πεδία

- id_individual: Ακέραιος θετικός αριθμός που δίνεται αυτόματα από το DBMS, με χρήση κατάλληλου sequence. Είναι το πρωτεύον κλειδί της σχέσης.
- individual_name: Αλφαριθμητικό. Είναι το τοπικό όνομα του ατόμου. Δεν επιτρέπεται να είναι κενό (not null). Στα δύο εικονικά άτομα που χρησιμοποιούνται ως ρίζα και τέλος της ιεραρχίας των δηλώσεων των μεταβατικών ρόλων, δίνονται τα δεσμευμένα ονόματα "_TOP_" και "not(_TOP_)" αντίστοιχα.
- i_type_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τους τύπους του ατόμου είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).

- i_diff_complete: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι η πληροφορία που βρίσκεται αποθηκευμένη στη βάση σχετικά με τα διαφορετικά, από το συγκεκριμένο άτομο, άτομα είναι πλήρης. Δεν επιτρέπεται να είναι κενό (not null).
- unique_pseudomodel: Δυαδική μεταβλητή. Προκαθορισμένη τιμή false. Αν είναι true σημαίνει ότι το συγκεκριμένο άτομο δεν διαθέτει άλλο ψευδομοντέλο πέρα από αυτό που είναι φορτωμένο στη βάση. Δεν επιτρέπεται να είναι κενό (not null).
- namespace_id: Ακέραιος θετικός αριθμός. Είναι ξένο κλειδί από τη σχέση "Namespaces" και δείχνει σε ποια οντολογία ανήκει το άτομο. Δεν επιτρέπεται να είναι κενό (not null). Στα εικονικά άτομα "_TOP_" και "not(_TOP_)" δίνεται η τιμή 1. Ανήκουν δηλαδή στο ψευδο-namespace.

Περιορισμοί μοναδικότητας (Υποψήφια κλειδιά)

- {individual_name , namespace_id}: Δύο άτομα που ανήκουν στην ίδια οντολογία δεν επιτρέπεται να έχουν το ίδιο τοπικό όνομα.

«Συμμετρικοί» Πίνακες

Οι πίνακες αυτοί σχετίζουν δύο εγγραφές κάποιου βασικού πίνακα μεταξύ τους. Γι' αυτό το λόγο, οι εγγραφές τους αποτελούνται από δυο πεδία που περιέχουν τα ids των εγγραφών των βασικών πινάκων ως ξένα κλειδιά. Τα δύο αυτά πεδία μαζί αποτελούν και το πρωτεύον κλειδί κάθε πίνακα, δηλαδή κάθε εγγραφή στους πίνακες αυτούς είναι μοναδική. Οι συμμετρικοί πίνακες του σχήματος είναι οι παρακάτω:

- Πίνακας Concept Disjointness

  Οι δύο εγγεγραμμένες κλάσεις είναι ασύμβατες (disjoint)
- Πίνακας Concept Completion

  Οι δύο εγγεγραμμένες κλάσεις είναι συμπληρωματικές (complement)
- Πίνακας Role Disjointness

  Οι δύο εγγεγραμμένοι ρόλοι είναι ασύμβατοι (disjoint)

- Πίνακας Inversion

  Οι δύο εγγεγραμμένοι ρόλοι είναι αντίστροφοι (inverse)

- Πίνακας Individuals Equivalence

  Τα δύο εγγεγραμμένα άτομα είναι ισοδύναμα (same)

- Πίνακας Individuals Difference

  Τα δύο εγγεγραμμένα άτομα είναι διαφορετικά (different)

Παρατηρήσεις

- Δεν υπάρχει καμία εγγραφή αντίστροφη κάποιας άλλης στη σχέση, για αποφυγή επανάληψης της ίδιας πληροφορίας και οικονομία χώρου.
- Δεν υπάρχουν ισοδύναμες εγγραφές. Ισοδύναμες θεωρούνται οι εγγραφές που εκφράζουν ουσιαστικά την ίδια πληροφορία, καθώς τα μέλη τους σχετίζονται μεταξύ τους με σχέσεις ισοδυναμίας.
- Οι συμμετρικοί ρόλοι, που είναι ουσιαστικά ρόλοι αντίστροφοι με τον εαυτό τους, δεν εγγράφονται στη σχέση "Inversion". Αναγνωρίζονται από την τιμή του πεδίου "role_characteristics" του πίνακα "Roles".

Πίνακες Δηλώσεων

Περιέχουν τις δηλώσεις των κλάσεων και των ρόλων της οντολογίας.

- Πίνακας Concept Assertions

  Περιέχει τις δηλώσεις των κλάσεων της οντολογίας.

  Πεδία

  - id_concept: Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με το πεδίο id_individual της σχέσης.
  - id_individual: Ξένο κλειδί από τον πίνακα "Individuals". Κύριο κλειδί μαζί με το πεδίο id_concept της σχέσης.

  Παρατηρήσεις

  - Δεν υπάρχουν ισοδύναμες εγγραφές για οικονομία χώρου.

- Πίνακας Role Assertions

  Περιέχει τις δηλώσεις των ρόλων αντικειμένου της οντολογίας.

Πεδία

- id_role: Ξένο κλειδί από τον πίνακα "Roles". Είναι ο ρόλος για τον οποίο η εγγραφή είναι δήλωση. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_individual_subject: Ξένο κλειδί από τον πίνακα "Individuals". Είναι το υποκείμενο της δήλωσης. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_individual_object: Ξένο κλειδί από τον πίνακα "Individuals". Είναι το αντικείμενο της δήλωσης. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

Παρατηρήσεις

- Δεν υπάρχουν ισοδύναμες εγγραφές για οικονομία χώρου.

- Πίνακας Inferred Transitive Role Assertions

  Έχει ακριβώς την ίδια μορφή με τον πίνακα "Role Assertions" μόνο που περιέχει δηλώσεις μεταβατικών ρόλων που προκύπτουν κατά τη διαδικασία αποτίμησης ερωτημάτων.

- Πίνακας Datatype Role Assertions

  Περιέχει τις δηλώσεις των ρόλων τύπου δεδομένων της οντολογίας.

  Πεδία

  - id_role: Ξένο κλειδί από τον πίνακα "Roles". Είναι ο ρόλος για τον οποίο η εγγραφή είναι δήλωση. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_individual_subject: Ξένο κλειδί από τον πίνακα "Individuals". Είναι το υποκείμενο της δήλωσης. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - literal: Αλφαριθμητικό. Είναι η τιμή της σταθεράς της δήλωσης. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - type: Αλφαριθμητικό. Είναι ο τύπος του αντικειμένου της δήλωσης. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

  Παρατηρήσεις

  - Δεν υπάρχουν ισοδύναμες εγγραφές για οικονομία χώρου.

- Πίνακας Transitive Role Assertions

  Περιέχει τις δηλώσεις των μεταβατικών ρόλων της οντολογίας και τις ιεραρχίες που αυτές σχηματίζουν.

  Πεδία

  - id_role: Ξένο κλειδί από τον πίνακα "Roles". Είναι ο ρόλος για τον οποίο η εγγραφή είναι δήλωση. Κύριο κλειδί μαζί με το πεδίο id_individual.
  - id_individual: Ξένο κλειδί από τον πίνακα "Individuals". Είναι το άτομο της δήλωσης. Κύριο κλειδί μαζί με το πεδίο id_role.
  - t_label_pre: Ακέραιος θετικός αριθμός. Είναι το πρώτο στοιχείο (index) της ετικέτας της δήλωσης. Δεν επιτρέπεται να είναι κενό.
  - t_label_post: Ακέραιος θετικός αριθμός. Είναι το δεύτερο στοιχείο (post) της ετικέτας της δήλωσης. Δεν επιτρέπεται να είναι κενό.
  - id_father: Ακέραιος αριθμός. Παίρνει την τιμή της ιδιότητας id_individual του άμεσου προγόνου της δήλωσης, βάσει του επιλεγμένου spanning tree κατά την κατασκευή του σχήματος των ετικετών. Δεν επιτρέπεται να είναι κενό. Στην ρίζα του γράφου κάθε ιεραρχίας (στο εικονικό άτομο _TOP_, όπου υπάρχει σε δήλωση) δίνεται η τιμή -1.

  Περιορισμοί μοναδικότητας (Υποψήφια κλειδιά)

  - {id_role , t_label_post}: Το δεύτερο στοιχείο της ετικέτας κάθε δήλωσης ενός μεταβατικού ρόλου είναι μοναδικό για την ιεραρχία δηλώσεων αυτού του ρόλου.

  Παρατηρήσεις

  - Δεν υπάρχουν ισοδύναμες εγγραφές για οικονομία χώρου.

Πίνακες Ετικετών DAG

- Πίνακας DAG Concept Labels

  Περιέχει την πληροφορία του σχήματος ετικετών της ιεραρχίας των κλάσεων για τις ακμές του γράφου της ιεραρχίας που δεν ανήκουν στο συνδετικό δέντρο. Κάθε εγγραφή περιέχει μια κλάση και μια ετικέτα κάποιας άλλης κλάσης που διαδόθηκε (propagated) στην πρώτη, βάσει του αλγορίθμου κατασκευής του σχήματος ετικετών.

Πεδία

- id_concept: Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- c_label_pre_dag: Ακέραιος θετικός αριθμός. Είναι το πρώτο στοιχείο (index) της ετικέτας. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- c_label_post_dag: Ακέραιος θετικός αριθμός. Είναι το δεύτερο στοιχείο (post) της ετικέτας. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

- Πίνακας DAG Role Labels

  Περιέχει την πληροφορία του σχήματος ετικετών της ιεραρχίας των ρόλων για τις ακμές του γράφου της ιεραρχίας που δεν ανήκουν στο συνδετικό δέντρο. Ομοίως με τον πίνακα "DAG Concept Labels".

- Πίνακας DAG TRole Labels

  Περιέχει την πληροφορία του σχήματος ετικετών για τις ακμές των γράφων των ιεραρχιών των δηλώσεων των μεταβατικών ρόλων που δεν ανήκουν στα αντίστοιχα συνδετικά δέντρα. Κάθε εγγραφή περιέχει το id ενός ρόλου και ενός ατόμου, δηλαδή μια δήλωση, και μια ετικέτα κάποιας άλλης δήλωσης του συγκεκριμένου μεταβατικού ρόλου που διαδόθηκε στην πρώτη, βάσει του αλγορίθμου κατασκευής του σχήματος ετικετών.

  Πεδία

  - id_role: Ξένο κλειδί μαζί με το πεδίο "id_individual" από τον πίνακα "Transitive Role Assertions". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_individual: Ξένο κλειδί μαζί με το πεδίο "id_role"από τον πίνακα "Transitive Role Assertions". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - t_label_pre_dag: Ακέραιος θετικός αριθμός. Είναι το πρώτο στοιχείο (index) της ετικέτας. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - t_label_post_dag: Ακέραιος θετικός αριθμός. Είναι το δεύτερο στοιχείο (post) της ετικέτας. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

Πίνακες Ψευδομοντέλων

Σε αυτούς αποθηκεύονται τα ψευδομοντέλα των κλάσεων και των ατόμων της οντολογίας. Κάθε πλειάδα, εκ των τεσσάρων, ενός ψευδομοντέλου (όπως περιγράφηκε στο κεφάλαιο 2) περιέχεται στον αντίστοιχο ομώνυμο πίνακα. Όσον αφορά τα ψευδομοντέλα των εννοιών, πέρα από το ψευδομοντέλο μιας κλάσης, στους ίδιους πίνακες αποθηκεύεται και το ψευδομοντέλο της συμπληρωματικής της.

Για κάθε κλάση ή άτομο μπορεί να αποθηκευθεί παραπάνω από ένα ψευδομοντέλο. Για αυτόν ακριβώς τον λόγο, υπάρχουν τα πεδία id_conceptPseudomodel και id_individualPseudomodel. Αποθηκεύοντας περισσότερα του ενός ψευδομοντέλα, επιτυγχάνουμε σταδιακά την μόνιμη αποθήκευση του «ξεδιπλωμένου», από τον Tableau αλγόριθμο, ABox. Θεωρητικά, όσο περισσότερα ψευδομοντέλα υπάρχουν αποθηκευμένα για κάθε έννοια ή άτομο, τόσο μεγαλύτερη η πιθανότητα να απαντήσει το DBRS σε κάποιο ερώτημα, χωρίς την κλήση του Reasoner.

- Πίνακας Concept PseudoModel In

  Περιέχει τις κλάσεις που ανήκουν στην πλειάδα “In” του ψευδομοντέλου.

  Πεδία

  - id_concept: Το id της κλάσης στην οποία ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα “Concepts”. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για τη συγκεκριμένη κλάση. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptIn: Το id της κλάσης που περιέχεται στην πλειάδα “In”. Ξένο κλειδί από τον πίνακα “Concepts”. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - complement: Δυαδική μεταβλητή. Αν είναι “true” το ψευδομοντέλο ανήκει στην συμπληρωματική της κλάσης που έχει id την τιμή του πεδίου id_concept. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

- Πίνακας Concept PseudoModel NotIn

  Περιέχει τις κλάσεις που ανήκουν στην πλειάδα "Not In" του ψευδομοντέλου.

  Πεδία

  - id_concept: Το id της κλάσης στην οποία ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για τη συγκεκριμένη κλάση. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptNotIn: Το id της κλάσης που περιέχεται στην πλειάδα "Not In". Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - complement: Δυαδική μεταβλητή. Αν είναι "true" το ψευδομοντέλο ανήκει στην συμπληρωματική της κλάσης που έχει id την τιμή του πεδίου id_concept. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

- Πίνακας Concept PseudoModel Existence

  Περιέχει τους ρόλους που ανήκουν στο σύνολο "Existence" του ψευδομοντέλου.

  Πεδία

  - id_concept: Το id της κλάσης στην οποία ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για τη συγκεκριμένη κλάση. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_role: Το id του ρόλου που περιέχεται στην πλειάδα "Existence". Ξένο κλειδί από τον πίνακα "Roles". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - complement: Δυαδική μεταβλητή. Αν είναι "true" το ψευδομοντέλο ανήκει στην συμπληρωματική της κλάσης που έχει id την τιμή του πεδίου id_concept. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

- Πίνακας Concept PseudoModel Universial

  Περιέχει τους ρόλους που ανήκουν στο σύνολο "Universial" του ψευδομοντέλου.

  Πεδία

  - id_concept: Το id της κλάσης στην οποία ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_conceptPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για τη συγκεκριμένη κλάση. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - id_role: Το id του ρόλου που περιέχεται στην πλειάδα "Universial". Ξένο κλειδί από τον πίνακα "Roles". Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.
  - complement: Δυαδική μεταβλητή. Αν είναι "true" το ψευδομοντέλο ανήκει στην συμπληρωματική της κλάσης που έχει id την τιμή του πεδίου id_concept. Κύριο κλειδί μαζί με τα άλλα τρία πεδία της σχέσης.

- Πίνακας Individual PseudoModel In

  Περιέχει τις κλάσεις που ανήκουν στην πλειάδα "In" του ψευδομοντέλου.

  Πεδία

  - id_individual: Το id του ατόμου στο οποίο ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Individuals". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
  - id_individualPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για το συγκεκριμένο άτομο. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
  - id_conceptIn: Το id της κλάσης που περιέχεται στην πλειάδα "In". Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

- Πίνακας Individual PseudoModel NotIn

  Περιέχει τις κλάσεις που ανήκουν στην πλειάδα "Not In" του ψευδομοντέλου.

Πεδία

- id_individual: Το id του ατόμου στο οποίο ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Individuals". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_individualPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για το συγκεκριμένο άτομο. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_conceptNotIn: Το id της κλάσης που περιέχεται στην πλειάδα "Not In". Ξένο κλειδί από τον πίνακα "Concepts". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

- **Πίνακας Individual PseudoModel Existence**

Περιέχει τους ρόλους που ανήκουν στο σύνολο "Existence" του ψευδομοντέλου.

Πεδία

- id_: Το id του ατόμου στο οποίο ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Individuals". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_individualPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για τη συγκεκριμένο άτομο. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_role: Το id του ρόλου που περιέχεται στην πλειάδα "Existence". Ξένο κλειδί από τον πίνακα "Roles". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

- **Πίνακας Individual PseudoModel Universial**

Περιέχει τους ρόλους που ανήκουν στο σύνολο "Universial" του ψευδομοντέλου.

Πεδία

- id_: Το id του ατόμου στο οποίο ανήκει το ψευδομοντέλο. Ξένο κλειδί από τον πίνακα "Individuals". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

- id_individualPseudomodel: Ακέραιος θετικός αριθμός. Το id του ψευδομοντέλου για το συγκεκριμένο άτομο. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
- id_role: Το id του ρόλου που περιέχεται στην πλειάδα "Universial". Ξένο κλειδί από τον πίνακα "Roles". Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.

Πίνακες Caching Ερωτημάτων

Σε αυτούς αποθηκεύονται κάποια επιλεγμένα ερωτήματα, που κατά την αποτίμησή τους εστάλησαν στον Reasoner και η πληροφορία που αυτός επέστρεψε δεν μπορούσε να αποθηκευθεί ή απλώς υπήρχε ήδη στη βάση αλλά δεν γνωρίζαμε ότι ήταν πλήρης. Αυτό γίνεται, για να απαντηθούν (σε περίπτωση που υποβληθούν τα ίδια ή ισοδύναμά τους στο μέλλον) αποκλειστικά μέσω SQL.

- Πίνακας Binary Query Cacher

  Εδώ αποθηκεύονται δυαδικά (boolean) ερωτήματα δύο ορισμάτων (όχι μεταβλητές), που για την αποτίμησή τους εστάλησαν στον Reasoner και απαντήθηκαν false.

  Πεδία

  - id_query: Ακέραιος θετικός αριθμός. Καθορίζει τον τύπο του ερωτήματος. Δέχεται τιμές από 1 έως 5, ανάλογα αν το ερώτημα είναι τύπου DisjointWith() κλάσεων, DisjointWith() ρόλων, ComplementOf(), DiffernentFrom() ή TypeOf() αντίστοιχα. Κύριο κλειδί μαζί με τα άλλα δυο πεδία της σχέσης.
  - id_arg1: Ακέραιος θετικός αριθμός. Είναι το id της κλάσης, ρόλου ή ατόμου (ανάλογα με τον τύπο του ερωτήματος) του πρώτου ορίσματος του ερωτήματος που αποθηκεύεται. Κύριο κλειδί μαζί με τα άλλα δυο πεδία της σχέσης.
  - id_arg2: Ομοίως με το id_arg1 για το δεύτερο όρισμα του ερωτήματος.

  Παρατηρήσεις

  - Δεν υπάρχουν ισοδύναμες εγγραφές για οικονομία χώρου.

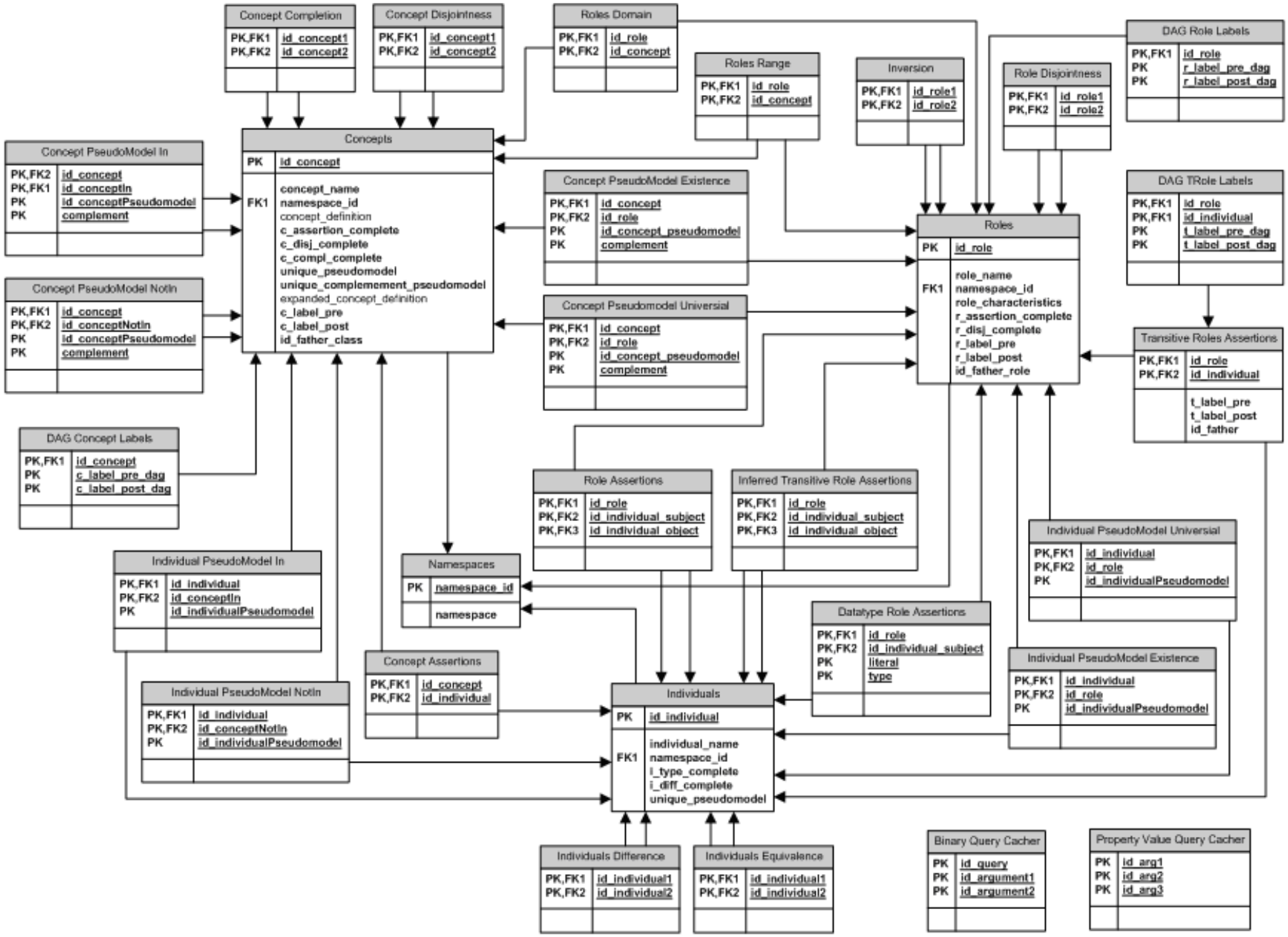
- Πίνακας Property Value Query Cacher

  Εδώ αποθηκεύονται ερωτήματα τύπου PropertyValue() για την αποτίμηση των οποίων έγινε κλήση του Reasoner. Αυτά μπορεί να είναι δύο ειδών:

  - Χωρίς μεταβλητή μεταξύ των ορισμάτων τους, που απαντήθηκαν αρνητικά.
  - Με ακριβώς μια μεταβλητή μεταξύ των ορισμάτων τους. Η μεταβλητή κωδικοποιείται με την τιμή 0 στο πεδίο που της αντιστοιχεί.

  Πεδία

  - id_arg1: Ακέραιος θετικός αριθμός. Είναι το id του ατόμου του ορίσματος-υποκειμένου του ερωτήματος που αποθηκεύεται. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
  - id_arg2: Ακέραιος θετικός αριθμός. Είναι το id του ρόλου του ερωτήματος που αποθηκεύεται. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.
  - id_arg3: Ακέραιος θετικός αριθμός. Είναι το id του ατόμου του ορίσματος-αντικειμένου του ερωτήματος που αποθηκεύεται. Κύριο κλειδί μαζί με τα άλλα δύο πεδία της σχέσης.



Σχήμα 5.7: Σχεσιακό Διάγραμμα της Βάσης Δεδομένων του DBRS

6

Υλοποίηση

Στο κεφάλαιο αυτό δίνουμε λεπτομέρειες σχετικά με την υλοποίηση του DBRS στην παρούσα έκδοσή του. Στην πρώτη ενότητα περιγράφονται επιλεκτικά διαδικασίες που επιτελούνται από το σύστημα είτε κατά τη φόρτωση είτε κατά τη διεξαγωγή ερωτημάτων, οι οποίες παρουσιάζουν αλγοριθμικό ενδιαφέρον. Στη δεύτερη ενότητα αναφέρονται οι πλατφόρμες πάνω στις οποίες ελέγχθηκε η λειτουργία του DBRS και τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξή του. Στη συνέχεια δίνονται οδηγίες για την εγκατάσταση του συστήματος και στο τέλος του κεφαλαίου περιγράφουμε αναλυτικά τις Java κλάσεις του DBRS και τα ευρετήρια της βάσης δεδομένων που χρησιμοποιεί εσωτερικά.

6.1 Αλγόριθμοι

Στην ενότητα αυτή περιγράφονται ενδεικτικά ορισμένες διαδικασίες που επιτελεί το DBRS οι οποίες παρουσιάζουν αλγοριθμικό ενδιαφέρον. Προς διευκόλυνση του αναγνώστη, δίνονται οι βασικές αρχές κάθε αλγορίθμου και αμέσως μετά ο αντίστοιχος ψευδοκώδικας ή το αντίστοιχο διαδικαστικό διάγραμμα ροής. Μεταξύ των δύο μορφών παρουσιάσης επιλέγεται κάθε φορά η πιο ευανάγνωστη προς διευκόλυνση του αναγνώστη.

6.1.1 Φόρτωση πλήρους πληροφορίας σχετικά με την ασυμβατότητα κλάσεων ή ρόλων της οντολογίας

Όπως αναφέρθηκε σε προηγούμενα κεφάλαια, το DBRS δίνει τη δυνατότητα στο χρήστη να φορτώσει εξαρχής στη βάση δεδομένων την πλήρη πληροφορία σχετικά με τις ασύμβατες κλάσεις ή ρόλους της οντολογίας. Μια πολύ απλή τεχνική θα ήταν να ζητούσαμε από τον Reasoner να μας επιστρέψει όλα τα ζεύγη ασύμβατων κλάσεων ή ρόλων μέσω του Tableau αλγορίθμου. Ωστόσο, η εύρεση υπονοούμενων σχέσεων ασυμβατότητας είναι ιδιαίτερα χρονοβόρα[39] και, γι'αυτό το λόγο, υλοποιήσαμε έναν αλγόριθμο που επιταχύνει την όλη διαδικασία, εκμεταλλευόμενος την ήδη κατασκευασμένη και αποθηκευμένη (μέσω σχήματος ετικετών) στη βάση δεδομένων ιεραρχία.

Η γενική ιδέα του αλγορίθμου είναι το «φιλτράρισμα» των υποψήφιων ασύμβατων κλάσεων (ρόλων) που στέλνονται στον Reasoner για την τελική απάντηση. Η κατασκευασμένη ιεραρχία (ετικέτες) διατρέχεται κατά πλάτος (BFS) και εντοπίζονται ασύμβατες ή μη ασύμβατες κλάσεις (ρόλοι) που δε χρειάζεται να σταλούν στον Reasoner. Ο αλγόριθμος είναι αναδρομικός και λειτουργεί ως εξής:

- Ελέγχεται, μέσω ετικετών, αν οι υποψήφιες/οι κλάσεις/ρόλοι έχουν κάποιο κοινό απόγονο. Αν ναι, τότε προφανώς δεν είναι ασύμβατες/οι.
- Ελέγχεται, μέσω ετικετών, αν οι υποψήφιες/οι κλάσεις/ρόλοι έχουν ασύμβατους προγόνους. Αν ναι, τότε είναι και αυτές/οί ασύμβατες/οι. Οι ασύμβατοι πρόγονοι είτε έχουν δηλωθεί ρητά στο αρχείο της οντολογίας ως

---

[39] Αυτό έχει να κάνει σε μεγάλο βαθμό με τη «φύση» της σχέσης. Ενδεικτικά αναφέρουμε ότι η σχέση ασυμβατότητας στερείται μεταβατικής ιδιότητας, σε αντίθεση για παράδειγμα με τη σχέση ισοδυναμίας. Αν έχουμε δύο ασύμβατες μεταξύ τους κλάσεις Α και Β, μια τρίτη κλάση Γ που είναι ασύμβατη με την Α δε σημαίνει ότι είναι απαραίτητα ασύμβατη και με τη Β. Επίσης, δυο κλάσεις με προγόνους που δεν είναι ασύμβατοι μεταξύ τους μπορεί να συνδέονται με σχέση ασυμβατότητας. Τα παραπάνω υποδεικνύουν ότι, για την εύρεση πλήρους πληροφορίας, πρέπει να ελεγχθούν πολλά υποψήφια ζεύγη κλάσεων. Ομοίως και για τους ρόλους.

τέτοιοι (και επομένως υπάρχουν στη βάση) είτε έχουν βρεθεί (μέσα από διαδικασία Tableau) κατά την εκτέλεση ενός προηγούμενου βήματος του παρόντος αλγορίθμου.

- Στην περίπτωση των ρόλων, τα σύνολα των ρόλων αντικειμένου και των ρόλων τύπου δεδομένων θεωρούνται εξ ορισμού ασύμβατα μεταξύ τους και δεν ελέγχονται για ασυμβατότητα.

Οι πρώτοι δύο έλεγχοι, παρά την απλότητά τους, δεν επιτελούνται κατ' ανάγκη από τους Reasoners. Αυτό συμβαίνει γιατί τα συστήματα συλλογιστικής ανάλυσης λειτουργούν αποκλειστικά στην κύρια μνήμη και κατά συνέπεια δεν μπορούν να «κρατάνε» (cache) μεγάλο μέρος προηγούμενων υπολογισμών. Ωστόσο, ακόμα και αν οι έλεγχοι αυτοί γίνονταν κατά την Tableau διαδικασία, η διάσχιση της ιεραρχίας των κλάσεων/ρόλων κατά BFS είναι ιδιαιτέρως δύσκολο να γίνει από το Reasoner, επειδή ακριβώς η οργάνωση του TBox της οντολογίας, στην εσωτερική βάση γνώσης που αυτός χρησιμοποιεί, γίνεται με ένα τελείως διαφορετικό τρόπο (απ' ότι στη ΒΔ του DBRS) όπως περιγράφηκε στο κεφάλαιο 2.

Διατρέχοντας την ιεραρχία κατά πλάτος (breadth-first) εκμεταλλευόμαστε με το βέλτιστο τρόπο τις γνωστές, σε κάθε βήμα του αλγορίθμου, σχέσεις ασυμβατότητας. Πιο συγκεκριμένα, η ασυμβατότητα μεταξύ δύο κλάσεων/ρόλων, που έχουν τοποθετηθεί στη συνολική ιεραρχία, μεταδίδεται και σε όλες τις/τους υποκλάσεις/υπορόλους τους. Έτσι, αν θέλουμε να εντοπίζουμε όσο το δυνατόν γρηγορότερα τέτοιες σχέσεις, πρέπει να ξεκινάμε την αναζήτηση από πάνω προς τα κάτω και να διατρέχουμε την ιεραρχία κατά πλάτος. Επίσης, με αυτόν τον τρόπο αποθηκεύουμε πολύ λιγότερη πληροφορία στη βάση δεδομένων, επειδή ακριβώς μας ενδιαφέρει μόνο το «ψηλότερο» στην ιεραρχία ασύμβατο ζεύγος και όχι τα «παρακάτω».

Στη συνέχεια δίνουμε υπό μορφή ψευδοκώδικα τον αλγόριθμο εύρεσης ασύμβατων κλάσεων. Η λογική είναι ίδια και για τους ρόλους (με ελάχιστες παραλλαγές) και παραλείπεται εσκεμμένα.

Αλγόριθμος

Σημειώνουμε εξαρχής ότι, όπου αναφερόμαστε σε αναγνωριστικό (id) κλάσης, εννοούμε αυτό που έχει δοθεί αυτόματα από το DBMS και είναι αποθηκευμένο στη

βάση δεδομένων του DBRS. Τα σύνολα αναγνωριστικών κάθε επιπέδου αποθηκεύονται σε ένα προσωρινό σχεσιακό πίνακα με όνομα «LevelConceptsTemp». Έτσι, κάθε φορά που ο αλγόριθμος χρειάζεται ένα τέτοιο σύνολο, αυτό ανασύρεται από τη βάση δεδομένων του DBRS.

```
loadDisjointConcepts():
OWLThingID := id καθολικής κλάσης
Εύρεση των άμεσων απογόνων της καθολικής κλάσης που ανήκουν στο συνδετικό δέντρο της ιεραρχίας:
      Κλάσεις_Επίπεδου_1 := Κενό Σύνολο;
      Για κάθε κλάση του πίνακα Concepts
      {
            Id_current_class := id κλάσης;

            Αν id_father_class = OWLThingID τότε:
            Επίπεδο_1 := Επίπεδο_1 ∪ {id_current_class};
      }
current_level := 1;
Τρέχον_επίπεδο := Επίπεδο_1;
Κλήση της μεθόδου loadDisjointConceptsRecursively():
{
      Για i := 1  έως i <= current_level κάνε:
      {
            Για κάθε ζεύγος αναγνωριστικών (id_1, id_2) τέτοιο ώστε id_1 ∈ Επίπεδο_i και id_2 ∈ Τρέχον_Επίπεδο
            {
                  Εύρεση ισοδυναμίας μεταξύ των δύο υποψηφίων κλάσεων μέσω labels;
                  Αν είναι ισοδύναμες τότε:
                        Συνέχεια στο επόμενο υποψήφιο ζεύγος;

                  Αποστολή ενός SQL ερωτήματος στο DBMS με το οποίο ελέγχεται αν οι δύο κλάσεις έχουν κοινούς απογόνους;
                  Αν έχουν κοινούς απογόνους τότε:
                        Συνέχεια στο επόμενο υποψήφιο ζεύγος;
                  διαφορετικά:
                  {
                        Αποστολή ενός SQL ερωτήματος στο DBMS με το οποίο ελέγχεται αν οι δύο κλάσεις έχουν ασύμβατους προγόνους ή μια τουλάχιστον από αυτές είναι μη ικανοποιήσιμη;
```

```
                        Αν δεν έχουν ασύμβατους προγόνους και
                        είναι και οι δύο ικανοποιήσιμες τότε:
                        {
                              Το ζεύγος στέλνεται στο Reasoner;
                              Αν προκύψει ασυμβατότητα τότε:
                                    Αποθηκεύεται το ζεύγος των ids
                                    στον πίνακα
                                    ConceptDisjointness
                        }
                        Συνέχεια στο επόμενο υποψήφιο ζεύγος;
                  }
            }
      }
      Αν υπάρχει νέο επίπεδο τότε:
      {
            Εύρεση αναγνωριστικών κλάσεων νέου επιπέδου;
            Κλήση μεθόδου loadDisjointConceptsRecursively();
      }
}
```

6.1.2 Κατασκευή διευρυμένου TBox

Στο τέλος της διαδικασίας φόρτωσης του TBox της οντολογίας στη βάση δεδομένων και ανάλογα με το αν το έχει επιλέξει ο χρήστης, εκτελείται η διαδικασία κατασκευής (και αποθήκευσης) του διευρυμένου (expanded) TBox. Ως διευρυμένο TBox ορίζεται εκείνο, στο δεξί μέρος όλων των αξιωμάτων του οποίου εμφανίζονται μόνον πρωταρχικές (primitive) κλάσεις. Ωστόσο, η διεύρυνση που επιτελείται στο DBRS αφορά μόνο στα αξιώματα ισοδυναμίας των κλάσεων, καθώς μόνον αυτά χρειάζεται να αποθηκεύσουμε.

Ο λόγος για τον οποίο το DBRS επιτελεί τη διαδικασία διεύρυνσης του TBox είναι επειδή μέσω των νέων διευρυμένων σχέσεων μπορεί να ταυτίσει αποθηκευμένες στη βάση δεδομένων κλάσεις με άλλες αυθαίρετες που υποβάλει ο χρήστης κατά τη διεξαγωγή ερωτημάτων και να επιταχύνει έτσι την αποτίμησή τους.

Η λογική του αλγορίθμου είναι σχετικά απλή. Αρχικά, στο πεδίο “concept_definition” κάθε εγγραφής στον πίνακα «Concepts» υπάρχει ο ορισμός της κλάσης σε μορφή αλφαριθμητικού, όπως ανασύρθηκε από το αρχείο της οντολογίας. Μόνο το αντίστοιχο πεδίο των πρωταρχικών κλάσεων είναι κενό. Η διαδικασία είναι επαναληπτική. Σε κάθε επανάληψη το όνομα κάθε κλάσης που συναντάται σε κάποιον ορισμό αντικαθίσταται από τον ορισμό της. Οι επαναλήψεις συνεχίζονται έως ότου δεν γίνει καμία αντικατάσταση, που σημαίνει ότι οι ορισμοί των κλάσεων

περιέχουν πλέον μόνον πρωταρχικές κλάσεις. Ο διευρυμένος ορισμός κάθε κλάσης, κανονικοποιείται και αποθηκεύεται στο πεδίο «expanded_concept_definition» της αντίστοιχης εγγραφής στον πίνακα «Concepts».

Είναι προφανές ότι κανονικά η διαδικασία αυτή εφαρμόζεται μόνον σε ακυκλικά TBoxes, δηλαδή σε TBoxes όπου το όνομα μιας κλάσης δεν περιέχεται ποτέ στον ορισμό της. Ωστόσο, οι δοκιμές μας έδειξαν ότι ο Pellet έχει τη δυνατότητα να χειριστεί περιπτώσεις κυκλικών TBoxes και, για αυτόν τον λόγο, επεκτείναμε τον αλγόριθμο ώστε να αναγνωρίζει κύκλους σε έναν ορισμό και να σταματάει έγκαιρα την διεύρυνσή του.

Πριν από την έναρξη της διαδικασίας διεύρυνσης, εντοπίζονται όλα τα σύνολα ισοδύναμων κλάσεων και, για κάθε κλάση σε καθένα από αυτά, ο ορισμός της τίθεται ίσος με την τομή των ορισμών όλων των κλάσεών που το συγκεκριμένο σύνολο περιέχει.

