ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
Τμήμα Ηλεκτρονικών Μηχανικών &
Μηχανικών Υπολογιστών

4D FLUENTS: Plug-in για το πρόγραμμα σχεδίασης οντολογιών Protégé

Εξεταστική επιτροπή:

Αν. Καθηγητής Ευριπίδης Πετράκης (επιβλέπων)

Επ. Καθηγητής Μιχαήλ Λαγουδάκης

Καθηγητής Μίνως Γαροφαλάκης

Λεωνίδας Πάρτσας

Χανιά 2012

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Με τον όρο Semantic Web[1] αναφερόμαστε στο επόμενο στάδιο του παγκοσμίου ιστού (World Wide Web) στον οποίο το νόημα της πληροφορίας και των υπηρεσιών θα ορίζεται πλήρως δίνοντας τη δυνατότητα:

i. στο “διαδίκτυο” να καταλαβαίνει και ικανοποιεί καλύτερα τις ανάγκες των χρηστών και
ii. στις μηχανές να επεξεργάζονται και χειρίζονται αποτελεσματικότερα τη πληροφορία που διακινείται.

Βασικός στόχος του semantic web είναι πλαισιώσει τη πληροφορία που βρίσκεται σήμερα στο διαδίκτυο με τη χρήση υπερδομών οι οποίες θα οργανώνουν και χαρακτηρίζουν τη πληροφορία ενώ παράλληλα θα είναι σε μορφή ικανή να επεξεργαστεί από εφαρμογές ώστε αυτές να μαζεύουν, αναλύουν και παρουσιάζουν τη πληροφορία με τον τρόπο που χρειάζεται.

Για παράδειγμα αν κάποιος ενδιαφέρεται να αγοράσει ένα αυτοκίνητο θα πρέπει, αφού ορίσει το ακριβές μοντέλο που ψάχνει, να βρει τι τιμές και υπηρεσίες παρέχει ο κάθε αντιπρόσωπος και να συγκρίνει τα αποτελέσματα μόνος του. Αν όμως οι πληροφορίες αυτές έχουν οργανωθεί με την βοήθεια υπερδομών δημιουργώντας μια οντολογία της αγοράς αυτοκινήτων τότε η αρμόδια εφαρμογή θα μπορεί να κάνει όλα τα παραπάνω από μόνη της και απλά να ενημερώσει τον χρήστη για το αποτέλεσμα της αναζήτησης του.

1.1 Ορισμός του προβλήματος

Ένα από τα βασικά προβλήματα που αντιμετωπίζουμε κατά την αναπαράσταση οντολογιών είναι η διαχείριση πληροφορίας που μεταβάλλεται μέσα στον χρόνο. Μάλιστα το πρόβλημα αυτό επιδεινώνεται ακόμα περισσότερο όταν για την αναπαράσταση κάνουμε χρήση γλωσσών που υποστηρίζουν μόνο δυαδικές σχέσεις, όπως είναι η OWL, η RDF και όσες άλλες ανήκουν στην κατηγορία της "περιγραφικής λογικής". Ακόμα και όταν η ίδια η σχέση είναι δυαδική, όπως το να κατοικεί κάποιος σε μια συγκεκριμένη διεύθυνση ή να είναι μέλος μιας οργάνωσης, το γεγονός ότι μπορεί να αλλάξει περιπλέκει τα πράγματα σε τέτοιο σημείο που οι άνωθεν γλώσσες επιλέγουν απλά να αγνοήσουν το πρόβλημα τελείως.

Έστω ότι έχουμε ένα σύστημα το οποίο με την βοήθεια της OWL σχηματίζει μια οντολογία με αντικείμενα και σχέσεις παρουσιάζοντας τη διαμονή των κατοίκων μιας πόλης. Σε μια τέτοια οντολογία μια πρόταση όπως η παρακάτω:

Ο Λεωνίδας κατοικεί στην οδό Ρέας 7

θα δημιουργούσε τα εξής δεδομένα:

- αντικείμενο(*Ρέας* τύπου Οδός)
- αντικείμενο(*Λεωνίδας* τύπου Κάτοικος)
  σχέση (κατοικείΣτηνΟδό *Ρέας*)

Το πρώτο και κύριο μειονέκτημα της αναπαράστασης του παραπάνω παραδείγματος είναι ότι αναφέρεται μόνο σε ένα σημείο στον χρόνο. Είναι αυτό που λέμε σύγχρονη (synchronic) αναπαράσταση. Ωστόσο σε πολλές περιοχές μελέτης, καθώς και στην ανθρώπινη λογική, σχέσεις τέτοιου τύπου είναι διαχρονικές (diachronic) δηλαδή αλλάζουν μέσα στον χρόνο. Για παράδειγμα η παρακάτω πρόταση:

Ο Λεωνίδας κατοικεί στην οδό Ρέας 7 από τον Ιούνιο του 1990

η οποία δείχνει καθαρά ότι ο Λεωνίδας πριν το 1990 κατοικούσε αλλού και ότι η σχέση «κατοικείΣτηνΟδό Ρέας» είναι αληθής για μια χρονική περίοδο με αρχή τον Ιούνιο του 1990.

Δυστυχώς η μεταφορά της αναπαράστασης μιας σχέσης από σύγχρονη σε διαχρονική δημιουργεί πολλά προβλήματα στη περίπτωση που χρησιμοποιούμε OWL. Με άλλα λόγια γλώσσες σαν την OWL δεν μπορούν να αναπαραστήσουν fluents.

Fluents ονομάζουμε τις σχέσεις που θεωρούνται αληθείς για ένα συγκεκριμένο χρονικό διάστημα και μόνο τότε.

Ο πιο διαδεδομένος τρόπος για να δημιουργηθεί ένα fluent είναι με την αλλαγή μια σχέσης από δυαδική σε τριαδική όπου το τρίτο όρισμα είναι η χρονική περίοδος για την οποία η σχέση είναι αληθής.

Έτσι στο παράδειγμα μας η σχέση κατοικείΣτηνΟδό (Λεωνίδας, Ρέας) γίνεται κατοικείΣτηνΟδό (Λεωνίδας, Ρέας, t1) όπου t1 είναι η χρονική περίοδος με αρχή την ημερομηνία 1/6/1990.

Ωστόσο ο τρόπος αυτός δεν είναι αποδεκτός για γλώσσες όπως η OWL που υποστηρίζουν μόνο δυαδικές σχέσεις.

1.2 Προτεινόμενη λύση και συνεισφορά της εργασίας

Στο επόμενο κεφάλαιο θα δούμε και θα αναλύσουμε τρόπους αναπαράστασης της χρονικά μεταβαλλόμενης πληροφορίας ανάμεσα στους οποίους είναι και ο τρόπος που διαλέξαμε εμείς ως λύση στο πρόβλημα που αναφέραμε παραπάνω.

Η λύση μας βασίζεται στην προσέγγιση των οντολογιών τεσσάρων διαστάσεων[2] κατά την οποία οι οντότητες υπάρχουν μέσα στον χρόνο όπως ακριβώς υπάρχουν τα αντικείμενα μέσα στον χώρο, καταλαμβάνοντας χωροχρόνο. Κατά την προσέγγιση αυτή κάθε οντότητα αποτελείται από αυθαίρετο αριθμό προσωρινών κομματιών όπου το κάθε κομμάτι αντιπροσωπεύει την οντότητα για συγκεκριμένη χρονική περίοδο. Έτσι αν δυο οντότητες σχηματίζουν μια fluent σχέση στην ουσία συμμετέχουν σε αυτή με τα προσωρινά τους κομμάτια και συγκεκριμένα με το κομμάτι εκείνο για το οποίο η χρονική περίοδος την οποία εκπροσωπεί καθιστά τη σχέση αληθή.

Για να μεταφερθεί η συγκεκριμένη λύση σε μια οντολογία θα πρέπει ο χρήστης να δημιουργήσει μια σειρά από κλάσεις και σχέσεις μέσω των οποίων θα χειρίζεται τα αντικείμενα του και συγκεκριμένα τα «κομμάτια» των αντικειμένων του για δεδομένες χρονικές περιόδους. Κάτι τέτοιο μπορεί στην αρχή να φανεί εύκολο αλλά όσο μια οντολογία μεγαλώνει η συντήρηση της καθίσταται δυσκολότερη. Η εργασία μας αναλαμβάνει τόσο τη δημιουργία των απαραίτητων κλάσεων και σχέσεων όσο και των αντικειμένων που θα συμμετέχουν στις fluent σχέσεις. Επίσης αναλαμβάνει τη σωστή εφαρμογή των κανόνων που θέτει η προσέγγιση μας και τους ελέγχους που πρέπει να γίνονται πάνω στα δεδομένα της οντολογίας ώστε να υπάρχει συνέπεια καθόλη την ανάπτυξη της οντολογίας. Παράλληλα η εργασία καλύπτει και το κενό που έχει το Protégé ως πρόγραμμα στο:

i. να παρουσιάσει μια fluent σχέση με τρόπο που να δείχνει στον χρήστη όλα τα «κομμάτια» των αντικειμένων που σχετίζονται στο ίδιο γραφικό περιβάλλον και
ii. να δημιουργεί, να μεταβάλει και να διαγράφει «κομμάτια» των αντικειμένων με τρόπο εύκολο μέσα από ένα γραφικό περιβάλλον χωρίς να χρειαστεί ο χρήστης να κινηθεί σε πολλά σημεία του προγράμματος.

1.3 Δομή της εργασίας

Το υπόλοιπο της εργασίας είναι διαχωρισμένο ως εξής:

Κεφάλαιο 2 - Σχετική εργασία: Σχετική δουλειά την οποία μελετήσαμε και βασιστήκαμε για την εργασία μας όπως το τι είναι η οντολογία, γλώσσες οντολογιών και εργαλεία χειρισμού οντολογιών.

Κεφάλαιο 3 - Χειρισμός της χρονικά εξελισσόμενης πληροφορίας στο Protégé: Αναφέρουμε τι προσφέρει, τι δεν προσφέρει και πως επεκτείναμε τη λειτουργία του Protégé εφαρμόζοντας τη λύση που επιλέξαμε. Επίσης γίνεται λεπτομερής αναφορά στο plug-in που δημιουργήσαμε.

Κεφάλαιο 4 - Συμπεράσματα και επεκτάσεις: Αναφέρονται τα αποτελέσματα και τα συμπεράσματα που εξήχθησαν από την εργασία. Ακόμη αναφέρονται σημεία στα οποία υπάρχει δυνατότητα για μελλοντική εργασία.

Κεφάλαιο 2

Σχετική εργασία

2.1 Οντολογίες

"Γνώση ονομάζουμε ένα σύνολο από δεδομένα με σημασιολογικό περιεχόμενο. Η οντολογία είναι ο τρόπος για να αναπαραστήσουμε τη γνώση αυτή."

Τι είναι λοιπόν μια οντολογία; Αν και ο όρος οντολογία[3] (ontology) έχει τις ρίζες του στη φιλοσοφία και έχει ερμηνευτεί διαφορετικά σε άλλους τομείς, στον τομέα της επιστήμης των υπολογιστών τον χρησιμοποιούμε για να αναφερθούμε σε ένα μοντέλο μέσω του οποίου περιγράφουμε τον κόσμο ή μια περιοχή αυτού (domain). Συγκεκριμένα, μέσω του μοντέλου ορίζουμε για μια περιοχή μελέτης ποιες είναι οι βασικές έννοιες που την διέπουν, τις ιδιότητες αυτών και φυσικά τις σχέσεις που σχηματίζουν μεταξύ τους. Έτσι αν κάποιος χρησιμοποιήσει το μοντέλο για να μελετήσει τη περιοχή θα μπορεί πολύ εύκολα να εξάγει συμπεράσματα για αυτή αλλά και να αναπαραστήσει όσο το δυνατόν καλύτερα τη γνώση που έχει για τα αντικείμενα της.

2.1.1 Βασικά χαρακτηριστικά μιας οντολογίας

Πριν δημιουργήσουμε μια οντολογία είναι αναγκαίο να καταγράψουμε, σχεδιάσουμε και ορίσουμε τα βασικά χαρακτηριστικά της περιοχής την οποία σκοπεύουμε να αναπαραστήσουμε.

Αν και η δημιουργία μιας οντολογίας προαπαιτεί κάποιος να είναι άριστος γνώστης της περιοχής μελέτης που θα καλύψει μια οντολογία εμείς θα προσπαθήσουμε με ένα απλό παράδειγμα να παρουσιάσουμε τα βασικά χαρακτηριστικά της.