Αλγόριθμος

Σημειώνουμε εδώ ότι στα παρακάτω δεν λαμβάνονται υπ' όψιν οι ανώνυμες κλάσεις, οι OWLThing και OWLNothing και οι ισοδύναμες αυτών.

```
createExpandedTBox():

Νέα δομή Map1 που εκφράζει αντιστοίχιση id κλάσης με ορισμό;

Αποστολή SQL ερωτήματος στο DBMS και φόρτωση της Map1 με ένα
ζεύγος id κλάσης και του ορισμού της από κάθε σύνολο
ισοδύναμων κλάσεων της οντολογίας;

Για κάθε id κλάσης Class1 της Map1
{
   Νέα δομή Map2 που εκφράζει αντιστοίχιση id κλάσης με τον
   ορισμό της;

   Αποστολή SQL ερωτήματος και φόρτωση της Map2 με τα id και
   τους ορισμούς των ισοδύναμων κλάσεων της Class1,
   συμπεριλαμβανομένης της Class1, εξαιρώντας τις κλάσεις που
   έχουν ορισμό null;

   Ένωσε τους ορισμούς όλων των κλάσεων της Map2 σε μια And
   έκφραση τομής newDefinition;

   Κανονικοποίησε το newDefinition;
```

Αποστολή SQL ερωτήματος που θέτει το πεδίο "expanded_concept_definition" στον πίνακα "Concepts" όλων των ισοδύναμων κλάσεων της Class1 (συμπεριλαμβανομένης αυτής) ίσο με newDefinition;

}

Νέες δομές Map3, Map4 και Map5 που εκφράζουν αντιστοίχιση id κλάσης με το όνομα της κλάσης και ορισμό;

Αποστολή SQL ερωτήματος και φόρτωση της Map3 με τα ids των κλάσεων, τα ονόματά τους και την τιμή του πεδίου τους "concept_definition" ή την τιμή του πεδίου τους "expanded_concept_definition", αν αυτό δεν είναι κενό;

Αποστολή SQL ερωτήματος και φόρτωση της Map4 με τα ids των κλάσεων, τα ονόματά τους και την τιμή του πεδίου τους "concept_definition" ή την τιμή του πεδίου τους "expanded_concept_definition", αν αυτό δεν είναι κενό, κρατώντας μια κλάση από κάθε σύνολο ισοδύναμων κλάσεων;

Φόρτωση της Map5 με τα περιεχόμενα της Map4;

Νέα δυαδική μεταβλητή changed;

Επανέλαβε

{

changed:=false;

Για κάθε κλάση Class1, με όνομα name1 και ορισμό definition1 από τη Map3

{

Όπου βρεις name1 στον ορισμό definition1, αντικατέστησέ το με #name1;

Για κάθε κλάση Class2, με όνομα name2 και ορισμό definition2 από τη Map4

{

Νέο αλφαριθμητικό newDefinition;

Όπου βρεις name1 στον ορισμό definition2, αντικατέστησέ το με definition1. Δώσε το τελικό αποτέλεσμα στο newDefinition;

Αν newDefinition<>definition2

{

changed:=true;

```
                Βάλε στη Map5 τον newDefinition στη θέση του
                definition2 για την Class2;
            }
        }
        Αν changed=true
        {
            Αντικατέστησε όλες τις τιμές ορισμών του Map4 με
            τις νέες τιμές ορισμών του Map5;

            Βγες από τον βρόχο;
        }
    }
}Όσο ισχύει changed=true;

Για κάθε κλάση Class και τον ορισμό της definition από την
Map4
{
    Κανονικοποίησε το definition;

    Θέσε το πεδίο "expanded_concept_definition" της κλάσης
    Class ίσο με definition;
}
```

6.1.3 Διάδοση χαρακτηριστικών ρόλων

Αφού έχουν φορτωθεί στην βάση δεδομένων οι ρόλοι της οντολογίας (και η ιεραρχία τους), τα δηλωμένα χαρακτηριστικά τους και οι σχέσεις ισοδυναμίας και ασυμβατότητας μεταξύ τους, επιτελείται μια διαδικασία διάδοσης των αυτών χαρακτηριστικών. Η διαδικασία αυτή γίνεται με άξονα τις σχέσεις μεταξύ των ρόλων. Οι σχέσεις αυτές μπορεί να είναι είτε σχέσεις ισοδυναμίας είτε σχέσεις αντιστροφής είτε ιεραρχικές σχέσεις.

Στον Πίνακα 6.1 φαίνονται τα χαρακτηριστικά των ρόλων, για εκφραστικότητα OWL-DL, όπως αυτά έχουν κωδικοποιηθεί από το DBRS. Σε κάθε χαρακτηριστικό ή συνδυασμό χαρακτηριστικών δίνεται ένας μοναδικός κωδικός και κάθε ρόλος της οντολογίας έχει πάντα έναν από αυτούς τους κωδικούς αποθηκευμένο στο αντίστοιχο πεδίο του πίνακα «Roles». Όσο μεγαλύτερος είναι ο κωδικός αυτός σε απόλυτη τιμή, τόσο ισχυρότερο είναι το χαρακτηριστικό(ά). Για παράδειγμα, αν ένας ρόλος Α έχει κωδικό 4 και έχει δηλωθεί ως ισοδύναμος με έναν ρόλο Β που έχει κωδικό 2, τότε ο κωδικός του Β θα πρέπει να ενημερωθεί στην τιμή 4.

Χαρακτηριστικό / Συνδυασμός Χαρακτηριστικών	Κωδικός
Απλός Ρόλος Αντικειμένου (Object)	0
Μεταβατικός (Transitive)	1
Αντίστροφος (Inverse)	2
Λειτουργικός (Functional)	3
Συμμετρικός (Symmetric)	4
Αντίστροφος & Μεταβατικός	5
Αντίστροφος & Λειτουργικός	6
Μεταβατικός & Συμμετρικός	7
Αντίστροφος & Συμμετρικός	8
Αντίστροφος & Συμμετρικός & Μεταβατικός	9
Τύπου Δεδομένων (Datatype)	10
Τύπου Δεδομένων & Λειτουργικός	11

Πίνακας 6.1: Κωδικοποίηση χαρακτηριστικών ρόλων

Όπως προαναφέρθηκε, η διάδοση των χαρακτηριστικών των ρόλων γίνεται βάσει των σχέσεων μεταξύ τους. Η διαδικασία επιτελείται σε τρία βήματα:

1. Εντοπίζονται όλα τα σύνολα αντιστρόφων ρόλων της οντολογίας και σε όλους τους ρόλους κάθε συνόλου δίνεται ως κωδικός χαρακτηριστικών ο μεγαλύτερος μεταξύ των αρχικών. Αυτό γίνεται με τη λογική ότι δυο αντίστροφοι μεταξύ τους ρόλοι, πρέπει κατά τ' άλλα να έχουν ακριβώς τα ίδια χαρακτηριστικά.
2. Εντοπίζονται όλα τα σύνολα ισοδύναμων ρόλων της οντολογίας και σε όλους τους ρόλους κάθε συνόλου δίνεται ως κωδικός χαρακτηριστικών ο μεγαλύτερος μεταξύ των αρχικών. Αυτό γίνεται με τη λογική ότι δυο ισοδύναμοι μεταξύ τους ρόλοι, πρέπει να έχουν ακριβώς τα ίδια χαρακτηριστικά.
3. Όλοι οι υπορόλοι των λειτουργικών ρόλων ορίζονται ως λειτουργικοί.

Αλγόριθμος

Σημειώνουμε εδώ ότι όπου παρακάτω γράφουμε ΚΧ εννοούμε Κωδικός Χαρακτηριστικών.

```
updateRolesCharacteristics():
Κλήση της setInverseRoleCharacteristics(){
      Για κάθε ζεύγος ids ρόλων του πίνακα "Inversion"
      [role1,role2]
      {
         Αν ΚΧ role1 < ΚΧ role2
         {
            ΚΧ role1 := ΚΧ role2 ;
            Για κάθε ρόλο R ισοδύναμο του role1
            {
                ΚΧ R := ΚΧ role2 ;
            }
         }
         Διαφορετικά αν ΚΧ role1 > ΚΧ role2
         {
            ΚΧ role2 := ΚΧ role1 ;
            Για κάθε ρόλο R ισοδύναμο του role2
            {
                ΚΧ R := ΚΧ role1 ;
            }
         }
      }
}
Κλήση της setEquivalentRoleCharacteristics(){
      Σε όλους του ρόλους κάθε συνόλου ισοδύναμων ρόλων δίνεται
      ο μεγαλύτερος ΚΧ μεταξύ τους. Αυτό γίνεται με ένα SQL
      ερώτημα:
      update "Roles" as "Q"
      set role_characteristics="Z".role_characteristics
      from (select
      "S".role_characteristics,"S".r_label_pre,"S".r_label_post
                  from "Roles" as "S","Roles" as "T"
            where "S".r_label_pre="T".r_label_pre  and
            "S".r_label_post="T".r_label_post and
            "S".id_role<>"T".id_role) as "Z"
            where ("Q".r_label_pre="Z".r_label_pre and
      "Q".r_label_post="Z".r_label_post and
      "Q".role_characteristics<"Z".role_characteristics);
}
Κλήση της propagateRoleFunctionality(){
      Νέο σύνολο functionalRoles;
```

```
        Προστίθενται στο functionalRoles τα ids όλων των γνωστών
        λειτουργικών ρόλων;
        Για κάθε ρόλο R του συνόλου functionalRoles
        {
            Για κάθε υπορόλο R' του R
            {
                Αν KX R'=2
                {
                    KX R':=6;
                }
                Διαφορετικά αν KX R'=10
                {
                    KX R'=11;
                }
                Διαφορετικά
                {
                    KX R'=3;
                }
            }
        }
}
```

6.1.4 Αλγόριθμοι αποτίμησης ερωτημάτων

Παρακάτω περιγράφονται αναλυτικά οι σημαντικότεροι από τους αλγόριθμους που χρησιμοποιεί το DBRS για την αποτίμηση των ερωτημάτων που υποβάλλει ο χρήστης. Χρησιμοποιούνται συμβολισμοί της ψευδογλώσσας σύνταξης ερωτημάτων που περιγράφεται στην Παράγραφο 6.4.2. Οι αλγόριθμοι αφορούν την περίπτωση που το σύστημα λειτουργεί σε eager mode. Για κάποιους από τους παρακάτω αλγόριθμους δεν δίνεται ψευδοκώδικας, αλλά μια συστηματική παρουσίαση των λειτουργιών τους και στη θέση του ψευδοκώδικα δίνονται αναλυτικά διαδικαστικά διαγράμματα ροής.

6.1.4.1 Εύρεση δηλώσεων ρόλου αντικειμένου

Η εύρεση δηλώσεων ρόλων αντικειμένου που δεν είναι αποθηκευμένες στον πίνακα «Role Assertions» της βάσης δεδομένων του DBRS γίνεται με κλήση της μεθόδου returnObjectRoleAssertions() της κλάσης DBMSWrapper. Η μέθοδος αυτή δέχεται ως όρισμα το id ενός ρόλου της οντολογίας και επιστρέφει ένα σύνολο δισδιάστατων πινάκων που περιέχει τις δηλώσεις του ρόλου ως ζεύγη ids ατόμων (υποκείμενο-αντικείμενο). Εκμεταλλεύεται την ήδη αποθηκευμένη πληροφορία

σχετικά με την ισοδυναμία ρόλων, τους αντίστροφους ρόλους, τους συμμετρικούς ρόλους, την ιεραρχία ρόλων και την ισοδυναμία ατόμων. Πιο συγκεκριμένα μαζί με τις ρητές δηλώσεις του δεδομένου ρόλου επιστρέφονται και:

- Δηλώσεις των ισοδύναμων ρόλων του δεδομένου ρόλου.
- Δηλώσεις των υπορόλων του δεδομένου ρόλου.
- Δηλώσεις των αντιστρόφων ρόλων του ρόλου και των ισοδύναμων αυτών. Αυτές οι δηλώσεις επιστρέφονται βέβαια ανεστραμμένες.
- Δηλώσεις των αντιστρόφων ρόλων των υπορόλων του δεδομένου ρόλου. Επίσης επιστρέφονται ανεστραμμένες
- Αν κάποιος ρόλος είναι μεταβατικός, τότε οι δηλώσεις του υπολογίζονται βάσει της ήδη αποθηκευμένου σχήματος ετικετών των δηλώσεών του.

Ο αλγόριθμος που περιγράφουμε στη συνέχεια δεν είναι πλήρης (complete), με την έννοια ότι δεν επιστρέφει όλες τις δηλώσεις ενός ρόλου αντικειμένου της οντολογίας. Αποτελεί μια περίπτωση «ελαφριάς» συλλογιστικής ανάλυσης και χρησιμοποιείται από το DBRS για να ελαχιστοποιηθεί η πληροφορία που αποθηκεύεται στη βάση δεδομένων. Η πλήρης απάντηση σε ένα ερώτημα PropertyValue ενδεχομένως να χρειάζεται το συνδυασμό του αλγορίθμου αυτού με κάποια tableau διαδικασία που επιτελείται από τον Reasoner. Ο αλγόριθμος αυτός χρησιμοποιείται αποκλειστικά από τον αλγόριθμο «Αποτίμηση ερωτήματος PropertyValue» που περιγράφεται στην Παράγραφο 6.1.4.6, για την έυρεση των αποθηκευμένων στη βάση δεδομένων δηλώσεων κάποιου ρόλου αντικειμένου.

Αλγόριθμος

```
returnObjectRoleAsserions(int idRole):

Νέο σύνολο από δισδιάστατους πίνακες με όνομα assertions ;
Νέο σύνολο με όνομα roles ;
Πρόσθεσε στο roles το idRole ;
Πρόσθεσε στο σύνολο roles τα ids των ισοδύναμων ρόλων του R ;
Πρόσθεσε στο σύνολο roles τα ids των μη μεταβατικών υπορόλων
του R ;
```

```
Για κάθε σύνολο ισοδύναμων μεταβατικών υπορόλων του R
{
  Επέλεξε έναν και πρόσθεσε το id του στο σύνολο roles ;
}

Για κάθε ρόλο R΄ του συνόλου roles
{
  Αν R΄ μεταβατικός
  {
    Νέο σύνολο με όνομα individuals ;

    Πρόσθεσε στο individuals τα ids όλων των successors του
    ατόμου _TOP_ στην ιεραρχία δηλώσεων του R΄ (δηλαδή όλα τα
    άτομα της ιεραρχίας) με κλήση της
    returnObjectSuccessorsForTransitiveRole( R΄ , _TOP_ );

    Για κάθε άτομο subject του συνόλου individuals
    {
      Κλήση της returnObjectSuccessorsForTransitiveRole(R΄,
      subject) και για κάθε άτομο object, successor του ατόμου
      subject στην ιεραρχία δηλώσεων του R΄
      {
        Πρόσθεσε στο assertions τη δήλωση [subject,object];

        Αν R΄ συμμετρικός
        {
           Πρόσθεσε στο assertions τη δήλωση [object,subject];
        }
      }
    }
    Για κάθε ισοδύναμο ρόλο του R΄, συμπεριλαμβανομένου του R΄
    {
       Κάλεσε την returnInferredTransitiveRoleAssertions(R')
       και πρόσθεσε όλες τις επιπλέον δηλώσεις στο σύνολο
       assertions (Η μέθοδος αυτή επιστρέφει και ανάστροφες
       τις δηλώσεις αν ο R΄ είναι συμμετρικός);
    }
  }
  Διαφορετικά
  {
    Για κάθε δήλωση [a,b] του R΄ στον πίνακα "Role Assertions"
    της βάσης δεδομένων
    {
      Νέο σύνολο subjects;
      Νέο σύνολο objects;
      Πρόσθεσε στο subjects το id του a και των ισοδύναμών
      του;
```

```
      Πρόσθεσε στο objects το id του b και των ισοδύναμων του;

      Για κάθε άτομο subject του συνόλου subjects
      {
        Για κάθε άτομο object του συνόλου objects
        {
          Πρόσθεσε στο assertions τη δήλωση [subject,object];

          Αν R' συμμετρικός
          {
            Πρόσθεσε στο assertions τη δήλωση
            [object,subject];
          }
        }
      }
    }
    Για κάθε ρόλο Ri, αντίστροφο του R'
    {
       Για κάθε δήλωση [a,b] του Ri στον πίνακα "Role
       Assertions" της βάσης δεδομένων
       {
         Ομοίως με ακριβώς παραπάνω, με τις δηλώσεις να
         εισάγονται ανάστροφες στο σύνολο assertions.
       }
    }
  }
}
Επέστρεψε το σύνολο assertions ;
```

6.1.4.2 Εύρεση δηλώσεων κλάσης

Το ερώτημα Type(? , C), όπου C κάποια γνωστή κλάση της οντολογίας, αποτελεί ένα από τα βασικότερα ερωτήματα που μπορούν να τεθούν και, για την αποτίμησή του, η εκμετάλλευση όσο το δυνατόν περισσότερης, από την αποθηκευμένη στη βάση δεδομένων, πληροφορίας αποτελεί ένα πολύ ενδιαφέρον πρόβλημα. Το έργο αυτό αναλαμβάνει η μέθοδος returnConceptIndividuals() που δέχεται ως όρισμα το id μιας κλάσης και επιστρέφει ένα σύνολο με ids ατόμων που ανήκουν σε αυτήν. Η μέθοδος αυτή εκμεταλλεύεται την ήδη αποθηκευμένη πληροφορία σχετικά με την ισοδυναμία και ιεραρχία κλάσεων, την ισοδυναμία και ιεραρχία ρόλων, το πεδίο, το εύρος ρόλων και την ισοδυναμία ατόμων. Χρησιμοποιείται αποκλειστικά από τον αλγόριθμο «Αποτίμηση ερωτήματος Type» που περιγράφεται στην Παράγραφο

6.1.4.5 για την εύρεση των αποθηκευμένων στη βάση δεδομένων δηλώσεων μιας γνωστής κλάσης. Πιο συγκεκριμένα η μέθοδος:

- Επιστρέφει ρητές δηλώσεις της δεδομένης κλάσης και των ισοδύναμών της από τον πίνακα «Concept Assertions» , προσθέτοντας στα άτομα του αποτελέσματος και τα ισοδύναμα αυτών.
- Βρίσκει τις δηλώσεις των ρόλων που έχουν δηλωμένο ως πεδίο ή εύρος τους (στους πίνακες «Roles Domain» και «Roles Range») την δεδομένη κλάση και επιστρέφει το υποκείμενο ή το αντικείμενο της δήλωσης αντιστοίχως. Εδώ λαμβάνεται υπ' όψιν και πληροφορία περί αντίστροφων ρόλων.

Ο αλγόριθμος που περιγράφουμε στη συνέχεια δεν είναι πλήρης (complete), με την έννοια ότι δεν επιστρέφει όλες τις δηλώσεις μιας κλάσης της οντολογίας. Αποτελεί μια περίπτωση «ελαφριάς» συλλογιστικής ανάλυσης και χρησιμοποιείται από το DBRS για να ελαχιστοποιηθεί η πληροφορία που αποθηκεύεται στη βάση δεδομένων. Η πλήρης απάντηση σε ένα ερώτημα Type ενδεχομένως να χρειάζεται το συνδυασμό του αλγορίθμου αυτού με κάποια tableau διαδικασία που επιτελείται από τον Reasoner.

Αλγόριθμος

```
returnConceptIndividuals(int idConcept):

Νέο σύνολο individualIds που θα επιστραφεί στο τέλος;
Νέο σύνολο concepts;
Πρόσθεσε στο σύνολο concepts το idConcept;
Πρόσθεσε στο σύνολο concepts τα ids των υποκλάσεων της
idConcept;
Νέο σύνολο haveConceptDomain;
Νέο σύνολο haveConceptRange;

Για κάθε κλάση C του συνόλου concepts, λαμβάνοντας μια κλάση
από κάθε σύνολο ισοδύναμων κλάσεων που αυτό περιέχει για
λόγους απόδοσης
{
   Κάλεσε την selectRolesByDomain(C) και πρόσθεσε τους ρόλους
   που αυτή επιστρέφει στο σύνολο haveConceptDomain;
```

```
   Κάλεσε την selectRolesByRange(C) και πρόσθεσε τους ρόλους
   που αυτή επιστρέφει στο σύνολο haveConceptRange;
}

Για κάθε ρόλο R του συνόλου haveConceptDomain
{
   Πρόσθεσε στο σύνολο haveConceptDomain τους υπορόλους του
   ρόλου R;
}

Για κάθε ρόλο R του συνόλου haveConceptRange
{
   Πρόσθεσε στο σύνολο haveConceptRange τους υπορόλους του
   ρόλου R;
}

Για κάθε ρόλο R του συνόλου haveConceptDomain
{
   Πρόσθεσε στο σύνολο haveConceptRange έναν αντίστροφο του
   ρόλου R (αν έχει);
}

Για κάθε ρόλο R του συνόλου haveConceptRange
{
   Πρόσθεσε στο σύνολο haveConceptDomain έναν αντίστροφο του
   ρόλου R (αν έχει);
}

Για κάθε ρόλο R του συνόλου haveConceptDomain, λαμβάνοντας
μόνο έναν ρόλο από κάθε σύνολο ισοδύναμων ρόλων που αυτό
περιέχει για λόγους απόδοσης
{
   Κάλεσε την returnSubjects(R) και πρόσθεσε τα άτομα που αυτή
   επιστρέφει στο σύνολο individuals;
}

Για κάθε ρόλο R του συνόλου haveConceptRange, λαμβάνοντας μόνο
έναν ρόλο από κάθε σύνολο ισοδύναμων ρόλων που αυτό περιέχει
για λόγους απόδοσης
{
   Κάλεσε την returnObjectsOfObjectRole(R) και πρόσθεσε τα
   άτομα που αυτή επιστρέφει στο σύνολο individuals;
}

Επέστρεψε το σύνολο individuals;
```

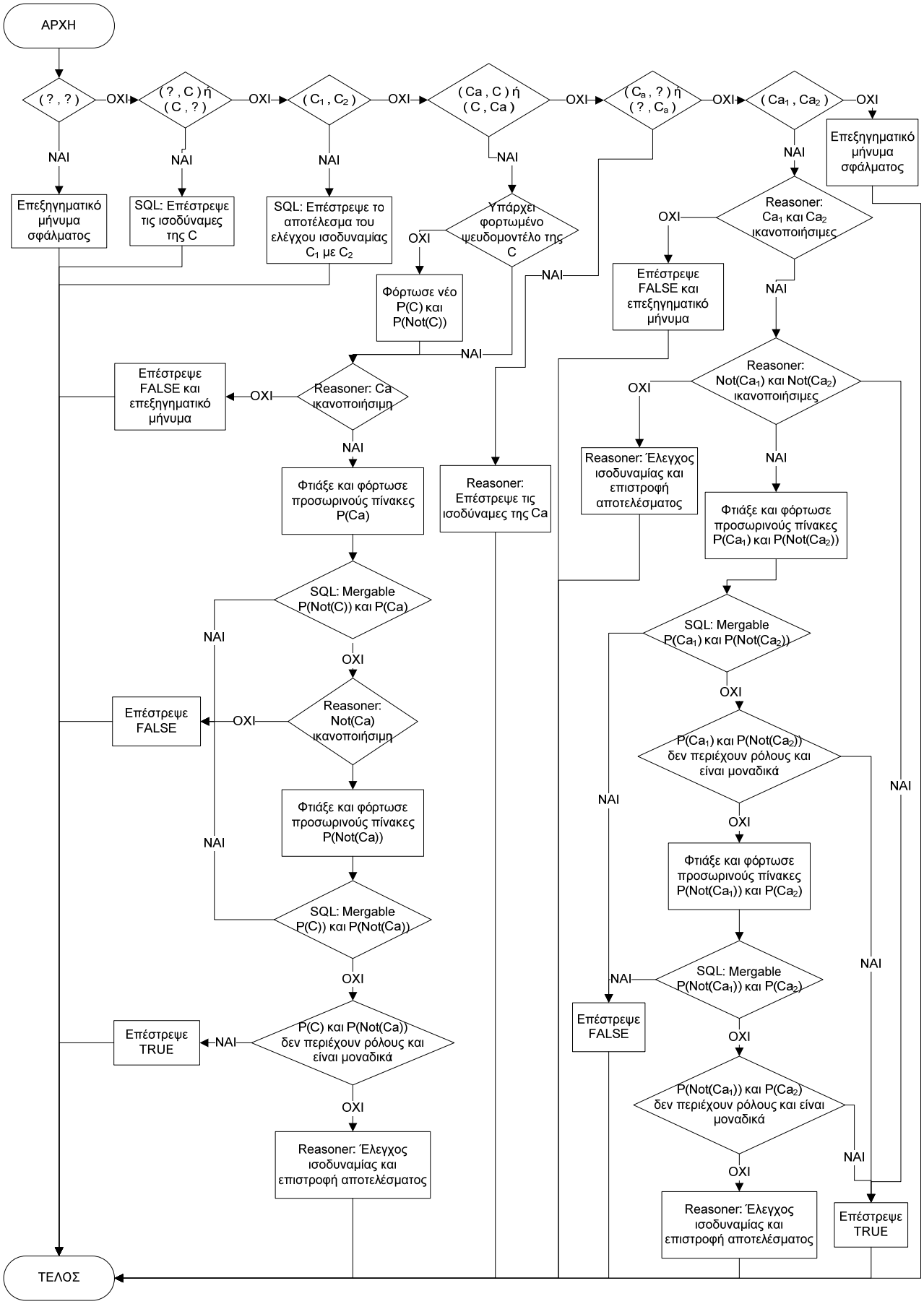
6.1.4.3 Αποτίμηση ερωτήματος EquivalentClass

- Στην περίπτωση που στα ορίσματα περιέχεται μια αυθαίρετη κλάση C_a και μια γνωστή κλάση C, λαμβάνονται (αν δεν υπάρχει ήδη ψευδομοντέλο της C αποθηκευμένο στη βάση δεδομένων) από τον Reasoner τα ψευδομοντέλα της C και της συμπληρωματικής της και αποθηκεύονται στη βάση δεδομένων. Κατ' αντιστοιχία, αποθηκεύονται προσωρινά στη βάση δεδομένων τα ψευδομοντέλα της C_a και της συμπληρωματικής της. Τα τελευταία λαμβάνονται επίσης από τον Reasoner, αφού προηγουμένως έχει ελεγχθεί η ικανοποιησιμότητα της C_a[40].

  Στη συνέχεια, εκτελούνται τα mergable tests μεταξύ C και Not(Ca) και μεταξύ Not(C) και C_a. Αν οποιοδήποτε από τα δύο απαντήσει true, τότε οι C και C_a δεν είναι ισοδύναμες. Αν τουλάχιστον ένα απαντήσει false, ελέγχεται αν το ζεύγος των ψευδομοντέλων είναι μοναδικό και δεν περιέχει ρόλους. Σε αυτήν την περίπτωση, οι C και C_a είναι ισοδύναμες. Σε κάθε άλλη περίπτωση, το ερώτημα στέλνεται στον Reasoner και τα προσωρινώς αποθηκευμένα ψευδομοντέλα σβήνονται.
- Στην περίπτωση που στα ορίσματα περιέχονται δύο αυθαίρετες κλάσεις, ακολουθείται ακριβώς όμοια διαδικασία με παραπάνω, με τη διαφορά ότι τώρα όλα τα ψευδομοντέλα (και των τεσσάρων εννοιακών εκφράσεων) αποθηκεύονται προσωρινά στη βάση δεδομένων.
- Στην περίπτωση που το ένα όρισμα είναι αυθαίρετη κλάση και το άλλο ερωτηματικό (?), το ερώτημα στέλνεται απ' ευθείας στον Reasoner, αφού πρώτα ελεγχθεί η ικανοποιησιμότητα της έκφρασης για την ενδεχόμενη επιστροφή κατάλληλου επεξηγηματικού μηνύματος.

---

[40] Για την ακρίβεια, ο έλεγχος ικανοποιησιμότητας της C_a ισοδυναμεί με την κατασκευή ενός ψευδομοντέλου γι' αυτήν. Αν προκύψει ότι δεν υπάρχει ψευδομοντέλο και άρα η C_a δεν είναι ικανοποιήσιμη, τότε επιστρέφεται στον χρήστη επεξηγηματικό μήνυμα.

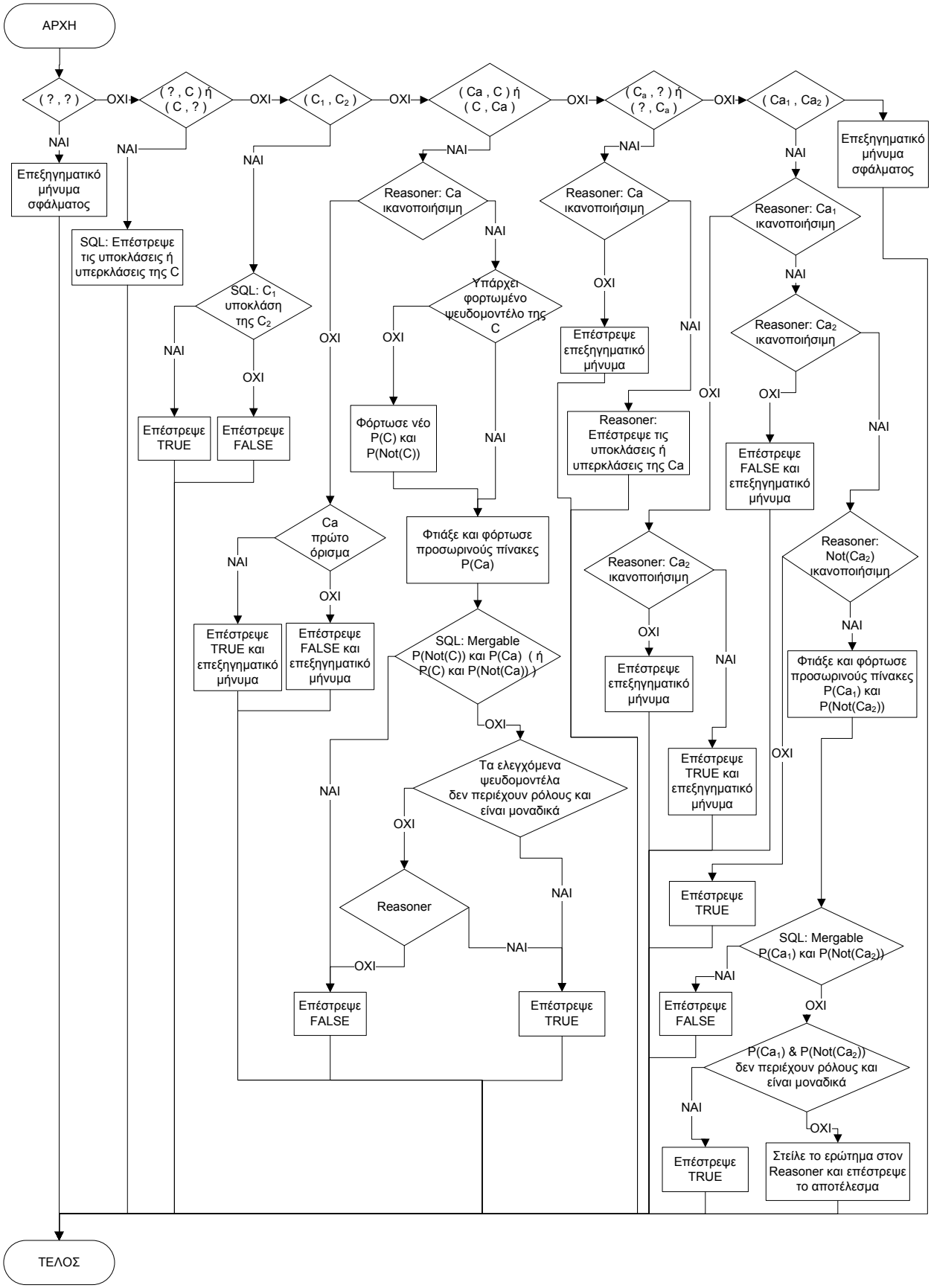


Σχήμα 6.1: Διάγραμμα ροής του αλγορίθμου αποτίμησης ερωτήματος EquivalentClass

6.1.4.4 Αποτίμηση ερωτήματος SubClassOf

- Στην περίπτωση όπου στα ορίσματα περιέχεται μια αυθαίρετη κλάση C_a και μια γνωστή κλάση C, λαμβάνονται (αν δεν υπάρχει ήδη ψευδομοντέλο της C αποθηκευμένο στη βάση δεδομένων) από τον Reasoner τα ψευδομοντέλα της C και της συμπληρωματικής της και αποθηκεύονται. Ομοίως, αποθηκεύονται προσωρινά τα ψευδομοντέλα της C_a και της συμπληρωματικής της. Τα τελευταία λαμβάνονται επίσης από τον Reasoner, αφού προηγουμένως έχει ελεγχθεί η ικανοποιησιμότητα της C_a.

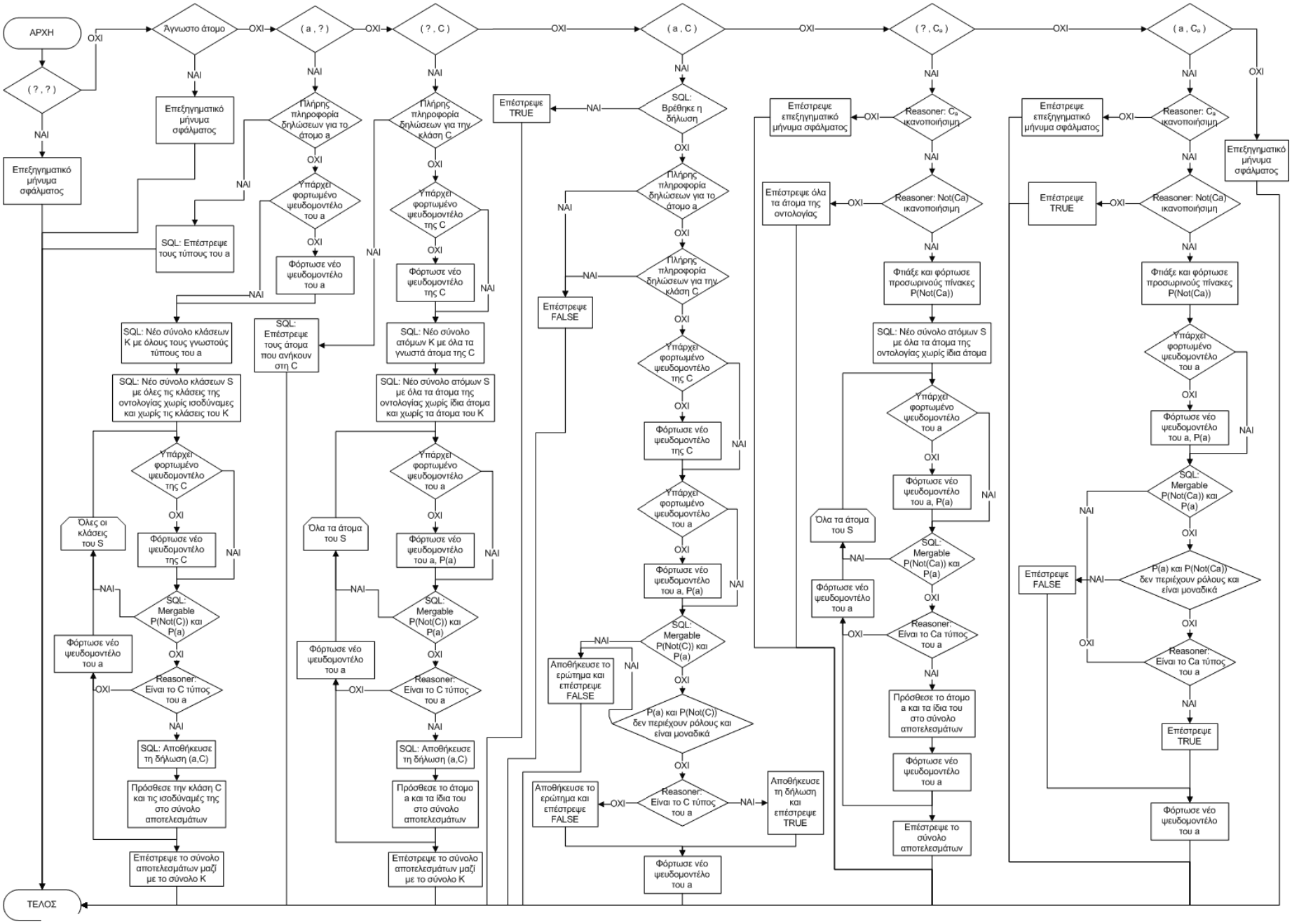
  Στη συνέχεια, εκτελείται το mergable test είτε μεταξύ C και Not(C_a), είτε μεταξύ Not(C) και C_a (ανάλογα με την σειρά των ορισμάτων) και αν απαντήσει true, το ερώτημα αποτιμάται ως false. Αν απαντήσει false, ελέγχεται αν τα ψευδομοντέλα των κλάσεων μεταξύ των οποίων έγινε το mergable test είναι μοναδικά και δεν περιέχουν ρόλους. Σε αυτήν την περίπτωση, το ερώτημα αποτιμάται ως false. Σε κάθε άλλη περίπτωση, το ερώτημα στέλνεται στον Reasoner και, με το τέλος της διαδικασίας αποτίμησης, σβήνονται τα προαναφερθέντα προσωρινώς αποθηκευμένα ψευδομοντέλα.
- Στην περίπτωση όπου στα ορίσματα περιέχονται δύο αυθαίρετες κλάσεις ακολουθείται ακριβώς όμοια διαδικασία με παραπάνω, με τη διαφορά ότι τώρα όλα τα ψευδομοντέλα (και των τεσσάρων εννοιακών εκφράσεων) αποθηκεύονται προσωρινά και σβήνονται με το τέλος της όλης διαδικασίας αποτίμησης.
- Στην περίπτωση που το ένα όρισμα είναι αυθαίρετη κλάση και το άλλο ερωτηματικό (?), το ερώτημα στέλνεται για αποτίμηση απ' ευθείας στον Reasoner, αφού πρώτα ελεγχθεί η ικανοποιησιμότητα της έκφρασης για την ενδεχόμενη επιστροφή κατάλληλου επεξηγηματικού μηνύματος.