Έστω λοιπόν ότι η οντολογία μας θα παρέχει πληροφορίες γύρω από την περιοχή των τεχνών. Τέσσερα είναι τα βήματα που πρέπει να ακολουθήσει κάποιος:

- *Βήμα 1:* Αναγνωρίζουμε και ορίζουμε τις βασικές έννοιες της περιοχής και για κάθε μια από αυτές δημιουργούμε ένα σετ, συνήθως το ονομάζουμε κλάση (class), το οποίο στη συνέχεια θα περικλείει όλα τα αντικείμενα που ανήκουν στην έννοια. Έτσι για το παράδειγμα μας βασικές έννοιες όπως αυτή του καλλιτέχνη και του έργου τέχνης θα μπορούσαν να αντιπροσωπευθούν από την κλάσεις Artist και PieceOfArt αντίστοιχα.

- *Βήμα 2:* Αναγνωρίζουμε και ορίζουμε δομές μέσω των οποίων παρουσιάζουμε την ιεραρχία και τις συσχετίσεις που υπάρχουν ανάμεσα στις έννοιες που ορίσαμε. Έτσι για το παράδειγμα μας μπορούμε να θεωρήσουμε ότι για την έννοια του καλλιτέχνη η έννοια του ζωγράφου αποτελεί ένα υποσύνολο και άρα η σχετική κλάση του θα οριστεί ως υποκλάση της κλάσης Artist: κλάση Painter υποκλάση της Artist. Αντίστοιχα η έννοια του πίνακα αποτελεί ένα υποσύνολο των έργων τέχνης και άρα η κλάση Painting αποτελεί ένα υποσύνολο της κλάσης PieceOfArt.

- *Βήμα 3:* Αναγνωρίζουμε και ορίζουμε τις ιδιότητες των εννοιών. Οι ιδιότητες χωρίζονται στα χαρακτηριστικά μέσω των οποίων περιγράφουμε μια έννοια και στις αντιστοιχίσεις μέσω των οποίων παρουσιάζουμε τις σχέσεις ανάμεσα στις έννοιες της οντολογίας μας. Έτσι για το παράδειγμα μας και για την κλάση Painter μερικά από τα χαρακτηριστικά που θα τον περιέγραφαν θα ήταν το όνομα του, η ηλικία του, η περιοχή καταγωγής του: pName, pAge, pCityOfBirth. Παράλληλα μια από τις αντιστοιχίσεις θα μπορούσε να είναι η σχέση που υπάρχει ανάμεσα στον ζωγράφο και κάποιον πίνακα που δημιούργησε: η hasCreated συνδέει τις έννοιες Painter και Painting.

- *Βήμα 4:* Αναγνωρίζουμε και ορίζουμε αξιώματα/κανόνες μέσω των οποίων εξάγουμε συμπεράσματα για την περιοχή που μελετάμε ενώ παράλληλα μπορεί να δημιουργούμε και καινούργιες έννοιες ή σχέσεις. Για παράδειγμα αν ένα αντικείμενο της κλάσης Artist συσχετίζεται με αντικείμενα της κλάσης Painting μέσω της σχέσης hasCreated τότε μέσω ενός κανόνα μπορούμε να συμπεράνουμε ότι το εν λόγο αντικείμενο είναι ανήκει και στην κλάση Painter.

2.1.2 Λόγοι δημιουργίας μια οντολογίας

Γιατί όμως δημιουργούμε οντολογίες; Αν και κάθε τομέας που χρησιμοποιεί την έννοια της οντολογίας μπορεί να έχει και τους δικούς του λόγους οι βασικοί είναι οι παρακάτω:

1. Με μια οντολογία οργανώνουμε και θέτουμε τους ορισμούς που περιγράφουν τη περιοχή που μελετάμε και δημιουργούμε έτσι ένα λεξιλόγιο το οποίο με τη σειρά του μας βοηθάει στο (α) να μοιραζόμαστε όλοι την ίδια γνώση και (β) να υπάρχει μια κοινή κατανόηση.
   Για παράδειγμα, συνεχίζοντας με την οντολογία της προηγούμενης ενότητας, το αντικείμενο Vincent van Gogh θα αποτελεί για όλους κοινή γνώση ότι πρόκειται για ζωγράφο από την Δανία που έζησε στο δεύτερο μισό του 19^{ου} αιώνα.
2. Μια οντολογία είναι μια συλλογή πληροφοριών αλλά παράλληλα είναι και μια συλλογή μεταδεδομένων που αναπαριστούν σαφώς τη σημασιολογία των πληροφοριών με τρόπο κατανοητό από τους υπολογιστές. Άρα με μια οντολογία παρέχουμε τη πληροφορία τόσο σε ανθρώπους όσο και σε μηχανές.
3. Έχοντας μια έτοιμη οντολογία σημαίνει ότι έχουμε έτοιμη τη γνώση γύρω από μια περιοχή και άρα μπορούμε να την επαναχρησιμοποιήσουμε σε μελλοντικές μας εργασίες. Για παράδειγμα αν μια ομάδα δημιουργήσει μια οντολογία που πραγματεύεται τις τέχνες και συγκεκριμένα του πίνακες ζωγράφων τότε θα μπορούμε να τη "εισάγουμε" στη δικιά μας και με αυτό τον τρόπο να (α) γλυτώσουμε χρόνο/δουλειά και (β) διατηρήσουμε το γεγονός ότι υπάρχει μια κοινή σταθερά σε ένα αντικείμενο, πχ: ότι ο πίνακας «Ο γρύλος κάτω από το φεγγάρι» αποτελεί έργο του ισπανού ζωγράφου Miro.
4. Δημιουργώντας μια οντολογία δημιουργούμε σαφείς ορισμούς γύρω από έννοιες, ιδιότητες και σχέσεις με αποτέλεσμα αν αργότερα θέλουμε να προσθέσουμε ή να αλλάξουμε κάτι στην υπάρχουσα πληροφορία να κατανοούμε εύκολα και γρήγορα που πρέπει να γίνει η αλλαγή/προσθήκη. Παράλληλα η καινούργια πληροφορία εφαρμόζεται σε ένα μέρος μόνο αλλά γνωστοποιείται σε όλους όσους χρησιμοποιούν τη συγκεκριμένη οντολογία άρα υπάρχει και μια συνέπεια όσον αφορά τη γνώση πάνω σε μια περιοχή.
5. Χρησιμοποιώντας μια οντολογία ως μια κοινή βάση πληροφοριών για μια περιοχή η δημιουργία τρίτων εφαρμογών γίνεται ακόμα πιο εύκολη εφόσον πλέον γίνεται χρήση της πληροφορίας ανεξάρτητα από το που προήλθε αυτή.
6. Τέλος με τη δημιουργία οντολογιών λύνουμε πολλά από τα σύγχρονα προβλήματα που αντιμετωπίζουμε πλέον όπως (α) την ραγδαία αύξηση της ψηφιακής πληροφορίας που κάνει δυσκολότερη τη πρόσβαση, και την εύρεση της και (β) το γεγονός ότι με το διαδίκτυο έρχονται σε επαφή πολλοί και διαφορετικοί άνθρωποι που ίσως να μη μιλάνε και την ίδια γλώσσα, με μια οντολογία και με κάποιες τεχνικές μετάφρασης η ίδια πληροφορία μπορεί να χρησιμοποιηθεί από όλους ανεξάρτητα από το που προέρχονται.

2.1.3 Διαφορές ανάμεσα σε οντολογίες και βάσεις δεδομένων

Ο τρόπος σκέψης για τη δημιουργία μιας οντολογίας είναι παραπλήσιος με αυτόν για μια βάση δεδομένων ωστόσο μια βάση δεδομένων είναι ένα σύνολο από πίνακες οι οποίοι κρατούν πληροφορία είτε για κάποιες οντότητες είτε για τις μεταξύ τους σχέσεις. Αντίθετα μια οντολογία περιέχει συντακτικά και σημασιολογικά πλουσιότερη πληροφορία η οποία περιγράφεται με φυσική γλώσσα σε ημιδομημένο κείμενο για να τη κατανοεί και ο υπολογιστής. Επίσης μια οντολογία πρέπει να είναι διαδικτυακής αρχιτεκτονικής γιατί χρησιμοποιείται για το διαμοιρασμό της πληροφορίας.

2.2 Γλώσσες αναπαράστασης οντολογιών

Υπάρχουν πολλές γλώσσες για να αναπαραστήσουμε μια οντολογία τις οποίες μπορούμε να κατατάξουμε σε τρεις κατηγορίες. Η πρώτη κατηγορία περιλαμβάνει τις λεγόμενες παραδοσιακές γλώσσες που βασίζονται σε έννοιες όπως η λογική πρώτης τάξης, η λογική πλαισίων ή τέλος η περιγραφική λογική. Χαρακτηριστικά παραδείγματα της κατηγορίας αυτής είναι γλώσσες όπως η Carin, η Flogic, η Loom. Μια δεύτερη κατηγορία, στην οποία και θα σταθούμε αργότερα, είναι οι λεγόμενες web-based γλώσσες η σύνταξη των οποίων βασίζεται στην XML και έχουν ως σκοπό την αναπαράσταση οντολογιών σε μέσα που χρησιμοποιούνται στο διαδίκτυο. Η τρίτη και τελευταία κατηγορία περιλαμβάνει γλώσσες που αναπτύχθηκαν για να αναπαραστήσουν συγκεκριμένες οντολογίες και για να χρησιμοποιηθούν σε συγκεκριμένες εφαρμογές. Παραδείγματα των γλωσσών αυτών είναι η CycL, η GRAIL και η NKRL.

Όλες οι παραπάνω γλώσσες μπορούν να χρησιμοποιηθούν για να αναπαραστήσουν μια οντολογία αλλά μεταξύ τους υπάρχουν πολλές και βασικές διαφορές σε τομείς όπως:

- η σύνταξη
- η ορολογία που χρησιμοποιούν (πχ: το γνώρισμα σε μια γλώσσα μπορεί να το ορίζεται ως attribute ενώ σε μια άλλη ως slot)
- στην εκφραστικότητα (η κάθε γλώσσα μπορεί να αναπαραστήσει μέχρι έναν συγκεκριμένο βαθμό την οντολογία που μας ενδιαφέρει ενώ υπάρχουν και περιπτώσεις που η αναπαράσταση μια έννοιας δεν είναι δυνατή)
- στη σημασιολογία (η ίδια δήλωση μπορεί να έχει διαφορετική ερμηνεία ανάμεσα στις γλώσσες που αναφέραμε)

2.2.1 OWL

Ο κύριος όγκος των web-based γλωσσών βασίζεται στην XML η οποία από μόνη της θεωρείται μοντέλο δεδομένων χαμηλού επιπέδου και δεν μπορεί να χρησιμοποιηθεί ούτε για τη δημιουργία οντολογιών εξειδικευμένου πεδίου, ούτε για τη δημιουργία οντολογικών λεξιλογίων αλλά ούτε και σε βασικές οντολογικές αρχές μοντελοποίησης. Επίσης δεν είναι κατάλληλη για διαμοιραζόμενες πηγές στον παγκόσμιο ιστό ενώ δεν διαθέτει μηχανή συμπερασματολογίας. Για τον λόγο αυτό έγιναν πολλές προσπάθειες να αναπτυχθούν καινούργιες γλώσσες οι οποίες θα βασίζονταν πάνω στην XML αλλά θα παρείχαν επιπλέον μέσα για τη κατασκευή μιας οντολογίας.

Μερικές από αυτές είναι η SHOE(XML)[4], η XOL[5], η OML[6] και η RDF-RDFs[7] η οποία μάλιστα συστήνεται και από την W3C για τη δημιουργία οντολογιών γιατί εκτός του ότι παρέχει ένα τυποποιημένο μοντέλο δεδομένων και ένα τυποποιημένο XML συντακτικό, είναι ικανή να δημιουργήσει εκτεταμένα λεξιλόγια ενώ παράλληλα υπάρχουν πολλά APIs και parsers για να χρησιμοποιήσει όποιος ενδιαφέρεται. Ωστόσο η χρήση της για τη δημιουργία μιας οντολογίας δεν είναι και η καλύτερη δυνατή επιλογή γιατί (α) είναι πολύ αδύναμη στη σημασιολογική αναπαράσταση, (β) δεν μπορεί να περιγράψει καλά το νόημα της πληροφορίας και (γ) δε διαθέτει μηχανή συμπερασματολογίας.

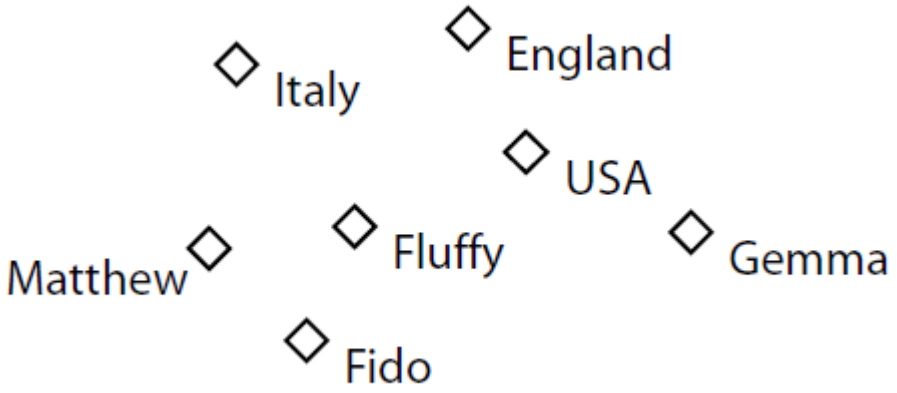
Τους παραπάνω περιορισμούς έρχεται να καλύψει η **OWL**, ή αλλιώς **Web Ontology Language**, η οποία σχεδιάστηκε έχοντας ως βάση της την RDFs και με στόχο την δημιουργία μιας γλώσσας η οποία θα χρησιμοποιείται καθαρά και μόνο για την επεξεργασία της πληροφορίας ενός εγγράφου από εφαρμογές και όχι τόσο για την αναπαράσταση της σε ανθρώπους. Με την OWL αυτό που επιτυγχάνουμε είναι η δημιουργία λεξιλογίων που περιέχουν την ρητή έννοια των όρων που θέλουμε καθώς επίσης και τις σχέσεις μεταξύ των όρων αυτών. Με άλλα λόγια επιτυγχάνουμε τη πλήρη περιγραφή μιας οντολογίας.

Όπως είδαμε παραπάνω η κάθε γλώσσα έχει τη δικιά της ορολογία και σημασιολογία. Για την OWL οι βασικές έννοιες που αντιπροσωπεύει είναι οι εξής:

- **Αντικείμενο / Στιγμιότυπο (individual ή instance)**
  Ορισμός:
  Αντικείμενο ονομάζουμε αυτό που έχουμε ορίσει / χαρακτηρίσει / περιγράψει πλήρως.

  Σε κώδικα OWL:
  <ClassName rdf:ID="IndividualName" />

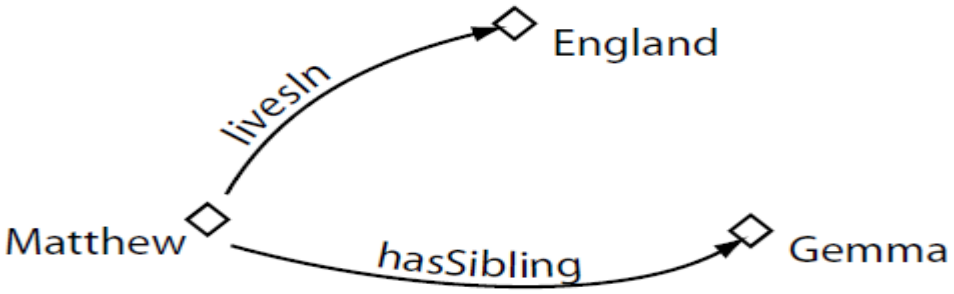
  Παράδειγμα:
  Μια χώρα όπως η Ιταλία (Italy) ή ένα συγκεκριμένο άτομο όπως ο Matthew είναι αντικείμενα. Γραφικά θα μπορούσαμε να θεωρήσουμε πως παρόμοια αντικείμενα αναπαριστούνται όπως φαίνεται στο σχήμα:



- **Γνώρισμα (attribute) ή ιδιότητα**
  Ορισμός:
  Γνώρισμα ονομάζουμε τη δυαδική σχέση μέσω της οποίας δυο αντικείμενα ενώνονται μεταξύ τους. Στην OWL έχουμε δυο είδη γνωρισμάτων. Έχουμε τα Object properties τα οποία περιγράφουν τη σχέση ανάμεσα σε δυο αντικείμενα και έχουμε και τα Datatype properties τα οποία περιγράφουν τη σχέση ανάμεσα σε ένα αντικείμενο και σε μια τιμή.

  Σε κώδικα OWL:
  <owl:ObjectProperty rdf:ID="PropertyName">
  <rdfs:domain rdf:resource="#ClassName1"/>
  <rdfs:range rdf:resource="#ClassName2"/>
  </owl:ObjectProperty> <owl:DatatypeProperty rdf:ID="PropertyName">
  <rdfs:domain rdf:resource="#Classname" />
  <rdfs:range rdf:resource="&xsd;DataType"/>
  </owl:DatatypeProperty>

  Παράδειγμα (συνεχίζοντας το προηγούμενο παράδειγμα προσθέτουμε για τα δύο είδη σχέσεων και από ένα γνώρισμα):
  Για το είδος των Object properties έχουμε το γνώρισμα hasSibling. Μέσω αυτού ενώνουμε τα αντικείμενα Matthew και Gemma οδηγώντας κάποιον στο λογικό συμπέρασμα πως οι Matthew και Gemma είναι συγγενείς. Γραφικά θα μπορούσε να αναπαρασταθεί όπως φαίνεται παρακάτω:

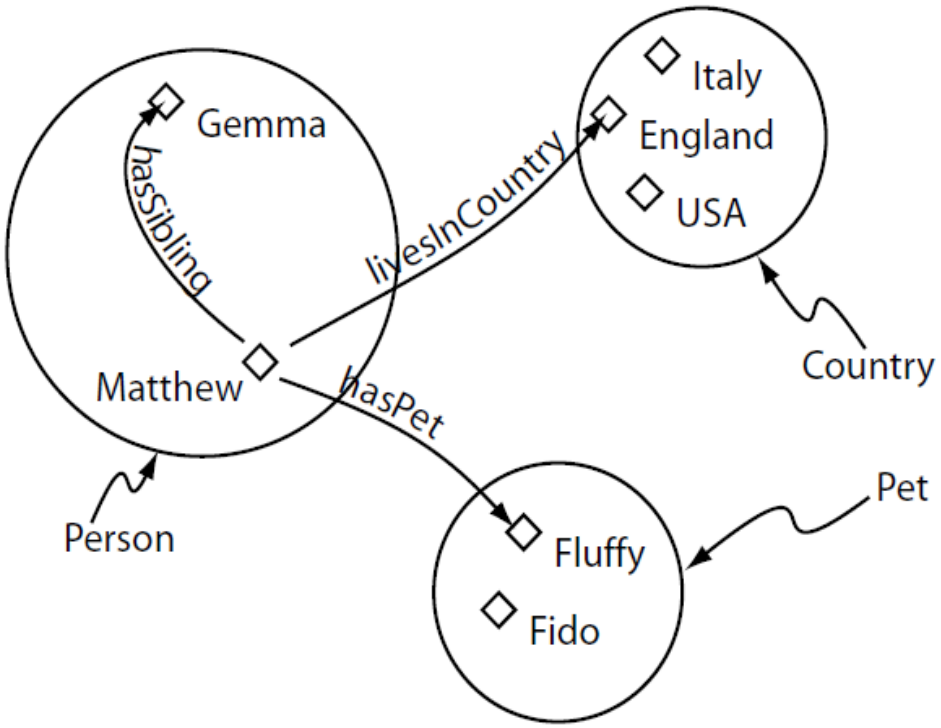


Για το είδος των Datatype properties προσθέτουμε το γνώρισμα colorOfEyesIs. Μέσω του γνωρίσματος αυτού μπορούμε να συνδέσουμε το αντικείμενο Matthew με την τιμή Brown (datatype=string) εξάγοντας έτσι το σαφές συμπέρασμα πως τα μάτια του Matthew είναι καφέ.

- **Κλάση (class)**
  Ορισμός:
  Κλάση ονομάζουμε ένα σύνολο (σετ) με αντικείμενα. Για να ανήκει ένα αντικείμενο σε κάποια κλάση θα πρέπει να πληρεί συγκεκριμένες προϋποθέσεις τις οποίες και θέτουμε κατά τη δημιουργία της κλάσης. Οι κλάσεις μπορούν να οργανωθούν σε υπερκλάσεις και υποκλάσεις με τις δεύτερες να προσθέτουν περισσότερες λεπτομέρειες στις πρώτες και έτσι να αποτελούν σύνολα πιο συγκεκριμένων αντικειμένων.

  Σε κώδικα OWL:
  ```
  <owl:Class rdf:ID="ClassName">
    [<rdfs:subClassOf rdf:resource="#SuperClassName" />]
    [...]
  </owl:Class>
  ```

  Παράδειγμα:
  Παρακάτω μπορούμε να δούμε και γραφικά ένα παράδειγμα κλάσεων στο οποίο οι τρεις κλάσεις Person, Country και Pet περικλύουν συγκεκριμένα αντικείμενα. Για παράδειγμα, τα αντικείμενα Matthew και Gemma που είναι άνθρωποι πληρούν τις προυποθέσεις που έχει θέσει η κλάση Person:



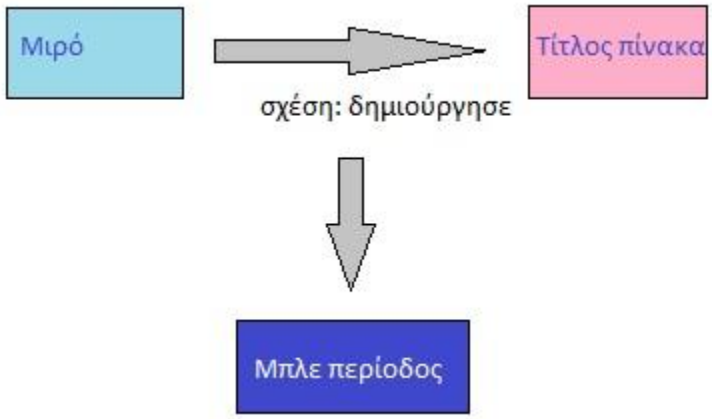
2.3 Παράσταση Ν-υαδικών σχέσεων

Όπως γνωρίζουμε σε γλώσσες όπως η RDF και η OWL ένα γνώρισμα είναι μια δυαδική σχέση που ενώνει δυο αντικείμενα ή ένα αντικείμενο με μια τιμή. Ωστόσο σε πολλές των περιπτώσεων γεννιέται η ανάγκη να συνδέσουμε ένα αντικείμενο με παραπάνω από ένα αντικείμενα ή τιμές. Αυτού του είδους οι σχέσεις ονομάζονται Ν-υαδικές σχέσεις[8] και τις συναντάμε κυρίως όταν θέλουμε να αναπαραστήσουμε μερικά από τα γνωρίσματα μιας σχέσης (πχ: τη βεβαιότητα μας για αυτή) ή όταν θέλουμε να αναπαραστήσουμε τις σχέσεις ανάμεσα σε πολλά αντικείμενα όπως πχ ενός πωλητή, ενός αγοραστή και ενός αντικειμένου προς πώληση.

Ο τρόπος για μεταφέρουμε μια ν-υαδική σχέση σε γλώσσες όπως η OWL βασίζεται στην ιδέα της αναπαράστασης της σχέσης μέσα από μια καινούργια κλάση η οποία με την σειρά της έχει της δικές της δυαδικές σχέσεις ενώνοντας έτσι το αρχικό αντικείμενο με τα υπόλοιπα ν-αντικείμενα.

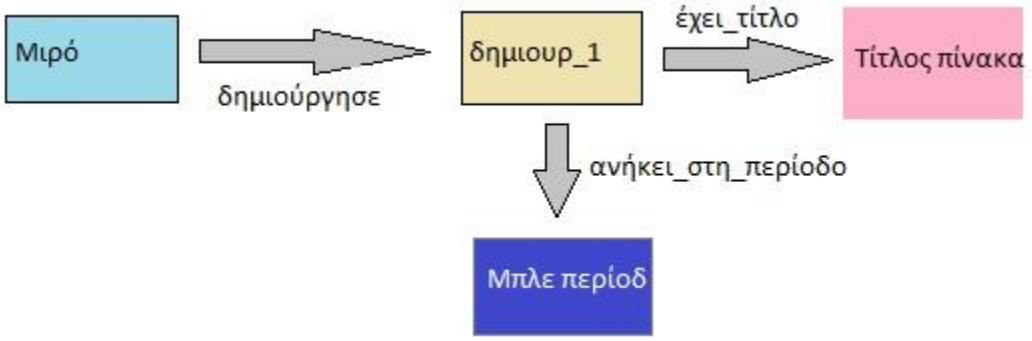
Για να το κατανοήσουμε καλύτερα ας δούμε το παρακάτω παράδειγμα. Η πρόταση:

ο ζωγράφος Miro δημιούργησε τον πίνακα «Τίτλος πίνακα» ενώ βρισκόταν στην γνωστή του «μπλε περίοδο».



όπως μπορούμε να δούμε και από το σχήμα δεν αποτελεί μια απλή δυαδική σχέση εφόσον το γνώρισμα «δημιούργησε» περιγράφεται με την σειρά του από ένα ακόμα γνώρισμα που το κατηγοριοποιεί σε συγκεκριμένη περίοδο.