Σχήμα 6.2: Διάγραμμα ροής του αλγόριθμου αποτίμησης ερωτήματος SubClassOf

6.1.4.5 Αποτίμηση ερωτήματος Type

- Ορίζεται ένα δυαδικό (boolean) πεδίο για κάθε αποθηκευμένη στη βάση δεδομένων κλάση το οποίο, αν είναι true, σημαίνει ότι η αποθηκευμένη πληροφορία σχετικά με τις δηλώσεις της κλάσης που αντιστοιχεί στη συγκεκριμένη εγγραφή είναι πλήρης. Οπότε, αν αυτό το πεδίο για μια κλάση C έχει τιμή true, τα ερωτήματα τύπου Type(?,C) και Type(a,C), όπου a άτομο της οντολογίας, απαντώνται αποκλειστικά με χρήση της βάσης δεδομένων. Αντιθέτως, αν το πεδίο έχει την τιμή false, τότε στην πρώτη περίπτωση θα επιστρατευθεί σίγουρα ο Reasoner, ενώ στη δεύτερη θα επιστρατευθεί μόνο αν, μέσω του αλγορίθμου «ελαφριάς» συλλογιστικής ανάλυσης που περιγράφουμε στην Παράγραφο 6.1.4.2, δεν βρεθεί η πληροφορία στη βάση δεδομένων.
- Ορίζεται ένα δυαδικό πεδίο για κάθε αποθηκευμένο στη βάση δεδομένων άτομο το οποίο, αν είναι true, σημαίνει ότι η αποθηκευμένη πληροφορία σχετικά με τους τύπους του ατόμου που αντιστοιχεί στη συγκεκριμένη εγγραφή είναι πλήρης. Οπότε, αν αυτό το πεδίο για ένα άτομο έχει τιμή true, τα ερωτήματα τύπου Type(a,?) και Type(a,C), όπου C γνωστή κλάση της οντολογίας, απαντώνται αποκλειστικά με χρήση της βάσης δεδομένων. Αντιθέτως, αν το πεδίο έχει την τιμή false, τότε στην πρώτη περίπτωση θα επιστρατευθεί σίγουρα ο Reasoner, ενώ στη δεύτερη θα επιστρατευθεί μόνο αν, μέσω του αλγορίθομου «ελαφριάς» συλλογιστικής ανάλυσης που περιγράφουμε στην Παράγραφο 6.1.4.2 , δεν βρεθεί η πληροφορία στη βάση δεδομένων.
- Αν η πληροφορία δηλώσεων για κάποια κλάση C και η πληροφορία τύπων για κάποιο άτομο a δεν είναι πλήρης και τεθεί ένα ερώτημα τύπου Type(a,C) το οποίο αποτιμάται τελικά από τον Reasoner ως false, τότε το ερώτημα κωδικοποιείται και αποθηκεύεται, ώστε την επόμενη φορά που θα τεθεί (ακριβώς το ίδιο ή κάποιο ισοδύναμό του), να απαντηθεί μόνο με χρήση της βάσης δεδομένων.
- Στην περίπτωση ενός ερωτήματος Type(a,?), αν η πληροφορία τύπων για το άτομο a είναι πλήρης, τότε το ερώτημα θα απαντηθεί με χρήση της βάσης δεδομένων. Σε αντίθετη περίπτωση, επιλέγονται όλες οι κλάσεις της οντολογίας οι οποίες δεν είναι γνωστοί τύποι του a και τοποθετούνται σε ένα



Σχήμα 6.3: Διάγραμμα ροής του αλγορίθμου αποτίμησης ερωτήματος Type

σύνολο, φροντίζοντας να μην επιλεχθούν ισοδύναμες κλάσεις. Στη συνέχεια, αν δεν υπάρχει ήδη αποθηκευμένο κάποιο ψευδομοντέλο του a, λαμβάνεται από τον Reasoner και αποθηκεύεται. Έπειτα, εκτελείται επαναληπτικά το mergable test μεταξύ του ψευδομοντέλου[41] της συμπληρωματικής κάθε κλάσης που επιλέχθηκε προηγουμένως και του ψευδομοντέλου^3 του ατόμου a. Αν το mergable test απαντήσει false, τότε η κλάση πιθανώς να είναι τύπος του a, οπότε καλείται ο Reasoner. Η νέα πληροφορία που προκύπτει αποθηκεύεται στη βάση δεδομένων.

Η διαδικασία αυτή γίνεται για να μειωθεί ο αριθμός των κλάσεων τις οποίες θα χρειαστεί να ελέγξει ο Reasoner σχετικά με το αν είναι τύποι του a, καθώς η αποτίμηση αυτή είναι συνήθως εξαιρετικά απαιτητική και χρονοβόρα.

Στην περίπτωση ενός ερωτήματος Type(?,C), αν η πληροφορία δηλώσεων για την κλάση C είναι πλήρης, τότε το ερώτημα θα απαντηθεί πλήρως με χρήση της βάσης δεδομένων.

Σε αντίθετη περίπτωση, επιλέγονται όλα τα άτομα της οντολογίας τα οποία δεν είναι ήδη γνωστό αν ανήκουν στη C και τοποθετούνται σε ένα σύνολο, φροντίζοντας να μην επιλεχθούν ταυτόσημα άτομα (same individuals). Στη συνέχεια, αν δεν υπάρχει ήδη αποθηκευμένο κάποιο ψευδομοντέλο της C και της συμπληρωματικής της, λαμβάνεται από τον Reasoner και αποθηκεύεται. Έπειτα εκτελείται επαναληπτικά το mergable test μεταξύ του ψευδομοντέλου κάθε ατόμου που επιλέχθηκε προηγουμένως και του ψευδομοντέλου της συμπληρωματικής της C. Αν το mergable test απαντήσει false, τότε το άτομο πιθανώς να ανήκει στην C, οπότε καλείται ο Reasoner. Η νέα πληροφορία που προκύπτει αποθηκεύεται στη βάση δεδομένων.

Η διαδικασία αυτή γίνεται για να μειωθεί ο αριθμός των ατόμων τα οποία θα χρειαστεί να ελέγξει ο Reasoner σχετικά με το αν ανήκουν στην κλάση C, καθώς η αποτίμηση αυτή είναι συνήθως εξαιρετικά απαιτητική και χρονοβόρα.

- Στην περίπτωση ενός ερωτήματος Type(?,Ca) ακολουθείται ακριβώς όμοια διαδικασία με παραπάνω, με τη διαφορά ότι τώρα τα ψευδομοντέλα της

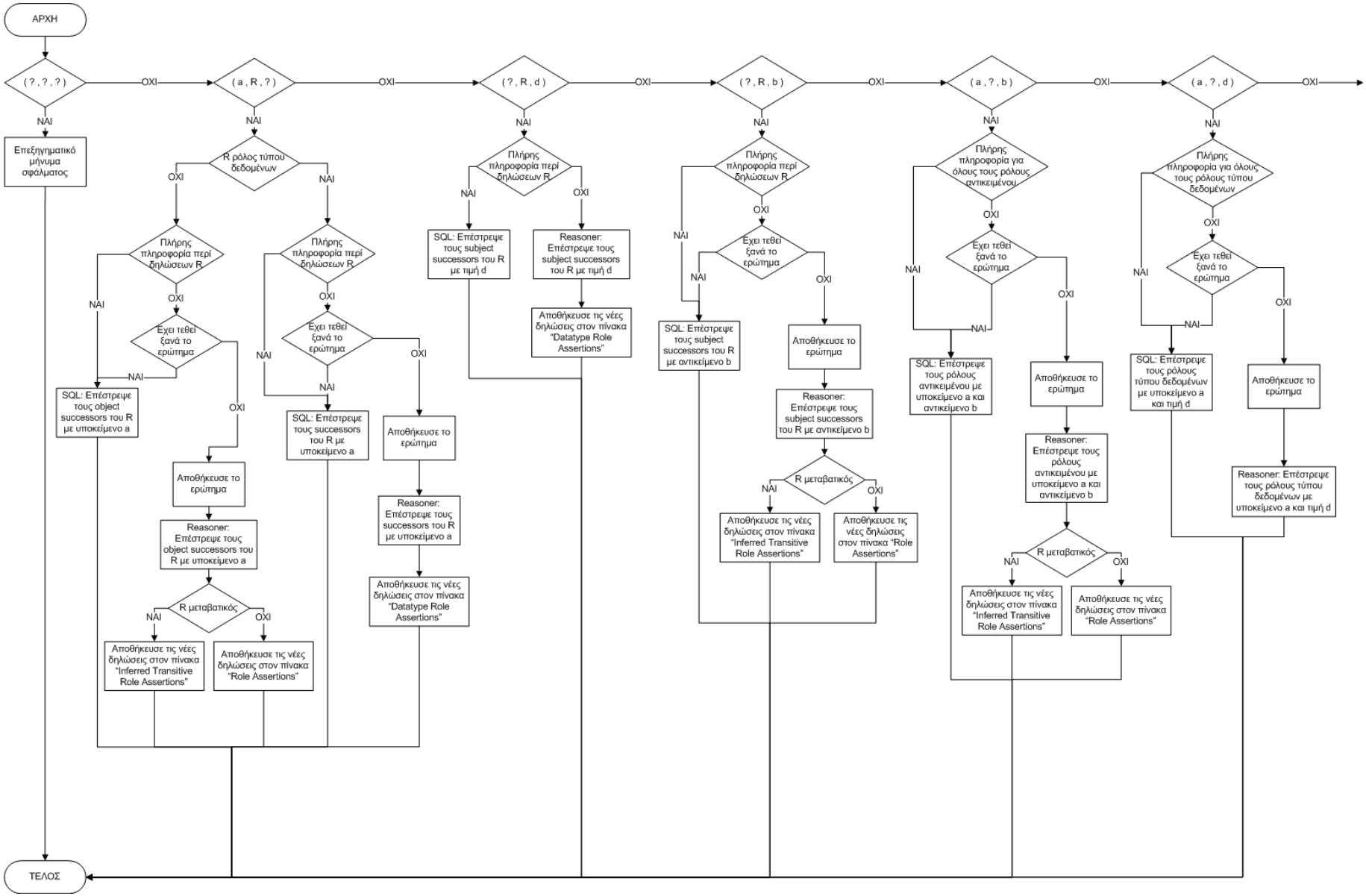
[41] Το οποίο αν δεν υπάρχει ήδη στη βάση ζητείται την στιγμή εκείνη από τον Reasoner και αποθηκεύεται.

αυθαίρετης κλάσης C_a αποθηκεύονται προσωρινά στη βάση δεδομένων και σβήνονται με το τέλος της όλης διαδικασίας. Εδώ, η πληροφορία που προκύπτει δεν αποθηκεύεται.

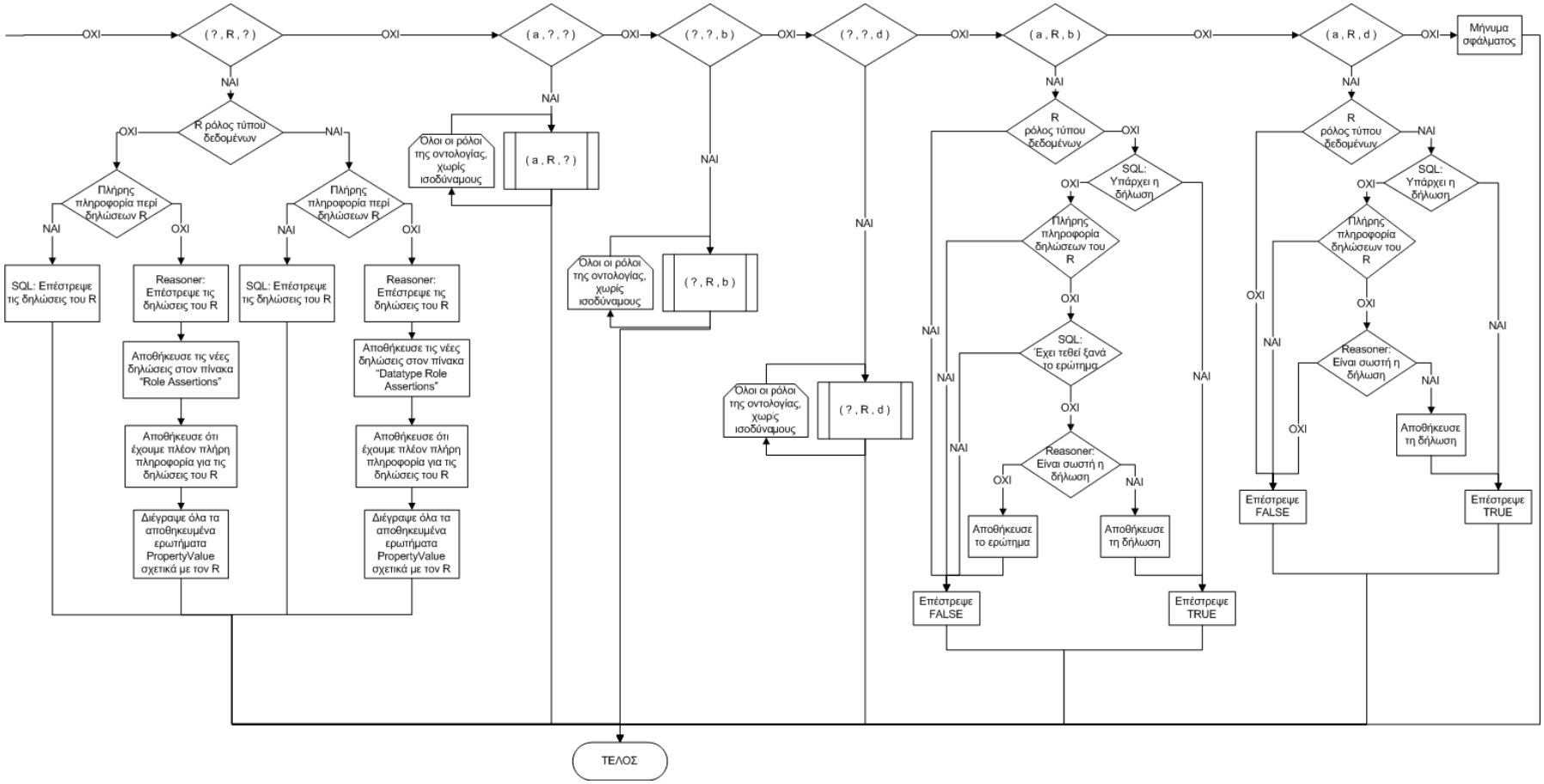
6.1.4.6 Αποτίμηση ερωτήματος PropertyValue

- Ορίζεται ένα δυαδικό (boolean) πεδίο για κάθε αποθηκευμένο στη βάση δεδομένων ρόλο το οποίο, αν έχει την τιμή true, σημαίνει ότι η αποθηκευμένη πληροφορία δηλώσεων του ρόλου στον οποίο αντιστοιχεί η συγκεκριμένη εγγραφή είναι πλήρης. Συνεπώς, αν αυτό το πεδίο έχει την τιμή true για κάποιον ρόλο R, τα ερωτήματα PropertyValue(a,R,b), PropertyValue(?,R,b), PropertyValue(a,R,?) και PropertyValue(?,R,?) απαντώνται αποκλειστικά με χρήση SQL ερωτημάτων, με χρήση του αλγορίθμου της Παραγράφου 6.1.4.1
- False ερωτήματα τύπου PropertyValue(a,R,b) τα οποία στέλνονται στον Reasoner για αποτίμηση (επειδή η αποθηκευμένη πληροφορία δηλώσεων του ρόλου R δεν είναι πλήρης), κωδικοποιούνται και αποθηκεύονται ώστε να μην χρειαστεί η επανάκληση του Reasoner σε περίπτωση που ξανατεθούν μελλοντικά τα ίδια ή ισοδύναμά τους.
- Ερωτήματα τύπου PropertyValue(a,R,?) και PropertyValue(?,R,b) τα οποία στέλνονται στον Reasoner για αποτίμηση (επειδή η αποθηκευμένη πληροφορία δηλώσεων του ρόλου R δεν είναι πλήρης), κωδικοποιούνται και αποθηκεύονται ώστε την επόμενη φορά που

  θα τεθούν, τα ίδια ή ισοδύναμά τους, να απαντηθούν αποκλειστικά με χρήση της βάσης δεδομένων.
- Οι δηλώσεις ρόλων αντικειμένου και ρόλων τύπου δεδομένων αποθηκεύονται διακριτά στη βάση δεδομένων, καθώς απαιτούν διαφορετικά πεδία για την αποθήκευση της αντίστοιχης πληροφορίας.



Σχήμα 6.4: Διάγραμμα ροής του αλγορίθμου αποτίμησης ερωτήματος PropertyValue



Σχήμα 6.5: Διάγραμμα ροής του αλγορίθμου αποτίμησης ερωτήματος PropertyValue

- Οι δηλώσεις των μεταβατικών ρόλων αποθηκεύονται ξεχωριστά από τις δηλώσεις των υπολοίπων ρόλων αντικειμένου, καθώς απαιτούν διαφορετικό αριθμό πεδίων στη βάση δεδομένων για την κωδικοποίηση της πληροφορίας ιεραρχίας βάσει του σχήματος ετικετών.
- Οι νέες δηλώσεις μεταβατικών ρόλων, που ανακύπτουν έπειτα από κλήση του Reasoner, αποθηκεύονται ξεχωριστά από τις υπάρχουσες, δηλαδή δεν προστίθενται απ' ευθείας (με την εύρεσή τους) στο σχήμα ετικετών. Αντίθετα αποθηκεύονται σε ξεχωριστό πίνακα και στη συνέχεια, όποτε ο χρήστης το επιθυμήσει, μπορεί να γίνει η ανακατασκευή του σχήματος ετικετών ενός μεταβατικού ρόλου με κλήση της αντίστοιχης εφαρμογής (βλέπε Παράγραφο 5.1.8) και να προστεθούν σ' αυτό.

6.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Το DBRS έχει δοκιμασθεί και λειτουργεί κανονικά σε λειτουργικό σύστημα Windows XP, Windows Vista, Ubuntu 7.10 και Ubuntu 8.04. Παρόλα αυτά μπορεί να «τρέξει» σε οποιοδήποτε εμπορικό λειτουργικό σύστημα καθώς είναι γραμμένο σε Java και χρησιμοποιεί το DBMS PostgreSQL, διανομές των οποίων κυκλοφορούν για τα περισσότερα εμπορικά λειτουργικά συστήματα.

Πιο αναλυτικά, το DBRS υλοποιήθηκε με το Java SDK 1.6 αλλά μπορεί να «τρέξει» με οποιαδήποτε έκδοση του Java JRE από 1.4 και πάνω. Ο server της βάσης δεδομένων που χρησιμοποιήθηκε είναι ο PostgreSQL Server 8.2, ενώ δοκιμάστηκε επιτυχώς και η νέα έκδοση 8.3.

Για την ανάπτυξη της βάσης δεδομένων χρησιμοποιήθηκε ο pgAdmin III 1.6.3 , ενώ για τη διασύνδεση με τον PostgreSQL Server χρησιμοποιήθηκε ο οδηγός JDBC 8.2-505 (για την έκδοση 8.2 του server) και ο JDBC 8.3-603 (για την έκδοση 8.3).

Ο Reasoner που χρησιμοποιήθηκε είναι ο Pellet v. 1.5.1. Η συγγραφή του κώδικα έγινε με χρήση του αναπτυξιακού περιβάλλοντος Eclipse v. 3.3.2 Europa. Η επεξεργασία των οντολογιών για τις δοκιμές του συστήματος κατά την ανάπτυξη, αλλά και κατά τον έλεγχο, έγινε με χρήση του Protégé Ontology Editor v. 3.3.1 και v. 4 Alpha.

Αξίζει να σημειωθεί ότι τα πακέτα PostgreSQL Server, pgAdmin, Pellet, Eclipse, Protégé είναι όλα «ανοικτού» κώδικα και υπόκεινται στο GPL.

6.3 Εγκατάσταση Συστήματος

Θα χρειαστεί αρχικά να εγκαταστήσετε τα εξής:

- PostgreSQL Server 8.2 ή ανώτερος
- Pellet 1.5.1
- Java SDK 1.4 ή ανώτερο
- Eclipse 3.3.x

Επίσης, θα χρειαστεί να κατεβάσετε τον, αντίστοιχης έκδοσης του PostgreSQL Server, JDBC Driver.

Διαδικασία εγκατάστασης:

i) Αποσυμπιέστε τα περιεχόμενα του συμπιεσμένου αρχείου pellet-1.5.1.zip σε κάποιο σημείο του filesystem σας. Έστω ότι το αποσυμπιέζετε στο C:\

ii) Κάντε copy τον φάκελο ntua με όλα τα περιεχόμενά του και επικολλήστε τον μέσα στον φάκελο C:\pellet-1.5.1\src\

iii) Ανοίξτε το Eclipse και δημιουργήστε ένα νέο workspace.

iv) Πηγαίνετε File->New->Project… και μετά Java->Java Project From Existing Ant Buildfile, πατήστε next, μετά Browse… και επιλέξτε το αρχείο C:\pellet-1.5.1\build.xml και μετά πατήστε Finish.

v) Πηγαίνετε Project->Properties->Java Build Path->Libraries και πατήστε Add External JARs… Επιλέξτε το JAR αρχείο του JDBC Driver που έχετε κατεβάσει και πατήστε open και έπειτα OK.

vi) Πηγαίνετε στην κλάση ntua.DBRS.SystemInitializer.java και ορίστε για τις πρώτες τρεις final String μεταβλητές:

```
"C:\\pellet-1.5.1\\src\\ntua\\DBRS\\CreateDatabaseTables.sql"
"C:\\pellet-1.5.1\\src\\ntua\\DBRS\\CreateDatabaseIndexes.sql"
"C:\\pellet-1.5.1\\src\\ntua\\DBRS\\Transitive Assertions\\"
```

και για τις επόμενες δύο το όνομα και τον κωδικό του διαχειριστή του DBMS αντίστοιχα.

vii) Πηγαίνετε στην κλάση ntua.DBRS.SystemUtils.java και ορίστε την final String μεταβλητή *filesPath* ομοίως με παραπάνω:

"C:\\pellet-1.5.1\\src\\ntua\\DBRS\\Transitive Assertions\\"

viii) Πηγαίνετε στην κλάση org.mindswap.pellet.CachedNode.java και ορίστε στο τέλος τη νέα μέθοδο:

```
public Individual returnNode(){
        return node;
}
```

ix) Κάντε δεξί κλικ πάνω στην κλάση ntua.DBRS.SystemGUI και πατήστε Run As-> Open Run Dialog… επιλέξετε την καρτέλα Arguments και προσθέστε στα Program Arguments: -Xmx*NUMBER*m , όπου στη θέση του *NUMBER* γράφετε το μέγιστο μέγεθος μνήμης σε MB που θα επιτραπεί να δεσμεύσει ο Java Heap (Προτεινόμενη τιμή >1000).

x) Κάντε δεξί κλικ πάνω στην κλάση ntua.DBRS.SystemGUI και πατήστε Run As-> Java Application.

6.4 Λεπτομέρειες Υλοποίησης

Στην ενότητα αυτή περιγράφουμε αναλυτικά το σχήμα ετικετών για την αναπαράσταση των ιεραρχιών στη βάση δεδομένων, τα ευρετήρια που χρησιμοποιούμε για την βελτιστοποίηση της ταχύτητας απάντησης σε SQL ερωτήματα της βάσης δεδομένων, την ψευδογλώσσα σύνταξης ερωτημάτων και τις κλάσεις του κώδικα.

6.4.1 Σχήμα Ετικετών

Για την αποθήκευση του σχήματος ετικετών της ιεραρχίας των κλάσεων στην βάση δεδομένων του DBRS, ορίσαμε τρία νέα πεδία για κάθε εγγραφή του πίνακα «Concepts». Ένα πεδίο (c_label_pre) που περιλαμβάνει την τιμή index της ετικέτας, ένα πεδίο (c_label_post) που περιλαμβάνει την τιμή post της ετικέτας και ένα πεδίο (id_father_class) που περιλαμβάνει το id της κλάσης που είναι πατέρας, στο επιλεγμένο κατά την κατασκευή του σχήματος spanning tree, της κλάσης στην οποία αντιστοιχεί η εγγραφή. Το τελευταίο πεδίο χρησιμεύει μόνον για ερωτήματα σχετικά

με «αδέρφια» και «άμεσους απογόνους» ή «άμεσους προγόνους»[42]. Επίσης, υπάρχει η σχέση «DAG Concept Labels», η οποία έχει τρία πεδία ακεραίων: {id_concept,c_label_pre_dag,c_label_post_dag}. Σε κάθε εγγραφή αποθηκεύεται το id ενός κόμβου και η ετικέτα του κόμβου που αντιστοιχήθηκε στον πρώτο, βάσει του αλγορίθμου. Η διαδικασία κατασκευής του σχήματος έχει ως εξής:

i. Επιλέγεται μια κλάση από κάθε σύνολο ισοδύναμων κλάσεων για να συμπεριληφθεί στην αρχική κατασκευή του σχήματος.

ii. Αποθηκεύεται στην κύρια μνήμη ο γράφος της ιεραρχίας των κλάσεων σε έναν δισδιάστατο πίνακα. Ο πίνακας αυτός περιλαμβάνει όλες τις ακμές του γράφου σε μορφή ζευγών κόμβων, βάσει των ids τους.

iii. Καλείται η μέθοδος που διατάσσει τους κόμβους του γράφου σε τοπολογική σειρά (topological order) και αποθηκεύει την διάταξη σε έναν μονοδιάστατο πίνακα. Η διαδικασία κατασκευής της τοπολογικής σειράς γίνεται, μέσω SQL, με αναδιάταξη πινάκων και χρησιμοποιεί δύο προσωρινές σχέσεις της βάσης, από τις οποίες στη μια αντιγράφεται ο δισδιάστατος πίνακας του γράφου της ιεραρχίας που κατασκευάστηκε στο βήμα 2 και στην άλλη αποθηκεύεται η τοπολογική διάταξη των κόμβων.

iv. Στη συνέχεια, με χρήση της τοπολογικής σειράς που υπολογίστηκε στο προηγούμενο βήμα, υπολογίζεται το βέλτιστο συνδετικό δέντρο (optimum spanning tree) του γράφου. Ο αλγόριθμος τρέχει στην κύρια μνήμη και αποτελεί απ' ευθείας υλοποίηση του αλγορίθμου που περιγράφεται στο [ABJ89].

v. Γίνεται αναδρομική διάσχιση κατά αντίστροφη σειρά (post order) του συνδετικού δέντρου και οι κόμβοι αποθηκεύονται (σε αυτή τη σειρά) σε έναν μονοδιάστατο πίνακα. Η θέση κάθε κόμβου σε αυτόν τον πίνακα είναι και η τιμή post της ετικέτας του.

vi. Κατασκευάζεται ένας μονοδιάστατος πίνακας μήκους όσο το πλήθος των

---

[42] Οι τιμές των πεδίων c_label_post δεν χρησιμοποιούνται στο DBRS ως πρωτεύοντα κλειδιά της σχέσης, όπως θα μπορούσαν σύμφωνα με την τυπική υλοποίηση, καθώς δεν είναι μοναδικά. Αυτό γιατί, στην παρούσα έκδοση, η πληροφορία περί ισοδύναμων κλάσεων αποθηκεύεται δίνοντας ίδιες τιμές στα αντίστοιχα πεδία ετικετών.

κόμβων του γράφου, όπου αποθηκεύονται οι τιμές των πεδίων index κάθε ετικέτας. Η τιμή index της ετικέτας κάθε κόμβου είναι η μικρότερη post τιμή των απογόνων του κόμβου στο συνδετικό δέντρο.

vii. Φορτώνονται στη βάση οι ετικέτες των κλάσεων.

viii. Ενημερώνονται, μέσω SQL, οι τιμές των πεδίων id_father_class όλων των κλάσεων. Η διαδικασία γίνεται με χρήση ενός προσωρινού πίνακα όπου είχε αποθηκευθεί το συνδετικό δέντρο κατά την κατασκευή του.

ix. Σύμφωνα με τον αλγόριθμο, η ετικέτα κάθε κόμβου στον οποίο καταλήγει ακμή/ακμές του γράφου που δεν ανήκει/ανήκουν στο συνδετικό δέντρο, διαδίδεται (propagate) αναδρομικά σε όλους τους προγόνους του μέσω αυτής/αυτών της/των ακμής/ακμών. Οι διαδιδόμενες ετικέτες αποθηκεύονται στη σχέση «DAG Concept Labels». Κατά τη διαδικασία αυτή, εφαρμόζεται συμπίεση για μείωση της προς αποθήκευση στη βάση πληροφορίας. Αν το διάστημα της ετικέτας που διαδίδεται σε κάποιον κόμβο περιέχεται στο διάστημα της ετικέτας αυτού του κόμβου, τότε δεν αποθηκεύεται στη σχέση «DAG Concept Labels».

Ακριβώς όμοια διαδικασία ακολουθείται για την ιεραρχία των ρόλων της οντολογίας και για τις ιεραρχίες των ατόμων που μετέχουν σε δηλώσεις μεταβατικών ρόλων, με μόνη διαφορά, στην περίπτωση των μεταβατικών ρόλων, ότι πρωτεύον κλειδί κάθε κόμβου είναι το id του ρόλου μαζί με το id του ατόμου που μετέχει στη δήλωση.

Στην περίπτωση που υπάρχουν ισοδύναμοι μεταβατικοί ρόλοι, το σύνολο των δηλώσεών τους σχηματίζει μια (εικονική) ιεραρχία ατόμων. Την ίδια παρατήρηση μπορούμε να κάνουμε και για κάθε σύνολο αντίστροφων (μεταξύ τους) μεταβατικών ρόλων. Γι' αυτόν ακριβώς το λόγο, κατασκευάζουμε μια ιεραρχία για κάθε σύνολο ισοδύναμων ή/και αντιστρόφων ρόλων.

6.4.2 Ψευδογλώσσα Σύνταξης Ερωτημάτων

Για την σύνταξη των ερωτημάτων που υποβάλει ο χρήστης στη βάση δεδομένων του DBRS, υλοποιήθηκε μια γλώσσα η οποία υιοθετεί τα πρωταρχικά (primitive) ερωτήματα της SPARQL-DL [SP07] και κάποια επιπλέον που θεωρήθηκαν χρήσιμα Αυτά είναι τα ερωτήματα σχετικά με το πεδίο και το εύρος ρόλων και τα εποπτικά ερωτήματα.

Παρακάτω συμβολίζουμε:

- ‘C’ όνομα γνωστής κλάσης (named concept) ή σύνθετη εννοιακή έκφραση
- ‘R’ όνομα ρόλου
- ‘a’ και ‘b’ ονόματα ατόμων

Πρωταρχικά ερωτήματα SPQRQL-DL :

- EquivalentClass(C1 , C2)

  Ερώτημα σχετικά με ισοδυναμία κλάσεων.

- ComplementOf(C1 , C2)

  Ερώτημα σχετικά με συμπληρωματικότητα κλάσεων.

- SubClassOf(C1 , C2)

  Ερώτημα σχετικά με υπαγωγή κλάσεων. Ρωτάται αν η κλάση που αντιπροσωπεύει το πρώτο όρισμα υπάγεται στην κλάση που αντισπροσωπεύει το δεύτερο.

- DisjointWith(C1 , C2)

  Ερώτημα σχετικά με ασυμβατότητα κλάσεων.

- DisjointWith(R1 , R2)

  Ερώτημα σχετικά με συμπληρωματικότητα ρόλων.

- EquivalentProperty(R1 , R2)

  Ερώτημα σχετικά με ισοδυναμία ρόλων.

- SubPropertyOf(R1 , R2)

  Ερώτημα σχετικά με υπαγωγή ρόλων. Ρωτάται αν ο ρόλος που αντιπροσωπεύει το πρώτο όρισμα υπάγεται στον ρόλο που αντισπροσωπεύει το δεύτερο.

- InverseOf(R1 , R2)

  Ερώτημα σχετικά με αντιστροφή ρόλων.

- SymmetricProperty(R)

  Ερώτημα σχετικά με συματρικότητα ρόλου.

- TransitiveProperty(R)

  Ερώτημα σχετικά με μεταβατικότητα ρόλου.

- FunctionalObjectProperty(R)

  Ερώτημα σχετικά με λειτουργικότητα ρόλου αντικειμένου.

- FunctionalDatatypeProperty(R)

  Ερώτημα σχετικά με λειτουργικότητα ρόλου τύπου δεδομένων.

- InverseFunctionalProperty(R)

  Ερώτημα σχετικά με αντιστροφή ρόλου τύπου δεδομένων.

- ObjectProperty(R)

  Ερώτημα αν ο ρόλος R είναι αντικειμενικός.

- DatatypeProperty(R)

  Ερώτημα αν ο ρόλος R είναι τύπου δεδομένων.

- Type(a , C)

  Ερώτημα σχετικά με δηλώσεις κλάσεων.

- SameAs(a , b)

  Ερώτημα σχετικά με ταυτότητα ατόμων.

- DifferentFrom(a , b)

  Ερώτημα σχετικά με διαφορετικότητα ατόμων.

- PropertyValue(a , R , b)

  Ερώτημα σχετικά με δήλωση ρόλου.

Ερωτήματα σχετικά με πεδίο και εύρος ρόλων:

- Domain(R , C)

  Ερώτημα σχετικά με το πεδίο ρόλου.

- Range(R , C)

  Ερώτημα σχετικά με το έυρος ρόλου.

Εποπτικά ερωτήματα:

- DescriptionRelatedAxioms(C)

  Επιστρέφει όλα τα αξιώματα της οντολογίας στα οποία απαντάται η κλάση C.

- PropertyRelatedAxioms(R)

  Επιστρέφει όλα τα αξιώματα της οντολογίας στα οποία απαντάται ο ρόλος R.

- IndividualRelatedAxioms(a)

  Επιστρέφει όλα τα αξιώματα της οντολογίας στα οποία απαντάται το άτομο a.

Σημειώνουμε εδώ ότι, κατά τη σύνταξη ενός ερωτήματος, τα ορίσματά του μπορούν να αντικατασταθούν από τη μεταβλητή '?', φροντίζοντας το ερώτημα να μην περιέχει μόνον μεταβλητές (με εξαίρεση τα ερωτήματα ενός ορίσματος).

6.4.3 Ευρετήρια της βάσης δεδομένων

Παρακάτω αναφέρονται τα ευρετήρια (indexes) που επιλέχθηκαν για κάθε πίνακα της βάσης δεδομένων. Οι επιλογές των ευρετηρίων έγιναν με βασικό γνώμονα τη βελτίωση της ταχύτητας απάντησης στα πιο συνηθισμένα SQL ερωτήματα που γίνονται στη βάση δεδομένων κατά τη χρήση του DBRS, εκμεταλλευόμενοι την ποικιλία των ευρετηρίων που προσφέρει η PostgreSQL.

Βασικοί Πίνακες

Εδώ έχουμε:

- Πίνακας Namespaces

  Εδώ τα ερωτήματα που γίνονται είναι ταυτοποίησης είτε βάσει του πεδίου namespace_id είτε βάσει του πεδίου namespace. Βάσει αυτών ορίσαμε ένα hash ευρετήριο για κάθε πεδίο της σχέσης.
- Πίνακας Concepts

  Εδώ γίνονται τα εξής ερωτήματα:
  - Ταυτοποίησης βάσει του πεδίου concept_id. Ορίσαμε hash ευρετήριο για αυτό το πεδίο.
  - Ταυτοποίησης βάσει των πεδίων concept_name και namespace_id μαζί. Ορίσαμε ένα b-tree ευρετήριο για τα δύο αυτά πεδία μαζί, καθώς είναι ο μόνος τύπος ευρετηρίου που υποστηρίζει συνδυασμό δυο ή περισσοτέρων πεδίων.
  - Σύγκρισης για τα πεδία c_label_pre και c_label_post (ακέραιοι). Συγκεκριμένα, για ένα ερώτημα ιεραρχίας όπου θέλουμε να βρούμε

αν μια κλάση A είναι υποκλάση της B, θα πρέπει να ισχύει η πρόταση ((A.c_label_pre>=B.c_label_post AND A.c_label_post<B.c_label_post) OR (A.c_label_pre>B.c_label_post AND A.c_label_post<=B.c_label_post)) . Για τη βελτιστοποίηση αυτού του ερωτήματος ορίσαμε ένα b-tree για το πεδίο c_label_pre και ένα για το πεδίο c_label_post.

- Ταυτοποίησης βάσει του πεδίου id_father_class. Ορίσαμε ένα hash ευρετήριο για αυτό το πεδίο.

- Πίνακας Roles

  Ομοίως με τον πίνακα "Concepts".

- Πίνακας Individuals

  Ομοίως με τα δύο πρώτα ευρετήρια του πίνακα "Concepts" για τα πεδία id_individual, individual_name και namespace_id αντίστοιχα.

<u>«Συμμετρικοί» Πίνακες</u>

Τα ερωτήματα που γίνονται είναι ταυτοποίησης βάσει είτε και των δύο πεδίων μαζί, είτε μόνο του ενός. Για τη βελτιστοποίηση της απόδοσης υλοποιήσαμε τα παρακάτω:

- Σε όλους τους συμμετρικούς πίνακες υπάρχει ο περιορισμός η τιμή του πρώτου πεδίου να είναι μικρότερη από την τιμή του δεύτερου. Με αυτόν τον τρόπο επιταχύνεται η αποτίμηση ερωτημάτων ταυτοποίησης βάσει και των δυο πεδίων μαζί και αποκλείονται οι αντίστροφες διπλοεγγραφές.

  Αν για παράδειγμα ο χρήστης θέτει το ερώτημα SameAs(a , b), όπου έστω ότι τα άτομα a και b έχουν id_individual 6 και 4 αντίστοιχα, τότε η αναζήτηση θα γίνει μόνον για την εγγραφή [4,6] και όχι και για την [6,4], εφόσον αυτή θα είναι αδύνατον να υπάρχει.

- Τα δεδομένα στις σχέσεις είναι ομαδοποιημένα (clustered) βάσει του πρώτου πεδίου. Αυτό επιταχύνει την απάντηση ερωτημάτων ταυτοποίησης βάσει είτε και των δύο πεδίων μαζί, είτε μόνο του πρώτου πεδίου. Ορίζουμε b-tree ευρετήριο για τα δύο αυτά πεδία μαζί.

Η παραπάνω διάταξη των δεδομένων βελτιστοποιεί μεν τους δυο τύπους ερωτημάτων που αναφέρθηκαν, όμως επιβαρύνει την αποτίμηση ερωτημάτων ταυτοποίησης βάσει του δεύτερου πεδίου. Για το λόγο αυτό ορίζουμε επιπλέον και ένα hash ευρετήριο για το δεύτερο πεδίο.

Πίνακες Δηλώσεων

Εδώ έχουμε:

- Πίνακας Concept Assertions

  Η περίπτωση είναι παρόμοια με αυτή των συμμετρικών πινάκων. Έχουμε ερωτήματα ταυτοποίησης βάσει του πεδίου id_concept, ερωτήματα ταυτοποίησης βάσει του πεδίου id_individual και ερωτήματα ταυτοποίησης βάσει και των δυο πεδίων μαζί.

  Η πρώτη και η τρίτη κατηγορία ερωτημάτων βελτιστοποιείται με τον ορισμό ενός b-tree ευρετηρίου για τα δύο πεδία μαζί και με ομαδοποίηση (clustering) βάσει του πεδίου id_concept. Η δεύτερη κατηγορία βελτιστοποιείται με τον ορισμό ενός hash ευρετηρίου για το πεδίο id_individual.

- Πίνακας Role Assertions

  Εδώ γίνονται τα εξής ερωτήματα:

  - Ταυτοποίησης βάσει του πεδίου id_individual_subject
  - Ταυτοποίησης βάσει του πεδίου id_individual_object
  - Ταυτοποίησης βάσει του πεδίου id_role
  - Ταυτοποίησης βάσει και των τριών πεδίων μαζί
  - Ταυτοποίησης βάσει των πεδίων id_role και id_individual_subject
  - Ταυτοποίησης βάσει των πεδίων id_role και id_individual_object

  Για την επιτάχυνση του πρώτου τύπου ερωτημάτων ορίζουμε ένα ευρετήριο hash για το πεδίο id_individual_subject. Ομοίως και για τον δεύτερο τύπο.

  Για την επιτάχυνση των τεσσάρων τελευταίων τύπων ερωτημάτων, ορίζουμε ένα ευρετήριο τύπου b-tree για τα τρία πεδία μαζί, με ομαδοποίηση (clustering) στο πεδίο id_role.

- Πίνακας Inferred Transitive Role Assertions

  Ομοίως με τον πίνακα "Role Assertions".

- Πίνακας Datatype Role Assertions

  Ομοίως με τον πίνακα "Role Assertions", με τη διαφορά ότι στη θέση του πεδίου id_individual_object έχουμε τα δύο πεδία literal και type μαζί.