Για τον λόγο αυτό δημιουργούμε την κλάση «Δημιούργημα» της οποίας το αντικείμενο «δημιουργ_1» θα αποτελέσει τον ενδιάμεσο κρίκο ανάμεσα στα αντικείμενα «Μιρό», «Τίτλος πίνακα» και «Μπλέ περίοδος» με τις σχέσεις «δημιούργησε», «έχει_τίτλο» και «ανήκει_στην_περίοδο» αντίστοιχα:



2.4 Παράσταση χρονικής πληροφορίας

Ένα από τα βασικά προβλήματα που αντιμετωπίζουμε κατά την αναπαράσταση οντολογιών είναι η διαχείριση πληροφορίας που μεταβάλλεται μέσα στον χρόνο. Μάλιστα το πρόβλημα αυτό επιδεινώνεται ακόμα περισσότερο όταν για την αναπαράσταση κάνουμε χρήση γλωσσών που υποστηρίζουν μόνο δυαδικές σχέσεις, όπως είναι η OWL, η RDF και όσες άλλες ανήκουν στην κατηγορία της "περιγραφικής λογικής". Ακόμα και όταν η ίδια η σχέση είναι δυαδική, όπως το να κατοικεί κάποιος σε μια συγκεκριμένη διεύθυνση ή να είναι μέλος μιας οργάνωσης, το γεγονός ότι η πληροφορία μπορεί να αλλάξει με τον χρόνο περιπλέκει τα πράγματα σε τέτοιο σημείο που οι άνωθεν γλώσσες επιλέγουν απλά να αγνοήσουν το πρόβλημα τελείως.

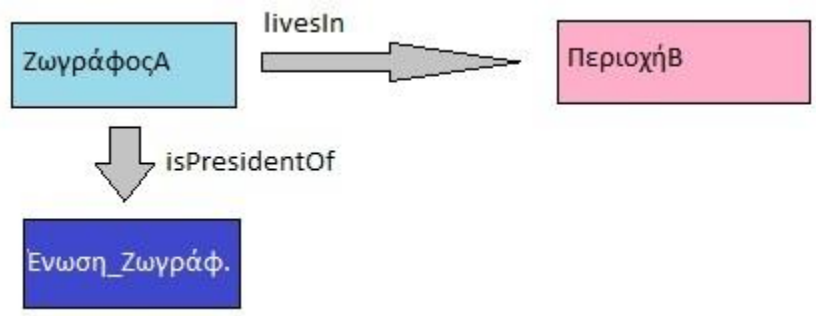
Στο πεδίο των οντολογιών υπάρχουν δυο βασικά γκρουπ που αντιμετωπίζουν το πρόβλημα του χρόνου απαντώντας διαφορετικά στην ερώτηση «πως αντιμετωπίζουμε το γεγονός ότι η ίδια οντότητα μπορεί να εμφανιστεί διαφορετική σε διαφορετικές χρονικές στιγμές» .

Το πρώτο γκρουπ υιοθετεί μια τρισδιάστατη όψη κατά την οποία έχουμε τις οντότητες που έχουν φυσική υπόσταση όπως ένας άνθρωπος ή μια καρέκλα και τα συμβάντα όπως το να κάτσει κάποιος σε μια καρέκλα ή το πάει ένας άνθρωπος στη δουλειά του. Οι μεν οντότητες έχουν μια συνεχή παρουσία καθόλη τη διάρκεια του χρόνου, ενώ τα συμβάντα έχουν προσωρινές υποστάσεις που παραμένουν αληθείς για συγκεκριμένο χρονικό διάστημα.

Το δεύτερο γκρουπ υιοθετεί μια όψη τεσσάρων διαστάσεων κατά την οποία οποία οι οντότητες υπάρχουν μέσα στον χρόνο όπως ακριβώς υπάρχουν τα αντικείμενα μέσα στον χώρο, καταλαμβάνοντας χωροχρόνο. Κατά την προσέγγιση αυτή κάθε οντότητα αποτελείται από αυθαίρετο αριθμό προσωρινών κομματιών όπου το κάθε κομμάτι αντιπροσωπεύει την οντότητα για συγκεκριμένη χρονική περίοδο. Έτσι αν δυο οντότητες σχηματίζουν μια fluent σχέση στην ουσία συμμετέχουν σε αυτή με τα προσωρινά τους κομμάτια και συγκεκριμένα με το κομμάτι εκείνο για το οποίο η χρονική περίοδος την οποία εκπροσωπεί καθιστά τη σχέση αληθή. Έστω ότι έχουμε την πρόταση:

«ο **ΖωγράφοςΑ** κάτοικος της **ΠεριοχήςΒ** αποτέλεσε τον πρόεδρο της **ένωσης ζωγράφων**.»

Η πρόταση αυτή σχηματικά μπορεί να παρουσιαστεί όπως παρακάτω



ενώ από μόνη της παράγει τον εξής ψευδοκώδικα σε OWL:

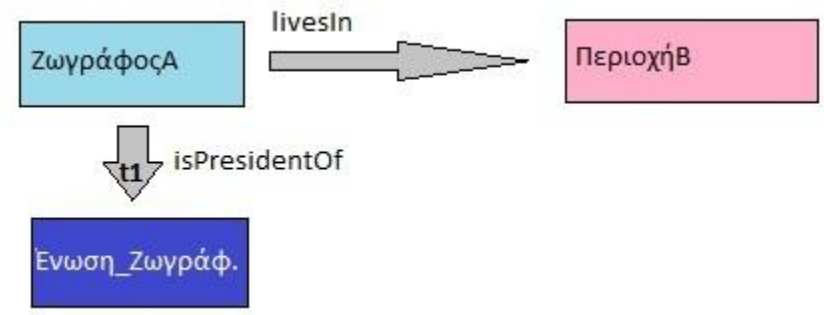
- Individual(**Ένωση_Ζωγράφων** *type*(Organization))
- Individual(**ΠεριοχήΒ** *type*(City))
- Individual(**ΖωγράφοςΑ** type(Painter))
  value(livesIn **ΠεριοχήΒ**)
  value(isPresidentOf **Ένωση_Ζωγράφων**)

Το πρώτο και κύριο μειονέκτημα της αναπαράστασης του παραπάνω παραδείγματος είναι ότι αναφέρεται μόνο σε ένα σημείο στον χρόνο. Είναι αυτό που λέμε σύγχρονη (synchronic) αναπαράσταση. Ωστόσο σε πολλές περιοχές μελέτης, καθώς και στην ανθρώπινη λογική, σχέσεις τέτοιου τύπου είναι διαχρονικές (diachronic) δηλαδή αλλάζουν μέσα στον χρόνο. Για παράδειγμα η πρόταση:

«ο **ΖωγράφοςΑ** κάτοικος της **ΠεριοχήςΒ** αποτέλεσε τον πρόεδρο της **ένωσης ζωγράφων** από τις 01/01/1970.»

δείχνει καθαρά ότι ο συγκεκριμένος καλλιτέχνης δεν ήταν πάντα ο πρόεδρος της ένωσης ζωγράφων και ότι η αρχική πρόταση δεν ισχύει για χρονικές στιγμές πριν από τις 01/01/1970.

Δυστυχώς η μεταφορά της αναπαράστασης μιας σχέσης από σύγχρονη σε διαχρονική δημιουργεί πολλά προβλήματα στη περίπτωση που χρησιμοποιούμε γλώσσες σαν την OWL των οποίων τα γνωρίσματα δεν υποστηρίζουν σχέσεις με πάνω από δυο αντικείμενα και έτσι μια λύση σαν την παρακάτω δεν είναι άμεσα υλοποιήσιμη:



- Individual(**ΖωγράφοςΑ** type(Painter))
  isPresidentOf(**Ένωση_Ζωγράφων, t1**)
  όπου t1 = χρονική περίοδος με αρχή τις 01/01/1970

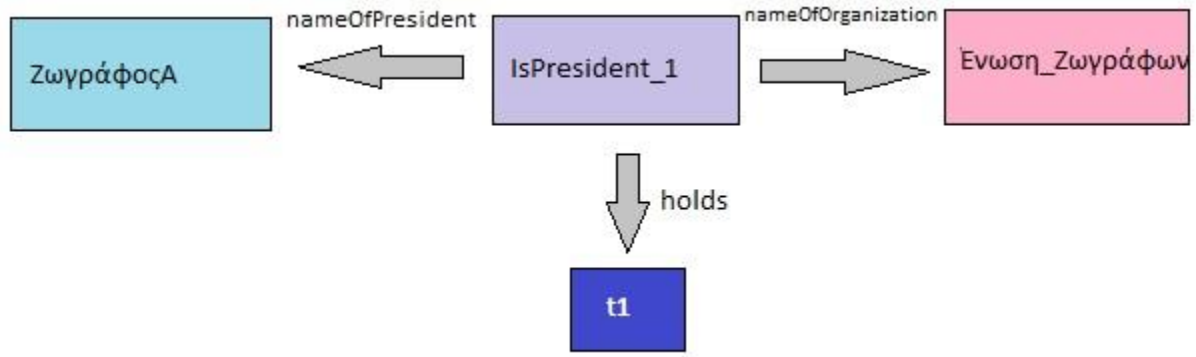
2.4.1 Λύσεις του προβλήματος

Το πρόβλημα αυτό είναι γνωστό εδώ και πολλά χρονιά και κατά καιρούς έχουν γίνει διάφορες προσεγγίσεις για τη λύση του. Δυο από αυτές αναλύονται παρακάτω:

i) Reification[2]

Στη συγκεκριμένη προσέγγιση όταν έχουμε μια σχέση όπου εμπλέκονται πάνω από δυο οντότητες αντί να ορίσουμε ένα καινούργιο γνώρισμα για να την αναπαραστήσουμε, ορίζουμε μια καινούργια κλάση η οποία και παίζει τον ρόλο της σχέσης δίνοντας της έτσι την ικανότητα να έχει και αυτή με τη σειράς της τα δικά της δυαδικά γνωρίσματα. Με τον τρόπο αυτό και μέσω των καινούργιων γνωρισμάτων όλα τα αντικείμενα της αρχικής σχέσης συνδέονται μεταξύ τους. Ας δούμε τα παραπάνω στο παράδειγμα μας:

Εδώ η σχέση isPresidentOf χρειάζεται τρία ορίσματα. Το πρώτο είναι ένα αντικείμενο της κλάσης Painter, το δεύτερο ένα αντικείμενο της κλάσης Organization και το τρίτο είναι η χρονική περίοδος για την οποία η σχέση είναι αληθής (αντικείμενο κάποιας τρίτης κλάσης). Για τον λόγο αυτό αντικαθιστούμε στην οντολογία μας την σχέση αυτή με την κλάση IsPresident ενώ παράλληλα ορίζουμε για αυτή και τρία γνωρίσματα: (α) το nameOfPresident που δέχεται αντικείμενα της Painter, (β) το nameOfOrganization που δέχεται αντικείμενα της Organization και (γ) το holds που δέχεται αντικείμενα χρονικής περιόδου. Με βάση αυτά σχηματικά έχουμε κάτι σαν το παρακάτω:



ενώ σε ψευδοκώδικα OWL:

- Individual(**Ένωση_Ζωγράφων** *type*(Organization))
- Individual(**ΠεριοχήΒ** *type*(City))
- Individual(**ΖωγράφοςΑ** type(Painter))
- Individual(**IsPresident_1** type(IsPresident))
  value(nameOfOrganization **Ένωση_Ζωγράφων**)

  value(nameOfPresident **ΖωγράφοςΑ**)
  value(holds **t1**)

  όπου t1 = χρονική περίοδος με αρχή τις 01/01/1970

Η λύση αυτή, αν και αποτελεί μια καθαρή και ευκολονόητη προσέγγιση, δημιουργεί άλλα προβλήματα.

- Πολλαπλασιασμός αντικειμένων: ο σύνηθες τρόπος για να αναπαραστήσουμε μια σχέση ανάμεσα σε δυο αντικείμενα απαιτεί τη δημιουργία ενός αντικειμένου για κάθε τύπο και τον σχηματισμό της δυαδικής σχέσης. Με την λύση του reification εκτός των δύο αντικείμενων πρέπει να δημιουργήσουμε και ένα αντικείμενο που θα υποδεικνύει τη χρονική περίοδο αλλά και ένα αντικείμενο που θα αντιπροσωπεύει την ίδια την σχέση. Επίσης ο σχηματισμός των δυαδικών σχέσεων αυξάνεται δύο εφόσον κάθε αντικείμενο τύπου "σχέσης" έχει και τρεις δικούς του δυαδικούς ρόλους. Οι παραπάνω αριθμοί μπορεί να φαίνονται μικροί όταν έχουμε μικρές οντολογίες αλλά στη περίπτωση συστημάτων που δημιουργούν οντολογίες με εκατομμύρια κλάσεις και σχέσεις ο χειρισμός και η κλιμάκωση της οντολογίας καθίσταται πολύ δύσκολη.
- Δημιουργία περιττών αντικειμένων: από τη στιγμή που η κάθε σχέση αντιπροσωπεύεται από ένα αντικείμενο υπάρχει πιθανότητα να δημιουργηθεί πάνω από ένα αντικείμενα που στην ουσία περιγράφουν την ίδια σχέση κάτι το οποίο θα προκαλέσει πρόβλημα στη χρήση των reasoners που κάνουν την παραδοχή "διαφορετική ονομασία = διαφορετικό αντικείμενο".
- Οντολογία η οποία μπερδεύει κατά τη χρήση της: όταν δημιουργούμε μια κλάση ή ένα αντικείμενο είναι σημαντικό να ορίζουμε και τι αντιπροσωπεύουν. Η παρουσία των σχέσεων-αντικειμένων και των ρόλων τους μπορεί να δημιουργήσει σύγχυση κατά τον χειρισμό συμβάντων και fluents.
- Περιορισμένη χρήση των OWL reasoners: εμπειρικές έρευνες έχουν δείξει ότι ακόμα και ένα μια απλή συνάρτηση reasoning όπως είναι το inverse, για τις σχέσεις, μπορεί

να επιδράσει σημαντικά στον χρήστη όσον αφορά τις συσχετίσεις που κάνει στο μυαλό του. Για παράδειγμα η inverse σχέση της ownerOf είναι η hasOwner και ανάμεσα στις δύο δε μπορούμε να είμαστε σίγουροι ποια θα είναι πιο χρήσιμη στον χρήστη για να εξάγει συμπεράσματα. Έχοντας δημιουργήσει μια σχέση με τον τρόπο του reification έχουμε αποκλείσει τη δυνατότητα δημιουργίας inverse σχέσεων και άρα έχουμε περιορίσει την ευελιξία του χρήστη. Εκτός του inverse η μέθοδος του reification αποκλείει και τη χρήση συναρτήσεων όπως transitive, symmetric, functional και inversefunctional.

ii) The Four-Dimensional approach[2]

Η βασική ιδέα πίσω από την προσέγγιση των οντολογιών τεσσάρων διαστάσεων (4D ontologies) παρουσιάζει τις οντότητες να υπάρχουν μέσω στον χρόνο όπως ακριβώς υπάρχουν τα αντικείμενα μέσα στον χώρο, καταλαμβάνοντας χωροχρόνο.

Κατά την προσέγγιση αυτή κάθε οντότητα αποτελείται από αυθαίρετο αριθμό προσωρινών κομματιών όπου το κάθε κομμάτι αντιπροσωπεύει την οντότητα για συγκεκριμένη χρονική περίοδο. Έτσι αν δυο οντότητες σχηματίζουν μια fluent σχέση στην ουσία συμμετέχουν σε αυτή με τα προσωρινά τους κομμάτια και συγκεκριμένα με το κομμάτι εκείνο για το οποίο η χρονική περίοδος την οποία εκπροσωπεί καθιστά τη σχέση αληθή.

Η παραπάνω ιδέα μεταφέρεται στην κάθε οντολογία με:

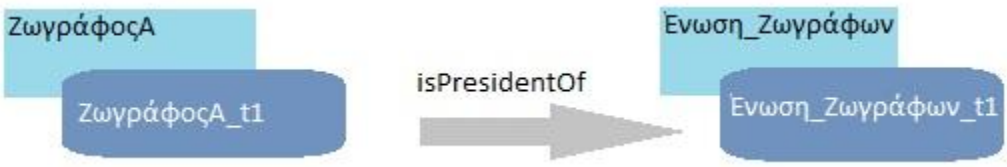
- μια επιπλέον κλάση, την οποία συνήθως ονομάζουμε TimeSlice και της οποίας τα αντικείμενα αποτελούν τα προσωρινά κομμάτια των οντοτήτων
- μια επιπλέον κλάση, την οποία συνήθως ονομάζουμε TimeInterval και της οποίας τα αντικείμενα αποτελούν το χρονικό διάστημα για το οποίο κάποιο προσωρινό κομμάτι ισχύει έναντι των άλλων
- μια επιπλέον σχέση, την οποία συνήθως ονομάζουμε tsTimeSliceOf και η οποία έχει ως domain την TimeSlice και ως range μια οποιαδήποτε άλλη κλάση εκτός της TimeInterval. Με τη σχέση αυτή συνδέουμε αντικείμενα της TimeSlice με αντίκειμενα άλλων κλάσεων ορίζοντας έτσι προσωρινά κομμάτια για τα αντικείμενα αυτά.
- μια επιπλέον σχέση, την οποία συνήθως ονομάζουμε tsTimeInterval και η οποία έχει ως domain την TimeSlice και ως range την TimeInterval. Με τη σχέση αυτή συνδέουμε αντικείμενα της TimeSlice με αντικείμενα της TimeInterval θέτωντας έτσι τη χρονική περίοδο την οποία και αντιπροσωπεύει το συγκεκριμένο time-slice.
- μια επιπλέον σχέση για κάθε άλλη σχέση που δημιουργούμε με την οποία συνδέουμε time-slices μεταξύ τους.

Έτσι με βάση τα παραπάνω το παράδειγμα μας έχει ως εξής:

- Individual(**t1** *type* TimeInterval)
  value(tiStart, 01/01/1970)
- Individual(**Ένωση_Ζωγράφων_t1** *type* TimeSlice)
  value(tsTimeSliceOf, Ένωση_Ζωγράφων)
  value(tsTimeInterval, t1)

- Individual(**ΖωγράφοςΑ_t1** type TimeSlice)
  value (tsTimeSliceOf, **ΖωγράφοςΑ**)
  value (tsTimeInterval, **t1**)
  value(isPresidentOf, **Ένωση_Ζωγράφων_t1**)
  όπου isPresidentOf = fluent γνώρισμα που συνδέει time slices αντικειμένων.

Έτσι αντί να συνδέουμε τα αντικείμενα ΖωγράφοςΑ και Ένωση_Ζωγράφων απευθείας, συνδέουμε τα χρονικά κομμάτια τους για την περίοδο t1 η οποία ξεκινάει στις 01/01/1970:



Σίγουρα ο αριθμός νέων σχέσεων δεν είναι καλύτερος από το να χρησιμοποιούσαμε τη μέθοδο του Reification, ωστόσο εδώ μπορούμε να εκμεταλλευτούμε όλη τη δύναμη του reasoning ενώ η πιθανότητα λάθους μειώνεται αισθητά εφόσον τα πράγματα είναι πιο ευκολονόητα. Μάλιστα το ποσοστό πέφτει ακόμα περισσότερο με το plug-in μας εφόσον ένα μεγάλο μέρος των κινήσεων που πρέπει να κάνει ο χρήστης αυτοματοποιείται ενώ παράλληλα διενεργούνται και έλεγχοι που σε διαφορετική περίπτωση μπορεί να διέφευγαν του χρήστη.

2.5 Εργαλεία χειρισμού οντολογιών

Υπάρχουν πολλά εργαλεία χειρισμού οντολογιών[9] το καθένα από τα οποία έχει τις δικές του λειτουργικότητες και χρησιμοποιείται από διαφορικές ομάδες μελετητών.

Μερικοί από τους λόγους που διαφοροποιούν τα εργαλεία αυτά είναι:

- ο βαθμός ανεξαρτησίας του γραφικού περιβάλλοντος από την παραγόμενη γλώσσα ώστε να μην υπάρχουν περιορισμοί ή παρερμηνείες κατά την παρουσίαση των δεδομένων ή την πλοήγηση μέσα σε αυτά
- οι δυνατότητες που έχουν στον τομέα της εισαγωγής και εξαγωγής άλλων γλωσσών αναπαράστασης με απώτερο σκοπό τον έλεγχο ανάμεσα σε έννοιες
- την ύπαρξη εσωτερικών λειτουργιών για εξαγωγή δεδομένων και για δημιουργία απαντήσεων από μια βάση γνώσεων

Το Protégé, που θα δούμε στην επόμενη ενότητα και για το οποίο φτιάξαμε το plug-in, είναι ίσως το πιο γνωστό ανάμεσα στα εργαλεία χειρισμού οντολογιών ο αριθμός των οποίων είναι αρκετά μεγάλος. Τα περισσότερα είναι γραμμένα σε java και αποτελούν standalone

προγράμματα, παρόλα αυτά η λίστα περιέχει και εργαλεία που μπαίνουν ως plug-in σε άλλα προγράμματα ή που τρέχουν ως web services. Μερικά από αυτά είναι:

- *Anzo for Excel*[10]. Εργαλείο από την Cambridge semantics που παράγει οντολογίες από spreadsheet του excel. Είναι ένα από τα ελάχιστα που δεν είναι ανοικτού κώδικα.
- *Neologism*[11]. Γραμμένο σε PHP χρησιμοποιείται ως extension του drupal και υποστηρίζει κυρίως RDF καθώς και τμήματα της OWL. Δωρεάν και ανοικτού κώδικα.
- *Emftext*[12]. Άλλο ένα εργαλείο ανοικτού κώδικα που υποστηρίζει τόσο RDF όσο και OWL. Γραμμένο σε java εγκαθιστάτε ως plug-in του Eclipse.
- *OWLGrEd*[13]: Plug-in για το Protégé που χρησιμοποιεί UML για να παρουσιάσει και μεταβάλει μια οντολογία.

2.5.1 Protégé

Το Protégé[14] είναι πλατφόρμα ελεύθερου λογισμικού που βοηθάει μια συνεχώς αυξανόμενη κοινότητα να δημιουργήσει, να μεταβάλει και να προβάλει οντολογίες. Αν και το ίδιο είναι πλήρως παραμετροποιήσιμο ώστε ο κάθε χρήστης να αισθάνεται πιο άνετα στη χρήση του, προσφέρει και ένα API μέσω του οποίου μπορούν να δημιουργηθούν διάφορα είδη plug-in ώστε να εμπλουτιστούν οι λειτουργικότητες που προσφέρει στον τελικό χρήστη. Τα plug-in μπορούν να είναι από ένα επιπλέον tab στο γραφικό περιβάλλον του προγράμματος μέχρι έναν καινούργιο τρόπο εξαγωγής της οντολογίας σε συγκεκριμένο format.

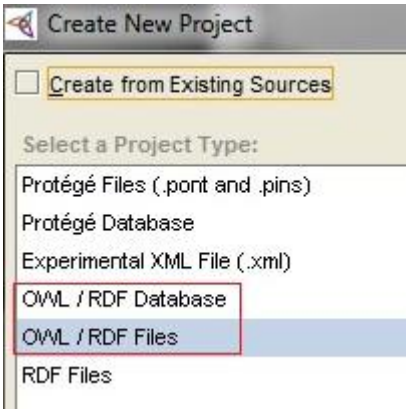
Συγκεκριμένα ένας προγραμματιστής μπορεί να δημιουργήσει για το Protégé:

- καινούργια tabs
- καινούργια γραφικά components για αναπαράσταση δεδομένων αντικειμένων
- καινούργιο τρόπο αποθήκευσης των δεδομένων ολόκληρου του project
- καινούργιο τρόπο να κάνεις εισαγωγή δεδομένων σε ένα project από αρχεία που δημιούργησε μια τρίτη εφαρμογή
- καινούργια μέθοδο εξαγωγής των δεδομένων του project στο φορμάτ που επιθυμείς, σε αρχείο ή κάποια βάση δεδομένων
- καινούργιο τρόπο διαχείρισης των project με προσθήκες τόσο στο γραφικό μενού του Protégé όσο και με την προσθήκη εργασιών που εκτελούνται σε κρίσιμα σημεία ενός project όπως είναι το άνοιγμα του, το κλείσιμο του ή ακόμα και η αποθήκευση του

Η πλατφόρμα υποστηρίζει δύο τρόπους ανάπτυξης οντολογιών:

1. Έναν frame-based τρόπο μέσω του Protégé-Frames editor κατά τον οποίο μια οντολογία αποτελείται από κλάσεις για την περιγραφή των εννοιών, slots για τα γνωρίσματα και τις σχέσεις και instances για τα αντικείμενα.
2. Έναν OWL-based τρόπο μέσω του Protégé-OWL editor κατά τον οποίο η οντολογία αποτελείται από περιγραφές των κλάσεων, των γνωρισμάτων και των αντικειμένων βάση των οποίων εξάγονται χρήσιμα συμπεράσματα για την περιοχή μελέτης.