- Πίνακας Transitive Role Assertions

  Ο πίνακας αυτός είναι όμοιος με τον πίνακα "Concepts" (με την έννοια ότι περιλαμβάνει πληροφορία ιεραρχίας), με τη διαφορά ότι στη θέση του id_concept ως κόμβο του γράφου, έχουμε τα δύο πεδία id_role και id_individual μαζί. Συνεπώς ορίζουμε τα παρακάτω ευρετήρια:

  - B-tree ευρετήριο για τα πεδία id_role και id_individual μαζί. Ομαδοποίηση βάσει του πεδίου id_role.
  - B-tree ευρετήριο για τα πεδία id_role και id_father μαζί.
  - B-tree ευρετήριο για το πεδίο t_label_pre.
  - B-tree ευρετήριο για το πεδίο t_label_post.
  - Hash ευρετήριο για το πεδίο id_individual για τα ερωτήματα ταυτοποίησης βάσει μόνο αυτού του πεδίου.

Πίνακες Ετικετών DAG

Εδώ έχουμε:

- Πίνακας DAG Concept Labels

  Εδώ γίνονται τα εξής ερωτήματα:

  - Ταυτοποίησης βάσει του πεδίου id_concept. Ορίζουμε ένα b-tree ευρετήριο για αυτό το πεδίο και επίσης ομαδοποιούμε (cluster) τα δεδομένα βάσει αυτού του πεδίου.
  - Σύγκρισης στα πεδία c_label_pre_dag και c_label_post_dag. Ορίζουμε ένα b-tree ευρετήριο για το καθένα από αυτά.

- Πίνακας DAG Role Labels

  Ομοίως με τον πίνακα "DAG Concept Labels".

- Πίνακας DAG TRole Labels

  Ομοίως με τους δύο παραπάνω πίνακες. Ορίζουμε:

  - Ένα b-tree ευρετήριο για τα πεδία id_role και id_individual μαζί και ομαδοποιούμε τα δεδομένα βάσει του πεδίου id_role.
  - Ένα b-tree ευρετήριο για καθένα από τα πεδία t_label_pre_dag και t_label_post_dag.

Πίνακες Ψευδομοντέλων

Εδώ έχουμε:

- Πίνακας Concept PseudoModel In

  Εδώ γίνονται τα εξής ερωτήματα:

  - Ταυτοποίησης βάσει των πεδίων id_concept και complement μαζί. Ορίζουμε ένα b-tree ευρετήριο για τα δύο αυτά πεδία και ομαδοποιούμε (cluster) τα δεδομένα βάσει του πεδίου id_concept.
  - Ταυτοποίησης βάσει του πεδίου id_conceptIn. Ορίζουμε ένα hash ευρετήριο για αυτό το πεδίο.
- Πίνακας Concept PseudoModel NotIn

  Ομοίως με τον πίνακα "Concept PseudoModel In", όπου αντί για id_conceptIn έχουμε id_conceptNotIn.
- Πίνακας Concept PseudoModel Existence

  Ομοίως με τον πίνακα "Concept PseudoModel In", όπου αντί για id_conceptIn έχουμε id_role.
- Πίνακας Concept PseudoModel Universial

  Ομοίως με τον πίνακα "Concept PseudoModel Existence".
- Πίνακας Individual PseudoModel In

  Εδώ γίνονται τα εξής ερωτήματα:

  - Ταυτοποίησης βάσει του πεδίου id_individual. Ορίζουμε ένα b-tree ευρετήριο για το πεδίο αυτό και επίσης ομαδοποιούμε (cluster) τα δεδομένα βάσει του πεδίου αυτού.

- Ταυτοποίησης βάσει του πεδίου id_conceptIn. Ορίζουμε ένα hash ευρετήριο για αυτό το πεδίο.

- Πίνακας Individual PseudoModel NotIn

  Ομοίως με τον πίνακα "Individual PseudoModel In", όπου αντί για id_conceptIn έχουμε id_conceptNotIn.

- Πίνακας Individual PseudoModel Existence

  Ομοίως με τον πίνακα "Individual PseudoModel In", όπου αντί για id_conceptIn έχουμε id_role.

- Πίνακας Individual PseudoModel Universial

  Ομοίως με τον πίνακα "Individual PseudoModel Existence".

Πίνακες Caching Ερωτημάτων

Εδώ έχουμε:

- Πίνακας Binary Query Cacher

  Εδώ γίνονται τα εξής ερωτήματα:

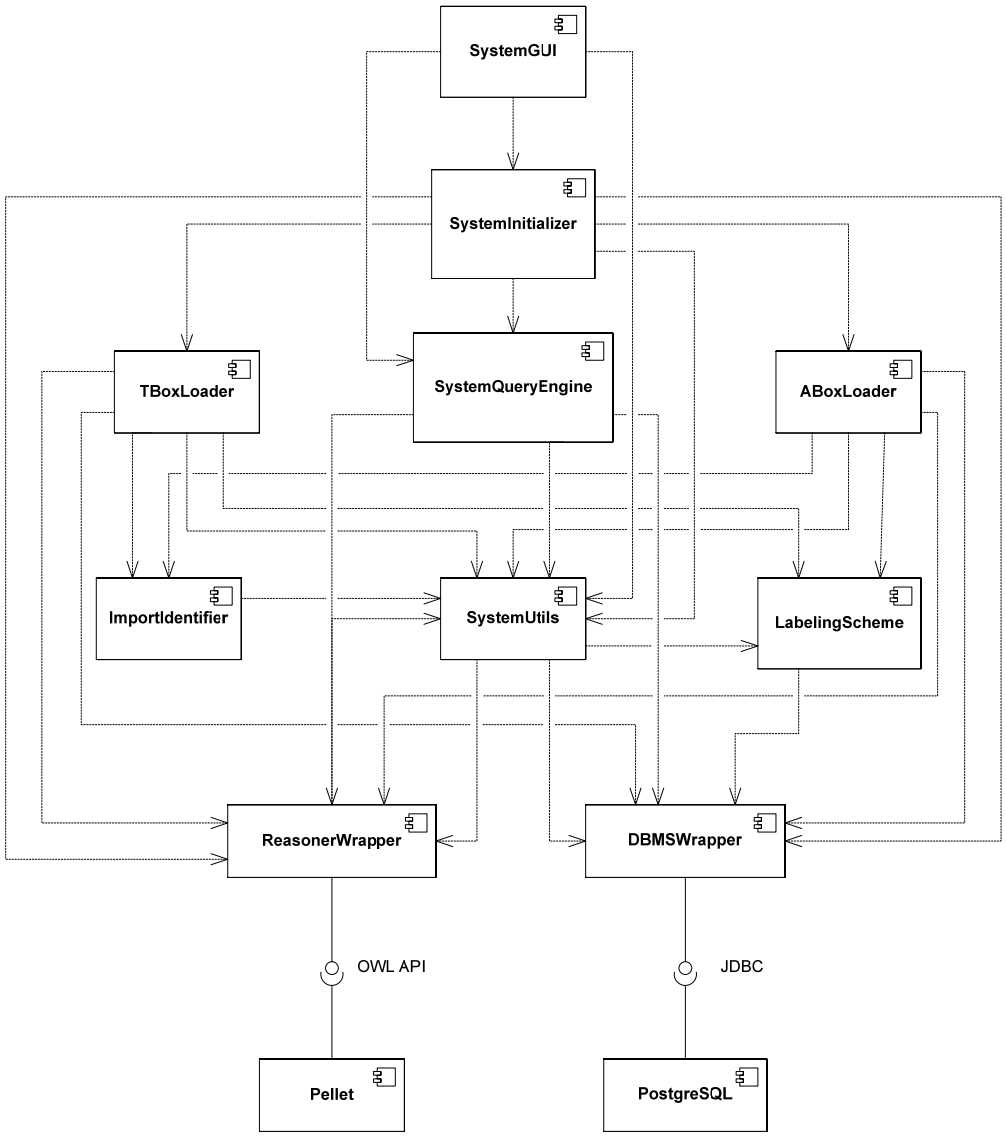
  - Ταυτοποίησης βάσει και των τριών πεδίων id_query, id_arg1 και id_arg2 μαζί. Ορίζουμε ένα b-tree για τα τρία πεδία και ομαδοποιούμε (cluster) τα δεδομένα βάσει του πεδίου id_query.
  - Ερωτήματα ταυτοποίησης βάσει του πεδίου id_arg1. Ορίζουμε ένα hash ευρετήριο για αυτό το πεδίο.
  - Ερωτήματα ταυτοποίησης βάσει του πεδίου id_arg2. Ορίζουμε ένα hash ευρετήριο για αυτό το πεδίο.

- Πίνακας Property Value Query Cacher

  Εδώ γίνονται ερωτήματα είτε βάσει του πεδίου id_arg1, είτε βάσει του πεδίου id_arg2, είτε βάσει του πεδίου id_arg3. Συνεπώς, ορίζουμε ένα hash ευρετήριο για κάθε ένα από τα τρία αυτά πεδία. Επίσης, ομαδοποιούμε (cluster) τα δεδομένα βάσει του πεδίου id_arg2.

6.4.4 Περιγραφή Java Κλάσεων

Στην ενότητα αυτή περιγράφονται συνοπτικά οι Java κλάσεις του κώδικα (source) του DBRS. Στο Σχήμα 6.6 φαίνεται το ψηφιδικό διάγραμμα (component diagram) του συστήματος, χρησιμοποιώντας συμβάσεις της γλώσσας UML. Τα διακεκομμένα βέλη δηλώνουν εξάρτηση (dependency) της ψηφίδας αφετηρίας από την ψηφίδα προορισμού. Όπως έχει ήδη αναφερθεί, οι κλάσεις ReasonerWrapper και DBMSWrapper υλοποιούν (implement) τις διαπροσωπίες OWL API και JDBC API αντιστοίχως. Κάθε ψηφίδα του διαγράμματος αντιστοιχεί σε μια κλάση του DBRS, με εξαίρεση τα εξωτερικά συστήματα PostgreSQL (DBMS) και Pellet (Reasoner).



Σχήμα 6.6: Ψηφιδικό διάγραμμα του συστήματος DBRS

6.4.4.1 *Η κλάση SystemGUI*

Παρακάτω, αναφέρονται μόνο τα μέλη της κλάσης που παρουσιάζουν ενδιαφέρον λόγω του μεγάλου πλήθους τους.

Μέλη της κλάσης

- `JPanel topPanel`

  Το πάνω μέρος του παραθύρου του γραφικού περιβάλλοντος.

- `JPanel bottomPanel`

  Το κάτω μέρος του παραθύρου του γραφικού περιβάλλοντος.

- `JTextArea queryText`

  Το πεδίο όπου πληκτρολογεί ο χρήστης το ερώτημα.

- `JTextField pathText`

  Το πεδίο όπου πληκτρολογεί ο χρήστης την τοποθεσία του αρχείου της οντολογίας.

- `JButton runQuery`

  Το κουμπί με το πάτημα του οποίου ξεκινάει η διαδικασία αποτίμησης του ερωτήματος που έχει πληκτρολογήσει ο χρήστης.

- `JButton queryHelp`

  Το κουμπί με το πάτημα του οποίου εμφανίζεται το παράθυρο βοήθειας σύνταξης ερωτημάτων.

- `JButton ontClean`

  Το κουμπί με το πάτημα του οποίου ξεκινάει η εφαρμογή εκκαθάρισης βάσης δεδομένων.

- `JButton ontLoad`

  Το κουμπί με το πάτημα του οποίου ξεκινάει η εφαρμογή φόρτωσης οντολογίας.

- `JButton browseButton`

  Το κουμπί με το πάτημα του οποίου ξεκινάει η εφαρμογή αναζήτησης αρχείου οντολογίας στο σύστημα αρχείων του χρήστη.

- `JButton viewDBsButton`

  Το κουμπί με το πάτημα του οποίου ξεκινάει η εφαρμογή εποπτείας υπαρχουσών βάσεων δεδομένων.

- `JButton aboutButton`

  Το κουμπί με το πάτημα του οποίου εμφανίζονται γενικές πληροφορίες για το DBRS και τους δημιουργούς του.

- `JRadioButton inferredTAssertions`

  Το κουμπί με την ενεργοποίηση του οποίου ξεκινάει η εφαρμογή ανανέωσης του σχήματος ετικετών των ατόμων που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

- `JCheckBox conceptDisjointness`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί η πλήρης πληροφορία περί ασυμβατότητας κλάσεων κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox roleDisjointness`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί η πλήρης πληροφορία περί ασυμβατότητας ρόλων κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox individualDifference`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί η πλήρης πληροφορία περί διαφορετικότητας ατόμων κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox conceptCompletion`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί η πλήρης πληροφορία περί συμπληρωματικότητας κλάσεων κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox conceptPseudoModels`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί ένα ψευδομοντέλο για κάθε κλάση της οντολογίας κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox individualPseudoModels`

Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα φορτωθεί ένα ψευδομοντέλο για κάθε άτομο της οντολογίας κατά την εκτέλεση της εφαρμογής φόρτωσης της οντολογίας.

- `JCheckBox expandedTBox`

  Το κουτί επιλογής, με το τσεκάρισμα του οποίου θα κατασκευασθεί και θα αποθηκευθεί το διευρυμένο TBox της οντολογίας.

- `JCheckBox lazyMode`

  Το κουτί επιλογής με την επιλογή του οποίου το σύστημα μπαίνει σε «οκνυρή» λειτουργία.

- `JTextArea resultsPane`

  Το πεδίο όπου προβάλλονται τα αποτελέσματα των ερωτημάτων που θέτει ο χρήστης και τα μηνύματα του συστήματος.

- `SystemInitializer systemInitializer`

  Το αντικείμενο κλάσης SystemInitializer το οποίο δημιουργείται κατά τη φόρτωση της οντολογίας και κρατείται για την πρόσβαση στη μηχανή ερωτημάτων.

Μέθοδοι της κλάσης

- `private SystemGUI()`

  Δημιουργία νέου αντικειμένου της κλάσης SystemGUI και κλήση της μεθόδου setupUI().

- `private void setupUI()`

  Κατασκευή των στοιχείων (containers και components) του GUI και δημιουργία των απαιτούμενων listeners.

- `private static void createAndShowGUI()`

  Ορισμός διαστάσεων GUI και εμφάνισή του.

- `public void actionPerformed(ActionEvent evt)`

  Σύλληψη, αναγνώριση γεγονότος (action event) και κλήση αντίστοιχης μεθόδου. Ένα γεγονός μπορεί να είναι επιλογή (ή και αποεπιλογή όσον αφορά στα checkboxes) ενός εκ των:

  - «Disjoint Concepts info» checkbox

- «Disjoint Properties info» checkbox
- «Different Individuals info» checkbox
- «Complement Concepts info» checkbox
- «Concept PseudoModels» checkbox
- «Individual PseudoModels» checkbox
- «Expanded TBox» checkbox
- «Lazy Mode» checkbox
- «Update Transitive Assertions» radio button
- «Load Selected Ontology» button
- «Clean Existent Database» button
- «View Databases» button
- «Browse…» button
- «Execute Query» button
- «Help» button

- `private void displayHelp()`

  Εμφάνιση παραθύρου βοήθειας (Help frame).

- `private void handleQuery()`

  Κλήση μεθόδου executeQuery(String query) του αντικειμένου τύπου SystemQueryEngine, όπως αυτό έχει αρχικοποιηθεί από το αντικείμενο τύπου SystemInitializer. Ως όρισμα περνιέται το ερώτημα που υποβάλλει ο χρήστης υπό μορφή αλφαριθμητικού (query) και το αποτέλεσμα προβάλλεται απευθείας στο GUI.

- `private void cleanDataBase()`

  Εκκαθάριση των δεδομένων της βάσης που επιλέγει ο χρήστης. Δημιουργεί τρία νέα παράθυρα εισαγωγής (input dialog) για την εισαγωγή του ονόματος της βάσης δεδομένων, του ονόματος του χρήστη και του κωδικού πρόσβασης.

- `private void updateTAssertions()`

  Κλήση της μεθόδου updateTAssertions(String tRoleURI) του αντικειμένου SystemInitializer για την ανανέωση της ιεραρχίας των ατόμων που συνδέονται μέσω μεταβατικού ρόλου. Το URI του ρόλου (tRoleURI) δίνεται από το χρήστη μέσω παραθύρου εισαγωγής. Κατά τη διαδικασία αυτή,

δημιουργούνται τέσσερα επιπλέον παράθυρα για την επιβεβαίωση συνέχισης της διαδικασίας (option dialog) και την εισαγωγή ονόματος βάσης δεδομένων, ονόματος χρήστη και κωδικού πρόσβασης.

- `private void initializeSystem()`

  Κατασκευή νέου αντικειμένου της κλάσης SystemInitializer για την αρχικοποίηση του συστήματος, βάσει των παραμέτρων που έχουν οριστεί από το χρήστη μέσω του GUI. Κατά τη διαδικασία αυτή, δημιουργούνται τρία νέα παράθυρα για την εισαγωγή του ονόματος της βάσης δεδομένων, του ονόματος του χρήστη και του κωδικού πρόσβασης.

- `private void browseFileSystem()`

  Δημιουργία παραθύρου αναζήτησης αρχείων στο σύστημα του χρήστη και αποθήκευση της διαδρομής (path) του επιλεγμένου αρχείου.

- `private void viewDataBases()`

  Δημιουργία παραθύρου προβολής των υπαρχουσών βάσεων δεδομένων και των ιδιοκτητών τους. Κατά τη διαδικασία, εμφανίζονται δυο επιπλέον παράθυρα για την εισαγωγή ονόματος χρήστη και κωδικού πρόσβασης.

- `public static void main(String[] args)`

  Κλήση μεθόδου createAndShowGUI().

6.4.4.2 *Η κλάση SystemInitializer*

Μέλη της κλάσης

- `static final String dbTablesScriptPath`

  Εδώ αποθηκεύεται η διαδρομή (path) του script αρχείου κατασκευής του σχήματος της βάσης δεδομένων του DBRS.

- `static final String dbIndexesScriptPath`

  Εδώ αποθηκεύεται η διαδρομή (path) του script αρχείου κατασκευής των ευρετηρίων της βάσης δεδομένων του DBRS.

- `static final String filesPath`

  Εδώ αποθηκεύεται η διαδρομή (path) του φακέλου όπου βρίσκονται τα text αρχεία αποθήκευσης δηλώσεων των μεταβατικών ρόλων για τους οποίους έχει κατασκευαστεί σχήμα ετικετών.

- `static final String adminUserName`

  Εδώ αποθηκεύεται το όνομα του διαχειριστή του DBMS.

- `static final String adminPassword`

  Εδώ αποθηκεύεται ο κωδικός του διαχειριστή του DBMS.

- `String ontologyPath`

  Η τοποθεσία του αρχείου της οντολογίας (URL ή διαδρομή στο σύστημα αρχείων).

- `DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

- `JTextArea resultPane`

  Το πεδίο του GUI όπου γράφονται τα αποτελέσματα της αποτίμησης των ερωτημάτων και τα μηνύματα του συστήματος.

- `ReasonerWrapper reasoner`

  Αντικείμενο της κλάσης ReasonerWrapper για τη σύνδεση με τον Reasoner.

- `OWLOntologyManager manager`

  Αντικείμενο της κλάσης OWLOntologyManager, απαραίτητο για την κατασκευή του μοντέλου της οντολογίας στην κύρια μνήμη.

- `OWLOntology ontology`

  Αντικείμενο της κλάσης OWLOntology. Είναι η φορτωμένη στην κύρια μνήμη οντολογία.

- `SystemQueryEngine queryEngine`

  Αντικείμενο της κλάσης SystemQueryEngine. Είναι η μηχανή αποτίμησης ερωτημάτων.

- `private String userName`

  Το όνομα του χρήστη.

- `private String userPassword`

  Ο κωδικός του χρήστη.

- `private String dbName`

  Το όνομα της βάσης δεδομένων.

Μέθοδοι της κλάσης

- **public** SystemInitializer(String path, String user_Name, String user_Password, String db_Name, **boolean** disjConcepts, **boolean** complConcepts, **boolean** concPM, **boolean** disjRoles, **boolean** diffInds, **boolean** indPM, **boolean** createExpTBox, **boolean** lazyMode, JTextArea pane)

  Δημιουργία νέου αντικειμένου κλάσης SystemInitializer για την αρχικοποίηση του συστήματος και τη δημιουργία (όταν απαιτείται) και φόρτωση της βάσης δεδομένων. Αφού έχει ολοκληρωθεί η διαδικασία φόρτωσης, γίνεται αρχικοποίηση της Μηχανής Ερωτημάτων μέσω της δημιουργίας ενός νέου αντικειμένου της κλάσης SystemQueryEngine. Δέχεται τα ορίσματα:

  - path: Η διαδρομή του αρχείου της οντολογίας ή το URI της, σε μορφή αλφαριθμητικού.
  - user_Name: Το όνομα του χρήστη που απαιτείται για τη σύνδεση με το PostgreSQL, σε μορφή αλφαριθμητικού.
  - user_Password: Ο κωδικός του χρήστη που απαιτείται για τη σύνδεση με το PostgreSQL, σε μορφή αλφαριθμητικού.
  - db_Name: Το όνομα της βάσης δεδομένων που υπάρχει ή πρόκειται να δημιουργηθεί για να φορτωθεί η οντολογία, σε μορφή αλφαριθμητικού.
  - disjConcepts: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με την ασυμβατότητα κλάσεων.
  - complConcepts: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με την συμπληρωματικότητα κλάσεων.
  - concPM: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) ενός ψευδομοντέλου για κάθε κλάση της οντολογίας.
  - disjRoles: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με την ασυμβατότητα ρόλων.
  - diffInds: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με την διαφορετικότητα ατόμων.

- `indPM`: Δυαδική παράμετρος για τη φόρτωση ή όχι (true/false) ενός ψευδομοντέλου για κάθε άτομο της οντολογίας.
- `createExpTBox`: Δυαδική παράμετρος για τη κατασκευή ή όχι (true/false) του διευρυμένου TBox της οντολογίας.
- `cleanDB`: Δυαδική παράμετρος για την εκκαθάριση ή όχι (true/false) της βάσης δεδομένων.
- `lazyMode`: Δυαδική παράμετρος για την επιλογή του τρόπου απάντησης ερωτημάτων μεταξύ «οκνηρής» ή «πρόθυμης» (true/false) λειτουργίας. Μεταβιβάζεται στο αντικείμενο της κλάσης SystemQueryEngine.
- `updateTAssertions`: Δυαδική παράμετρος για τη ανανέωση της ιεραρχίας των ατόμων που συνδέονται μέσω κάποιου μεταβατικού ρόλου (tRoleURI).
- `tRoleURI`: Το URI του μεταβατικού ρόλου της οντολογίας η ιεραρχία των ατόμων του οποίου θέλουμε να ανανεωθεί, σε μορφή αλφαριθμητικού.
- `pane`: Το αντικείμενο τύπου JTextArea του GUI, στο οποίο προβάλλονται τα μηνύματα του συστήματος, εν προκειμένω σχετικά με την πρόοδο της διαδικασίας φόρτωσης.

- **`public`** `SystemInitializer(String user_Name, String user_Password, String db_Name, JTextArea pane)`

  Ένας δεύτερος κατασκευαστής της κλάσης ο οποίος χρησιμοποιείται μόνο για τις κλήσεις των μεθόδων cleanDB() και updateTAssertions(), όταν αυτό ζητηθεί μέσω του GUI. Τα ορίσματά του αντιστοιχούν στα ομώνυμα ορίσματα του προηγούμενου κατασκευαστή.

- **`public void`** `updateTAssertions(String tRoleURI)`

  Καλεί τη μέθοδο updateTransitiveRoleAssertionsLabelingScheme() της κλάσης SystemUtils για την ανανέωση της ιεραρχίας των ατόμων που συνδέονται μέσω του μεταβατικού ρόλου με URI tRoleURI. Το URI αυτό δίνεται ως παράμετρος σε μορφή αλφαριθμητικού, μέσω του GUI, από το χρήστη.

- `public void cleanDB()`

  Καθαρίζει τη βάση από τα δεδομένα της. Εκτός από τα δεδομένα της βάσης, γίνεται κλήση της deleteTransitiveAssertionsFiles() της κλάσης SystemUtils και σβήνονται όλα τα βοηθητικά text αρχεία που περιέχουν δηλώσεις μεταβατικών ρόλων.

- `private boolean checkDBInfo()`

  Ελέγχει αν η βάση δεδομένων, τα στοιχεία της οποίας έχουν δοθεί νωρίτερα από το χρήστη μέσω του GUI, περιέχει δεδομένα. Αν είναι φορτωμένη με κάποια άλλη οντολογία από αυτή που επιχειρεί να φορτώσει ο χρήστης, επιστρέφει false. Σε αντίθετη περίπτωση ή στην περίπτωση που η βάση δεν υπάρχει ή είναι άδεια, επιστρέφει true.

- `public SystemQueryEngine getQueryEngine()`

  Επιστρέφει το αντικείμενο SystemQueryEngine.

- `public short getOntologyNamespaceDBID()`

  Επιστρέφει το id του namespace της οντολογίας που αρχικοποιείται, όπως αυτό είναι αποθηκευμένο στη βάση δεδομένων.

- `private void createDataBase(DBMSWrapper checkDBHandler)`

  Δημιουργεί, αν δεν υπάρχει ήδη, ένα νέο ρόλο στο DBMS με τα στοιχεία του χρήστη. Στη συνέχεια, κατασκευάζει τη βάση δεδομένων και τα ευρετήριά της, εκτελώντας τα δύο αντίστοιχα script αρχεία.

6.4.4.3 Η κλάση TBoxLoader

Μέλη της κλάσης

- `final DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

- `final ReasonerWrapper reasoner`

  Αντικείμενο της κλάσης ReasonerWrapper για τη σύνδεση με τον Reasoner.

- `final OWLOntologyManager manager`

  Αντικείμενο της κλάσης OWLOntologyManager, για πρόσβαση στα στοιχεία του μοντέλου της οντολογίας στην κύρια μνήμη.

- `final OWLDataFactory factory`

  Αντικείμενο της κλάσης OWLDataFactory, για την κατασκευή τύπων του OWL API.

- `final OWLOntology ontology`

  Αντικείμενο της κλάσης OWLOntology. Είναι η φορτωμένη στην κύρια μνήμη οντολογία.

Μέθοδοι της κλάσης

- `public TBoxLoader(DBMSWrapper inputdbHandler, ReasonerWrapper inputreasoner, OWLOntologyManager inputmanager, OWLOntology inputontology, boolean disjConcepts, boolean disjRoles, boolean complConcepts, boolean concPM, boolean createExpTBox)`

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης. Δέχεται ως ορίσματα:

  - `inputdbHandler`: Αντικείμενο κλάσης DBMSWrapper μέσω του οποίου γίνεται όλη η επικοινωνία με τη βάση δεδομένων.
  - `inputreasoner`: Αντικείμενο κλάσης ReasonerWrapper μέσω του οποίου γίνεται όλη η επικοινωνία με τον Pellet.
  - `inputmanager`: Αντικείμενο κλάσης OWLOntologyManager, απαραίτητο για τον χειρισμό της φορτωμένης στην κύρια μνήμη οντολογίας.
  - `inputontology`: Αντικείμενο κλάσης OWLOntology. Είναι το φορτωμένο στην κύρια μνήμη μοντέλο της οντολογίας.
  - `disjConcepts`: Δυαδική παράμετρος που καθορίζει τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με ασυμβατότητα κλάσεων.
  - `disjRoles`: Δυαδική παράμετρος που καθορίζει τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με ασυμβατότητα ρόλων.
  - `complConcepts`: Δυαδική παράμετρος που καθορίζει τη φόρτωση ή όχι (true/false) της πλήρους πληροφορίας σχετικά με συμπληρωματικότητα κλάσεων.

- `concPM`: Δυαδική παράμετρος που καθορίζει τη φόρτωση ή όχι (true/false) στη βάση δεδομένων ενός ψευδομοντέλου για κάθε κλάση της οντολογίας.
- `createExpTBox`: Δυαδική παράμετρος που καθορίζει την κατασκευή και φόρτωση στη βάση δεδομένων του διευρυμένου TBox της οντολογίας.

- **`private void`** `loadNamespaces()`

  Αποθηκεύεται το namespace της οντολογίας καθώς και όλων των οντολογιών που γίνονται imported στο αρχείο της. Επίσης αποθηκεύονται και άλλα απαραίτητα namespaces, όπως το namespace των xml τύπων και το namespace των κλάσεων OWLThing και OWLNothing. Σε κάθε namespace δίνεται ένα μοναδικό id.

- **`private void`** `createConceptsHierarchy( HashSet<OWLClass> ontologyClasses, HashSet<OWLClass> equivalentClasses )`

  Είναι υπεύθυνη για την κατασκευή και αποθήκευση στη βάση δεδομένων της ιεραρχίας των κλάσεων της οντολογίας. Δέχεται τα ορίσματα ontologyClasses και equivalentClasses τύπου Java Set που περιέχουν αντικείμενα τύπου OWLClass. Το πρώτο περιέχει τις κλάσεις τις οντολογίας και το δεύτερο επιλεγμένες ισοδύναμες κλάσεις οι οποίες δεν χρησιμοποιούνται κατά την αρχική κατασκευή της ιεραρχίας και προστίθενται σε αυτήν στο τέλος (βλέπε επόμενη μέθοδο). Κατά την κλήση της μεθόδου ακολουθείται η παρακάτω διαδικασία:

  - Αφαιρούνται οι περιττές ισοδύναμες κλάσεις οι οποίες δεν χρειάζονται για την κατασκευή της ιεραρχίας. Η εύρεσή τους γίνεται με τη βοήθεια του Pellet.
  - Ζητούνται από τον Pellet οι απ' ευθείας απόγονοι κάθε κλάσης. Με αυτή την τεχνική λαμβάνονται οι αντίστοιχες ακμές του γράφου της ιεραρχίας, οι οποίες αποθηκεύονται στον προσωρινό πίνακα "ConceptsTemp".
  - Κατασκευάζεται με χρήση του πίνακα "ConceptsTemp" το Interval Labeling Scheme για την αναπαράσταση και αποθήκευση της πλήρους ιεραρχίας των κλάσεων στη βάση δεδομένων. Η διαδικασία αυτή ξεκινάει με την δημιουργία ενός νέου αντικειμένου της κλάσης

LabelingScheme. Μόλις ολοκληρωθεί ο πίνακας "ConceptsTemp" είναι πλέον περιττός και διαγράφεται (drop).

- `private void addEquivalentClasses( HashSet<OWLClass> equivalentClasses )`

  Είναι υπεύθυνη για την αποθήκευση, στην ήδη κατασκευασμένη ιεραρχία, των ισοδύναμων κλάσεων που αρχικά δεν τοποθετήθηκαν σε αυτήν. Σε αυτές δίνονται τα ίδια ακριβώς labels με τις ισοδύναμές τους, των οποίων τα labels κατασκευάστηκαν στο ακριβώς προηγούμενο βήμα.

- `private void loadDisjointConcepts( HashSet<OWLClass> ontologyClasses )`

  Είναι υπεύθυνη για την αποθήκευση στη βάση δεδομένων της πλήρους πληροφορίας περί ασυμβατότητας των κλάσεων της οντολογίας. Δέχεται το όρισμα ontologyClasses που είναι το ίδιο με αυτό που περιγράφηκε στη μέθοδο createConcpetsHierarchy.

  Η μέθοδος στέλνει στον Pellet ζεύγη κλάσεων ώστε να ελεγχθεί αν είναι ασύμβατες. Τα ζεύγη ασύμβατων κλάσεων που επιστρέφονται αποθηκεύονται στον πίνακα "Concept Disjointness" της βάσης δεδομένων. Η διαδικασία βελτιστοποιείται με τη χρήση κατάλληλου αλγόριθμου για την αποστολή στον Pellet όσο το δυνατόν λιγότερων ζευγών κλάσεων προς έλεγχο.

  Στο τέλος της διαδικασίας τα πεδία "c_disj_complete" του πίνακα " Concepts " για όλες τις κλάσεις της οντολογίας γίνονται true, καθώς η πληροφορία περί ασυμβατότητας κλάσεων είναι πλέον πλήρης.

- `private void loadExplicitDisjointConcepts()`

  Φορτώνει στον πίνακα "Concept Disjointness" της βάσης δεδομένων μόνον τα ζεύγη ids των κλάσεων που αναφέρονται ρητά στο αρχείο της οντολογίας ως ασύμβατες. Αν κάποια γνωστή κλάση δηλώνεται ως ασύμβατη με κάποια αυθαίρετη κλάση, τότε η αυθαίρετη κλάση αποθηκεύεται με όνομα 'anonymous' στον πίνακα "Concepts" της βάσης δεδομένων, ώστε να της δοθεί ένα μοναδικό id και το ζεύγος να μπορέσει να αποθηκευθεί στη συνέχεια στον πίνακα "Concept Disjointness".

- **private void** loadDisjointConceptsRecursively(Object[] disjointCandidatesIds, HashSet<String> unsatConceptIds, **int** topID, **int** notTopID, **int** level)

  Καλείται από τη μέθοδο loadDisjointConcepts() και υλοποιεί τον αλγόριθμο της Παραγράφου 6.1.1 για την αποδοτική φόρτωση της πλήρους πληροφορίας περί ασυμβατότητας κλάσεων, χρησιμοποιώντας αναδρομική τεχνική.

- **private void** loadComplementConcepts(HashSet<OWLClass> ontologyClasses, HashSet<OWLClass> equivalentClasses)

  Είναι υπεύθυνη για την αποθήκευση στη βάση δεδομένων της πλήρους πληροφορίας περί συμπληρωματικότητας των κλάσεων της οντολογίας. Δέχεται τα ορίσματα ontologyClasses και equivalentClasses τα οποία είναι τα ίδια με αυτά που περιγράφηκαν για τη μέθοδο createConceptsHierarchy().

  Η μέθοδος στέλνει στον Pellet ζεύγη κλάσεων ώστε να ελεγχθεί αν είναι συμπληρωματικές. Η διαδικασία βελτιστοποιείται ελαχιστοποιώντας τον αριθμό των ζευγών που στέλνονται για έλεγχο. Η ελαχιστοποίηση αυτή γίνεται με δύο τρόπους. Με χρήση μόνο μιας κλάσης από κάθε σύνολο ισοδύναμων κλάσεων και με εκμετάλλευση των ήδη αποθηκευμένων ορισμών των κλάσεων, από τους οποίους μπορεί, μέσω μιας διαδικασίας «ελαφριάς» συλλογιστικής ανάλυσης, να εξαχθεί σχέση συμπληρωματικότητας..

- **private void** createRolesHierarchy(Set<OWLProperty> ontologyRoles, HashSet<OWLProperty> equivalentRoles)

  Είναι υπεύθυνη για την κατασκευή και αποθήκευση στη βάση δεδομένων της ιεραρχίας των ρόλων της οντολογίας. Δέχεται τα ορίσματα ontologyRoles και equivalentRoles τύπου Java Set που περιέχουν αντικείμενα τύπου OWLProperty. Το πρώτο περιέχει τους ρόλους τις οντολογίας και το δεύτερο επιλεγμένους ισοδύναμους ρόλου οι οποίοι δεν χρησιμοποιούνται κατά την κατασκευή της ιεραρχίας και προστίθενται σε αυτήν στο τέλος (βλέπε επόμενη μέθοδο). Κατά την κλήση της μεθόδου ακολουθείται η παρακάτω διαδικασία:

  - Αφαιρούνται οι περιττοί ισοδύναμοι ρόλοι οι οποίοι δεν χρειάζονται για την κατασκευή της ιεραρχίας. Η εύρεσή τους γίνεται με τη βοήθεια του Pellet.
  - Ζητούνται από τον Pellet οι απ' ευθείας απόγονοι κάθε ρόλου. Με αυτή την τεχνική λαμβάνονται οι αντίστοιχες ακμές του γράφου της

ιεραρχίας, οι οποίες αποθηκεύονται στον προσωρινό πίνακα "RolesTemp".

- Κατασκευάζεται με χρήση του πίνακα "RolesTemp" το Interval Labeling Scheme για την αναπαράσταση και αποθήκευση της πλήρους ιεραρχίας των ρόλων στη βάση δεδομένων. Η διαδικασία αυτή ξεκινάει με την δημιουργία ενός νέου αντικειμένου της κλάσης LabelingScheme. Μόλις ολοκληρωθεί, ο πίνακας "RolesTemp" είναι πλέον περιττός και διαγράφεται (drop).

- `private void addEquivalentRoles( HashSet<OWLProperty> equivalentRoles )`

  Είναι υπεύθυνη για την αποθήκευση στην ήδη κατασκευασμένη ιεραρχία των ισοδύναμων ρόλων που αρχικά δεν τοποθετήθηκαν σε αυτήν. Σε αυτούς δίνονται τα ίδια ακριβώς labels με τους ισοδύναμούς τους, των οποίων τα labels κατασκευάστηκαν στο ακριβώς προηγούμενο βήμα.

- `private void handleInverseRoles( HashSet<OWLProperty> equivalentRoles )`

  Είναι υπεύθυνη για την αποθήκευση της πληροφορίας περί αντίστροφων ρόλων (inverse roles). Η εύρεση των ζευγών αντίστροφων ρόλων γίνεται με τη βοήθεια του Pellet. Η πληροφορία που εξάγεται αποθηκεύεται στον πίνακα "Inversion" της βάσης δεδομένων, με τη μορφή ζευγών ids ρόλων. Η διαδικασία βελτιστοποιείται με την αποφυγή αποστολής στον Pellet ισοδύναμων ρόλων που έχουν ήδη σταλεί και της μη αποθήκευσης στον πίνακα "Inversion" συμμετρικών ρόλων. Επίσης, ανανεώνεται το πεδίο "role_characteristics" στον πίνακα "Roles" της βάσης δεδομένων για όποιον ρόλο διαθέτει αντίστροφο, καθώς και για τους συμμετρικούς ρόλους που ανευρίσκονται.

- `private void handleDomainRangeAndSetCharacteristics()`

  Αποθηκεύει τη ρητά δηλωμένη πληροφορία περί πεδίου και εύρους, ενώ επιτελεί και μια πρώτη ενημέρωση των χαρακτηριστικών των ρόλων.

- `private void manipulateObjectPropertyDomainAxiom( int index, String axiom )`

  Καλείται από τη μέθοδο handleDomainRangeAndSetCharacteristics() και χειρίζεται αξιώματα πεδίου ρόλου.

- `private void manipulateObjectPropertyRangeAxiom( int index, String axiom )`

  Καλείται από τη μέθοδο handleDomainRangeAndSetCharacteristics() και χειρίζεται αξιώματα εύρους ρόλου.

- `private void manipulateTransitiveObjectPropertyAxiom( int index, String axiom )`

  Καλείται από τη μέθοδο handleDomainRangeAndSetCharacteristics() και ανανεώνει τα χαρακτηριστικά των μεταβατικών ρόλων.

- `private void manipulateFunctionalObjectPropertyAxiom( int index, String axiom )`

  Καλείται από τη μέθοδο handleDomainRangeAndSetCharacteristics() και ανανεώνει τα χαρακτηριστικά των λειτουργικών ρόλων αντικειμένου.

- `private void manipulateInverseFunctionalObjectPropertyAxiom( int index, String axiom )`

  Καλείται από τη μέθοδο handleDomainRangeAndSetCharacteristics() και ανανεώνει τα χαρακτηριστικά των αντίστροφων λειτουργικών ρόλων.