Εμείς βασιστήκαμε και αναπτύξαμε το plug-in μας πάνω στον δεύτερο editor ο οποίος ενεργοποιείται αυτόματα αν κατά την δημιουργία καινούργιου project ο χρήστης επιλέξει μια από τις δυο OWL / RDF επιλογές όπως φαίνεται και στην εικόνα παρακάτω:



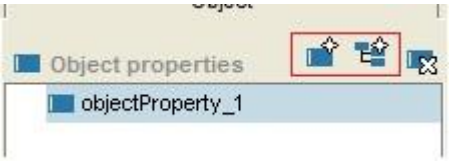
Η αρχική οθόνη του προγράμματος είναι πολύ απλή και χωρίζεται στην μπάρα του μενού, μια βοηθητική μπάρα με συντομεύσεις λειτουργιών και τέλος μια σειρά από tabs κάθε ένα από τα οποία κρατάει και συγκεκριμένες λειτουργικότητες για δημιουργία συγκεκριμένων κομματιών σε μια οντολογία.

Συγκεκριμένα:

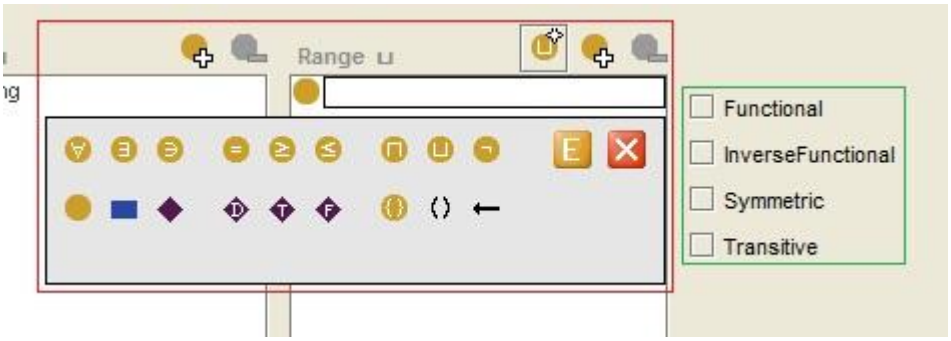
1. Στο tab **OWLClasses** ο χρήστης μπορεί να δημιουργήσει τις κλάσεις της οντολογίας του με την ιεραρχία που επιθυμεί

αλλά και να προσθέσει τους απαραίτητους περιορισμούς όπως τις κλάσεις με τις οποίες είναι disjoint

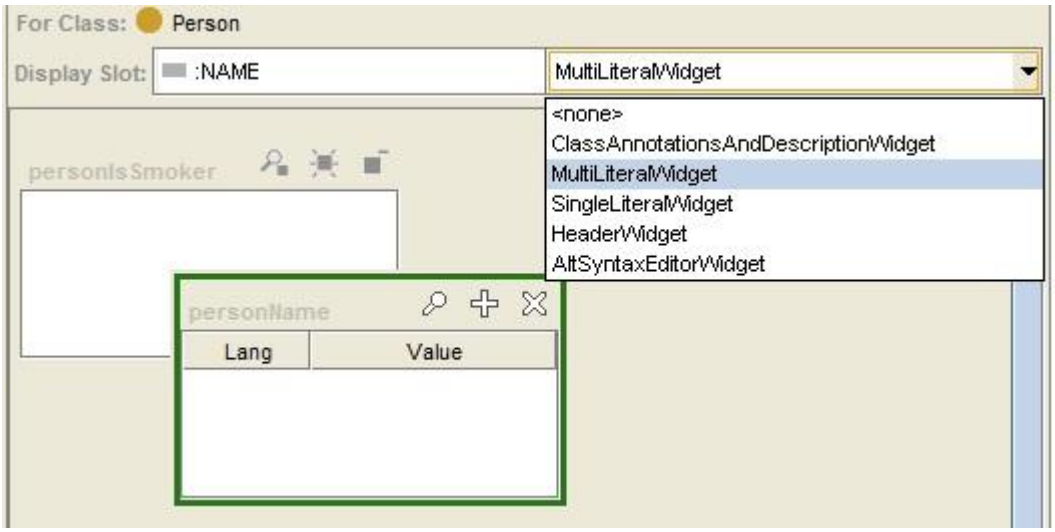
2. Στο tab **Properties** δίνεται η δυνατότητα στον χρήστη να προσθέσει τα γνωρίσματα και τις σχέσεις που επιθυμεί ορίζοντας πλήρως τόσο την ιεραρχία των γνωρισμάτων



όσο και το ακριβές range της σχέσης (κόκκινο περίγραμμα) ή το είδος του γνωρίσματος



3. Στο tab Individuals ο χρήστης καλείται να δημιουργήσει τα αντικείμενα της οντολογίας του αλλά και να θέσει τιμές στα γνωρίσματα και στις σχέσεις που έχουν τεθεί για αυτά.
4. Τέλος στο tab Forms το Protégé επιτρέπει στον χρήστη να ορίσει την σειρά, την διάταξη και το είδος των γραφικών components που θα χρησιμοποιηθούν για την αναπαράσταση των σχέσεων και τον ορισμό των τιμών των γνωρισμάτων



Κεφάλαιο 3

Χειρισμός χρονικά εξελισσόμενης πληροφορίας στο Protégé

Όπως είδαμε και παραπάνω η δημιουργία μιας οντολογίας τεσσάρων διαστάσεων βασίζεται στη δημιουργία κλάσεων και σχέσεων απώτερος σκοπός των οποίων είναι να παρουσιάζουν τις σχέσεις ανάμεσα σε αντικείμενα συγκεκριμένης χρονικής περιόδου. Κάτι τέτοιο μπορεί να γίνει σε όλα τα γραφικά εργαλεία όσο ο χρήστης ακολουθεί συγκεκριμένα και προσεκτικά βήματα. Ωστόσο τόσο η συντήρηση όσο και η παρουσίαση της οντολογίας αποτελούν μειονέκτημα που κανένα εργαλείο, συμπεριλαμβανομένου και του Protégé, δεν έχει καταφέρει να λύσει.

Το plug-in μας αναλαμβάνει να καλύψει την ανάγκη αυτή με μια σειρά γραφικών components που βοηθούν τον χρήστη τόσο να δημιουργήσει όσο και να παρουσιάσει κλάσεις, σχέσεις και αντικείμενα ενώ παράλληλα διατηρεί τον παραγόμενο OWL κώδικα καθαρό και χωρίς δεσμεύσεις σε συγκεκριμένα εργαλεία, όπως το plug-in μας, ώστε να μπορεί να μεταφερθεί και χρησιμοποιηθεί από τον χρήστη όπως και όπου αυτός επιθυμεί.

Για να αντιληφθούμε καλύτερα τις προσθήκες του plug-in μας στο Protégé καλό είναι να πάρουμε τα πράγματα από την αρχή και μέσα από ένα παράδειγμα να παρουσιάσουμε τα συν και τα πλην τόσο του Protégé όσο και του plug-in μας.

Έστω ότι θέλουμε να κατασκευάσουμε μια οντολογία στην οποία θα παρουσιάζονται στοιχεία ατόμων όπως το που εργάζονται, το τι αμάξι οδηγούν, το όνομα τους και το αν είναι καπνιστές. Πρόκειται για μια πολύ απλή οντολογία ιδανική για να παρουσιάσουμε όσα αναφέραμε παραπάνω.

Οι βασικές κλάσεις της οντολογίας μας θα είναι οι τρεις παρακάτω:

- **Person**
- **Company**
- **Car**

Πολύ γενικές κλάσεις που δεν χρειάζονται περαιτέρω ανάλυση.

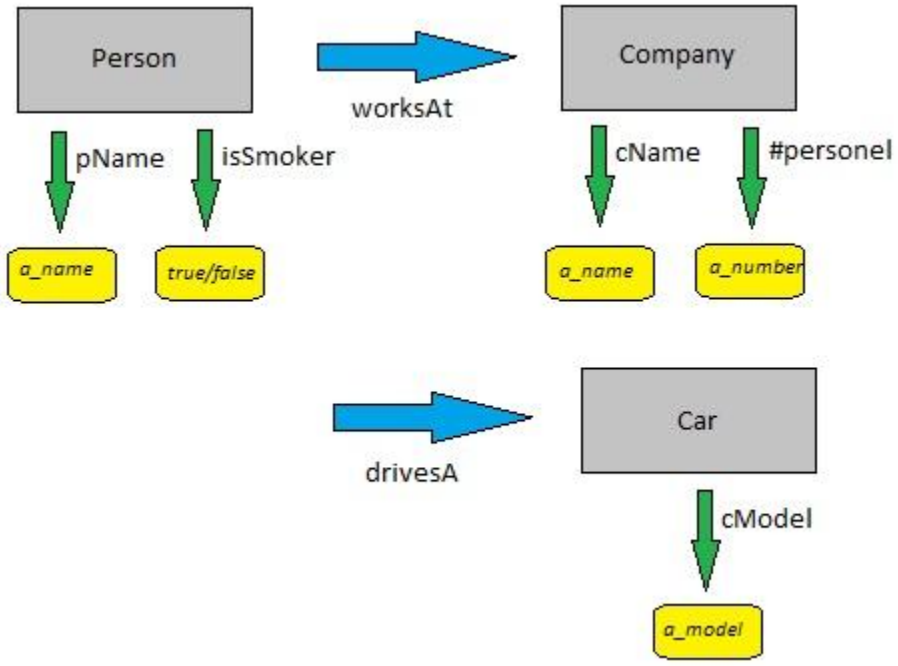
Οι σχέσεις που θα χρειαστεί να δημιουργήσουμε είναι:

- η **worksAt** η οποία θα συνδέει αντικείμενα της κλάσης Person (domain) με αντικείμενα της κλάσης Company (range) και στην ουσία θα παρουσιάζει σε ποια εταιρεία εργάζεται το κάθε άτομο της οντολογίας, και
- η **drivesA** η οποία θα συνδέει αντικείμενα της κλάσης Person (domain) με αντικείμενα της κλάσης Car (range) παρουσιάζοντας έτσι το μοντέλο του αυτοκινήτου που κάποιο άτομο της οντολογίας οδηγεί.

Τέλος, για να καλύψουμε όλα τα στοιχεία μιας οντολογίας, θα δημιουργήσουμε και κάποια γνωρίσματα. Συγκεκριμένα:

- το **personName**, ένα απλό αλφαριθμητικό (range) το οποίο υποδεικνύει την ονομασία ενός αντικείμενου της κλάσης Person (domain)
- το **personIsSmoker**, ένα γνώρισμα τύπου boolean (range) μέσω του οποίου ένα αντικείμενο της κλάσης Person (domain) θα χαρακτηρίζεται ως καπνιστής ή όχι.
- το **companyName**, ένα απλό αλφαριθμητικό (range) το οποίο υποδεικνύει την ονομασία ενός αντικείμενου της κλάσης Company (domain)
- το **companyPersonelNumber**, ένα γνώρισμα ακέραιου τύπου (range) που παρουσιάζει τον αριθμό των εργαζομένων σε μια εταιρεία (αντικείμενο της κλάσης Company - domain), και τέλος
- το **carModel**, ένα απλό αλφαριθμητικό (range) το οποίο υποδεικνύει την ονομασία ενός αντικείμενου της κλάσης Car (domain)

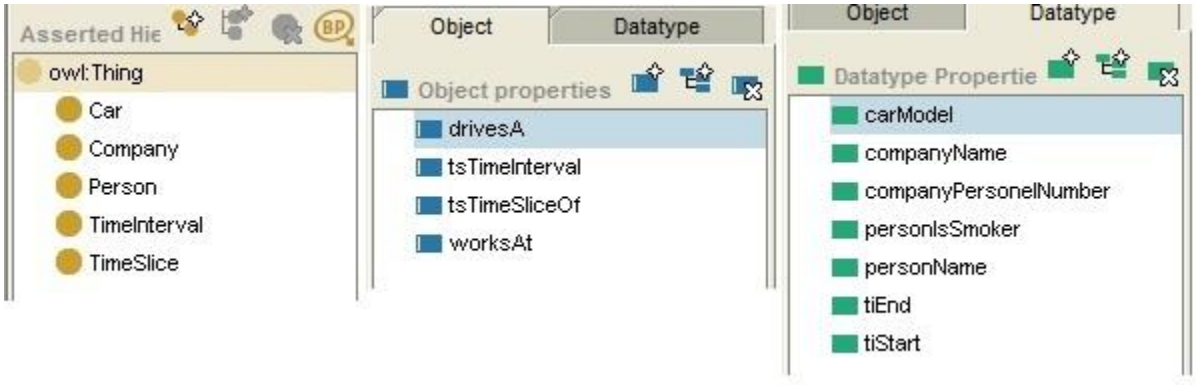
Σχηματικά η οντολογία μας παρουσιάζεται παρακάτω:



Φυσικά, επειδή θέλουμε η οντολογία μας να παρουσιάζει χρονικά εξελισσόμενη πληροφορία, θα δημιουργήσουμε και τις κλάσεις και τα γνωρίσματα που αναφέραμε παραπάνω μελετώντας την προσέγγιση των τεσσάρων διαστάσεων (βλ: 2.4.1).