- `private void handleDatatypeRoles()`

  Ανανεώνει τα χαρακτηριστικά των ρόλων τύπου δεδομένων.

- `private void updateDomainAndRange()`

  Ανανεώνει τα πεδία και εύρη ανάλογα με την ήδη αποθηκευμένη πληροφορία περί ισοδυναμίας, αντιστροφής και συμμετρικότητας ρόλων.

- `private void updateRolesCharacteristics()`

  Ανανεώνει τα χαρακτηριστικά των ρόλων της οντολογίας, εκτελώντας τον αλγόριθμο της Παραγράφου 6.1.3.

- `private void loadDisjointRoles(Set<OWLProperty> ontologyRoles)`

  Είναι υπεύθυνη για την φόρτωση της πλήρους πληροφορίας περί ασυμβατότητας ρόλων. Δέχεται το όρισμα ontologyRoles που είναι το ίδιο με αυτό που περιγράφτηκε στη μέθοδο createRolesHierarchy(). Στέλνει στον Pellet ζεύγη ρόλων ώστε να ελεγχθεί αν είναι ασύμβατοι. Τα ids των ζευγών ασύμβατων ρόλων αποθηκεύονται στον πίνακα “Role Disjointness” της βάσης δεδομένων. Η διαδικασία βελτιστοποιείται με τη χρήση κατάλληλου αλγόριθμου για την αποστολή στον Pellet όσο το δυνατόν λιγότερων ζευγών ρόλων προς έλεγχο.

Στο τέλος της διαδικασίας τα πεδία “r_disj_complete” του πίνακα “Roles” για όλους τους ρόλους της οντολογίας γίνονται true, καθώς η πληροφορία περί ασυμβατότητας ρόλων είναι πλέον πλήρης.

- `private void loadExplicitDisjointRoles()`

  Φορτώνει στον πίνακα “Role Disjointness” της βάσης δεδομένων μόνον τα ids των ζευγών των ρόλων που αναφέρονται ρητά στο αρχείο της οντολογίας ως ασύμβατα.

- `private void loadDisjointRolesRecursively( Object[] disjointCandidatesIds, int topID, int notTopID, int level )`

  Καλείται από τη μέθοδο loadDisjointRoles() και υλοποιεί τον αλγόριθμο της Παραγράφου 6.1.1 για την αποδοτική φόρτωση της πλήρους πληροφορίας περί ασυμβατότητας ρόλων, χρησιμοποιώντας αναδρομική τεχνική.

- `private void loadConceptsDefinitions()`

  Είναι υπεύθυνη για την αποθήκευση των ορισμών των κλάσεων. Οι ορισμοί, όπου υπάρχουν, αποθηκεύονται στο πεδίο “concept_definition” του πίνακα “Concepts” της βάσης δεδομένων.

- `private void createExpandedTBox()`

  Κατασκευάζει, με χρήση κατάλληλου αλγορίθμου (Παράγραφος 6.1.2), τον διευρυμένο ορισμό κάθε κλάσης και τον αποθηκεύει στο πεδίο “expanded_concept_definition” του πίνακα “Concepts”.

- `private void loadConceptPseudoModels()`

  Αποθηκεύει στους πίνακες “Concept PseudoModel In”, “Concept PseudoModel NotIn”, “Concept PseudoModel Existence” και “Concept PseudoModel Universial” της βάσης δεδομένων ένα ψευδομοντέλο για κάθε κλάση της οντολογίας.

  Τα ψευδομοντέλα λαμβάνονται μέ κλήση της μεθόδου returnAllConceptPseudoModelsFillers() της κλάσης ReasonerHandler. Η μέθοδος αυτή ανήκει στον ReasonerHandler, καθώς τα ψευδομοντέλα για κάθε κλάση ζητούνται από τον Pellet. Μαζί με το ψευδομοντέλο κάθε κλάσης, αποθηκεύεται και το ψευδομοντέλο της συμπληρωματικής της το οποίο επίσης υπολογίζει ο Pellet. Σε κάθε ψευδομοντέλο κλάσης δίνεται ένα μοναδικό id, καθώς αργότερα μπορεί να προστεθούν επιπλέον ψευδομοντέλα για την ίδια κλάση.

6.4.4.4 *Η κλάση ABoxLoader*

Μέλη της κλάσης

- `final DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

- `final ReasonerWrapper reasoner`

  Αντικείμενο της κλάσης ReasonerWrapper για τη σύνδεση με τον Reasoner.

- `final OWLOntologyManager manager`

  Αντικείμενο της κλάσης OWLOntologyManager, για πρόσβαση στα στοιχεία της φορτωμένης στην κύρια μνήμη οντολογίας.

- `final OWLOntology ontology`

  Αντικείμενο της κλάσης OWLOntology. Είναι η φορτωμένη στην κύρια μνήμη οντολογία.

Μέθοδοι της κλάσης

- `public ABoxLoader(String dbName, DBMSWrapper inputdbHandler, ReasonerWrapper inputreasoner, OWLOntologyManager inputmanager, OWLOntology inputontology, boolean diffInds, boolean indPM)`

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης. Δέχεται ως ορίσματα:

  - `dbName`: Το όνομα της βάσης δεδομένων.
  - `inputdbHandler`: Αντικείμενο κλάσης DBMSWrapper μέσω του οποίου γίνεται όλη η επικοινωνία της κλάσης με τη βάση δεδομένων.
  - `inputreasoner`: Αντικείμενο κλάσης ReasonerWrapper μέσω του οποίου γίνεται όλη η επικοινωνία της κλάσης με τον Pellet.
  - `inputmanager`: Αντικείμενο κλάσης OWLOntologyManager, απαραίτητο για τον χειρισμό της φορτωμένης στην κύρια μνήμη οντολογίας.
  - `inputontology`: Αντικείμενο κλάσης OWLOntology. Είναι το φορτωμένο στην κύρια μνήμη μοντέλο της οντολογίας.

- `diffInds`: Δυαδική μεταβλητή, που καθορίζει την φόρτωση ή όχι (true/false) της πλήρους πληροφορίας περί διαφορετικότητας ατόμων.
- `indPM`: Δυαδική μεταβλητή, που καθορίζει την φόρτωση ή όχι (true/false) ενός ψευδομοντέλου για κάθε άτομο της οντολογίας.

- **`private void`** `loadIndividuals()`

  Αποθηκεύει στον πίνακα "Individuals" της βάσης δεδομένων τα URIs όλων των ατόμων που αναφέρονται στο αρχείο της οντολογίας. Σε κάθε άτομο δίνεται ένα μοναδικό id.

- **`private void`** `handleSameIndividuals()`

  Ζητάει από τον Pellet τα ισοδύναμα άτομα κάθε ατόμου της οντολογίας και αποθηκεύει την εξαγόμενη πληροφορία στον πίνακα "Individuals Equivalence" σε μορφή ζευγών ids.

- **`private void`** `handleDifferentIndividuals()`

  Ζητάει από τον Pellet τα διαφορετικά άτομα κάθε ατόμου της οντολογίας και αποθηκεύει την εξαγόμενη πληροφορία στον πίνακα "Individuals Difference" σε μορφή μοναδικών ζευγών ids. Η διαδικασία βελτιστοποιείται με την αποστολή στον Pellet ενός ατόμου από κάθε σύνολο ισοδύναμων ατόμων.

  Στο τέλος της διαδικασίας τα πεδία "i_diff_complete" του πίνακα "Individuals" για όλα τα άτομα της οντολογίας γίνονται true, καθώς η πληροφορία περί διαφορετικότητας ατόμων είναι πλέον πλήρης.

- **`private void`** `handleExplicitDifferentIndividuals()`

  Ανασύρει από το αρχείο της οντολογίας όλες τις ρητές δηλώσεις διαφορετικών ατόμων και τις αποθηκεύει στον πίνακα "Individuals Difference" σε μορφή ζευγών ids.

- **`private void`** `loadConceptAssertions()`

  Αποθηκεύει την πληροφορία σχετικά με τις δηλώσεις κλάσεων (concept assertions). Η πληροφορία αποθηκεύεται στον πίνακα "Concept Assertions" με τη μορφή ζευγών ids κλάσεων και ατόμων.

- **private void** `loadRoleAssertions()`

  Αποθηκεύει την πληροφορία σχετικά με τις δηλώσεις μη μεταβατικών ρόλων αντικειμένου (non-transitive object role assertions). Η πληροφορία αποθηκεύεται στον πίνακα "Role Assertions" με τη μορφή τριάδων ids που αποτελούνται από το id του ρόλου και τα ids του ατόμου υποκείμενου και του ατόμου αντικείμενου αντιστοίχως.

  Τέλος, σβήνονται από τον πίνακα "Role Assertions" οι πιθανές διπλές δηλώσεις που προκύπτουν από συμμετρικότητα και αντιστροφή ρόλων.

- **private void** `createTransitiveRolesAssertionsHierarchy(String dbName)`

  Κατασκευάζει, για κάθε μεταβατικό ρόλο (transitive role), την ιεραρχία των ατόμων που συμμετέχουν σε αυτόν με την επανάληψη για κάθε μεταβατικό ρόλο της παρακάτω διαδικασίας:

  - Αφαιρούνται τα περιττά ισοδύναμα άτομα τα οποία δεν χρειάζονται για την κατασκευή της ιεραρχίας.
  - Επιλέγεται ένας ρόλος από το σύνολο ισοδύναμων και αντιστρόφων ρόλων του δεδομένου ρόλου για την κατασκευή της ιεραρχίας.
  - Υπολογίζονται όλες οι δηλώσεις του ρόλου που επιλέχθηκε (από τις ρητές δηλώσεις του ίδιου, των ισοδύναμών του και των αντιστρόφων του) και αποθηκεύονται σε μορφή ακμών στον προσωρινό πίνακα "Transitive Roles Temp Table".
  - Δημιουργείται ένα νέο αντικείμενο της κλάσης LabelingScheme, το οποίο, με χρήση του πίνακα "Transitive Roles Temp Table", κατασκευάζει το Interval Labeling Scheme για την αναπαράσταση και αποθήκευση της πλήρους ιεραρχίας των δηλώσεων του μεταβατικού ρόλου στη βάση δεδομένων.
  - Διαγράφεται ο πίνακας "Transitive Roles Temp Table".

- **private void** `loadDatatypeRoleAssertions()`

  Αποθηκεύει την πληροφορία σχετικά με τις δηλώσεις ρόλων τύπου δεδομένων (data type role assertions). Η πληροφορία αποθηκεύεται στον πίνακα "Datatype Role Assertions" ως εγγραφές που αποτελούνται από το id του ρόλου, το id του ατόμου υποκειμένου, τον τύπο και την τιμή του αντικειμένου ως αλφαριθμητικά.

- `private void loadIndividualPseudoModels()`

  Αποθηκεύει στους πίνακες “Individual PseudoModel In”, “Individual PseudoModel NotIn”, “Individual PseudoModel Existence” και “Individual PseudoModel Universial” της βάσης δεδομένων ένα ψευδομοντέλο για κάθε άτομο της οντολογίας.

  Τα ψευδομοντέλα λαμβάνονται με κλήση της μεθόδου returnAllIndividualPseudoModelsFillers() της κλάσης ReasonerHandler. Σε κάθε ψευδομοντέλο ατόμου δίνεται ένα μοναδικό id, καθώς αργότερα μπορεί να προστεθούν επιπλέον ψευδομοντέλα για το ίδιο άτομο.

6.4.4.5 *Η κλάση ImportIdentifier*

Μέλη της κλάσης

- `OWLOntologyManager manager`

  Αντικείμενο της κλάσης OWLOntologyManager για πρόσβαση στα στοιχεία της οντολογίας.

- `OWLOntology ontology`

  Αντικείμενο της κλάσης OWLOntology. Είναι η φορτωμένη στην κύρια μνήμη οντολογία.

Μέθοδοι της κλάσης

- `public ImportIdentifier( OWLOntology inputOntology, OWLOntologyManager inputManager)`

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης. Δέχεται δύο ορίσματα:

  - inputOntology: Αντικείμενο κλάσης OWLOntology. Είναι το φορτωμένο στην κύρια μνήμη μοντέλο της οντολογίας.
  - inputManager: Αντικείμενο κλάσης OWLOntologyManager, απαραίτητο για τον χειρισμό της φορτωμένης στην κύρια μνήμη οντολογίας.

- `public boolean identifyConceptSource( URI conceptURI, String axiom )`

  Δέχεται ως ορίσματα το URI μιας κλάσης (conceptURI) και ένα αξίωμα σε μορφή αλφαριθμητικού (axiom), στο οποίο περιέχεται το τοπικό όνομα της

κλάσης (χωρίς το namespace). Επιστρέφει true, αν η κλάση της οποίας το τοπικό όνομα υπάρχει στο αξίωμα είναι η κλάση με URI το conceptURI.

- `public boolean identifyObjectPropertySource( URI objectPropertyURI, String axiom )`

  Δέχεται ως ορίσματα το URI ενός ρόλου (roleURI) και ένα αξίωμα σε μορφή αλφαριθμητικού (axiom), στο οποίο περιέχεται το τοπικό όνομα του ρόλου (χωρίς το namespace). Επιστρέφει true, αν ο ρόλος του οποίου το τοπικό όνομα υπάρχει στο αξίωμα είναι ο ρόλος με URI το roleURI.

- `public boolean identifyDataTypePropertySource( URI dataTypePropertyURI, String axiom )`

  Λειτουργεί με τον ίδιο τρόπο που λειτουργεί η μέθοδος identifyObjectPropertySource(), αλλά για ρόλους τύπου δεδομένων.

- `public boolean identifyIndividualSource( URI individualURI, String axiom )`

  Δέχεται ως ορίσματα το URI ενός ατόμου (individualURI) και ένα αξίωμα σε μορφή αλφαριθμητικού (axiom), στο οποίο περιέχεται το τοπικό όνομα του ατόμου (χωρίς το namespace). Επιστρέφει true, αν το άτομο του οποίου το τοπικό όνομα υπάρχει στο αξίωμα είναι το άτομο με URI το individualURI.

6.4.4.6 *Η κλάση LabelingScheme*

Μέλη της κλάσης

- `final DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

- `private final int[][] graphTable`

  Δισδιάστατος πίνακας ακεραίων που περιέχει τις ακμές του γράφου της εκάστοτε ιεραρχίας ως ζεύγη ids των κόμβων του.

- `private final int[][] optimumSpanningTree`

  Δισδιάστατος πίνακας ακεραίων που περιέχει τις ακμές του βέλτιστου συνδετικού δέντρου του γράφου της εκάστοτε ιεραρχίας ως ζεύγη ids των κόμβων του δέντρου.

- **private final int**[] postorderedNodes

  Μονοδιάστατος πίνακας ακεραίων που περιέχει τα ids των κόμβων του εκάστοτε γράφου της ιεραρχίας, σε post order σειρά.

Μέθοδοι της κλάσης

- **public** LabelingScheme(DBMSWrapper inputDbHandler, **boolean** forConcepts)

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης και χρησιμοποιείται για την κατασκευή του σχήματος ετικετών της ιεραρχίας των κλάσεων και των ρόλων της οντολογίας. Δέχεται ως όρισμα ένα αντικείμενο DBMSWrapper για τη σύνδεση με τη βάση δεδομένων και μια δυαδική μεταβλητή, ανάλογα με την οποία (αν είναι true ή false) κατασκευάζεται το σχήμα ετικετών των κλάσεων ή των ρόλων αντίστοιχα. Αρχικά, φορτώνονται από τη βάση δεδομένων σε έναν δισδιάστατο πίνακα ακεραίων όλες οι ακμές του γράφου της ιεραρχίας και στη συνέχεια καλούνται κατάλληλες μέθοδοι για την κατασκευή του σχήματος ετικετών.

- **public** LabelingScheme(DBMSWrapper inputDbHandler, **int** transitiveRoleId)

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης και χρησιμοποιείται για την κατασκευή του σχήματος ετικετών της ιεραρχίας των ατόμων που συμμετέχουν σε δηλώσεις μεταβατικών ρόλων της οντολογίας. Δέχεται ως όρισμα ένα αντικείμενο DBMSWrapper για τη σύνδεση με τη βάση δεδομένων και το id ενός μεταβατικού ρόλου (όπως αυτό είναι αποθηκευμένο στη βάση), του οποίου το σχήμα ετικετών της ιεραρχίας των δηλώσεων θα κατασκευαστεί. Αρχικά, φορτώνονται από τη βάση δεδομένων σε έναν δισδιάστατο πίνακα ακεραίων όλες οι ακμές του γράφου της ιεραρχίας και στη συνέχεια καλούνται κατάλληλες μέθοδοι για την κατασκευή του σχήματος ετικετών.

- **private int**[] topologicalSortForConcepts()

  Αναλύει το δισδιάστατο πίνακα ακεραίων, ο οποίος περιέχει τις ακμές του γράφου της ιεραρχίας των κλάσεων της οντολογίας και επιστρέφει έναν μονοδιάστατο πίνακα ακεραίων που περιέχει τους κόμβους του σε τοπολογική σειρά (topological order). Η διάταξη των κόμβων γίνεται βάσει

του αλγορίθμου που δίνεται στο [ABJ89] και ο οποίος έχει υλοποιηθεί ώστε να εκτελείται σε SQL.

- `private int[] topologicalSortForRoles(int[][] graphTable)`

  Ομοίως με την προηγούμενη μέθοδο, για την ιεραρχία όμως των ρόλων.

- `private int[] topologicalSortForTRoles(int[][] graphTable )`

  Ομοίως με την προηγούμενη μέθοδο, για την ιεραρχία όμως των ατόμων που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

- `private int[][] getOptimumSpanningTree( int[] sortedNodes, boolean transitive )`

  Δέχεται ως ορίσματα τον μονοδιάστατο πίνακα ακεραίων που περιέχει τους κόμβους του γράφου σε τοπολογική σειρά και μια δυαδική μεταβλητή, η οποία αν είναι true σημαίνει ότι αναφερόμαστε σε ιεραρχία δηλώσεων μεταβατικού ρόλου. Επιστρέφει έναν δισδιάστατο πίνακα ακεραίων που περιέχει τις ακμές του βέλτιστου συνδετικού δέντρου το οποίο υπολογίζεται με τον αλγόριθμο που δίνεται στο [ABJ89] και λαμβάνοντας υπ' όψιν τον δισδιάστατο πίνακα που περιέχει τις ακμές του γράφου της ιεραρχίας. Τέλος, ο πίνακας που περιέχει το συνδετικό δέντρο αποθηκεύεται σε ένα προσωρινό πίνακα της βάσης δεδομένων, για χρήση από τις επόμενες μεθόδους.

- `private void postorderSortedOptimumSpanningTree(int position, int node)`

  Διασχίζει αναδρομικά κατά post order τους κόμβους του βέλτιστου συνδετικού δέντρου και αποθηκεύει τη διάταξη σε μορφή μονοδιάστατου πίνακα ακεραίων.

- `private void loadConceptLabels()`

  Επιτελεί την βασική εργασία κατασκευής του σχήματος ετικετών της ιεραρχίας των κλάσεων, υλοποιώντας τον αλγόριθμο [ABJ89]. Η πληροφορία αποθηκεύεται στον πίνακα "Concepts" της βάσης δεδομένων.

- `private void loadRoleLabels()`

  Ομοίως με προηγουμένως, για τους ρόλους όμως της οντολογίας.

- `private void loadTRoleLabels()`

  Ομοίως με προηγουμένως, για τα άτομα όμως που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

- `private void dagConceptsPropagation()`

  Ολοκληρώνει την κατασκευή του σχήματος ετικετών για την ιεραρχία των κλάσεων με την διάδοση των ετικετών κάθε κόμβου του γράφου στους προγόνους του (πρόγονοι με τους οποίους συνδέεται διαμέσου ακμών που δεν ανήκουν στο συνδετικού δέντρου). Η διαδικασία αυτή επιτελείται με την κλήση της αναδρομικής μεθόδου conceptsPropagateRecursively().

- `private void dagRolesPropagation()`

  Ομοίως με προηγουμένως, για την ιεραρχία όμως των ρόλων.

- `private void dagTRolesPropagation(int transitiveRoleId)`

  Ομοίως με προηγουμένως, για την ιεραρχία όμως των ατόμων που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

- `private void conceptsPropagateRecursively(int nonTreeEdges[][], int childId, int[] labels )`

  Υλοποιεί την αναδρομική τεχνική διάδοσης των ετικετών της ιεραρχίας των κλάσεων. Αποθηκεύει τη νέα πληροφορία στον πίνακα "DAG Concept Labels".

- `private void rolesPropagateRecursively(int nonTreeEdges[][], int childId, int[] labels )`

  Ομοίως με προηγουμένως, για την ιεραρχία όμως των ρόλων.

- `private void tRolesPropagateRecursively(int transitiveRoleId, int nonTreeEdges[][], int childId, int[] labels )`

  Ομοίως με προηγουμένως, για την ιεραρχία όμως των ατόμων που συμμετέχουν σε δηλώσεις μεταβατικού ρόλου.

6.4.4.7 *Η κλάση SystemUtils*

Μέλη της κλάσης

- `static final String filesPath`

  Η διαδρομή των text αρχείων αποθήκευσης δηλώσεων των μεταβατικών ρόλων για τους οποίους έχει κατασκευαστεί σχήμα ετικετών.

- `OWLDataFactory dataFactory`

  Αντικείμενο της κλάσης OWLDataFactory, για την κατασκευή τύπων του OWL API.

- `final DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

Μέθοδοι της κλάσης

- `public SystemUtils(DBMSWrapper inputDBMSWrapper, OWLDataFactory factory)`

  Δημιουργεί ένα νέο αντικείμενο της κλάσης. Δέχεται ως όρισμα ένα αντικείμενο κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων και ένα αντικείμενο τύπου OWLDataFactory, απαραίτητο για την κατασκευή αντικειμένων τύπου OWL API.

- `public OWLDescription makeOWLDescriptionFromString(String expression, String ontologyNamespace)`

  Μετατρέπει μια έκφραση υπό μορφή αλφαριθμητικού σε αντικείμενο τύπου OWL API, καλώντας την makeOWLDescriptionRecursively(). Δέχεται ως επιπλέον όρισμα το namespace της οντολογίας στην οποία αναφέρεται η έκφραση σε μορφή αλφαριθμητικού (ontologyNamespace).

- `private OWLDescription makeOWLDescriptionRecursively(String expression, String ontologyNamespace)`

  Καλείται από την makeOWLDescriptionFromString() και υλοποιεί μια αναδρομική τεχνική για την μετατροπή μιας έκφρασης από αλφαριθμητικό σε αντικείμενο OWL API.

- `public void updateInverseRolesDomainAndRange()`

  Ανανεώνει το πεδίο και εύρος των ρόλων βάσει των σχέσεων αντιστροφής.

- `public void updateSymmetricRolesDomainAndRange()`

  Ανανεώνει το πεδίο και εύρος βάσει των σχέσεων συμμετρικότητας.

- `public boolean updateTransitiveRoleAssertionsLabelingScheme(String dbName, String tRoleURI)`

  Ανακατασκευάζει το σχήμα ετικετών ενός επιλεγμένου από τον χρήστη μεταβατικού ρόλου.

- `private HashSet<String[]> createUpdatedTransitiveAssertionsSet(String dbName, String roleName, short namespaceId)`

  Καλείται από την ποηγούμενη μέθοδο και κατασκευάζει το σύνολο ακμών του νέου γράφου ιεραρχίας για τον οποίο πρέπει να ανακατασκευαστεί το σχήμα ετικετών.

- `public void loadConceptPseudoModels( int conceptDBID, ReasonerWrapper reasonerWrapper )`

  Ανασύρει από τον Pellet το ψευδομοντέλο της κλάσης με id (στη βάση δεδομένων) το conceptDBID και το αποθηκεύει. Δέχεται, επίσης, ως όρισμα τον reasonerWrapper της οντολογίας στην οποία αναφερόμαστε.

- `public void loadIndividualPseudoModels( int individualDBID, ReasonerWrapper reasonerWrapper )`

  Φορτώνει στη βάση δεδομένων το ψευδομοντέλο του ατόμου της οντολογίας με id (στη βάση δεδομένων) το individualDBID. Δέχεται, επίσης, ως όρισμα τον reasonerWrapper της οντολογίας στην οποία αναφερόμαστε.

- `private boolean isAtomicConcept(String expression, String ontologyNamespace)`

  Επιστρέφει true, αν το αλφαριθμητικό expression είναι γνωστή κλάση της οντολογίας με namespace ontologyNamespace.

- `public int matchArbitraryConceptToExistentDefinition(String arbitraryExpression)`

  Δέχεται ως όρισμα μια εννοιακή έκφραση και αν βρει κάποια γνωστή κλάση με τον ίδιο ορισμό επιστρέφει το id της. Σε αντίθετη περίπτωση επιστρέφει -1. Η ταυτοποίηση ορισμών γίνεται έπειτα από διεύρυνση και κανονικοποίησή τους, προκειμένου να αυξηθούν οι πιθανότητες επιτυχίας. Η μέθοδος χρησιμοποιείται όταν ο χρήστης εισάγει μια αυθαίρετη κλάση ως όρισμα σε κάποιο ερώτημα, ώστε το ερώτημα να μην αποσταλεί κατ' ευθείαν στον reasoner, αλλά να επιχειρηθεί να απαντηθεί πρώτα από το DBRS.

- `public String expandDefinition(String expression)`

  Δέχεται ως όρισμα μια εννοιακή έκφραση και την επιστρέφει σε διευρυμένη μορφή. Η διεύρυνση γίνεται με επαναληπτικές αντικαταστάσεις των εννοιών που περιέχονται στην έκφραση από τους ορισμούς τους, έως ότου η έκφραση

να περιέχει πλέον αποκλειστικά ατομικές κλάσεις (atomic concept), πρωταρχικές κλάσεις δηλαδή, που δεν διαθέτουν ορισμό.

- `public HashSet<String> returnDescriptionRelatedAxioms(String desc, short namespaceId)`

  Δέχεται ως ορίσματα μια έκφραση σε μορφή αλφαριθμητικού και το id του URI της οντολογίας στην οποία αναφέρεται η έκφραση αυτή. Επιστρέφει ένα Java Set με τα αξιώματα της οντολογίας, μέσα στα οποία συναντάται η έκφραση, σε μορφή αλφαριθμητικών.

- `public static SortedSet<String> getExpressionsParts(String expression, boolean analyzeRoles)`

  Δέχεται ως όρισμα μια έκφραση σε μορφή αλφαριθμητικού (expression) και επιστρέφει ένα sorted Java Set με τα μέρη του πρώτου επιπέδου της σε μορφή αλφαριθμητικού.

  Για παράδειγμα, αν definition: And(Or(C_3 C_4) C_1 C_2), τότε result: [C_1 , C_2 , Or(C_3 C_4)]. Επίσης, η μέθοδος δέχεται άλλο ένα όρισμα δυαδικής μορφής (analyzeRoles), το οποίο καθορίζει αν η μέθοδος θα αναλύσει ή όχι (true/false) και τις εκφράσεις ρόλων. Για παράδειγμα, αν definition: ObjectSome(hasChild woman) και analyzeRoles=true, τότε result: [hasChild , woman].

- `public static String cleanPart(String part)`

  Καθαρίζει ένα αλφαριθμητικό από χαρακτήρες κενού στην αρχή και το τέλος του.

- `public static boolean isAnd(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση τομής (π.χ. And(Or(C_3 C_4) C_1 C_2)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isOr(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ένωσης (π.χ. Or(C_3 C_4)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isNot(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση άρνησης (π.χ. Not(Or(C_3 C_4) C_1 C_2)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isRoleExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ρόλου (π.χ. ObjectSome(R C)) και σε αυτή την περίπτωση επιστρέφει true. Αναγνωρίζει τα προθέματα:

  - ObjectSome
  - ObjectAll
  - ObjectMin
  - ObjectMax
  - DataSome
  - DataAll
  - DataMin
  - DataMax

- `public static boolean isObjectSomeExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση μερικής συμμετοχής ρόλου αντικειμένου (π.χ. ObjectSome(R C)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isObjectAllExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ολικής συμμετοχής ρόλου αντικειμένου (π.χ. ObjectAll(R C)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isObjectMinExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση κατώτατου ορίου συμμετοχής ρόλου αντικειμένου (π.χ. ObjectMin(1 R C)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isObjectMaxExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ανώτατου ορίου συμμετοχής ρόλου αντικειμένου (π.χ. ObjectMax(3 R C)) και σε αυτή την περίπτωση επιστρέφει true.

- `public static boolean isDataSomeExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση μερικής συμμετοχής ρόλου τύπου δεδομένων (π.χ. DataSome(R string)) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isDataAllExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ολικής συμμετοχής ρόλου τύπου δεδομένων (π.χ. DataAll(R integer)) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isDataMinExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση κατώτατου ορίου συμμετοχής ρόλου τύπου δεδομένων (π.χ. DataMin(1 R string)) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isDataMaxExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ανώτατου ορίου συμμετοχής ρόλου τύπου δεδομένων (π.χ. DataMax(3 R short)) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isObjectValueExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ανάθεσης τιμής (π.χ. ObjectValue(brother1 "Nikos")) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isObjectEnumerationExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση απαρίθμησης (π.χ. ObjectEnumeration(Red White Rose)) και σε αυτή την περίπτωση επιστρέφει true.

- **public static boolean** `isNominalExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ανάθεσης τιμής ή έκφραση απαρίθμησης και σε περίπτωση που ισχύει κάποιο από τα δύο επιστρέφει true.

- **public static boolean** `isRoleCardinalityExpression(String expression)`

  Αναγνωρίζει αν η έκφραση είναι έκφραση ανωτάτου ή κατωτάτου ορίου συμμετοχής ρόλου και σε αυτή την περίπτωση επιστρέφει true.

- **public static** `String normalizeExpression(String expression)`

  Δέχεται ως όρισμα μια έκφραση σε μορφή αλφαριθμητικού (expression) και την επιστρέφει σε κανονικοποιημένη μορφή, καλώντας τις αναδρομικές

μέθοδους normalizeAndExpressionRecursively() και normalizeOrExpression Recursively(). Για παράδειγμα, αν expression: And(And(Or(B A Or(C D) D) E) F) τότε result: And(E F Or(A B C D)).

- `private static String normalizeAndExpressionRecursively(String expression)`

  Κανονικοποιεί μια έκφραση τομής που της δίνεται ως όρισμα σε μορφή αλφαριθμητικού, υλοποιώντας αναδρομική τεχνική.

- `private static String normalizeOrExpressionRecursively(String expression)`

  Κανονικοποιεί μια έκφραση ένωσης που της δίνεται ως όρισμα σε μορφή αλφαριθμητικού, υλοποιώντας αναδρομική τεχνική.

- `public static String getNegationNormalForm(String expression)`

  Δέχεται ως όρισμα μια έκφραση άρνησης σε μορφή αλφαριθμητικού (expression) και την επιστρέφει σε κανονικοποιημένη μορφή. Για παράδειγμα, αν

  expression: Not(ObjectSome(R C)), τότε result: ObjectAll(R C).

- `public static String replace(String expression, String toReplace, String replacement)`

  Επιστρέφει το αλφαριθμητικό expression, έχοντας αντικαταστήσει σε αυτό όλες τις ταυτόσημες με το αλφαριθμητικό toReplace λέξεις, με το αλφαριθμητικό replacement.

- `public static void createTransitiveAssertionsFile(String dbName, String roleName, short namespaceId)`

  Δημιουργεί το text αρχείο που θα αποθηκευτούν οι δηλώσεις ενός μεταβατικού ρόλου για τον οποίο θα κατασκευαστεί σχήμα ετικετών.

- `public static void deleteTransitiveAssertionsFiles(File path)`

  Διαγράφει ένα αρχείο που δημιούργησε η προηγούμενη εφαρμογή.

- `public static void fillTransitiveAssertionsFile(String dbName, String roleName, short namespaceId, HashSet<String[]> tRoleAssertions)`

  Εισάγει δηλώσεις, υπό μορφή ζευγών ids, στο text αρχείο.

- **public static** String getOWLEntityNamespace(URI entityURI)

  Δέχεται ως όρισμα το URI μιας οντότητας της οντολογίας και επιστρέφει το namespace της σε μορφή αλφαριθμητικού.

- **public static** String getOWLEntityLocalName(URI entityURI)

  Δέχεται ως όρισμα το URI μιας οντότητας της οντολογίας και επιστρέφει το τοπικό της όνομα σε μορφή αλφαριθμητικού.

- **public static** HashSet<String[]> viewDataBases(String userName, String userPassword)

  Δέχεται ως ορίσματα το όνομα και τον κωδικό του χρήστη, σε μορφή αλφαριθμητικών και επιστρέφει ένα Java Set το οποίο περιέχει δισδιάστατους πίνακες αλφαριθμητικών, με το όνομα και τον ιδιοκτήτη κάθε βάσης δεδομένων που υπάρχει στο DBMS. Λειτουργεί καλώντας τη μέθοδο getDataBasesInfo() ενός νέου αντικείμενου κλάσης DBMSWrapper, με ορίσματα userName και userPassword.

6.4.4.8 Η κλάση SystemQueryEngine

Μέλη της κλάσης

- String ontologyNamespace

  Το namespace της οντολογίας.

- **static short** *ontologyNamespaceDBId*

  Το id του namespace της οντολογίας στη βάση δεδομένων.

- String queryString

  Το ερώτημα όπως το πληκτρολόγησε ο χρήστης.

- String argument1

  Το πρώτο όρισμα του ερωτήματος, όπως το πληκτρολόγησε ο χρήστης.

- String argument2

  Το δεύτερο όρισμα του ερωτήματος, όπως το πληκτρολόγησε ο χρήστης.

- String argument3

  Το δεύτερο όρισμα του ερωτήματος, όπως το πληκτρολόγησε ο χρήστης.

- **short** queryType

  Ο τύπος του ερωτήματος όπως αναγνωρίσθηκε από την initializeQueryClassMembers().

- `SortedSet<String> querySetAnswer`

  Εδώ αποθηκεύονται τα αποτελέσματα της αποτίμησης του ερωτήματος.

- `ReasonerWrapper reasoner`

  Αντικείμενο της κλάσης ReasonerWrapper για τη σύνδεση με τον Reasoner.

- `boolean lazyMode`

  Δυαδική μεταβλητή, αν η τιμή της οποίας είναι true, τα ερωτήματα αποτιμώνται αποκλειστικά με χρήση του Reasoner.

- `OWLDataFactory dataFactory`

  Αντικείμενο της κλάσης OWLDataFactory για την κατασκευή τύπων του OWL API.

- `DBMSWrapper dbHandler`

  Αντικείμενο της κλάσης DBMSWrapper για τη σύνδεση με τη βάση δεδομένων.

- `SystemUtils utils`

  Αντικείμενο της κλάσης SystemUtils για την πρόσβαση στα εργαλεία της κλάσης.

Μέθοδοι της κλάσης

- `public SystemQueryEngine(short ontNamespaceDBId, ReasonerWrapper inputReasoner, OWLDataFactory factory, boolean lazy, String databaseName, String username, String password)`

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης. Επίσης, κατασκευάζει και ένα νέο αντικείμενο κλάσης DBMSWrapper για τη σύνδεση με το DBMS. Δέχεται τα παρακάτω ορίσματα:

  - `ontNamespaceDBId`: Είναι το id της οντολογίας (στην βάση δεδομένων) στην οποία αναφέρονται τα ερωτήματα.
  - `inputReasoner`: Είναι ένα στιγμιότυπο του Pellet, απαραίτητο για την αποστολή ερωτημάτων προς αυτόν.
  - `factory`: Είναι μια μηχανή κατασκευής δεδομένων τύπου OWL API.
  - `lazy`: Είναι μια δυαδική μεταβλητή, αν η τιμή της οποίας είναι true, σημαίνει ότι η Μηχανή Ερωτημάτων θα αποτιμάει τα ερωτήματα δια της «οκνηρής» μεθόδου.

- `databaseName`: Είναι το όνομα της βάσης δεδομένων, σε μορφή αλφαριθμητικού, απαραίτητο για τη διασύνδεση με το DBMS.
- `username`: Είναι το όνομα του χρήστη, σε μορφή αλφαριθμητικού, απαραίτητο για τη διασύνδεση με το DBMS.
- `password`: Είναι ο κωδικός του χρήστη, σε μορφή αλφαριθμητικού, απαραίτητος για τη διασύνδεση με το DBMS.

- **`private void`** `initializeQueryClassMembers()`

  Αρχικοποιεί, στην αρχή της διαδικασίας αποτίμησης κάθε νέου ερωτήματος, τα πεδία της κλάσης που περιέχουν το ερώτημα σε μορφή αλφαριθμητικού (String queryString), τα ορίσματα του ερωτήματος (String argument1, argument2, argument3), τον τύπο του ερωτήματος (short queryType) και το σύνολο απάντησης (SortedSet<String> querySetAnswer).

- **`private boolean`** `fillClassQueryMembers(String query)`

  Δέχεται ως όρισμα το ερώτημα που έχει υποβληθεί από το χρήστη σε μορφή αλφαριθμητικού και αναγνωρίζει τον τύπο του και τις παραμέτρους του. Για την αναγνώριση του ερωτήματος χρησιμοποιεί τις παρακάτω μεθόδους, η λειτουργία των οποίων δηλώνεται από το όνομά τους:

  - **`private boolean`** `isDatatypeArgument(String argument)`
  - **`private boolean`** `isEquivalentClassQuery(String query)`
  - **`private boolean`** `isSubClassOfQuery(String query)`
  - **`private boolean`** `isDisjointWithQuery(String query)`
  - **`private boolean`** `isComplementOfQuery(String query)`
  - **`private boolean`** `isEquivalentPropertyQuery(String query)`
  - **`private boolean`** `isSubPropertyQuery(String query)`
  - **`private boolean`** `isInverseOfQuery(String query)`
  - **`private boolean`** `isSymmetricPropertyQuery(String query)`
  - **`private boolean`** `isTransitivePropertyQuery(String query)`
  - **`private boolean`** `isInverseFunctionalPropertyQuery(String query)`
  - **`private boolean`** `isFunctionalObjectPropertyQuery(String query)`
  - **`private boolean`** `isObjectPropertyQuery(String query)`
  - **`private boolean`** `isDatatypePropertyQuery(String query)`

- **private boolean** isFunctionalDatatypePropertyQuery(String query)
- **private boolean** isDomainQuery(String query)
- **private boolean** isRangeQuery(String query)
- **private boolean** isTypeQuery(String query)
- **private boolean** isSameAsQuery(String query)
- **private boolean** isDifferentFromQuery(String query)
- **private boolean** isPropertyValueQuery(String query)
- **private boolean** isDescriptionRelatedAxiomsQuery(String query)
- **private boolean** isRoleRelatedAxiomsQuery(String query)
- **private boolean** isIndividualRelatedAxiomsQuery(String query)

- **public** SortedSet<String> executeQuery(String query)

  Ανάλογα με τον τύπο του ερωτήματος, καλεί την αντίστοιχη μέθοδο αποτίμησης.

Οι παρακάτω μέθοδοι αποτελούν σημαντικό τμήμα του DBRS και η αναλυτική περιγραφή της λειτουργίας τους έχει δοθεί στο Κεφάλαιο 5. Στην αρχή αποτίμησης κάθε ερωτήματος, αναγνωρίζονται τα ορίσματά του με κλήση κατάλληλων μεθόδων της κλάσης DBMSWrapper, ανάμεσα στα:

- Γνωστή κλάση της οντολογίας
- Γνωστός ρόλος της οντολογίας
- Γνωστό άτομο της οντολογίας
- Αυθαίρετη κλάση
- Μεταβλητή (?)

Για τα παρακάτω χρησιμοποιούμε του εξής συμβλισμούς:

- C_1, C_2 ήδη ορισμένες κλάσεις της οντολογίας που υπάρχουν στη βάση δεδομένων
- R_1, R_2 ρόλοι της οντολογίας
- a, b άτομα της οντολογίας
- C_a αυθαίρετη κλάση της OWL-DL
- $?$ μεταβλητή

- `private void handleEquivalentClassQuery()`

  Χειρίζεται τις περιπτώσεις:

  - EquivalentClass(C_1 , C_2)
  - EquivalentClass(C_{a1} , C_{a2})
  - EquivalentClass(C_1 , ?)
  - EquivalentClass(C_a , ?)

- `private void handleEquivalentClassOneArbitraryQuery( int argument1DBID, String argument2 )`

  Χειρίζεται την περίπτωση:

  - EquivalentClass(C_1 , C_a)

- `private void handleComplementOfQuery()`

  Χειρίζεται τις περιπτώσεις:

  - ComplementOf(C_{a1} , C_{a2})
  - ComplementOf(C_a , ?)

- `private void handleComplementOfOneArgumentQuery(int argumentDBID)`

  Χειρίζεται την περίπτωση:

  - ComplementOf(C_1 , ?)

- `private void handleComplementOfTwoArgumenstQuery( int argument1DBID, int argument2DBID )`

  Χειρίζεται την περίπτωση:

  - ComplementOf (C_1 , C_2)

- `private void handleComplementOfOneArbitraryQuery( int argument1DBID, String argument2 )`

  Χειρίζεται την περίπτωση:

  - ComplementOf (C_1 , C_a)

- `private void handleSubClassOfQuery()`

  Χειρίζεται τις περιπτώσεις:

  - SubClassOf(C_1 , C_2)
  - SubClassOf(C_{a1} , C_{a2})
  - SubClassOf(C_1 , ?)

- SubClassOf(C_a , ?)
- SubClassOf(? , C_1)
- SubClassOf(? , C_a)

- `private void handleSubClassOfOneArbitraryQuery(int conceptDBID, String arbitraryExpression, boolean arbitraryFirst)`

  Χειρίζεται την περίπτωση:

  - SubClassOf(C_1 , C_a)

- `private void handleDisjointWithQuery()`

  Έπειτα από τη διαδικασία αναγνώρισης των ορισμάτων, καλεί είτε την handleDisjointWithClassesQuery(), είτε την handleDisjointWithRolesQuery(), ανάλογα με το αν τα ορίσματα είναι κλάσεις ή ρόλοι της οντολογίας αντίστοιχα.

- `private void handleDisjointWithClassesQuery(int argument1DBID, int argument2DBID)`

  Χειρίζεται τις περιπτώσεις:

  - DisjointWith(C_{a1} , C_{a2})
  - DisjointWith(C_a , ?)

- `private void handleDisjointWithClassesOneArgumentQuery(int argumentDBID)`

  Χειρίζεται την περίπτωση:

  - DisjointWith(C_1 , ?)

- `private void handleDisjointWithClassesTwoArgumentsQuery(int argument1DBID,int argument2DBID)`

  Χειρίζεται την περίπτωση:

  - DisjointWith(C_1 , C_2)

- `private void handleDisjointWithForClassesOneArbitraryQuery(int argument1DBID, String argument2)`

  Χειρίζεται την περίπτωση:

  - DisjointWith(C_1 , C_a)

- `private void handleDisjointWithRolesQuery(int argument1DBID, int argument2DBID)`

  Χειρίζεται την περίπτωση:

  - DisjointWith(R_1 , R_2)

- `private void handleDisjointWithRolesOneArgumentQuery(int argumentDBID)`

  Χειρίζεται την περίπτωση:

  - DisjointWith(R_1 , ?)

- `private void handleEquivalentPropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - EquivalentProperty(R_1 , R_2)
  - EquivalentProperty(R_1 , ?)

- `private void handleSubPropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - SubPropertyOf(R_1 , R_2)
  - SubPropertyOf(R_1 , ?)
  - SubPropertyOf(? , R_1)

- `private void handleInverseOfQuery()`

  Χειρίζεται τις περιπτώσεις:

  - InverseOf(R_1 , R_2)
  - InverseOf(R_1 , ?)

- `private void handleSymmetricQuery()`

  Χειρίζεται τις περιπτώσεις:

  - SymmetricProperty(R_1)
  - SymmetricProperty (?)

- `private void handleTransitiveQuery()`

  Χειρίζεται τις περιπτώσεις:

  - TransitiveProperty(R_1)
  - TransitiveProperty (?)

- `private void handleFunctionalObjectPropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - FunctionalObjectProperty(R_1)
  - FunctionalObjectProperty (?)

- `private void handleFunctionalDatatypePropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - FunctionalDatatypeProperty(R_1)
  - FunctionalDatatypeProperty (?)

- `private void handleInverseFunctionalQuery()`

  Χειρίζεται τις περιπτώσεις:

  - InverseFunctionalProperty(R_1)
  - InverseFunctionalProperty (?)

- `private void handleObjectPropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - ObjectProperty(R_1)
  - ObjectProperty (?)

- `private void handleDatatypePropertyQuery()`

  Χειρίζεται τις περιπτώσεις:

  - DatatypeProperty(R_1)
  - DatatypeProperty (?)

- `private void handleDomainQuery()`

  Χειρίζεται τις περιπτώσεις:

  - Domain(C)
  - Domain(C_a)

- `private void handleRangeQuery()`

  Χειρίζεται τις περιπτώσεις:

  - Range(C)
  - Range(C_a)

- `private void handleTypeQuery()`

  Χειρίζεται τις περιπτώσεις:

  - Type(a , C_a)
  - Type(? , C_a)
  - Type(a , ?)

- `private void handleTypeOneArgumentQuery(int argument2DBID)`

  Χειρίζεται την περίπτωση:

  - Type(? , C)

- `private void handleTypeTwoArgumentsQuery(int argument1DBID, int argument2DBID)`

  Χειρίζεται την περίπτωση:

  - Type(a , C)

- `private void handleSameAsQuery()`

  Χειρίζεται τις περιπτώσεις:

  - SameAs(a , b)
  - SameAs(a , ?)

- `private void handleDifferentFromQuery()`

  Χειρίζεται την περίπτωση:

  - DifferentFrom(a , b)

- `private void handleDifferentFromOneArgumentQuery(int argumentDBID, String individualName)`

  Χειρίζεται την περίπτωση:

  - DifferentFrom (a , ?)

- `private void handlePropertyValueQuery()`

  Χειρίζεται τις περιπτώσεις:

  - PropertyValue(a , R , b)
  - PropertyValue(a , ? , b)
  - PropertyValue(? , R , ?)
  - PropertyValue(a , ? , ?)
  - PropertyValue(? , ? , b)

- `private void handlePropertyValueReturnSubjectsQuery(int argument2DBID, int argument3DBID, boolean printRole)`

  Χειρίζεται την περίπτωση:

  - PropertyValue(? , R , b)

- `private void handlePropertyValueReturnObjectsQuery(int argument1DBID, int argument2DBID, boolean printRole)`

  Χειρίζεται την περίπτωση:

  - PropertyValue(a , R , ?)

- `private void handleDescriptionRelatedAxiomsQuery()`

  Χειρίζεται τις περιπτώσεις:

  - DescriptionRelatedAxioms(C_a)
  - DescriptionRelatedAxioms(C)

- `private void handleRoleRelatedAxiomsQuery()`

  Χειρίζεται την περίπτωση:

  - RoleRelatedAxioms(R)

- `private void handleIndividualRelatedAxiomsQuery()`

  Χειρίζεται την περίπτωση:

  - IndividualRelatedAxioms(a)

- `private HashSet<String> selectConceptCandidates(String individualURI, HashSet<String> knownConcepts)`

  Εκτελεί mergable test και επιλέγει τις κλάσεις της οντολογίας που πιθανώς είναι τύποι ενός δοσμένου ατόμου.

- `private HashSet<String> selectIndividualCandidates()`

  Εκτελεί mergable test και επιλέγει τα άτομα της οντολογίας που πιθανώς ανήκουν σε μια γνωστή ή αυθαίρετη κλάση.

- `private HashSet<String> selectIndividualCandidates(String conceptURI, HashSet<String> knownIndividuals)`

  Εκτελεί mergable test και επιλέγει τα άτομα της οντολογίας που πιθανώς ανήκουν σε μια γνωστή κλάση με URI το conceptURI.

6.4.4.9 *Η κλάση ReasonerWrapper*

Μέλη της κλάσης

- `final ReasonerWrapper reasoner`

  Αντικείμενο της κλάσης ReasonerWrapper για τη σύνδεση με τον Reasoner.

- `final OWLDataFactory factory`

  Αντικείμενο της κλάσης OWLDataFactory, για την κατασκευή τύπων του OWL API.

Μέθοδοι της κλάσης

- `public ReasonerWrapper(OWLOntologyManager manager)`

  Κατασκευάζει ένα νέο αντικείμενο της κλάσης. Δέχεται ως όρισμα ένα αντικείμενο κλάσης OWLOntologyManager, απαραίτητο για τη διασύνδεση με το φορτωμένο στη μνήμη μοντέλο της οντολογίας και για τη χρήση της μηχανής κατασκευής αντικειμένων OWL API (OWLDataFactory) που διαθέτει. Επίσης, κατασκευάζει ένα νέο αντικείμενο της κλάσης Reasoner του Pellet, απαραίτητο για τη σύνδεση με τον Pellet.

- `public ATermAppl convertOWLClassToATermAppl(OWLClass owlConcept)`

  Μετατρέπει ένα αντικείμενο κλάσης οντολογίας τύπου OWLClass σε αντικείμενο τύπου ATermAppl.

- `public ATermAppl convertOWLPropertyToATermAppl( OWLProperty owlProperty )`

  Μετατρέπει ένα αντικείμενο ρόλου τύπου OWLProperty σε αντικείμενο τύπου ATermAppl.

- `public ATermAppl convertOWLDataPropertyToATermAppl( OWLDataProperty owlProperty )`

  Μετατρέπει ένα αντικείμενο ρόλου τύπου δεδομένων τύπου OWLDataProperty, σε αντικείμενο τύπου ATermAppl.

- `public ATermAppl convertOWLIndividualToATermAppl( OWLIndividual owlIndividual )`

  Μετατρέπει ένα αντικείμενο ατόμου τύπου OWLIndividual σε αντικείμενο τύπου ATermAppl.

- `public ATermAppl convertOWLTypedConstantToATermAppl( OWLTypedConstant owlConstant )`

  Μετατρέπει ένα αντικείμενο literal τύπου OWLTypedConstant σε αντικείμενο τύπου ATermAppl.

- `public ATermAppl makeATermApplFromString(String expression, String ontologyNamespace)`

  Δέχεται ως ορίσματα μια έκφραση και το namespace της οντολογίας στην οποία αυτή ανήκει (σε μορφή αλφαριθμητικών) και επιστρέφει την έκφραση σε μορφή ATermAppl.

- `private ATermAppl makeATermApplRecursively(String expression, SystemUtils utils, String ontologyNamespace)`

  Δέχεται ως ορίσματα μια έκφραση και το namespace της οντολογίας στην οποία αυτή ανήκει, σε μορφή αλφαριθμητικών, καθώς και ένα αντικείμενο τύπου SystemUtils. Μετατρέπει την έκφραση σε μορφή ATermAppl με χρήση αναδρομικής τεχνικής και την επιστρέφει. Καλείται από την makeATermApplFromString().

- `public boolean isSatisfiable(String expression, String ontologyNamespace)`

  Δέχεται ως ορίσματα μια εννοιακή έκφραση και το namespace της οντολογίας στην οποία αυτή ανήκει (σε μορφή αλφαριθμητικών) και επιστρέφει true αν η έκφραση είναι ικανοποιήσιμη.

- `public HashSet<OWLClass> getSubClasses( OWLClass owlConcept, boolean direct )`

  Δέχεται ως όρισμα ένα αντικείμενο κλάσης τύπου OWLClass και επιστρέφει ένα σύνολο Java Set, το οποίο περιέχει τις υποκλάσεις αυτού, υπό μορφή OWLClass επίσης. Αν επιπλέον το δυαδικό όρισμα direct είναι true, επιστρέφονται μόνον οι απ' ευθείας υποκλάσεις.

- `public HashSet<OWLClass> getSubClasses( String owlConcept, String ontologyNamespace, boolean direct )`

  Επιτελεί ακριβώς την ίδια λειτουργία με την προηγούμενη μέθοδο, μόνο που αντί για ένα όρισμα τύπου OWLClass, δέχεται δυο ορίσματα σε μορφή αλφαριθμητικού. Το τοπικό όνομα της κλάσης (owlConcept) και το namespace της οντολογίας στην οποία ανήκει (ontologyNamespace).

- `public HashSet<OWLClass> getSuperClasses( OWLClass owlConcept, boolean direct )`

  Ομοίως με την αντίστοιχη getSubClasses(), αλλά για υπερκλάσεις.

- `public HashSet<OWLClass> getSuperClasses( String owlConcept, String ontologyNamespace, boolean direct )`

  Ομοίως με την αντίστοιχη getSubClasses(), αλλά για υπερκλάσεις.

- `public HashSet<OWLProperty> getSuperProperties( OWLProperty owlProperty, boolean direct )`

  Ομοίως με την αντίστοιχη getSuperClasses(), αλλά για ρόλους.

- `public HashSet<OWLProperty> getSubProperties( OWLProperty owlProperty, boolean direct )`

  Ομοίως με την αντίστοιχη getSubClasses(), αλλά για ρόλους.

- `public HashSet<OWLClass> getDisjointClasses(OWLClass owlClass)`

  Δέχεται ως όρισμα ένα αντικείμενο κλάσης οντολογίας (τύπου OWLClass) και επιστρέφει ένα σύνολο Java Set με τις ασύμβατες αυτής, σε μορφή OWLClass.

- `public HashSet<OWLProperty> getInverses(OWLProperty owlProperty)`

  Δέχεται ως όρισμα ένα αντικείμενο ρόλου (τύπου OWLProperty) και επιστρέφει ένα σύνολο Java Set με τους αντίστροφους αυτού, σε μορφή OWLProperty.

- `public HashSet<OWLObjectProperty> getDisjointObjectProperties( OWLObjectProperty owlProperty )`

  Δέχεται ως όρισμα ένα αντικείμενο ρόλου αντικειμένου (τύπου OWLObjectProperty) και επιστρέφει ένα σύνολο Java Set με τους ασύμβατους αυτού, σε μορφή OWLObjectProperty.

- `public HashSet<OWLDataProperty> getDisjointDataProperties( OWLDataProperty owlProperty )`

  Δέχεται ως όρισμα ένα αντικείμενο ρόλου τύπου δεδομένων (τύπου OWLDataProperty) και επιστρέφει ένα σύνολο Java Set με τους ασύμβατους αυτού, σε μορφή OWLDataProperty.

- `public boolean isDisjointClass( OWLClass owlClass1, OWLClass owlClass2 )`

  Δέχεται ως ορίσματα δύο κλάσεις της οντολογίας τύπου OWLClass και επιστρέφει true αν είναι ασύμβατες μεταξύ τους.

- `public HashSet<OWLIndividual> getIndividualsWithObjectProperty( OWLObjectProperty prop, OWLIndividual owlObject )`

  Δέχεται ως ορίσματα έναν ρόλο αντικειμένου τύπου OWLObjectProperty και ένα άτομο τύπου OWLIndividual και επιστρέφει ένα σύνολο Java Set με τα άτομα που συμμετέχουν στον συγκεκριμένο ρόλο ως υποκείμενα και που συνδέονται με αντικείμενο το άτομο-όρισμα.

- `public HashSet<OWLIndividual> getIndividualsWithDataProperty( OWLDataProperty prop, OWLTypedConstant owlObject )`

  Δέχεται ως ορίσματα έναν ρόλο τύπου δεδομένων (τύπου OWLDataProperty) και ένα literal τύπου OWLTypedConstant και επιστρέφει ένα σύνολο Java Set με τα άτομα που συμμετέχουν στον συγκεκριμένο ρόλο ως υποκείμενα και που συνδέονται με τιμή το literal-όρισμα.

- `public HashSet<HashSet<String[]>> returnAllConceptPseudomodelsFillers( HashMap<String,HashSet<String>> equivalentsMap , HashSet<String> conceptsThatDontHaveEquivalents )`

  Επιστρέφει ένα ψευδομοντέλο για κάθε κλάση της οντολογίας. Δέχεται ως ορίσματα μια αντιστοίχηση τύπου Java Map που περιλαμβάνει σύνολα ισοδύναμων κλάσεων και ένα σύνολο τύπου Java Set που περιλαμβάνει τις κλάσεις που δεν έχουν ισοδύναμες. Όλες οι κλάσεις περιγράφονται από τα URIs τους και είναι τύπου αλφαριθμητικού.

- `public HashSet<String[]> returnConceptPseudomodelsFillers( String conceptURI )`

  Δέχεται ως όρισμα το URI μιας κλάσης της οντολογίας (σε μορφή αλφαριθμητικού) και επιστρέφει το ψευδομοντέλο της.

- `public HashSet<String[]> returnConceptPseudomodelsFillers( ATermAppl conceptAppl )`

  Δέχεται ως όρισμα το μια κλάση της οντολογίας (σε μορφή ATermAppl) και επιστρέφει το ψευδομοντέλο της.

- `public HashMap<String,Boolean> returnIndividualsPseudomodelsUniqueness()`

  Επιστρέφει μια αντιστοίχηση Java Map μεταξύ του URI κάθε ατόμου της οντολογίας (σε μορφή αλφαριθμητικού) και μιας δυαδικής μεταβλητής που δείχνει αν το ψευδομοντέλο αυτού του ατόμου είναι μοναδικό ή όχι (τιμή true ή false αντίστοιχα).

- `public boolean checkIndividualsPseudomodelUniqueness( String individualURI )`

  Δέχεται το URI ενός ατόμου της οντολογίας σε μορφή αλφαριθμητικού και επιστρέφει true αν το ψευδομοντέλο αυτού είναι μοναδικό.

- `public boolean checkConceptsPseudomodelUniqueness( String conceptURI )`

  Δέχεται το URI μιας κλάσης της οντολογίας σε μορφή αλφαριθμητικού και επιστρέφει true αν το ψευδομοντέλο αυτής είναι μοναδικό.

- `public boolean checkConceptsPseudomodelUniqueness( ATermAppl concept )`

  Δέχεται το URI μιας κλάσης της οντολογίας σε μορφή ATermAppl και επιστρέφει true αν το ψευδομοντέλο αυτής είναι μοναδικό.

- `public HashSet<HashSet<String[]>> returnAllIndividualPseudomodelsFillers()`

  Επιστρέφει ένα ψευδομοντέλο για κάθε άτομο της οντολογίας. Όλα τα άτομα περιγράφονται από τα URIs τους και είναι τύπου αλφαριθμητικού.

- `public HashSet<String[]> returnIndividualPseudomodelsFillers( String individualURI )`

  Δέχεται ως όρισμα το URI ενός ατόμου της οντολογίας (σε μορφή αλφαριθμητικού) και επιστρέφει το ψευδομοντέλο του.

- `public HashSet<String> sortOutConceptIndividuals( String conceptURI, HashSet<String> individualsURIs )`

  Δέχεται ως όρισμα το URI μιας κλάσης της οντολογίας (σε μορφή αλφαριθμητικού) και ένα σύνολο Java Set με URIs ατόμων της οντολογίας (επίσης σε μορφή αλφαριθμητικού) που πιθανώς ανήκουν στην κλάση. Με χρήση mergable test απορρίπτει ορισμένα από αυτά και επιστρέφει ένα νέο, μικρότερο σύνολο.

- `public HashSet<String> sortOutConceptIndividuals( ATermAppl concept, HashSet<String> individualsURIs )`

  Ομοίως με από πάνω, μόνο που η κλάση-όρισμα είναι τύπου ATermAppl.

- `public HashSet<String> sortOutIndividualTypes( String individualURI, HashSet<String> conceptsURIs )`

  Δέχεται ως όρισμα το URI ενός ατόμου της οντολογίας (σε μορφή αλφαριθμητικού) και ένα σύνολο Java Set με τα URIs κλάσεων της οντολογίας (σε μορφή αλφαριθμητικού) στις οποίες πιθανώς ανήκει το άτομο. Με κατάλληλα mergable tests απορρίπτει ορισμένες από αυτές και επιστρέφει ένα νέο, μικρότερο σύνολο.

- `public String selectSameIndividualInCompletion( HashSet<String> samesURIs )`

  Δέχεται ως όρισμα ένα σύνολο Java Set με τα URIs ισοδύναμων ατόμων τις οντολογίας (σε μορφή αλφαριθμητικού) και επιστρέφει το άτομο (ανάμεσα σε αυτά) του οποίου το ψευδομοντέλο είναι κατασκευασμένο τη δεδομένη στιγμή από τον Pellet.

6.4.4.10 Η κλάση DBMSWrapper

Μέλη της κλάσης

- `Connection db`

  Η σύνδεση με τη βάση δεδομένων

- `Statement sql`

  Μια δήλωση SQL για υποβολή ερωτημάτων.

- `DatabaseMetaData dbmd`

  Τα στοιχεία της βάσης δεδομένων με την οποία έχει πραγματοποιηθεί σύνδεση.

- `boolean dbExistence`

  Παίρνει την τιμή true μόνο αν πραγματοποιηθεί επιτυχής σύνδεση με τη βάση.

- `boolean dbUserExistence`

  Παίρνει την τιμή true μόνο αν πραγματοποιηθεί επιτυχής σύνδεση με τη βάση για το δοσμένο όνομα χρήστη.

Μέθοδοι της κλάσης

- **public** DBMSWrapper(String database, String username, String password)

  Ο κατασκευαστής της κλάσης. Πραγματοποιεί τη σύνδεση με τη βάση δεδομένων βάσει των στοιχείων που του δίνονται ως ορίσματα και αρχικοποιεί τα μέλη της κλάσης.

- **public** DBMSWrapper(String userName, String userPassword)

  Κατασκευαστής της κλάσης. Χρησιμοποιείται μόνο για τον έλεγχο ύπαρξης βάσης δεδομένων και χρήστη.

- **public boolean** dbExistenceChecker()

  Επιστρέφει την τιμή του μέλους της κλάσης dbExistence

- **public boolean** isDBEmpty()

  Επιστρέφει true αν η βάση δεδομένων με την οποία έχει πραγματοποιηθεί σύνδεση περιέχει δεδομένα.

- **public boolean** dbUserExistenceChecker()

  Επιστρέφει την τιμή του μέλους της κλάσης dbUserExistence

- **public void** closeConnection()

  Διακόπτει τη σύνδεση με τη βάση δεδομένων.

- **public** HashSet<String[]> getDataBasesInfo(String userName, String userPassword)

  Δέχεται ως ορίσματα ένα όνομα χρήστη και τον κωδικό του και, αν υπάρχει βάση δεδομένων γι' αυτά τα στοιχεία, επιστρέφει τα ονόματα και τους ιδιοκτήτες των υπαρχουσών βάσεων δεδομένων (εξαιρώντας αυτές που ανήκουν στον διαχειριστή).

- **public void** loadScript(String scriptFilePath, String userName)

  Δέχεται ως όρισμα την τοποθεσία ενός script αρχείου κατασκευής σχήματος βάσης δεδομένων (ή ευρετηρίων) και το όνομα του χρήστη στον οποίο θα ανήκει αυτή και το εκτελεί.

- **public void** createDatabase(String databaseName, String userName)

  Δέχεται ως ορίσματα το όνομα μιας νέας βάσης δεδομένων και το όνομα του ιδιοκτήτη της και την κατασκευάζει.

- `public void createDBUser(String userName, String userPassword)`

  Δέχεται ως ορίσματα το όνομα ενός νέου χρήστη και τον κωδικό του και τον εγγράφει στο DBMS.
- `public void cleanDatabase()`

  Εκκαθαρίζει τη βάση δεδομένων με την οποία υπάρχει σύνδεση.

Οι παρακάτω μέθοδοι αναλαμβάνουν την κατασκευή των προσωρινών πινάκων της βάσης δεδομένων. Οι πίνακες αυτοί δημιουργούνται για την εκτέλεση κάποιων διαδικασιών φόρτωσης κατά την αρχική φόρτωση της οντολογίας, αλλά και για την αποτίμηση ερωτημάτων, σε ερωτήματα όπου εφαρμόζονται τεχνικές ψευδομοντέλων με αυθαίρετες κλάσεις. Αναφέρονται ονομαστικά:

- `public void createConceptsTempTable()`
- `public void createRolesTempTable()`
- `public void createTransitiveRolesTempTable()`
- `public void createTempGraphTable()`
- `public void createTempSortedNodesTable()`
- `public void createTempSortedOptimumSpanningTreeTable()`
- `public void createLevelConceptsTempTable()`
- `public void createLevelRolesTempTable()`
- `public void createArbitraryConceptPseudomodelTempTables()`

Αντίστοιχα με αυτές υπάρχουν και οι μέθοδοι εισαγωγής στοιχείων και διαγραφής των προσωρινών πινάκων, οι οποίες παραλείπονται εσκεμμένα.

- `public boolean checkTempPseudomodelUniqueness(int id_pseudoModel)`
- `public boolean arbitraryConceptPseudoModelContainsRole(int id_conceptPseudoModel)`
- `public HashSet<String> getLevelConcepts(int level)`
- `public HashSet<String> getLevelRoles(int level)`

- `public void completeTransitiveRolesTempTable()`
- `public String selectConceptNameFromConceptsTemp(int conceptId)`
- `public int selectConceptIdFromConceptsTemp(String conceptName, short namespaceId)`
- `public String selectRoleNameFromRolesTemp(int roleId)`
- `public int selectRoleIdFromRolesTemp(String roleName, short namespaceId)`
- `public int selectTRoleIdFromTRolesTemp(String roleName, short namespaceId)`
- `public int[] getSortedNodes(int nodesNumber)`
- `public int findNumberOfIndividualsInTransitiveRolesTempTable()`
- `public int getNumberOfConceptsTempEdges()`
- `public int getNumberOfRolesTempEdges()`
- `public int getNumberOfTRolesTempEdges()`
- `public int[][] getConceptsGraphTable()`
- `public int[][] getRolesGraphTable()`
- `public int[][] getTransitiveRolesGraphTable()`
- `public void completeIndividualsTable()`

Οι παρακάτω μέθοδοι είναι μέθοδοι εισαγωγής στοιχείων στους μόνιμους πίνακες της βάσης δεδομένων και θα αναφερθούν μόνο ονομαστικά:

- `public void fillConceptsTable(String concept_name, String concept_definition, int c_label_pre, int c_label_post, int id_father_class, short namespace_id)`
- `public void fillConceptAssertionsTable(int id_concept, int id_individual)`
- `public void fillConceptDisjointnessTable(int id_concept1, int id_concept2)`
- `public void fillRoleDisjointnessTable(int id_role1, int id_role2)`

- `public void fillRolesTable(String role_name, short role_characteristics, int r_label_pre, int r_label_post, int id_father_role, short namespace_id)`
- `public void fillRoleAssertions(int id_role, int id_individual_subject, int id_individual_object)`
- `public void fillTransitiveRoleAssertionTable(int currentIndividual_id, int id_individual, int t_label_pre, int t_label_post, int id_father_individual)`
- `public void fillInferredTransitiveRoleAssertionTable(int id_role, int id_individual_subject, int id_individual_object)`
- `public void fillDagConceptLabels(int id_concept, int c_label_pre_dag, int c_label_post_dag, boolean compression)`

  Το όρισμα compression καθορίζει αν θα γίνει συμπίεση κατά την αποθήκευση των ετικετών ή όχι.
- `public void fillDagRoleLabels(int id_role, int r_label_pre_dag, int r_label_post_dag, boolean compression)`
- `public void fillDagTRoleLabels(int id_role, int id_individual, int t_label_pre_dag, int t_label_post_dag, boolean compression)`
- `public void fillIndividualsTable(String instance, short namespace_id)`
- `public void fillDomainTable(int id_role, int id_concept)`
- `public void fillRangeTable(int id_role, int id_concept)`
- `public void fillInversionTable(int idRole1, int idRole2)`
- `public void fillIndividualsEquivalenceTable(int id_individual1, int id_individual2)`
- `public void fillIndividualsDifferenceTable(int id_individual1, int id_individual2)`
- `public void fillDatatypeRoleAssertionsTable(int id_role, int id_individual_subject, String literal, String type)`
- `public void fillNamespacesTable(String namespace)`

- `public void fillConceptPseudoModelInTable(int id_conceptPseudomodel, int id_concept, String part, boolean complement)`

  Το όρισμα complement καθορίζει αν το ψευδομοντέλο που αποθηκεύεται ανήκει στην κλάση με id το id_concept (false) ή στη συμπληρωματική της (true).

- `public void fillConceptPseudoModelExistenceTable(int id_conceptPseudomodel, int id_concept, String part, boolean complement)`
- `public void fillConceptPseudoModelNotInTable(int id_conceptPseudomodel, int id_concept, String part, boolean complement)`
- `public void fillConceptPseudoModelUniversialTable(int id_conceptPseudomodel, int id_concept, String part, boolean complement)`
- `public void fillIndividualPseudoModelInTable(int id_individualPseudomodel, int id_individual, String part)`
- `public void fillIndividualPseudoModelExistenceTable(int id_individualPseudomodel, int id_individual, String part)`
- `public void fillIndividualPseudoModelNotInTable(int id_individualPseudomodel, int id_individual,String part)`
- `public void fillIndividualPseudoModelUniversialTable(int id_individualPseudomodel, int id_individual, String part)`
- `public void fillBinaryQueryCacherTable(int id_query, int id_arg1, int id_arg2)`
- `public void fillPropertyValueQueryCacherTable(int id_arg1, int id_arg2, int id_arg3)`
- `public void fillConceptCompletionTable(int id_concept1, int id_concept2)`
- `public void updateDagConceptLabels(int id_concept, int c_label_pre_dag, int c_label_post_dag)`
- `public void updateDagRoleLabels(int id_role, int r_label_pre_dag, int r_label_post_dag)`
- `public void updateDagTRoleLabels(int id_role, int t_label_pre_dag, int t_label_post_dag, int t_ancestor)`

- `public void updateConceptFather(String conceptName,short namespace_id , int fatherId)`
- `public void updateConceptDefinition(int id_concept, String definition)`
- `public void updateRoleFather(String roleName, short namespace_id, int fatherId)`
- `public void updateTRoleFather(int tRoleID, int individualID, int fatherId)`
- `public void updateRoleCharacteristics(String roleName, short namespace_id, short characteristics)`
- `public void updateRoleCharacteristics(int id_role, short characteristics)`

Οι παρακάτω μέθοδοι ανανεώνουν κάποια δεδομένα στους πίνακες της βάσης, εκτελώντας διαδικασίες ελαφριάς συλλογιστικής ανάλυσης:

- `public void updateEquivalentRolesDomain()`

  Ανανεώνει τα πεδία των ρόλων βάσει των σχέσεων ισοδυναμίας μεταξύ τους.
- `public void updateEquivalentRolesRange()`

  Ανανεώνει τα εύρη των ρόλων βάσει των σχέσεων ισοδυναμίας μεταξύ τους.
- `public void updateConceptsFatherId()`

  Θέτει στο πεδίο id_father_class κάθε εγγραφής του πίνακα “Concepts” το id της κλάσης που είναι άμεσος πρόγονος της και συνδέεται μαζί της μέσω του spanning tree του γράφου της ιεραρχίας των κλάσεων που επιλέχθηκε κατά την κατασκευή του σχήματος ετικετών. Η διαδικασία επιτελείται με ένα SQL ερώτημα, στο οποίο λαμβάνεται υπ’ όψιν η πληροφορία περί ισοδυναμίας κλάσεων (με χρήση των ετικετών) για να βελτιωθεί η απόδοσή του.
- `public void updateRolesFatherId()`

  Ομοίως με την προηγούμενη μέθοδο, για τους ρόλους.
- `public void updateTransitiveRolesAssertionsFatherId(int id_role)`

  Ομοίως με την προηγούμενη μέθοδο, για τους ρόλους.

- `public void setEquivalentRoleCharacteristics()`

  Θέτει τις τιμές των χαρακτηριστικών των ισοδύναμων ρόλων ίσες μεταξύ τους, με εκτέλεση ενός SQL ερωτήματος.

- `public void setInverseRoleCharacteristics()`

  Θέτει τις τιμές των χαρακτηριστικών κάθε ζεύγους ισοδύναμων ρόλων ίσες μεταξύ τους.

- `public void propagateRoleFunctionality()`

  Μεταδίδει την λειτουργικότητα των ρόλων στην συνολική ιεραρχία.

- `public void deleteUnecessaryRoleAssertions()`

  Διαγράφει πιθανές διπλο-αποθηκευμένες δηλώσεις ρόλων.

- `public void collectUpdateGarbage()`

  Διαγράφει προσωρινές εγγραφές που εισήχθησαν κατά την εκτέλεση των μεθόδων updateEquivalentRolesDomain() και updateEquivalentRolesRange() και είναι πλέον περιττές.

- `public void setConceptsAndItsEquivalentsExpandedDefinition(int id_concept,String expandedDefinition)`

  Δέχεται ως όρισμα το id μιας κλάσης και έναν ορισμό σε μορφή αλφαριθμητικού και το θέτει ως τιμή του πεδίου "expanded_concept_definition" της δεδομένης κλάσης.

Οι τρεις παρακάτω μέθοδοι χρησιμοποιούνται για την εκτέλεση επιμέρους διεργασιών που έχουν ως στόχο την ανανέωση του Πεδίου και του Εύρους ισοδύναμων ρόλων:

- `public HashSet<String[]> getFirstInverseRolesSet()`
- `public HashSet<String[]> getSecondInverseRolesSet()`
- `public HashSet<String[]> getSymmetricRolesSet()`

Οι παρακάτω μέθοδοι θέτουν την τιμή των πεδίων που υποδεικνύουν πληρότητα πληροφορίας και που αντιστοιχούν σε οντότητες (κλάσεις/ρόλοι/άτομα) της οντολογίας. Το είδος της πληροφορίας είναι προφανές από τα ονόματα των μεθόδων.

- `public void setAllConceptsDisjointsCompleteness(boolean complete)`
- `public void setAllConceptsComplementsCompleteness(boolean complete)`
- `public void setAllRolesDisjointsCompleteness(boolean complete)`
- `public void setAllIndividualsDifferentsCompleteness(boolean complete)`
- `public void setConceptAssertionsCompleteness(int id_concept, boolean complete)`
- `public void setConceptDisjointsCompleteness(int id_concept, boolean complete)`
- `public void setConceptComplementsCompleteness(int id_concept, boolean complete)`
- `public void setRoleAssertionsCompleteness(int id_role, boolean complete)`
- `public void setRoleDisjointsCompleteness(int id_role, boolean complete)`
- `public void setRoleDisjointsCompleteness(int id_role, boolean complete)`
- `public void setIndividualDifferentsCompleteness(int id_individual, boolean complete)`

Οι παρακάτω μέθοδοι ελέγχουν αν η πληροφορία (το είδος της πληροφορίας είναι προφανές από το όνομα των μεθόδων) που βρίσκεται αποθηκευμένη στη βάση δεδομένων είναι πλήρης.

- `public boolean isConceptAssertionsInfoComplete(int id_concept)`
- `public boolean isConceptCompletionInfoComplete(int id_concept)`
- `public boolean isConceptDisjointnessInfoComplete(int id_concept)`
- `public boolean allObjectRolesAssertionsInfoComplete()`
- `public boolean allDatatypeRolesAssertionsInfoComplete()`

- `public boolean isRoleAssertionsInfoComplete(int id_role)`
- `public boolean isRoleDisjointnessInfoComplete(int id_role)`
- `public boolean isIndividualTypeInfoComplete(int id_individual)`
- `public boolean isIndividualDifferenceInfoComplete(int id_individual)`

Οι παρακάτω μέθοδοι ανασύρουν πληροφορία (από τη βάση δεδομένων) σχετική με το TBox της οντολογίας:

- `public HashSet<String[]> findDescriptionInConceptDefinition(String description)`

  Επιστρέφει τα ids των κλάσεων των οποίων οι ορισμοί περιέχουν την έκφραση που δίνεται ως όρισμα.
- `private HashSet<String[]> findRoleInConceptDefinition(String roleName)`

  Επιστρέφει τα ids των κλάσεων των οποίων οι ορισμοί περιέχουν τον ρόλο που δίνεται ως όρισμα.
- `private int selectEquivalentTransitiveRoleWithAssertionsHierarchy(int roleId)`

  Επιστρέφει το id του ρόλου του οποίου η ιεραρχία δηλώσεων είναι αποθηκευμένη και είναι ισοδύναμος με τον δοσμένο ρόλο. Επιστρέφει -1 αν δεν υπάρχει ισοδύναμος ρόλος με αποθηκευμένη ιεραρχία δηλώσεων.
- `private int selectInverseTransitiveRoleWithAssertionsHierarchy(int roleId)`

  Επιστρέφει το id του ρόλου του οποίου η ιεραρχία δηλώσεων είναι αποθηκευμένη και είναι αντίστροφος του δοσμένου ρόλο. Επιστρέφει -1 αν δεν υπάρχει αντίστροφος ρόλος με αποθηκευμένη ιεραρχία δηλώσεων.
- `public HashSet<String> getUpdatableTRoles()`

  Επιστρέφει τα ids των ρόλων των οποίων εγγραφές υπάρχουν στον πίνακα "Inferred Transitive Role Assertions".

- `public boolean checkCommonDescendantConceptsExistence(int id_concept1, int id_concept2, int notTopId, HashSet<String> unsatConcepts)`

  Ελέγχει αν οι δύο κλάσεις των οποίων τα ids δίνονται ως ορίσματα έχουν κοινό απόγονο. Δέχεται ως επιπλέον ορίσματα το id της κλάσης "Nothing" και το σύνολο των μη ικανοποιήσιμων κλάσεων της οντολογίας.

- `public HashMap<Integer,String> returnIdsAndDefinitionsOfConceptsHavingEquivalents()`

  Επιστρέφει έναν χάρτη αντιστοίχισης με τα ids και τους ορισμούς όλων των κλάσεων που διαθέτουν τουλάχιστον μια ισοδύναμη κλάση. Στο αποτέλεσμα περιέχεται μόνο μια κλάση από κάθε σύνολο ισοδύναμων κλάσεων.