- ✓ Ένα από τα θετικά σημεία του plug-in μας είναι ότι αναλαμβάνει το ίδιο την δημιουργία των σχετικών κλάσεων και γνωρισμάτων φροντίζοντας παράλληλα και την προσθήκη των απαραίτητων περιορισμών όπως το ότι η κλάση TimeSlice πρέπει να είναι disjoint με την κλάση TimeInterval ή ότι το γνώρισμα tsTimeSliceOf δεν μπορεί να έχει για range αντικείμενα της κλάσης TimeInterval.

Έτσι στο τέλος των πρώτων ενεργειών μας η οντολογία θα παρουσιάζεται στο Protégé κάπως έτσι:



Αναμφισβήτητα μια από τις πρώτες κινήσεις που θα θέλει ένας χρήστης να πραγματοποιήσει είναι το να δημιουργήσει μια fluent σύνδεση ανάμεσα σε δυο στιγμιότυπα αντικειμένων. Έστω λοιπόν ότι θέλουμε να δείξουμε στην οντολογία μας το παρακάτω:

*ο **Leonidas**, για την περίοδο **1/1/2000 - 1/1/2010**, δούλευε στην εταιρεία **CompanyA***

Τα βήματα που θα πρέπει να ακολουθήσουμε είναι τα εξής:

1. Πλοηγούμαστε στο *tab-Individuals* όπου εκεί για την κλάση *Person* δημιουργούμε ένα αντικείμενο με την ονομασία **Leonidas**.
2. Στο ίδιο tab αλλά για την κλάση *Company* δημιουργούμε το αντικείμενο **CompanyA**.

Ως εδώ έχουμε δυο απλά αντικείμενα. Η κατευθείαν συσχέτιση τους δεν είναι λανθασμένη αλλά δεν παρουσιάζει τη σχέση σωστά, δηλαδή για τη χρονική περίοδο που θέλουμε. Για τον λόγο αυτό επόμενο στάδιο είναι η δημιουργία δυο time-slices τα οποία θα συνδεθούν με συγκεκριμένη χρονική περίοδο και με κάθε ένα από τα παραπάνω αντικείμενα:

3. Δημιουργία της χρονικής περιόδου. Παραμένοντας στο *tab-Individuals* δημιουργούμε ένα αντικείμενο για την κλάση *TimeInterval*, με ονομασία **010100_010110,** στο οποίο και δίνουμε τις τιμές **01/01/00**, **01/01/10** στα πεδία *tiStart*, *tiEnd* αντίστοιχα.
4. Δημιουργία των time-slices. Στο ίδιο tab αλλά αυτή τη φορά για την κλάση TimeSlice δημιουργούμε δυο αντικείμενα. Το πρώτο το ονομάζουμε **Leonidas_010100_010110**, το συσχετίζουμε με το αντικείμενο Leonidas (μέσω του γνωρίσματος tsTimeSliceOf) και θέτουμε ως ισχύουσα χρονική περίοδο την 010100_010110 που φτιάξαμε στο βήμα 3.

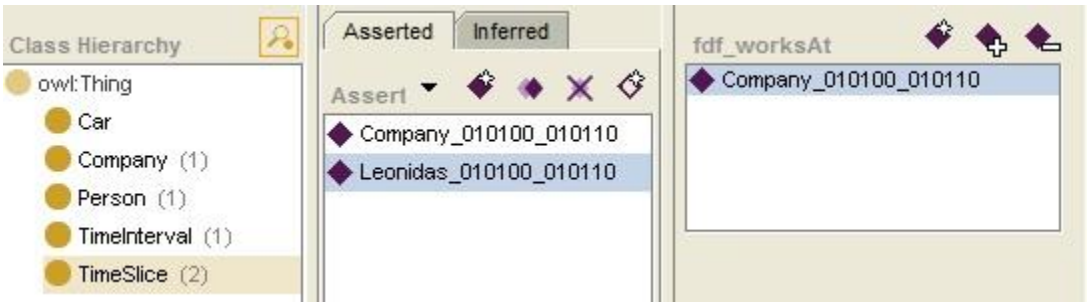
Το δεύτερο αντικείμενο το ονομάζουμε **CompanyA_010100_010110** ορίζοντας στα σχετικά γνωρίσματα τα αντικείμενα CompanyA και 010100_010110 αντίστοιχα.

Στο σημείο αυτό έχουμε έτοιμα τόσο τα αντικείμενα της πρότασης μας όσο και τα χρονικά τους στιγμιότυπα. Ωστόσο δε μπορούμε να τα συνδέσουμε άμεσα γιατί η σχέση worksAt δεν είναι fluent δηλαδή δεν συνδέει TimeSlices μεταξύ τους. Άρα στην όλη διαδικασία προσθέτουμε ένα ακόμα βήμα, αυτό της δημιουργίας ενός fluent γνωρίσματος για κάθε γνώρισμα που θέλουμε να έχει την ιδιότητα να παρουσιάζει συσχετίσεις χρονικών στιγμιότυπων.

5. Πλοήγηση στο tab-Properties και δημιουργία γνωρίσματος που θα έχει και για domain και για range την κλάση *TimeSlice*. Καλό είναι η ονομασία που διαλέγουμε να είναι παρόμοια με το αρχικό γνώρισμα για να το βρίσκουμε πιο εύκολα. Στην προκειμένη περίπτωση διαλέξαμε την **fdf_worksAt**.

Τώρα είμαστε σε θέση να δημιουργήσουμε τη σχέση που ορίσαμε στην πρόταση μας:

6. Πλοήγηση πίσω στο tab-Individuals, επιλογή της κλάσης TimeSlice ώστε να δούμε <u>όλα</u> τα χρονικά στιγμιότυπα, επιλογή του Leonidas_010100_010110 και τέλος συσχέτιση του αντικειμένου αυτού με το αντικείμενο CompanyA_010100_010110 μέσω της σχέσης fdf_worksAt.



Θεωρητικά λοιπόν μπορούμε να πούμε πως μεταφέραμε επιτυχώς την πρόταση που αναφέραμε παραπάνω στην οντολογία μας:

```
- <TimeSlice rdf:ID="Leonidas_010100_010110">
  - <fdf_worksAt>
    - <TimeSlice rdf:ID="Company_010100_010110">
      - <tsTimeInterval>
        - <TimeInterval rdf:about="#010100_010110">
            <tiEnd rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-01-01T23:01:26</tiEnd>
            <tiStart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2000-01-01T23:01:11</tiStart>
          </TimeInterval>
        </tsTimeInterval>
      - <tsTimeSliceOf>
          <Company rdf:ID="CompanyA"/>
        </tsTimeSliceOf>
      </TimeSlice>
    </fdf_worksAt>
  - <tsTimeSliceOf>
      <Person rdf:ID="Leonidas"/>
    </tsTimeSliceOf>
    <tsTimeInterval rdf:resource="#010100_010110"/>
  </TimeSlice>
```

Ωστόσο δε μπορούμε να παραβλέψουμε κάποια από τα αρνητικά της όλης προσέγγισης:

- Ξεκινάμε φυσικά από την άσχημη παρουσίαση του αποτελέσματος η οποία μάλιστα όσο αυξάνονται τα time-slices θα γίνεται ακόμα πιο δύσκολη για τον χρήστη να μελετήσει. Όλα τα time-slices, ανεξάρτητα σε πιο αντικείμενο ανήκουν, παρουσιάζονται στον ίδιο χώρο, ως αντικείμενα της κλάσης TimeSlice, χωρίς να δίνεται η δυνατότητα διάκρισης τους ώστε ο χρήστης να βρει εύκολα αυτό που ζητάει. Έτσι αν ήθελε να δει ποια time-slices ανήκουν στο αντικείμενο Leonidas θα έπρεπε, **αναγκαστικά**, να ανοίξει ένα-ένα τα αντικείμενα για να δει ποια είναι η τιμή του γνωρίσματος tsTimeSliceOf. Χρονοβόρα και άβολη διαδικασία.
- Ένα δεύτερο αρνητικό είναι τα πολλά βήματα που έπρεπε να ακολουθήσει ο χρήστης για να δημιουργήσει την σχέση τα οποία μάλιστα τον οδηγούσαν σε διαφορετικά σημεία του προγράμματος το καθένα από τα οποία είχε και διαφορετική σημασιολογική έννοια. Σε άλλο σημείο του προγράμματος για να δημιουργήσει αντικείμενα κλάσης, σε άλλο για την δημιουργία χρονικών περιόδων και χρονικών στιγμιότυπων και σε τρίτο για την δημιουργία της σχέσης που θέλαμε.
- Και σαν μην έφτανε η πολυπλοκότητα για την δημιουργία της σχέσης η όλη διαδικασία είναι τέτοια που μπορεί εύκολα να οδηγήσει σε σοβαρά λάθη όπως τη χρήση λάθους σχέσης. Μη ξεχνάμε ότι για να παρουσιάσουμε τη σχέση που θέλαμε αναγκαστήκαμε να δημιουργήσουμε ένα δεύτερο γνώρισμα το όποιο θα έπαιζε το ρόλο της fluent σχέσης. Φανταστείτε τώρα να κάναμε το ίδιο για **όλες** τις σχέσεις **όλων** των κλάσεων της οντολογίας μας. Θα καταλήγαμε η κλάση TimeSlice να έχει έναν μεγάλο αριθμό σχέσεων με κάθε μια από τις οποίες να συνδέει, **χωρίς έλεγχο**, οποιοδήποτε από τα αντικείμενα της TimeSlice. Έτσι αν ο ίδιος ο χρήστης δεν είναι προσεκτικός θα μπορούσε μια σχέση να αποτελείται όχι μόνο από αντικείμενα λάθος κλάσεων αλλά και από αντικείμενα διαφορετικών χρονικών περιόδων.

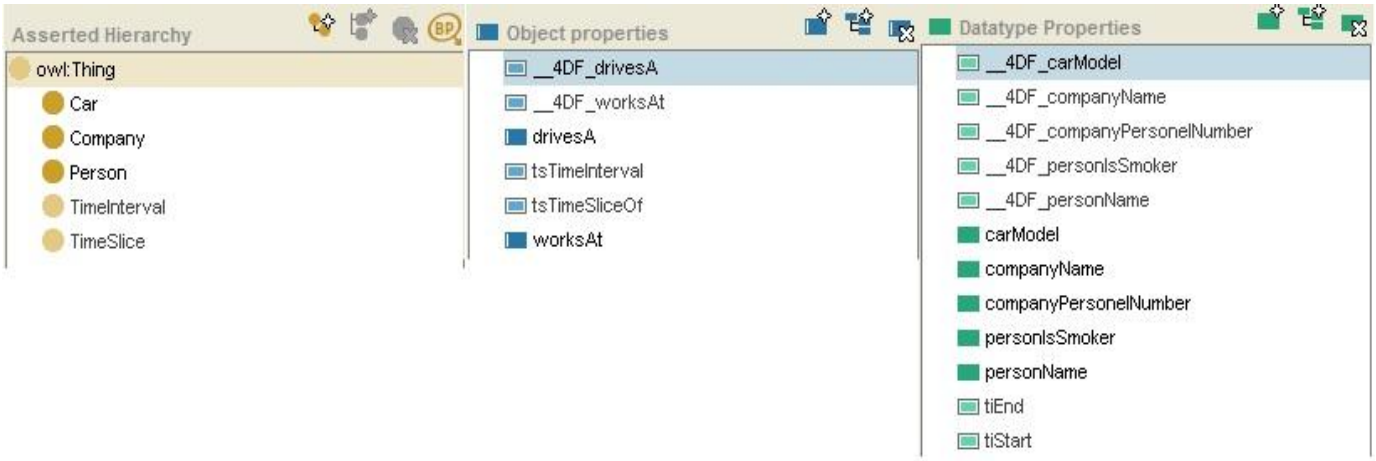
Ας δούμε τώρα την δημιουργία της ίδιας σχέσης μέσα όμως από την χρήση του plug-in μας. Όπως και παραπάνω πριν την δημιουργία της σχέσης θα πρέπει πρώτα να δημιουργήσουμε τις κλάσεις, τις σχέσεις και τα γνωρίσματα της οντολογίας μας. Η διαφορά με την προηγούμενη προσέγγιση είναι ότι τώρα θα είναι ενεργοποιημένο και το plug-in μας. Μάλιστα η ενεργοποίηση του έχει ως άμεσο αποτέλεσμα να δημιουργηθούν:

a. οι δυο βασικές κλάσεις TimeSlice και TimeInterval μαζί με τους απαραίτητους περιορισμούς (disjoints) που πρέπει να έχουν στην οντολογία μας
b. τα δυο βασικά object properties τους tsTimeSliceOf και tsTimeInteval
c. και τέλος τα δυο datatype properties tiStart και tiEnd.