- `public HashMap<Integer,String> returnEquivalentsIdsAndDefinitions(int id_concept)`

  Επιστρέφει έναν χάρτη αντιστοίχισης με τα ids και τους ορισμούς όλων των κλάσεων που διαθέτουν τουλάχιστον μια ισοδύναμη κλάση. Στο αποτέλεσμα περιέχονται όλες οι κλάσεις κάθε συνόλου ισοδύναμων κλάσεων.

- `public HashMap<Integer,String[]> selectConceptIdsNamesAndDefinitionsWithoutEquivalents()`

  Επιστρέφει έναν χάρτη αντιστοίχισης με τα ids και τους ορισμούς όλων των κλάσεων που δεν διαθέτουν ισοδύναμες κλάσεις.

- `public HashSet<String> treeAncestorsConcepts(int id_concept)`

  Επιστρέφει τα ids των κλάσεων που είναι πρόγονοι της δοσμένης κλάσης, λαμβάνοντας υπ' όψιν μόνο το spanning tree.

- `public HashSet<String> treeAncestorsTRoles(int transitiveRoleId, int id_individual)`

  Ομοίως με την treeAncestorsConcepts(), αλλά για άτομα που σχετίζονται με μεταβατικό ρόλο.

- `public HashSet<String> treeAncestorsRoles(int id_role)`

  Ομοίως με την treeAncestorsConcepts(), αλλά για ρόλους.

- `public HashSet<String> returnAncestorConcepts(int id_concept)`

  Επιστρέφει τα ids των κλάσεων που είναι πρόγονοι της δοσμένης κλάσης.

- `public HashSet<String> returnAncestorRoles(int id_role)`

  Επιστρέφει τα ids των ρόλων που είναι πρόγονοι του δοσμένου ρόλου.

- `public HashSet<String> returnDescendantConcepts(int id_concept)`

  Επιστρέφει τα ids των κλάσεων που είναι απόγονοι της δοσμένης κλάσης.

- `public HashSet<String> returnDescendantRoles(int id_role)`

  Επιστρέφει τα ids των ρόλων που είναι απόγονοι του δοσμένου ρόλου.

- `public HashSet<String> returnEquivalentConcepts(int id_concept)`

  Επιστρέφει τα ids των κλάσεων που είναι ισοδύναμες της δοσμένης κλάσης.

- `public HashSet<String> returnEquivalentRoles(int id_role)`

  Επιστρέφει τα ids των ρόλων που είναι ισοδύναμοι του δοσμένου ρόλου.

- `private HashSet<String> returnSatisfiableConcepts()`

  Επιστρέφει τα ids των ικανοποιήσιμων κλάσεων.

- `public HashSet<String> returnDisjointConcepts(int concept_id)`

  Επιστρέφει τα ids των κλάσεων που είναι ασύμβατες της δοσμένης κλάσης.

- `public HashSet<String> returnDisjointRoles(int role_id)`

  Επιστρέφει τα ids των ρόλων που είναι ασύμβατοι του δοσμένου ρόλου.

- `public boolean checkConceptSubsumption(int id_concept_subsumee, int id_concept_subsumer)`

  Ελέγχει αν ο πρώτος ρόλος είναι απόγονος του δεύτερου.

- `public boolean checkConceptEquivalence(int concept_id1, int concept_id2)`

  Ελέγχει αν οι δύο κλάσεις είναι ισοδύναμες.

- `public boolean checkRoleEquivalence(int id_role1, int id_role2)`

  Ελέγχει αν οι δύο ρόλοι είναι ισοδύναμοι.

- `public int checkConceptDisjointness(int concept_id1, int concept_id2)`

  Ελέγχει αν οι δύο κλάσεις είναι ασύμβατες και επιστρέφει 1 αν είναι, 0 αν δεν είναι και 2 αν δεν μπορεί να απαντήσει.

- `public int checkRoleDisjointness(int role_id1, int role_id2)`

  Ελέγχει αν οι δύο ρόλοι είναι ασύμβατοι και επιστρέφει 1 αν είναι, 0 αν δεν είναι και 2 αν δεν μπορεί να απαντήσει.

- `public boolean checkRoleInversion(int id_role1, int id_role2)`

  Ελέγχει αν οι δύο ρόλοι είναι αντίστροφοι.

- `public HashSet<String> returnInverseRoles(int id_role)`

  Επιστρέφει τα ids των ρόλων που είναι αντίστροφοι του δοσμένου ρόλου.

- `public boolean isSymmetricRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι συμμετρικός.

- `public boolean isInverseRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι αντίστροφος κάποιου άλλου.

- `public boolean isTransitiveRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι μεταβατικός.

- `public boolean isFunctionalRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι λειτουργικός.

- `public boolean isInverseFunctionalRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι αντίστροφος λειτουργικός.

- `public boolean isDatatypeRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι ρόλος τύπου δεδομένων ή όχι και επιστρέφει true ή false αντίστοιχα.

- `public boolean isFunctionalDatatypeRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι λειτουργικός ρόλος τύπου δεδομένων.

- `public boolean isObjectRole(int id_role)`

  Ελέγχει αν ο δεδομένος ρόλος είναι ρόλος αντικειμένου.

- `public HashSet<String> returnObjectRoles()`

  Επιστρέφει τα ids των ρόλων αντικειμένου της οντολογίας.

- `public HashSet<String> returnSymmetricRoles()`

  Επιστρέφει τα ids των συμμετρικών ρόλων της οντολογίας.

- `public HashSet<String> returnInverseRoles()`

  Επιστρέφει τα ids των αντίστροφων ρόλων της οντολογίας.

- `public HashSet<String> returnTransitiveRoles()`

  Επιστρέφει τα ids των μεταβατικών ρόλων της οντολογίας.

- `public HashSet<String> returnFunctionalObjectRoles()`

  Επιστρέφει τα ids των λειτουργικών ρόλων αντικειμένου της οντολογίας.

- `public HashSet<String> returnFunctionalDatatypeRoles()`

  Επιστρέφει τα ids των λειτουργικών ρόλων τύπου δεδομένων της οντολογίας.

- `public HashSet<String> returnInverseFunctionalRoles()`

  Επιστρέφει τα ids των αντιστρόφων λειτουργικών ρόλων της οντολογίας.

- `public HashSet<String> returnDatatypeRoles()`

  Επιστρέφει τα ids των ρόλων τύπου δεδομένων της οντολογίας.

- `public HashSet<String> getDirectAncestorsConcepts(int id_concept)`

  Επιστρέφει τα ids των απ' ευθείας προγόνων τις δοσμένης κλάσης.

- `public HashSet<String> getDirectAncestorsRoles(int id_role)`

  Επιστρέφει τα ids των απ' ευθείας προγόνων του δοσμένου ρόλου.

- `public HashSet<String> getDirectDescendantsConcepts(int id_concept)`

  Επιστρέφει τα ids των απ' ευθείας απογόνων τις δοσμένης κλάσης.

- `public HashSet<String> getDirectDescendantsRoles(int id_role)`

  Επιστρέφει τα ids των απ' ευθείας απογόνων του δοσμένου ρόλου.

- `public HashSet<String> getNearestCommonConceptAnscestors(int id_concept1, int id_concept2)`

  Επιστρέφει το id (ή τα ids) του κοντινότερου κοινού προγόνου των δύο δοσμένων κλάσεων.

- `public HashSet<String> getNearestCommonRoleAncestors(int id_role1, int id_role2)`

  Επιστρέφει το id (ή τα ids) του κοντινότερου κοινού προγόνου των δύο δοσμένων ρόλων.

- `public HashSet<String> getNearestCommonConceptDescendants(int id_concept1, int id_concept2)`

  Επιστρέφει το id (ή τα ids) του κοντινότερου κοινού απογόνου των δύο δοσμένων κλάσεων.

- `public HashSet<String> getNearestCommonRoleDescendants(int id_role1, int id_role2)`

  Επιστρέφει το id (ή τα ids) του κοντινότερου κοινού απογόνου των δύο δοσμένων ρόλων.

- `public HashSet<String> getConceptSiblings(int id_concept)`

  Επιστρέφει τα ids των αδερφών (βάσει του γράφου της ιεραρχίας) της δεδομένης κλάσης.

- `public HashSet<String> getRoleSiblings(int id_role)`

  Επιστρέφει τα ids των αδερφών (βάσει του γράφου της ιεραρχίας) του δεδομένου ρόλου.

- `public HashSet<String> getConceptLeaves(int id_concept)`

  Επιστρέφει τα φύλλα (βάσει του γράφου της ιεραρχίας) που προκύπτουν από την δεδομένη κλάση.

- `public HashSet<String> getRoleLeaves(int id_role)`

  Επιστρέφει τα φύλλα (βάσει του γράφου της ιεραρχίας) που προκύπτουν από τον δεδομένο ρόλο.

- `public int checkConceptCompletion(int id_concept1, int id_concept2)`

  Ελέγχει αν οι δύο δοσμένες κλάσεις είναι συμπληρωματικές και επιστρέφει 1 αν είναι, 0 αν δεν είναι και 2 αν δεν μπορεί να απαντήσει.

- `public HashSet<String> returnComplementConcepts(int id_concept)`

  Επιστρέφει τα ids των συμπληρωματικών κλάσεων της δοσμένης.

- `public HashSet<String> returnDomains(int id_role, boolean includeInverses)`

  Επιστρέφει τα ids των κλάσεων που αποτελούν το εύρος του δεδομένου ρόλου. Η δυαδική μεταβλητή includeInverses καθορίζει αν θα ληφθεί υπ'όψιν ή όχι κατά την αποτίμηση η πληροφορία περί αντιστρόφων ρόλων.

- `public HashSet<String> returnRanges(int id_role, boolean includeInverses)`

  Επιστρέφει τα ids των κλάσεων που αποτελούν το πεδίο του δεδομένου ρόλου. Η δυαδική μεταβλητή includeInverses καθορίζει αν θα ληφθεί υπ'όψιν ή όχι κατά την αποτίμηση η πληροφορία περί αντιστρόφων ρόλων.

Οι παρακάτω μέθοδοι ανασύρσουν πληροφορία (από τη βάση δεδομένων) σχετική με το ABox της οντολογίας:

- `public HashSet<int[]> returnObjectRoleAssertions(int id_role)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου και επιστρέφει όλες τις αποθηκευμένες δηλώσεις του, σε μορφή ζευγών ids ατόμων. Επιτελεί διαδικασίες ελαφριάς συλλογιστικής ανάλυσης (βλέπε Παράγραφο 6.1.4.1)

- `public HashSet<String[]> returnDatatypeRoleAssertions( int id_role)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ρόλου τύπου δεδομένων και επιστρέφει όλες τις αποθηκευμένες δηλώσεις του, σε μορφή ζευγών ids ατόμων.

- `private HashSet<String> returnSubjects(int id_role)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ρόλου και επιστρέφει τα ids όλων των ατόμων που συμμετέχουν ως υποκείμενα στις αποθηκευμένες δηλώσεις του ρόλου. Χρησιμοποιεί τις returnObjectRoleAssertions()και returnDatatypeRoleAssertions().

- `private HashSet<String> returnObjectsOfObjectRole(int id_role)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου και επιστρέφει τα ids όλων των ατόμων που συμμετέχουν ως αντικείμενα στις αποθηκευμένες δηλώσεις του ρόλου. Χρησιμοποιεί την returnObjectRoleAssertions().

- `public HashSet<int[]> returnInferredTransitiveRoleAssertions( int id_role)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός μεταβατικού ρόλου αντικειμένου και επιστρέφει τις δηλώσεις του, σε μορφή ζευγών ids ατόμων, που είναι αποθηκευμένες στον πίνακα “Inferred Transitive Role Assertions” της βάσης δεδομένων. Επιτελεί διαδικασίες ελαφριάς συλλογιστικής καθώς λαμβάνει υπ’ όψιν ταυτότητα ατόμων, ισοδυναμία και συμμετρικότητα ρόλων.

- `public HashSet<String> returnRolesWhereIndividualIsSubject( int id_individual)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids όλων των ρόλων σε κάποια δήλωση των οποίων συμμετέχει ως υποκείμενο.

Χρησιμοποιεί τις returnObjectRoleAssertions()και returnDatatypeRoleAssertions().

- `public HashSet<String> returnRolesWhereIndividualIsObject( int id_individual)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids όλων των ρόλων αντικειμένου σε κάποια δήλωση των οποίων συμμετέχει ως υποκείμενο. Χρησιμοποιεί την returnObjectRoleAssertions().

- `public HashSet<String> returnConnectingObjectRoles( int id_subject, int id_object)`

  Δέχεται ως ορίσματα τα ids (στη βάση δεδομένων) δύο ατόμων της οντολογίας και επιστρέφει τα ids όλων των ρόλων αντικειμένου, σε κάποια δήλωση των οποίων το πρώτο άτομο είναι υποκείμενο και το δεύτερο αντικείμενο. Χρησιμοποιεί την returnObjectRoleAssertions().

- `public HashSet<String> returnConnectingDatatypeRoles( int id_subject, String type, String literal)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ατόμου, έναν τύπο δεδομένων σε μορφή αλφαριθμητικού και μια τιμή (επίσης σε μορφή αλφαριθμητικού) και επιστρέφει τα ids όλων των ρόλων τύπου δεδομένων, σε κάποια δήλωση των οποίων το άτομο είναι υποκείμενο με τιμή literal τύπου type. Χρησιμοποιεί την

  returnDatatypeRoleAssertions().

- `public HashSet<String> returnSubjectSuccessorsForObjectRole( int id_role, int id_object)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου και το id ενός ατόμου και επιστρέφει τα ids των υποκειμένων των αποθηκευμένων δηλώσεων του δεδομένου ρόλου που συνδέονται με αντικείμενο το δεδομένο άτομο. Χρησιμοποιεί την returnObjectRoleAssertions().

- `public HashSet<String> returnObjectSuccessorsForObjectRole( int id_role, int id_subject)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου και το id ενός ατόμου και επιστρέφει τα ids των αντικειμένων των αποθηκευμένων δηλώσεων του δεδομένου ρόλου που συνδέονται με υποκείμενο το δεδομένο άτομο. Χρησιμοποιεί την returnObjectRoleAssertions().

- `private HashSet<String> returnSubjectSuccessorsForTransitiveRole( int id_role, int id_object, boolean includeSymmetricsAndInverses)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός μεταβατικού ρόλου αντικειμένου και το id ενός ατόμου και επιστρέφει τα ids των υποκειμένων των αποθηκευμένων δηλώσεων του δεδομένου ρόλου, που συνδέονται με αντικείμενο το δεδομένο άτομο. Επιτελεί διαδικασίες ελαφριάς συλλογιστικής ανάλυσης, καθώς για την αποτίμηση χρησιμοποιεί τις αποθηκευμένες ιεραρχίες δηλώσεων μεταβατικών ρόλων, πληροφορία σχετικά με ισοδυναμία ρόλων, αντίστροφων ρόλων, συμμετρικών ρόλων και ταυτότητα ατόμων. Το δυαδικό όρισμα καθορίζει αν κατά την αποτίμηση θα ληφθεί υπ' όψιν η πληροφορία σχετικά με αντίστροφους ρόλους και συμμετρικότητα ρόλων.

- `private HashSet<String> returnObjectSuccessorsForTransitiveRole( int id_role, int id_subject,boolean includeSymmetricsAndInverses)`

  Ομοίως με την παραπάνω, αλλά επιστρέφει αντικείμενα.

- `private HashSet<String> returnIndividualAncestorsInTRoleHierarchy( int id_role, int id_individual)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός μεταβατικού ρόλου αντικειμένου που έχει αποθηκευμένη ιεραρχία δηλώσεων και το id ενός ατόμου και επιστρέφει τα άτομα-προγόνους του δεδομένου ατόμου στην ιεραρχία δηλώσεων του ρόλου.

- `private HashSet<String> returnIndividualDescendantsInTRoleHierarchy( int id_role, int id_individual)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός μεταβατικού ρόλου αντικειμένου που έχει αποθηκευμένη ιεραρχία δηλώσεων και το id ενός ατόμου και επιστρέφει τα άτομα-απογόνους του δεδομένου ατόμου στην ιεραρχία δηλώσεων του ρόλου.

- `public HashSet<String> returnSubjectSuccessorsForDatatypeRole(int id_role, String type, String literal)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου τύπου δεδομένων και μια τιμή και τον τύπο της, και επιστρέφει τα ids των ατόμων των αποθηκευμένων δηλώσεων του δεδομένου ρόλου, που συνδέονται με τιμή τη δοσμένη (literal). Χρησιμοποιεί την returnDatatypeRoleAssertions().

- `public HashSet<String[]> returnObjectSuccessorsForDatatypeRole(int id_role, int id_subject)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου τύπου δεδομένων και το id ενός ατόμου και επιστρέφει τις τιμές και τους τύπους των αποθηκευμένων δηλώσεων του δεδομένου ρόλου, που αντισοιτχούν στο δοσμένο άτομο-υποκείμενο. Χρησιμοποιεί την returnDatatypeRoleAssertions().

- `public int areConnectedThroughObjectRole(int id_subject, int id_role, int id_object)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου και τα ids δύο ατόμων (ως υποκείμενο και αντικείμενο της δήλωσης αντίστοιχα) και επιστρέφει 1 αν βρεθεί η δήλωση, 0 αν η δήλωση είναι σίγουρο ότι δεν υπάρχει και 2 σε κάθε άλλη περίπτωση. Χρησιμοποιεί την returnObjectRoleAssertions().

- `public int areConnectedThroughDatatypeRole(int id_subject, int id_role, String type, String literal)`

  Δέχεται ως ορίσματα το id (στη βάση δεδομένων) ενός ρόλου αντικειμένου, το id ενός ατόμου, μια τιμή και τον τύπο της και επιστρέφει 1 αν βρεθεί η δήλωση, 0 αν η δήλωση είναι σίγουρο ότι δεν υπάρχει και 2 σε κάθε άλλη περίπτωση. Χρησιμοποιεί την returnDatatypeRoleAssertions().

- `public boolean checkPropertyValueQueryCacher(int id_arg1, int id_arg2, int id_arg3)`

  Ελέγχει αν η δεδομένη εγγραφή υπάρχει στον πίνακα “Property Value Query Cacher”.

- `public HashSet<String> returnConceptIndividuals(int id_concept)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) μιας κλάσης και επιστρέφει τα ids των ατόμων που ανήκουν στην κλάση αυτή, βάσει της αποθηκευμένης πληροφορίας. (βλέπε Παράγραφο 6.1.4.2)

- `public HashSet<String> returnIndividualTypes(int id_individual)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids των κλάσεων που είναι τύποι αυτού, βάσει της αποθηκευμένης πληροφορίας.

- `private HashSet<String> returnConceptAssertedIndividuals(int id_concept)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) μιας κλάσης και επιστρέφει τα ids των ατόμων που ανήκουν στην κλάση αυτή, χρησιμοποιώντας μόνο τις ρητές δηλώσεις, δηλαδή χωρίς την εκτέλεση καμίας λειτουργίας συλλογιστικής ανάλυσης.

- `private HashSet<String> returnIndividualExplicitTypes(int id_individual)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids των κλάσεων που είναι τύποι αυτού, χρησιμοποιώντας μόνο τις ρητές δηλώσεις, δηλαδή χωρίς την εκτέλεση καμίας λειτουργίας συλλογιστικής ανάλυσης.

- `public HashSet<String> returnEquivalentIndividuals(int individual_id)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids όλων των ατόμων ίδιων με αυτό.

- `public boolean areSame(int individual_id1, int individual_id2)`

  Δέχεται ως όρισμα τα ids (στη βάση δεδομένων) δυο ατόμων και ελέγχει αν είναι ίδια.

- `public int areDifferent(int individual_id1, int individual_id2)`

  Δέχεται ως όρισμα τα ids (στη βάση δεδομένων) δυο ατόμων και επιστρέφει 1 αν είναι διαφορετικά, 0 αν δεν είναι και 2 αν δεν μπορεί να απαντήσει βάσει της αποθηκευμένης πληροφορίας.

- `public HashSet<String> returnDifferents(int id_individual)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα ids όλων των ατόμων που είναι διαφορετικά με αυτό, βάσει της αποθηκευμένης πληροφορίας.

- `public int hasType(int id_individual, int id_concept)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και το id μιας κλάσης και επιστρέφει 1 αν το άτομο ανήκει στην κλάση, 0 αν δεν ανήκει και 2 αν δεν μπορεί να απαντήσει.

Οι παρακάτω μέθοδοι αφορούν στην υλοποίηση εποπτικών ερωτημάτων:

- `public HashSet<String> returnConceptRelatedAxioms(int conceptID)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) μιας κλάσης και επιστρέφει τα αξιώματα της οντολογίας, σε μορφή αλφαριθμητικού, στα οποία απαντάται η κλάση αυτή.

- `public HashSet<String> returnRoleRelatedAxioms(String roleName, short namespaceId)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ρόλου και επιστρέφει τα αξιώματα της οντολογίας, σε μορφή αλφαριθμητικού, στα οποία απαντάται ο ρόλος αυτός.

- `public HashSet<String> returnIndividualRelatedAxioms(String individualName, short namespaceId)`

  Δέχεται ως όρισμα το id (στη βάση δεδομένων) ενός ατόμου και επιστρέφει τα αξιώματα της οντολογίας, σε μορφή αλφαριθμητικού, στα οποία απαντάται το άτομο αυτό.

Οι παρακάτω μέθοδοι αφορούν στο χειρισμό ψευδομοντέλων ατόμων και κλάσεων:

- `public boolean checkIndividualPseudoModelExistence(HashSet<String[]> pseudoModel)`

  Ελέγχει αν το δοσμένο ψευδομοντέλο ατόμου υπάρχει στη βάση.

- `public boolean checkConceptPseudoModelExistence(HashSet<String[]> pseudoModel)`

  Ελέγχει αν το δοσμένο ψευδομοντέλο κλάσης υπάρχει στη βάση.

- `public void setIndividualUniquePseudomodel(int id_individual, boolean value)`

  Θέτει την τιμή του πεδίου "unique_pseudomodel" του δοσμένο ατόμου ίση με το όρισμα value.

- `public void setConceptUniquePseudomodel(int id_concept, boolean value, boolean complement)`

  Θέτει την τιμή του πεδίου "unique_pseudomodel" της δοσμένης κλάσης ίση με το όρισμα value.

- `public int returnMaxIPMID(int id_individual)`

  Επιστρέφει το μέγιστο id των αποθηκευμένων ψευδομοντέλων του δοσμένου ατόμου.

- `public boolean checkArbitraryPseudomodelExistence()`

  Ελέγχει αν το δοσμένο ψευδομοντέλο αυθαίρετης κλάσης υπάρχει στη βάση.

- `public int returnMaxCPMID(int id_concept, boolean complement)`

  Επιστρέφει το μέγιστο id των αποθηκευμένων ψευδομοντέλων της δοσμένης κλάσης.

- `public boolean conceptPseudoModelContainsRole(int id_concept, int id_conceptPseudoModel, boolean complement)`

  Ελέγχει αν το ψευδομοντέλο της δοσμένης κλάσης με το δοσμένο id περιέχει ρόλους (περιέχει δηλαδή τουλάχιστον μια εγγραφή στα tuples Universial και Existence). Η μεταβλητή complement καθορίζει αν αναφερόμαστε στο ψευδομοντέλο της κλάσης ή της συμπληρωματικής της.

- `public boolean checkConceptsPseudoModelUniqueness(int id_concept, boolean complement)`

  Ελέγχει αν το ψευδομοντέλο της δοσμένης κλάσης με το δοσμένο id είναι μοναδικό ή όχι. Η μεταβλητή complement καθορίζει αν αναφερόμαστε στο ψευδομοντέλο της κλάσης ή της συμπληρωματικής της.

- `public boolean individualPseudoModelContainsRole(int id_individual, int id_individualPseudoModel)`

  Ελέγχει αν το ψευδομοντέλο του δοσμένου ατόμου με το δεδομένο id περιέχει ρόλους (περιέχει δηλαδή τουλάχιστον μια εγγραφή στα tuples Universial και Existence).

- `public boolean checkIndividualsPseudoModelUniqueness(int id_individual)`

  Ελέγχει αν το ψευδομοντέλο του δοσμένου ατόμου με το δεδομένο id είναι μοναδικό ή όχι.

- `public HashSet<String> getConceptsIdsFromIndividualPseudoModelInTable(int individualId, int pseudoModelId)`

  Επιστρέφει τα ids των κλάσεων του πίνακα "In" του δοσμένου ψευδομοντέλου του δοσμένου ατόμου.

- `public HashSet<String> getConceptsIdsFromIndividualPseudoModelNotInTable(int individualId , int pseudoModelId)`

  Επιστρέφει τα ids των κλάσεων του πίνακα "Not In" του δοσμένου ψευδομοντέλου του δοσμένου ατόμου.

- `public HashSet<String> getRolesIdsFromIndividualPseudoModelExistenceTable(int individualId , int pseudoModelId)`

  Επιστρέφει τα ids των ρόλων του πίνακα "Existence" του δοσμένου ψευδομοντέλου του δοσμένου ατόμου.

- `public HashSet<String> getRolesIdsFromIndividualPseudoModelUniversialTable(int individualId , int pseudoModelId)`

  Επιστρέφει τα ids των ρόλων του πίνακα "Universial" του δοσμένου ψευδομοντέλου του δοσμένου ατόμου.

- `public HashSet<String> getConceptsIdsFromConceptPseudoModelInTable(int conceptId , int pseudoModelId)`

  Επιστρέφει τα ids των κλάσεων του πίνακα "In" του δοσμένου ψευδομοντέλου της δοσμένης κλάσης.

- `public HashSet<String> getConceptsIdsFromConceptPseudoModelNotInTable(int conceptId , int pseudoModelId)`

  Επιστρέφει τα ids των κλάσεων του πίνακα "Not In" του δοσμένου ψευδομοντέλου της δοσμένης κλάσης.

- `public HashSet<String> getRolesIdsFromConceptPseudoModelExistenceTable(int conceptId , int pseudoModelId)`

  Επιστρέφει τα ids των ρόλων του πίνακα "Existence" του δοσμένου ψευδομοντέλου της δοσμένης κλάσης.

- `public HashSet<String> getRolesIdsFromConceptPseudoModelUniversialTable(int conceptId , int pseudoModelId)`

  Επιστρέφει τα ids των ρόλων του πίνακα "Universial" του δοσμένου ψευδομοντέλου της δοσμένης κλάσης.

- `private boolean checkIPMExRecordExistence(int id_individualPseudomodel, int id_individual, int id_role)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Existence" του δοσμένου ψευδομοντέλου ατόμου υπάρχει στη βάση.

- `private boolean checkIPMUnRecordExistence(int id_individualPseudomodel, int id_individual, int id_role)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Universial" του δοσμένου ψευδομοντέλου ατόμου στη βάση.

- `private boolean checkCPMInRecordExistence(int id_conceptPseudomodel, int id_concept, int id_conceptIn, boolean complement)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "In" του δοσμένου ψευδομοντέλου κλάσης υπάρχει στη βάση.

- `private boolean checkCPMNotInRecordExistence(int id_conceptPseudomodel, int id_concept, int id_conceptNotIn, boolean complement)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Not In" του δοσμένου ψευδομοντέλου κλάσης υπάρχει στη βάση.

- `private boolean checkCPMExRecordExistence(int id_conceptPseudomodel, int id_concept, int id_role, boolean complement)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Existence" του δοσμένου ψευδομοντέλου κλάσης υπάρχει στη βάση.

- `private boolean checkCPMUnRecordExistence(int id_conceptPseudomodel, int id_concept, int id_role, boolean complement)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Universial" του δοσμένου ψευδομοντέλου κλάσης υπάρχει στη βάση.

- `private boolean checkACPMInRecordExistence(int id_conceptPseudomodel, int id_conceptIn)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "In" του δοσμένου ψευδομοντέλου αυθαίρετης κλάσης υπάρχει στη βάση.

- `private boolean checkACPMNotInRecordExistence(int id_conceptPseudomodel, int id_conceptNotIn)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Not In" του δοσμένου ψευδομοντέλου αυθαίρετης κλάσης υπάρχει στη βάση.

- `private boolean checkACPMExRecordExistence(int id_conceptPseudomodel, int id_role)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Existence" του δοσμένου ψευδομοντέλου αυθαίρετης κλάσης υπάρχει στη βάση.

- `private boolean checkACPMUnRecordExistence(int id_conceptPseudomodel, int id_role)`

  Ελέγχει αν η δοσμένη εγγραφή του πίνακα "Universial" του δοσμένου ψευδομοντέλου αυθαίρετης κλάσης υπάρχει στη βάση.

- `public int selectSameIndividualWithPseudomodel(int id_individual)`

  Επιστρέφει το id του ίδιου ατόμου με το δοσμένου, για το οποίο υπάρχει αποθηκευμένο ψευδομοντέλο. Αν δεν υπάρχει, επιστρέφει -1.

- `public int selectEquivalentConceptWithPseudomodel(int id_concept, boolean complement)`

  Επιστρέφει το id της ισοδύναμης κλάσης με τη δοσμένη, για την οποία υπάρχει αποθηκευμένο ψευδομοντέλο. Αν δεν υπάρχει, επιστρέφει -1.

- `public boolean executeSubsumptionMergableTest(int id_concept, boolean knownConceptIsFirst)`

  Εκτελεί το mergable test μεταξύ των ψευδομοντέλων της δοσμένης κλάσης και των ψευδομοντέλων της αυθαίρετης κλάσης. Η τιμή της δυαδικής μεταβλητής δηλώνει τη θέση της γνωστής κλάσης στο ερώτημα που τέθηκε.

- `public boolean executeTypeMergableTest(int id_individual, int id_concept)`

  Εκτελεί το mergable test μεταξύ των ψευδομοντέλων της δοσμένης κλάσης και των ψευδομοντέλων του δοσμένου ατόμου.

7

Έλεγχος και Αξιολόγηση

Στο κεφάλαιο αυτό παρουσιάζουμε σενάρια λειτουργίας του συστήματος με πραγματικές οντολογίες πολύ μεγάλου όγκου της τάξης των εκατομμυρίων τριάδων του RDF μοντέλου. Αρχικά, εξετάζουμε την επίδραση των παραμέτρων φόρτωσης στο χρόνο που καταναλώνει το DBRS να αποθηκεύσει την εκάστοτε οντολογία στη βάση δεδομένων και στη συνέχεια, στα πλαίσια του ελέγχου επιδόσεων, διαλέγουμε ένα μικρό σύνολο βασικών ερωτημάτων για να συγκρίνουμε την απόδοση του συστήματος με αυτή του Reasoner Pellet όταν λειτουργεί ως αυτόνομη εφαρμογή (stand-alone application).

7.1 Μεθοδολογία ελέγχου

Ο έλεγχος της λειτουργίας του DBRS περιλαμβάνει δύο σκέλη. Το πρώτο αφορά στην πληρότητα του συστήματος, δηλαδή στην «ορθή» (sound & complete) απάντηση των ερωτημάτων που τίθενται από τον χρήστη μέσω του GUI, ενώ το δεύτερο στην απόδοσή του τόσο κατά τη φάση της φόρτωσης (loading phase), όσο και κατά την αποτίμηση των ερωτημάτων (querying phase).

Η πληρότητα των αλγορίθμων που εκτελεί το DBRS μπορεί να ελεχθεί πρακτικώς και με οντολογίες μικρού όγκου, που έχουν όμως υψηλή εκφραστικότητα. Αντιθέτως,

ο έλεγχος της απόδοσης του συστήματος (benchmarking) προϋποθέτει τη χρήση οντολογιών πολύ μεγάλου όγκου (εκαταμμύρια RDF τριάδες), όπως αυτές που χρησιμοποιούνται σε πραγματικές εφαρμογές του Σημασιολογικού Ιστού. Άλλωστε, όπως έχει γίνει ήδη σαφές από τα προηγούμενα κεφάλαια, η περίπτωση που μας ενδιαφέρει είναι αυτή των μεγάλων οντολογιών, έστι ώστε να φανούν τα πλεονεκτήματα της χρήσης του DBMS.

7.2 Αναλυτική παρουσίαση ελέγχου

Η μεθοδολογία που παρουσιάζουμε επικεντρώνεται μόνο στον έλεγχο της απόδοσης του συστήματος και συγκεκριμένα στους χρόνους φόρτωσης οντολογιών και αποτίμησης ερωτημάτων. Η πληρότητα (soundness & completeness) των αλγορίθμων που εκτελούνται από το DBRS μπορεί να επιβεβαιωθεί θεωρητικά, βάσει των όσων αναλύσαμε στα Κεφάλαια 5 & 6 και δε θα μας απασχολήσει εδώ.

Στην Παράγραφο 7.2.2 εξετάζουμε την επίδραση του μεγέθους της οντολογίας και των παραμέτρων φόρτωσης στο συνολικό χρόνο που καταναλώνεται από το DBRS για να αποθηκεύσει την πληροφορία που επιθυμεί ο χρήστης. Στην Παράγραφο 7.2.3 ελέγχεται η απόδοση του συστήματος κατά την αποτίμηση TBox ερωτημάτων, ενώ στην Παράγραφο 7.2.4 η αντίστοιχη απόδοσή του για την περίπτωση των ABox ερωτημάτων.

Σημειώνουμε ότι η υποβολή των ερωτημάτων στον Pellet έγινε μέσω του OWL API, ενώ όλες οι μετρήσεις των Παραγράφων 7.2.2, 7.2.3 και 7.2.4 πραγματοποιήθηκαν σε προσωπικό υπολογιστή με επεξεργαστή Intel(R) Core(TM) 2 E6850 στα 3.00GHz, μνήμη RAM 2GB και λειτουργικό σύστημα Microsoft Windows XP Professional version 2002, Service Pack 2.

Επίσης, τόσο κατά τη φόρτωση όσο και κατά την αποτίμηση ερωτημάτων, η λειτουργία του DBRS έγινε μέσα από το περιβάλλον Eclipse SDK version 3.3.2 [Eclipse] και η έκδοση της Java που χρησιμοποιήθηκε ήταν η 1.6.0_06.

7.2.1 Οντολογίες που χρησιμοποιήθηκαν κατά τον έλεγχο

Για τον έλεγχο της απόδοσης του DBRS, χρησιμοποιούμε έξι οντολογίες διαφορετικού μεγέθους, τα χαρακτηριστικά των οποίων δίνονται στον Πίνακα 7.1.

Η πρώτη από αυτές, η Thesaurus [Thesaurus], έχει αναπτυχθεί στα πλαίσια ενός προγράμματος του Αμερικανικού Ινστιτούτου Καρκίνου (US National Cancer

Institute) και περιέχει μοντελοποιημένη γνώση ιατρικής φύσης. Δε διαθέτει καθόλου ABox (δηλώσεις κλάσεων ή ρόλων), αλλά διακρίνεται για το πολύ μεγάλο TBox (περίπου 62.000 κλάσεις και 200 ρόλοι) με πολύπλοκους ορισμούς κλάσεων και ταυτόχρονα «πυκνή» και «πλατιά» ιεραρχία. Τη χρησιμοποιούμε, όπως θα δούμε στην Παράγραφο 7.2.3, για τον έλεγχο της απόδοσης του DBRS κατά την αποτίμηση TBox ερωτημάτων.

Οι υπόλοιπες πέντε οντολογίες κατασκευάστηκαν με χρήση της εφαρμογής LUBM (Lehigh University BenchMark) [LUBM] που χρησιμοποιείται ευρέως για τον έλεγχο της σωστής λειτουργίας και την αξιολόγηση της επίδοσης Συστημάτων Διαχείρισης Οντολογιών και Μηχανών Συλλογιστικής Ανάλυσης. Αντίθετα από τη Thesaurus, όλες οι οντολογίες που κατασκευάσαμε με τη LUBM διαθέτουν πολύ μικρό TBox και μεγάλο (σε σύγκριση με το τελευταίο) ABox. Ως εκ τούτου, τις χρησιμοποιούμε, όπως θα δούμε στην Παράγραφο 7.2.4, για τον έλεγχο της απόδοσης του DBRS κατά την αποτίμηση ABox ερωτημάτων.

	Thesaurus	**LUBM-1**	**LUBM-2**	**LUBM-3**	**LUBM-4**	**LUBM-5**
Κλάσεις	62.354	43	43	43	43	43
Ρόλοι	197	32	32	32	32	32
Άτομα	3	12.169	26.671	38.611	54.259	70.675
Δηλώσεις κλάσεων	0	13.123	28.843	41.842	58.878	76.721
Δηλώσεις ρόλων αντικειμένου	0	49.246	113.250	166.343	235.995	308.750
Δηλώσεις ρόλων τύπου δεδομένων	3	28.074	64.425	94.493	133.748	175.028

Πίνακας 7.1: Ποσοτικά χαρακτηριστικά των οντολογιών που χρησιμοποιήθηκαν για τον έλεγχο της απόδοσης του DBRS

7.2.2 *Η επίδραση του μεγέθους της οντολογίας και των παραμέτρων φόρτωσης στο συνολικό χρόνο αποθήκευσης της πληροφορίας στη βάση δεδομένων του DBRS*

Σχετικά με την επίδραση των παραμέτρων φόρτωσης στο συνολικό χρόνο που απαιτείται, εξετάζουμε μόνο τις περιπτώσεις που ο χρήστης επιθυμεί να αποθηκεύσει από την αρχή ένα ψευδομοντέλο για κάθε κλάση[43] και ένα ψευδομοντέλο για κάθε άτομο της εκάστοτε οντολογίας. Η επιλογή μας αυτή οφείλεται σε δύο λόγους. Από τη μία, στην περίπτωση οντολογιών τέτοιου μεγέθους και πολυπλοκότητας, οι μόνες παράμετροι που μπορούν να επιλεχθούν, έτσι ώστε η φόρτωση σε ένα συνήθη προσωπικό υπολογιστή να ολοκληρωθεί σε ρεαλιστικό χρόνο, είναι οι δύο συγκεκριμένες. Από την άλλη, η φόρτωση των ψευδομοντέλων κατά την αρχικοποίηση του DBRS επηρεάζει, όπως θα δούμε στη συνέχεια, την απόδοσή του σε μια σειρά βασικών ερωτημάτων και συνεπώς έχει αξία να εξετάσουμε το «κόστος» μιας τέτοιας επιλογής.

Τα αποτελέσματα των μετρήσεων (σε δευτερόλεπτα) για κάθε μια από τις οντολογίες της Παραγράφου 7.2.1 δίνονται στον Πίνακα 7.2.

	Απλή Φόρτωση	**Φόρτωση με ένα ψευδομοντέλο για κάθε κλάση**	**Φόρτωση με ένα ψευδομοντέλο για κάθε άτομο**
Thesaurus	1.736	6.051	1.736
LUBM-1	332	333	460
LUBM-2	895	896	1.203
LUBM-3	1.862	1.863	2.287
LUBM-4	3.998	3.999	4.529
LUBM-5	8.861	8.862	9.620

Πίνακας 7.2: Χρόνοι φόρτωσης οντολογιών από το DBRS (seconds)

[43] Για την ακρίβεια, όταν ο χρήστης επιλέξει να φορτώσει από την αρχή στη βάση δεδομένων ένα ψευδομοντέλο για κάθε κλάση, το DBRS φορτώνει και το αντίστοιχο ψευδομοντέλο του συμπληρώματός της (complement class).

Όπως ήταν αναμενόμενο, η αύξηση του μεγέθους της οντολογίας συνεπάγεται και την αύξηση του συνολικού χρόνου φόρτωσής της στη βάση δεδομένων του DBRS. Αύξηση, σε σχέση με την απλή φόρτωση, παρατηρείται επίσης και στην περίπτωση αποθήκευσης των ψευδομοντέλων. Ενδεικτικά αναφέρουμε ότι, για τη Thesaurus, ο συνολικός αριθμός των εγγραφών στους τέσσερεις πίνακες των ψευδομοντέλων των κλάσεων είναι 1.689.024, ενώ ο αντίστοιχος αριθμός για τα ψευδομοντέλα των ατόμων στην περίπτωση της LUBM-5 είναι 305.962.

Όσον αφορά στις LUBM οντολογίες, η φόρτωση με ένα ψευδομοντέλο για κάθε κλάση διαρκεί (περίπου) όσο και η απλή φόρτωση, επειδή το TBox τους είναι πάρα πολύ μικρό. Το ίδιο ισχύει και στην περίπτωση της Thesaurus για τους χρόνους απλής φόρτωσης και φόρτωσης με ένα με ένα ψευδομοντέλο για κάθε άτομο (δεν υπάρχει ABox).

Τελειώνοντας, σημειώνουμε ότι ο χρόνος απλής φόρτωσης της Thesaurus αντιστοιχεί ουσιαστικά στον χρόνο αποθήκευσης του TBox. Η πολυπλοκότητα του τελευταίου μπορεί να επιβεβαιωθεί από μια πρόχειρη επισκόπηση της βάσης δεδομένων του DBRS μετά την ολοκλήρωση της διαδικασίας. Εκεί βρίσκουμε 10.177 ορισμούς κλάσεων στον πίνακα «Concepts» και 44.439 επιπλέον ετικέτες στον πίνακα «DAG Concept Labels» (για τις ακμές του ακυκλικού γράφου της ιεραρχίας των κλάσεων που δεν ανήκουν στο βέλτιστο συνδετικό δέντρο που επιλέχθηκε)[44].

7.2.3 Επιδόσεις συστήματος στην περίπτωση αποτίμησης TBox ερωτημάτων

Όπως προαναφέρθηκε, για την αξιολόγηση της επίδοσης του συστήματος στην περίπτωση των TBox ερωτημάτων χρησιμοποιούμε την οντολογία Thesaurus.

Τα ερωτήματα που υποβάλουμε ανήκουν σε δύο γενικές κατηγορίες και διαφέρουν μόνο ως προς τα ορίσματά τους. Η πρώτη κατηγορία περιλαμβάνει ερωτήματα ιεραρχίας κλάσεων (SubClassOf queries), ενώ η δεύτερη ερωτήματα

[44] Υπενθυμίσουμε εδώ ότι το DBRS «ζητά» από τον Pellet μόνο τους απευθείας απογόνους (direct descendants) για κάθε κλάση της οντολογίας (ομοίως και για τους ρόλους) και στη συνέχεια κατασκευάζει την πλήρη ιεραρχία με χρήση του αλγορίθμου που προτάθηκε στο [] και που ένα μέρος του υλοποιήθηκε με SQL.

ασυμβατότητας κλάσεων (DisjointWith queries)[45]. Κάθε ερώτημα υποβάλλεται δύο φορές, προκειμένου να εξετάσουμε την επίδραση του caching (τόσο από τον Pellet όσο και από το DBRS) στον εκάστοτε χρόνο αποτίμησης.

Στον Πίνακα 7.3 δίνονται τα αποτελέσματα των μετρήσεων (σε χιλιοστά του δευτερολέπτου) για τα είδη των ερωτημάτων που αναφέραμε. Συμβολίζουμε με C, C_1, C_2 γνωστές κλάσεις (named classes) της οντολογίας και με ? μεταβλητή (variable).

Η τελευταία γραμμή του Πίνακα 7.3 περιλαμβάνει το εποπτικό ερώτημα ClassRelatedAxioms(C), το οποίο επιστρέφει σε μορφή αξιωμάτων όλη την πληροφορία που υπάρχει (εκείνη τη στιγμή) στη βάση δεδομένων του DBRS και η οποία σχετίζεται με την κλάση C. Παρότι αντίστοιχο ερώτημα δεν έχει υλοποιηθεί για τον Pellet, το συμπεριλαμβάνουμε στον έλεγχο λόγω της σημαντικότητάς του. Αν αναλογιστεί κανείς ότι η βάση δεδομένων του DBRS «γεμίζει» με επιπλέον πληροφορία στις περιπτώσεις εξαγωγής (από τον Reasoner) νέων (υπονοούμενων) σχέσεων, μπορούμε με χρήση του συγκεκριμένου ερωτήματος να συνθέσουμε μια νέα «έκδοση» του αρχείου της οντολογίας που διαχειριζόμαστε, με περισσότερα ρητά (explicit) αξιώματα και κατά συνέπεια με μικρότερο «κόστος» ανάλυσης στην επόμενη φόρτωση από τον Reasoner.

Για τα ερωτήματα SubClassOf(C_1 , C_2) και DisjointWith(C_1 , C_2) χρησιμοποιούμε 10.000 τυχαία ζεύγη γνωστών κλάσεων. Στην περίπτωση των SubClassOf(? , C_2) και DisjointWith(? , C_2) χρησιμοποιούμε 10.000 τυχαίες κλάσεις, ενώ για το ερώτημα ClassRelatedAxioms(C) επιλέξαμε την κλάση «» που συμμετέχει σε αξιώματα. Όλα τα ερωτήματα στέλνονται σειριακά (το ένα μετά το άλλο) στον Pellet και το DBRS και τα αποτελέσματα που παρουσιάζουμε αντιστοιχούν στο μέσο χρόνο απάντησης ανά ερώτημα.

Διευκρινίζουμε ότι η υποβολή των TBox ερωτημάτων στον Pellet γίνεται αφού έχει υπολογιστεί η πλήρη ιεραρχία (classification) των κλάσεων της οντολογίας από αυτόν, πράγμα που φαίνεται και από τους μικρούς χρόνους αποτίμησης των σχετικών ερωτημάτων. Η επιλογή μας αυτή οφείλεται στο ότι η κατασκευή της πλήρους ιεραρχίας στη Βάση Γνώσης του Pellet γίνεται υποχρεωτικά κατά τη

[45] Δεν συμπεριλαμβάνουμε τα αντίστοιχα ερωτήματα ιεραρχίας και ασυμβατότητας για ρόλους, επειδή η διαδικασία απάντησή τους είναι ακριβώς ίδια με την περίπτωση των κλάσεων.

φόρτωση του TBox της οντολογίας στη Βάση Δεδομένων του DBRS και συγκεκριμένα όταν ζητούνται για πρώτη φορά οι απευθείας απόγονοι μιας κλάσης (βλέπε Κεφάλαια 5 & 6). Έτσι, και για την περίπτωση που ο Pellet λειτουργεί αυτόνομα, θεωρήσαμε ορθό να υποβάλλουμε ερωτήματα σε αυτόν αφού έχει πραγματοποιηθεί το classification του TBox.

	1η φορά (Pellet)	**2η φορά (Pellet)**	**1η φορά (Eager DBRS)**	**2η φορά (Eager DBRS)**
SubClassOf(C_1 , C_2)	0,2320	0,0320	1,4020	1,3900
SubClassOf(? , C)	1,7675	0,7407	14,9764	14,8546
DisjointWith(C_1 , C_2)	0,4120	0,0467	0,7321	0,2345
DisjointWith(? , C)	24,6731	5,3241	25,4562	6,3124
ClassRelatedAxioms(C)	---	---	55,8763	---

Πίνακας 7.3: Χρόνοι αποτίμησης ερωτημάτων TBox για την οντολογία Thesaurus (msecs)

Από τον Πίνακα 7.3, παρατηρούμε ότι ο χρόνος αποτίμησης των ερωτημάτων ιεραρχίας από το DBRS μεταβάλλεται ελάχιστα τη δεύτερη φορά. Αυτό οφείλεται στο ότι το σύστημα, σε αυτήν την περίπτωση, δεν επιτελεί κανένα είδος caching, αλλά απαντά με τον ίδιο τρόπο (μέσω ετικετών) τόσο την πρώτη όσο και τη δεύτερη φορά.

Από την άλλη, στα ερωτήματα ασυμβατότητας, ο χρόνος αποτίμησης από το DBRS μειώνεται σημαντικά τη δεύτερη φορά, εξαιτίας της χρήσης ευρετηρίων στον πίνακα «Disjoint Concepts», αλλά και λόγω του caching που γίνεται στον πίνακα «Binary Query Cacher».

Στην περίπωση του ερωτήματος SubClassOf(C_1 , C_a), εξετάζουμε τον χρόνο απάντησης του DBRS όταν υπάρχει από την αρχή αποθηκευμένο στη βάση δεδομένων ένα ψευδομοντέλο για κάθε κλάση, αλλά και όταν το αντίστοιχο ψευδομοντέλο (της γνωστής κλάσης) φορτώνεται κατά την αποτίμηση του

ερωτήματος (on-the-fly)[46]. Εδώ, η δεύτερη υποβολή των ερωτημάτων δεν έχει νόημα, καθότι η διαδικασία αποτίμησης που ακολουθείται από το DBRS είναι πάντα η ίδια.

Ο Πίνακας 7.4 περιέχει τα αποτελέσματα των μετρήσεων (σε δευτερόλεπτα) για το ερώτημα SubClassOf(C_1 , C_a). Συμβολίζουμε με C γνωστή κλάση και με C_a αυθαίρετη κλάση (arbitrary class) της μορφής «And(C_1 C_2)». Χρησιμοποιούμε 1.000 τυχαία ζεύγη κλάσεων (μια γνωστή και μια αυθαίρετη), ενώ, όπως και πριν, όλα τα ερωτήματα στέλνονται σειριακά στον Pellet και το DBRS και τα αποτελέσματα που παρουσιάζουμε αντιστοιχούν στο μέσο χρόνο απάντησης ανά ερώτημα.

	SubClassOf(C , C_a)
1η φορά (Pellet)	0,3554
2η φορά (Pellet)	0,1344
1η φορά με Ψευδομοντέλα (Eager DBRS)	0,6732
1η φορά χωρίς ψευδομοντέλα (Eager DBRS)	1,4814

Πίνακας 7.4: Χρόνοι αποτίμησης ερωτημάτος SubClassOf(C_1 , C_a) για την οντολογία Thesaurus (seconds)

7.2.4 Επιδόσεις συστήματος στην περίπτωση αποτίμησης ABox ερωτημάτων

Στην περίπτωση των ABox ερωτημάτων, η αξιολόγηση της επίδοσης του συστήματος γίνεται με την οντολογία LUBM-5.

Τα ερωτήματα που υποβάλουμε ανήκουν σε δύο γενικές κατηγορίες και διαφέρουν μόνο ως προς τα ορίσματά τους. Η πρώτη κατηγορία περιλαμβάνει ερωτήματα δηλώσεων κλάσεων (TypeOf queries), ενώ η δεύτερη ερωτήματα δηλώσεων ρόλων (PropertyValue queries). Κάθε ερώτημα υποβάλλεται δύο φορές,

[46] Ως γνωστόν, το ψευδομοντέλο της αυθαίρετης κλάσης φορτώνεται πάντα κατά την αποτίμηση.

προκειμένου να εξετάσουμε την επίδραση του caching στον εκάστοτε χρόνο αποτίμησης.

Στον Πίνακα 7.5 δίνονται τα αποτελέσματα των μετρήσεων (σε δευτερόλεπτα) για τα είδη των ερωτημάτων που αναφέραμε. Συμβολίζουμε με R , R_1 , R_2 ρόλους της οντολογίας, με a, b άτομα και με ? μεταβλητή (variable). Η τελευταία γραμμή του Πίνακα 7.5 περιλαμβάνει το εποπτικό ερώτημα IndividualRelatedAxioms(a), το οποίο επιστρέφει σε μορφή αξιωμάτων όλη την πληροφορία που υπάρχει (εκείνη τη στιγμή) στη βάση δεδομένων του DBRS και η οποία σχετίζεται με το άτομο a.

Για τα ερωτήματα PropertyValue(? , R , b) και PropertyValue(a , R , ?) χρησιμοποιούμε 10.000 τυχαία ζεύγη (ρόλος, άτομο). Στην περίπτωση του PropertyValue(a , ? , b) χρησιμοποιούμε 10.000 τυχαία ζεύγη ατόμων, ενώ για το ερώτημα PropertyValue(a , R , b) διαλέγουμε 10.000 τυχαίες τριάδες (άτομο, ρόλος, άτομο). Όλα τα ερωτήματα στέλνονται σειριακά στον Pellet και το DBRS και τα αποτελέσματα που παρουσιάζουμε αντιστοιχούν στο μέσο χρόνο απάντησης ανά ερώτημα. Τέλος, η μέτρηση που δίνουμε για το IndividualRelatedAxioms(a) αντιστοιχεί στο μέσο χρόνο απάντησης του συγκεκριμένου ερωτήματος για 100 τυχαία άτομα της οντολογίας.

	1η φορά (Pellet)	**2η φορά (Pellet)**	**1η φορά (Eager DBRS)**	**2η φορά (Eager DBRS)**
PropertyValue(a , R , b)	0,3423	0,0321	1,7651	0,4322
PropertyValue(? , R , b)	1,7632	0,3325	2,5421	0,8245
PropertyValue(a , R , ?)	1,6722	0,2787	2,6433	0,7674
PropertyValue(a , ? , b)	1,9862	0,5543	2,7891	0,9534
IndividualRelatedAxioms(a)	---	---	32,4312	---

Πίνακας 7.5: Χρόνοι αποτίμησης ερωτημάτων ABox για την οντολογία LUBM-5 (seconds)

Από τον Πίνακα 7.5, παρατηρούμε ότι, για όλα τα είδη ερωτημάτων PropertyValue, ο χρόνος αποτίμησης από το DBRS μειώνεται σημαντικά τη δεύτερη

φορά. Αυτό οφείλεται στο caching που γίνεται στον πίνακα «Property Value Query Cacher».

Στην περίπωση των ερωτημάτων TypeOf(a , C) και TypeOf(a , C_a), εξετάζουμε την απόδοση του DBRS όταν υπάρχει από την αρχή αποθηκευμένο στη βάση δεδομένων ένα ψευδομοντέλο για κάθε κλάση και ένα για κάθε άτομο, αλλά και όταν τα αντίστοιχα ψευδομοντέλα φορτώνονται κατά την αποτίμηση του ερωτήματος. Την ίδια ακριβώς διαδικασία ακολουθούμε και για τα ερωτήματα TypeOf(a , ?) και TypeOf(? , C), καθότι σε αυτά το DBRS επιτελεί ένα είδος «φιλτραρίσματος», στέλνοντας στον Pellet μόνο τις περιπτώσεις που δεν μπορεί να απαντήσει με χρήση του Mergable Test (βλέπε Κεφάλαιο 5).

Στον Πίνακα 7.4 δίνονται οι μετρήσεις (σε δευτερόλεπτα) για τα ερωτήματα TypeOf. Συμβολίζουμε με C γνωστή κλάση της οντολογίας, με C_a αυθαίρετη κλάση της μορφής «And(C_1 C_2)», με a άτομο και με ? μεταβλητή (variable).

Στην περίπτωση των TypeOf(a , C) και TypeOf(a , C_a) χρησιμοποιούμε 1.000 τυχαία ζεύγη (άτομο, κλάση). Για το ερώτημα TypeOf(a , ?) χρησιμοποιούμε 100 τυχαία άτομα και για το TypeOf(? , C) 100 τυχαίες κλάσεις. Όπως και πριν, όλα τα ερωτήματα στέλνονται σειριακά στον Pellet και το DBRS και τα αποτελέσματα που παρουσιάζουμε αντιστοιχούν στο μέσο χρόνο απάντησης ανά ερώτημα.

	TypeOf(a , C)	**TypeOf(a , C_a)**	**TypeOf(a , ?)**	**TypeOf(? , C)**
1η φορά (Pellet)	0,2721	0,6734	1,7654	1,9654
2η φορά (Pellet)	0,0124	0,0379	0,1732	0,2312
1η φορά με ψευδομοντέλα (Eager DBRS)	0,5631	1,4769	2,3241	2,6731
1η φορά χωρίς ψευδομοντέλα (Eager DBRS)	1,2348	1,4753	2,2748	2,0248
2η φορά (Eager DBRS)	0,0235	1,3955	0,6376	0,4327

Πίνακας 7.6: Χρόνοι αποτίμησης ερωτημάτων TypeOf για την οντολογία LUBM-5 (seconds)

8

Επίλογος

Στο παρόν κεφάλαιο επιχειρούμε μια σύντομη ανασκόπηση της διπλωματικής εργασίας, τη σύνοψη των αποτελεσμάτων της και την ανάδειξη της συνεισφοράς της σχετικά με τα προβλήματα που αντιμετώπισε. Επίσης, βάσει της εμπειρίας μας, προτείνουμε (στην Ενότητα 8.3) πιθανές μελλοντικές επεκτάσεις και γενικές βελτιώσεις του συστήματος DBRS.

8.1 Σύνοψη

Η προσπάθεια προσθήκης των μεταδεδομένων στον Παγκόσμιο (συντακτικό) Ιστό (World Wide Web) για την ανάπτυξη του Σημασιολογικού Ιστού (Semantic Web) γεννά μια σειρά από προκλήσεις που τα τελευταία χρόνια απασχολούν έντονα την επιστημονική κοινότητα. Ένα ιδιαιτέρως φλέγον ζήτημα είναι η δυνατότητα ολοκληρωμένης διαχείρισης μεταδεδομένων, οργανωμένων με τη μορφή οντολογιών (ontologies), που διακρίνονται για τον πολύ μεγάλο όγκο τους, αλλά και την πλούσια εκφραστικότητά τους (expressivity).

Στις πραγματικές εφαρμογές του Σημασιολογικού Ιστού, ο όγκος (και η πολυπλοκότητα) των οντολογιών τείνει να αυξάνεται. Αποτέλεσμα αυτής της αύξησης είναι η αδυναμία συλλογιστικής ανάλυσης των μεταδεδομένων (reasoning),

λόγω του αυξημένου κόστους που συνεπάγεται μια τέτοια διαδικασία σε κατανάλωση Κύριας Μνήμης, αλλά και η αδυναμία, σε ορισμένες περιπτώσεις, απλής φόρτωσής τους (loading) σε αυτήν. Αν μάλιστα στο τελευταίο προσθέσουμε και το γεγονός ότι οι σημασιολογικές σχέσεις είναι μέχρι στιγμής πρακτικώς αδύνατο να κατατμηθούν και να αναλυθούν επιμέρους («ή όλα ή τίποτα»), αντιλαμβανόμαστε εύκολα το λόγο για τον οποίο η επιστημονική έρευνα προσανατολίζεται στην ανάπτυξη τεχνικών που θα εκμεταλλεύονται μηχανισμούς δευτερεύουσας μνήμης (second storage mechanisms), έτσι ώστε να ξεπεραστούν τα απαγορευτικά όρια που παρουσιάζονται.

Στην κατεύθυνση αυτή, η παρούσα διπλωματική εργασία αποσκοπούσε στον προσδιορισμό των απαιτήσεων και την κατασκευή ενός συστήματος ολοκληρωμένης διαχείρισης πολύ μεγάλου όγκου σημασιολογικών σχημάτων (εκατομμύρια τριάδες του RDF μοντέλου), η εκφραστικότητα των οποίων ανήκει στο τμήμα $SHOIN^{(D)}$ της Περιγραφικής Λογικής (Description Logic - DL), ένα σύνολο λογικών φορμαλισμών που χρησιμοποιούνται για αναπαράσταση γνώσης και το οποίο περιγράψαμε εκτενώς μαζί με τις επικρατέστερες μεθόδους συλλογιστικής ανάλυσης σε αυτό. Το εύρος εκφραστικότητας $SHOIN^{(D)}$ αντιστοιχεί στη γλώσσα OWL-DL και το επιλέξαμε επειδή είναι αποκρίσιμο (decidable), αλλά και ταυτόχρονα επαρκές για την περιγραφή των χαρακτηριστικών και των σχέσεων των δεδομένων που υπάρχουν στο διαδίκτυο.

Κεντρική ιδέα της υλοποίησης ήταν ο συνδυασμός ενός Σχεσιακού Συστήματος Διαχείρισης Βάσεων Δεδομένων (Relational DataBase Management System - RDBMS) με μια Μηχανή Συλλογιστικής Ανάλυσης (Reasoner), έτσι ώστε η πρώτη λειτουργική μονάδα (RDBMS) να χρησιμοποιείται για την οργάνωση και τη μόνιμη αποθήκευση της σημασιολογικής πληροφορίας σε δευτερεύουσα μνήμη (Βάση Δεδομένων), ενώ η δεύτερη (Reasoner) για την εκτέλεση όλων εκείνων των απαραίτητων επαγωγικών διαδικασιών (inference procedures) που προϋποθέτει η διαχείριση των σημασιολογικών σχημάτων.

Συνοπτικά, η εργασία μας επικεντρώθηκε σε δύο βασικούς άξονες. Ο ένας σχετίζεται, όπως προαναφέρθηκε, με την ολοκληρωμένη διαχείριση σημασιολογικής πληροφορίας εκφραστικότητας $SHOIN^{(D)}$, δηλαδή με την πληρότητα του συστήματος, ενώ ο άλλος με την απόδοσή του (performance). Με άλλα λόγια, σε αντίθεση με πολλές από τις υπάρχουσες πλατφόρμες και σχετικές εργασίες,

επιχειρήσαμε την κατασκευή ενός συστήματος που όχι μόνο θα οργανώνει σε δευτερεύουσα μνήμη τα μεταδεδομένα θεωρώντας τα «ορθά», αλλά θα επιτελεί (συνδυάζοντας RDBMS με Reasoner) και όλες εκείνες τις διεργασίες, είτε «ελαφριάς» είτε «βαριάς» συλλογιστικής ανάλυσης, που είναι απαραίτητες τόσο για τον έλεγχο της λογικής συνέπειας (consistency) της προς αποθήκευση γνώσης, όσο και για την πλήρη (sound & complete) αποτίμηση ερωτημάτων σχετικά με αυτήν (ανοιχτός κόσμος -«open world»).

Ιδιαίτερη έμφαση δώσαμε στην κατασκευή ενός σχήματος βάσης δεδομένων που αφενός θα μοντελοποιεί τη $SHOIN^{(D)}$ πληροφορία χωρίς απώλειες, αφετέρου θα επιτρέπει τη γρήγορη αποτίμηση ερωτημάτων. Όσον αφορά στο τελευταίο, πέρα από τη χρήση της «κλασσικής» τεχνολογίας βάσεων δεδομένων (π.χ. Ευρετήρια), χρησιμοποιήσαμε και μια σειρά από άλλες τεχνικές βελτιστοποίησης. Ενδεικτικά παραδείγματα αποτελούν το Σχήμα Ετικετών (Labeling Scheme) για την αναπαράσταση και κωδικοποίηση των ιεραρχικών δομών (hierarchical status) στη βάση δεδομένων, αλλά και ο Επίπεδος Έλεγχος Συγχώνευσης (Flat Pseudomodel Mergable Test) για την υπό προϋποθέσεις απάντηση σε ερωτήματα «ανοιχτού κόσμου» από το DBMS.

8.2 Αποτελέσματα - Συνεισφορά

Μέχρι στιγμής, το DBRS έχει δοκιμαστεί με πραγματικές οντολογίες πολύ μεγάλου όγκου (όπως η Thesaurus [12] που περιλαμβάνει 1.500.000 RDF τριάδες) και λειτουργεί χωρίς προβλήματα και με ικανοποιητικούς χρόνους. Μια πρόχειρη σύγκριση της απόδοσης του συστήματος με αυτή της Μηχανής Συλλογιστικής Ανάλυσης Pellet, στην περίπτωση που η τελευταία λειτουργεί μόνη της (stand-alone application), παρουσιάστηκε στο Κεφάλαιο 7 για ένα μικρό σύνολο βασικών ερωτημάτων και τα αποτελέσματα είναι ενθαρρυντικά. Σύμφωνα με αυτά και δεδομένου ότι υπάρχουν ακόμη πολλά περιθώρια βελτιώσεων και επεκτάσεων, θεωρούμε ότι η έρευνα προς την κατεύθυνση που κι εμείς ακολουθήσαμε, δηλαδή του συνδυασμού της τεχνολογίας Βάσεων Δεδομένων με Μηχανές Συλλογιστικής Ανάλυσης, είναι πολλά υποσχόμενη και πρέπει να συνεχιστεί.

Στο σημείο αυτό, συνοψίζουμε τη συνεισφορά της διπλωματικής μας εργασίας στα παρακάτω:

- Κατασκευάσαμε ένα πλήρες μοντέλο Οντοτήτων-Συσχετίσεων για γνώση εκφραστικότητας $SHOIN^{(D)}$, από την επεξεργασία του οποίου προέκυψε το σχεσιακό σχήμα της βάσης δεδομένων του DBRS.
- Υλοποιήσαμε ένα μηχανισμό αμφίδρομης επικοινωνίας μεταξύ της Βάσης Γνώσης της Μηχανής Συλλογιστικής Ανάλυσης (που βρίσκεται στην Κύρια Μνήμη) και της βάσης δεδομένων του DBRS. Αποτελέσματα αυτής της επικοινωνίας είναι:
    - Επέκταση της Μηχανής Συλλογιστικής Ανάλυσης με δευτερεύουσα μνήμη, γεγονός που προσδίδει στις εφαρμογές περισσότερη σταθερότητα και αξιοπιστία (reliability) λόγω της μόνιμης αποθήκευσης.
    - Διευκόλυνση και επιτάχυνση, λόγω της τεχνολογίας Βάσεων Δεδομένων, των επαγωγικών διαδικασιών της Μηχανής Συλλογιστικής Ανάλυσης, χάρη στη χρήση τεχνικών όπως το Σχήμα Ετικετών και ο Επίπεδος Έλεγχος Συγχώνευσης Ψευδομοντέλων. Γενικώς, η βάση δεδομένων του DBRS χρησιμοποιείται όχι απλώς για τη μόνιμη αποθήκευση της πληροφορίας, αλλά και ως «φίλτρο», στέλνοντας στον Reasoner μόνο τις περιπτώσεις που δεν μπορεί να αποκλείσει και για τις οποίες η εκτέλεση του Tableau αλγόριθμου είναι αναπόφευκτη.
    - Δυνατότητα ενοποίησης πολλών διαφορετικών Βάσεων Γνώσης, ακόμα και από διαφορετικές Μηχανές Συλλογιστικής Ανάλυσης, σε μια κοινή βάση δεδομένων. Πολλοί χρήστες του DBRS μπορούν να συνδεθούν στην ίδια βάση δεδομένων, αρκεί να έχουν τα απαραίτητα δικαιώματα.
- Αναπτύξαμε αλγόριθμους "ελαφριάς" συλλογιστικής ανάλυσης για την εξαγωγή μέρους υπονοούμενης γνώσης (ανοιχτός κόσμος) από τη ΒΔ (κλειστός κόσμος).
- Αναπτύξαμε αλγόριθμους που συνδυάζουν DBMS και Reasoner για την επιτάχυνση της διαδικασίας φόρτωσης του DBRS, αλλά και της αποτίμησης ερωτημάτων από αυτό.
- Όσον αφορά στο βιβλιογραφικό κομμάτι, μελετήσαμε και παρουσιάσαμε έντεκα δημοφιλή συστήματα διαχείρισης οντολογιών.

8.3 Μελλοντικές επεκτάσεις

Ανάμεσα στις πιθανές μελλοντικές επεκτάσεις και βελτιώσεις του συστήματος DBRS είναι και οι παρακάτω:

- Αναβάθμιση του Υποσυστήματος Αποτίμησης ερωτημάτων, έτσι ώστε να αναγνωρίζει και να επεξεργάζεται την πρότυπη (W3C standard) και ευρέως διαδεδομένη γλώσσα ερωτήσεων SPARQL.
- Δυνατότητα ενημερώσεων της βάσης δεδομένων του DBRS, ανάλογα με τις ενημερώσεις που γίνονται στο αρχείο της οντολογίας, είτε χειρονακτικά (manually) είτε μέσω κάποιας διεπαφής, όπως το OWL API που προσφέρει αυτή τη δυνατότητα. Για παράδειγμα, το Σχήμα Ετικετών για την ιεραρχία των κλάσεων στη βάση δεδομένων πρέπει να ενημερώνεται σε περίπτωση που αφαιρεθεί μια ή περισσότερες κλάσεις από το αρχικό αρχείο της οντολογίας.
- Επέκταση των μεθόδων συγχώνευσης ψευδομοντέλων στη βάση δεδομένων του DBRS, έτσι ώστε να λαμβάνουν υπόψη τους την επιπλέον πληροφορία που προκύπτει από την ύπαρξη αντίστροφων ρόλων (I), απαριθμητικών (O) και τύπων δεδομένων (D).
- Ανάπτυξη τεχνικών για διευκόλυνση της Μηχανής Συλλογιστικής Ανάλυσης κατά τον αρχικό έλεγχο συνέπειας ολόκληρης της οντολογίας, έτσι ώστε να ξεπεραστούν τα απαγορευτικά όρια που θέτει το μέγεθος της Κύριας Μνήμης Μια πρόσφατα δημοσιευμένη προσπάθεια που κινείται σε αυτήν την κατεύθυνση και που παρουσιάζει ιδιαίτερο ενδιαφέρον μπορεί να βρεθεί στο [FKM+06].

9

Βιβλιογραφία

[BBC+99] P.A. Bernstein, Th. Bergstraesser, J. Carlson, S. Pal, P. Sanders, D. Shutt. Microsoft Repository Version 2 and the Open Information Model. To appear in Information Systems 24(2), 1999.

[AH04] Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2004) 67-92.

[BCG+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider: The Description Logic Handbook: Theory, Implementation, Applications. Cambridge University Press, Cambridge, UK, 2003.

[ACK+01] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis & K.Tolle, On Storing Voluminous RDF Description: The case of Web Portal Catalogs, In Proc. of the 4th International Workshop on the Web and Databases (WebDB2001) in conjunction with ACM SIGMOD'01 Conference, 2001.

[RA06] Maria del Mar Roldan-Garcia, Jose F. Aldana-Montes: A Survey on Disk Oriented Querying and Reasoning on the Semantic Web,

Proceedings of the 22nd International Conference on Data Engineering Workshops, 2006.

[ACK+00] Sofia Alexaki, Vassilis Christophides, Greg Karvounarakis, Dimitris Plexousakis, Karsten Tolle: The RDFSuite: Managing Voluminous RDF Description Bases. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece, 2000.

[KAC+02] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl: RQL: A declarative query language for RDF, Proceedings of the 11th International World Wide Web Conference (WWW2002). 2002.

[HLTB04] I.Horrocks, L.Li, D.Turi, and S.Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In Description Logic Workshop, 2004.

[BHT05] Bechhofer, S., Horrocks, I., Turi, D.: The OWL instance store: System description. In Proc. of the 20th Int. Conf. on Automated Deduction (CADE-20). Lecture Notes in Artificial Intelligence, Springer (2005).

[PH03] Zhengxiang Pan and Jeff Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. In Workshop on Practical and Scalable Semantic Systems, ISWC 2003.

[GHVD03] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic programs: Combining logic programs with Description Logic. In Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary,2003.

[WLS03] Timo Weithoener, Thorsten Liebig, and Guenther Specht: Storing and Querying Ontologies in Logic Databases, First International Workshop on Semantic Web and Databases, Sept. 7-8 2003.

[O05] Owens, A.. Semantic Storage: Overview and Assessment. Technical Report IRP Report 2005, Electronics and Computer Science, U of Southampton, 2005.

[WGA05] D. Wood, P. Gearon, and T. Adams. Kowari: A platform for semantic web storage and analysis. In Proceedings of the 14th International WWW Conference, 2005.

[Kowari] Kowari Technical Documentation, http://www.kowari.org/ .

[BK01] Jeen Broekstra and Arjohn Kampman, Sesame: A generic architecture for storing and querying RDF and RDF Schema, Technical report, Aidministrator Nederland b.v., October 2001.

[Sesame] Sesame Technical Documentation, http://www.openrdf.org/ .

[HG03] Harris, S., Gibbins, N.: 3store: Efficient bulk RDF storage. In: PSSS'03, Sanibel, Fl (2003) 1–15

[OWLIM] OWLIM Technical Documentation, http://www.ontotext.com/owlim .

[CDD+04] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, K. Wilkinson, Jena: Implementing the semantic web recommendations, in: Proc. of the 13th Int. World Wide Web Conference (WWW 2004), 2004.

[B02] B. McBride Jena IEEE Internet Computing, July/August, 2002.

[WSKR03] K. Wilkinson, C. Sayers, H. A. Kuno, D. Raynolds: Efficient RDF Storage and Retrieval in Jena2. In Proc. of SWDB'03 (co-located with VLDB'03).

[CDD+03] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. The Jena Semantic Web Platform: Architecture and design. Technical report, Hewlett Packard Laboratories, Palo Alto, California, 2003.

[R03] D. Reynolds, Jena Relational Database Interface – Performance Notes, in Jena 1.6.1 download: http://www.hpl.hp.com/semweb/download.htm .

[VOSM03] Raphael Volz, Daniel Oberle, Steffen Staab, Boris Motik, KAON SERVER - A Semantic Web Management System, WWW2003, May 20-24, 2003, Budapest, Hungary.

[W05] J.Y. Wang : Large Abox Store (LAS): Database Support for Tbox Queries,

Master thesis, Department of Computer Science and Software Engineering,

Concordia University 2005.

[C05] Cuiming Chen. Large ABox Store (LAS): Database Support for ABox Queries. Master thesis, Concordia University, Computer Science department, September 2005.

[HMT01] V. Haarslev, R. Moller, and A.-Y. Turhan. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In Proceedings of the International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, 2001, Siena, Italy, LNCS. Springer-Verlag, Berlin, June 2001.

[RDF] RDF Tutorial , http://www.w3.org/RDF .

[OWL] OWL Overview, http://www.w3.org/TR/owl-features .

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

[BLproposal] Tim Berners-Lee's original proposal to CERN: http://www.w3.org/History/1989/proposal-msw.html .

[Pellet] Pellet: The Open Source OWL - DL Reasoner, http://pellet.owldl.com/ .

[Postgres] PostgreSQL: open source database, http://www.postgresql.org .

[XML] Extensible Markup Language (XML): http://www.w3.org/XML/ .

[TH02] S. Tessaris, I. Horrocks : Abox Satisfiability Reduced to Terminological Reasoning in Expressive Description Logics. In Matthias Baaz and Andrei Voronkov, editors, Proc. Of the 9th int. conf. On logic for programming and automated reasoning (lpar'02), volume 2514 of Lecture Notes in Computer Science. Springer, 2002.

[BS01] Baader, F. and U. Sattler: An Overview of Tableau Algorithms for Description Logics. Studia Logica 69(1), 5–40. 2001.

[HS05] Horrocks, I., and Sattler, U. A Tableaux Decision Procedure for SHOIQ. In Proc. of 19th International Joint Conference on Artificial Intelligence (IJCAI 2005) , Morgan Kaufmann, Los Altos. 2005.

[HKS06] Horrocks, I., O.Kutz, and Sattler, U. The Even More Irresistible SROIQ. In Proc. of KR-06 (2006).

[HS02] Horrocks, I., and Sattler, U. Optimised reasoning for SHIQ. In Proc. of the 15th European Conf. on Artificial Intelligence (ECAI 2002). 2002.

[CHW04] C. Chen, V. Haarslev, and J. Wang. LAS: Extending Racer by a Large ABox Store. In Proc. of the Int. Description Logic Workshop (DL 2005), pages 41–50. 2004.

[DAML] The DARPA Agent Markup Language: http://www.daml.org/ .

[UML] The Unified Modeling Language™: http://www.uml.org/ .

[OWL1.1] OWL 1.1 Web Ontology Language: http://www.webont.org/owl/1.1/ .

[FKM+06] A.Fokou, A.Kershenbaum, L.Ma, E.Schonberg, and K.Srinivas: The summary Abox: Cutting ontologies down to size. Proc. of the Int. Semantic Web Conf.(ISWC 2006) 136–145. 2006.

[ORC] Using Oracle Database Features: http://www.oracle.com/technology/products/ias/toplink/doc/1013/main/_html/qryadv008.htm .

[D82] P. F. Dietz: Maintaining order in a linked list. In Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing (STOC'82) pages 122-127. 1982.

[W88] N. Wirth: Type extensions. ACM Trans. on Progr. Languages and Systems, 10(2):204-214. 1988.

[ABJ89] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In Proc. of the SIGMOD Inter. Conf. On Manag. Of Data, pages 253-262. 1989.

[LM2001] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In Proc. of 27th Inter. Conf. on Very Large Data Bases(VLDB'01). 2001.

[CPST03] Christophides, V., Plexousakis, D., Scholl, M., Tourtounis, S.: On labeling schemes for the semantic web. In: Proceedings of the twelfth international conference on World Wide Web, ACM Press (2003) 544-555. 2003.

[Redland] Redland rdf application framework: http://www.librdf.org .

[RDQL] RDQL - A Query Language for RDF: http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/ .

[MSP+04] L. Ma, Z. Su, Y. Pan, L. Zhang, T. Liu: RStar: An RDF Storage and Query System for Enterprise Resource Management. In Proc. of the ACM CIKM 2004.

[RACER] RACER, Renamed Abox and Concept Expression Reasoner: http://www.sts.tu-harburg.de/~r.f.moeller/racer/ .

[SPARQL] SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/ .

[Joseki] Joseki – A SPARQL server for Jena: http://www.joseki.org/ .

[OWLAPI] The OWL API: http://owlapi.sourceforge.net/ .

[RMI] Remote Method Invocation (RMI): http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp .

[JMX] Java Management Extensions (JMX) Technology: http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/ .

[JBoss] JBoss Enterprise Application Platform: http://www.jboss.com/products/platforms/application .

[FaCT] The FaCT System: http://www.cs.man.ac.uk/~horrocks/FaCT/ .

[AKS97] Abraham Silberschatz, Henry F. Korth, S. Sudarshan: Database system concepts, 4th Ed. ISBN 0-07-228363-7. 1997.

[JDBC] The JDBC API: http://java.sun.com/javase/technologies/database/ .

[ATerm] The ATerm Library: http://www.mindswap.org/2003/pellet/ .

[SP07] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In 3rd OWL Experiences and Directions Workshop (OWLED-2007). 2007.

[Eclipse] Eclipse: http://www.eclipse.org/ .

[Thesaurus] The Thesaurus ontology: http://ncicb.nci.nih.gov/download/evsportal.jsp .

[LUBM] Lehigh University Benchmark: http://swat.cse.lehigh.edu/projects/lubm/ .