ενώ ως έμμεσο αποτέλεσμα κάθε φορά που ο χρήστης προσθέτει μια καινούργια σχέση ή ένα γνώρισμα, το plug-in αναλαμβάνει να δημιουργήσει το αντίστοιχο fluent δίνοντας του ονομασία ίδια με αυτή του αρχικού γνωρίσματος προσθέτοντας όμως και το πρόθεμα *__4DF_*. Φυσικά κάθε αλλαγή στον ονομασία του αρχικού γνωρίσματος θα γίνει και στο fluent.

Να σημειώσουμε εδώ ότι όλες οι προσθήκες του plug-in γίνονται σε read only μορφή ώστε ο χρήστης να μην σβήσει ή αλλάξει κάτι καταλάθος.

Έτσι μετά από το αρχικό στάδιο η οντολογία μας στο Protégé δείχνει κάπως έτσι:

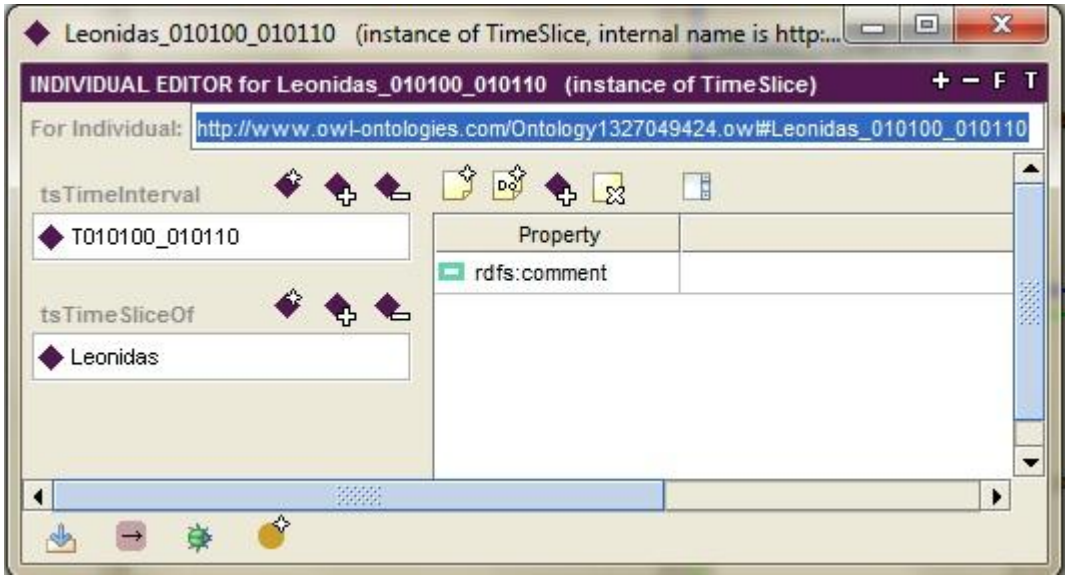


Η οντολογία μας είναι έτοιμη οπότε μπορούμε να δημιουργήσουμε την σχέση του παραδείγματος μας. Όπως και στην περίπτωση που δεν έχουμε το plug-in στο Protégé έτσι και εδώ ξεκινάμε τα ίδια πρώτα βήματα ώστε να δημιουργήσουμε τα αντικείμενα Leonidas και CompanyA:

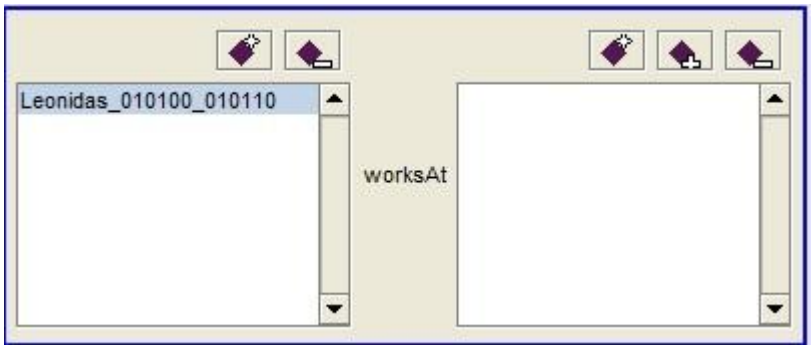
1. Πλοηγούμαστε στο *tab-Individuals* όπου εκεί για την κλάση *Person* δημιουργούμε ένα αντικείμενο με την ονομασία **Leonidas**.
2. Στο ίδιο tab αλλά για την κλάση *Company* δημιουργούμε το αντικείμενο **CompanyA**

Το σημείο αυτό είναι που κάνει το plug-in μας να ξεχωρίζει γιατί σε αντίθεση με πριν που ακολουθούσαν 3 ακόμα βήματα σε διαφορετικά σημεία του προγράμματος εμείς σε ένα και μόνο σημείο με ένα βήμα θα ολοκληρώσουμε τη δημιουργία της σχέσης:

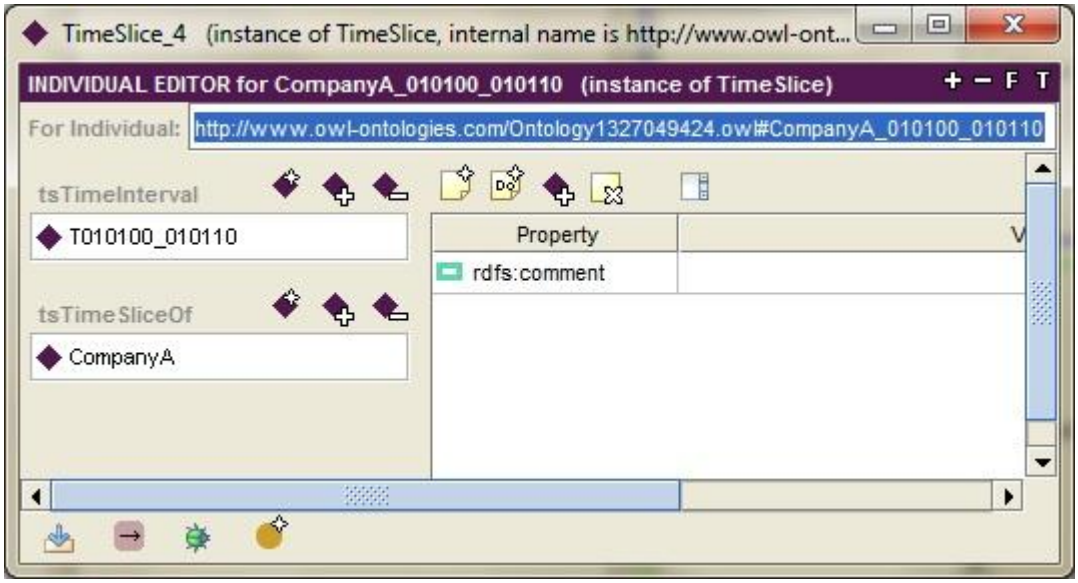
3. Παραμένουμε στο tab-Individuals και επανερχόμαστε στο αντικείμενο Leonidas της κλάσης Person. Στη σχέση worksAt επιλέγουμε την δημιουργία καινούργιου time-slice η οποία μας οδηγεί στη συμπλήρωση της παρακάτω φόρμας:



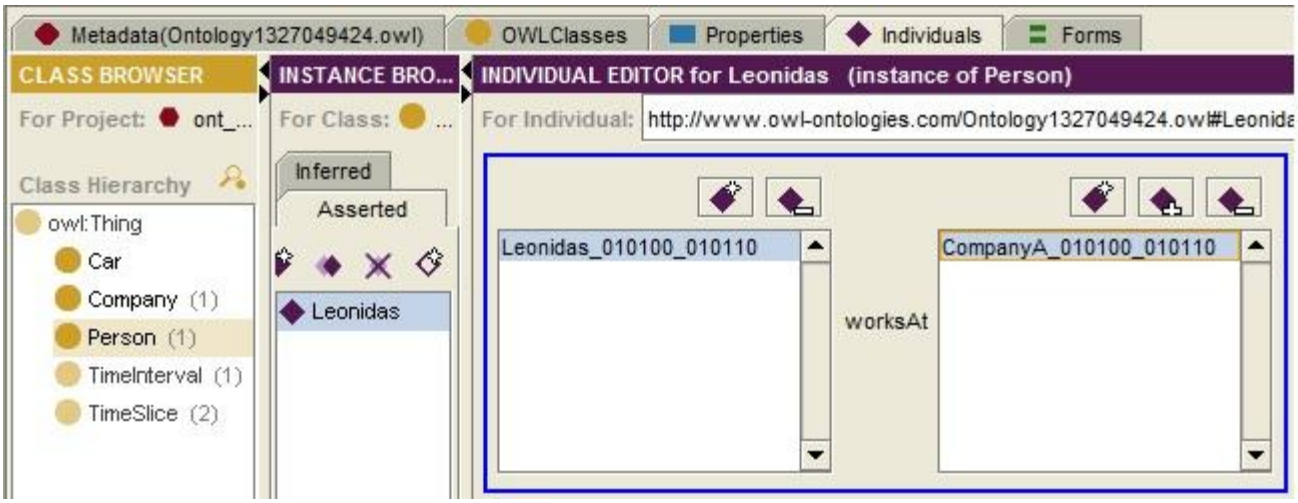
Το πεδίο tsTimeSliceOf έχει έρθει προσυμπληρωμένο με το αντικείμενο Leonidas άρα το μόνο που μένει είναι να δημιουργήσουμε μια χρονική περίοδο για το tsTimeInterval και να ονοματίσουμε το time-slice. Στο τέλος το component που φτιάξαμε για να δείχνει τις fluent σχέσεις θα μοιάζει κάπως έτσι:



Το μόνο που απομένει τώρα είναι να δημιουργήσουμε ένα time-slice για το αντικείμενο CompanyA και να το ενώσουμε με το Leonidas_010100_010110. Για να το πετύχουμε αυτό στο ίδιο component της σχέσης worksAt απλά πατάμε το κουμπί δημιουργίας για το range της σχέσης, διαλέγουμε το αντικείμενο που θέλουμε (εδώ το CompanyA) και απλά συμπληρώνουμε το όνομα που θέλουμε να δώσουμε στο αντικείμενο:



τόσο το πεδίο που δείχνει σε πιο αντικείμενο ανήκει το time-slice (tsTimeSliceOf) όσο και το πεδίο που δείχνει την χρονική περίοδο για την οποία ισχύει (tsTimeInterval) έχουν έρθει προσυμπληρωμένα. Το τελικό αποτέλεσμα είναι σίγουρα πολύ καλύτερο από αυτό της προηγούμενης προσέγγισης εφόσον τώρα ο χρήστης μπορεί με μια ματιά να καταλάβει ποια είναι η σχέση, ποια τα αντικείμενα της σχέσης και ποια τα χρονικά στιγμιότυπα των αντικείμενων:



Όσον αφορά τον κώδικα που παράχθηκε, όπως θα φαίνεται και παρακάτω, δεν διαφέρει σε τίποτα από αυτόν της πρώτης μεθόδου και άρα μπορεί να μεταφερθεί και σε άλλα προγράμματα παρόμοια του Protégé:

```
−<TimeSlice rdf:ID="Leonidas_010100_010110">
  −<__4DF_worksAt>
    −<TimeSlice rdf:ID="CompanyA_010100_010110">
      −<tsTimeInterval>
        −<TimeInterval rdf:ID="T010100_010110">
            <tsTimeInterval rdf:resource="#T010100_010110"/>
            <tiEnd rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2010-01-01T16:15:16</tiEnd>
            <tiStart rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2000-01-01T16:15:01</tiStart>
          </TimeInterval>
        </tsTimeInterval>
      −<tsTimeSliceOf>
        −<Company rdf:ID="CompanyA">
            <tsTimeInterval rdf:resource="#T010100_010110"/>
          </Company>
        </tsTimeSliceOf>
      </TimeSlice>
    </__4DF_worksAt>
    <tsTimeInterval rdf:resource="#T010100_010110"/>
  −<tsTimeSliceOf>
    −<Person rdf:ID="Leonidas">
        <tsTimeInterval rdf:resource="#T010100_010110"/>
      </Person>
    </tsTimeSliceOf>
  </TimeSlice>
```

Η προσέγγιση μας με βάση το plug-in κατάφερε να κάνει την όλη διαδικασία πιο γρήγορη, πιο ασφαλή και σίγουρα πιο απλή:

- Το γραφικό component που σχεδιάσαμε για το plug-in δίνει την ψευδαίσθηση στον χρήστη ότι χειρίζεται τις βασικές του σχέσεις, εδώ την worksAt, όταν στην πραγματικότητα δημιουργεί αντικείμενα και συνδυασμούς για τις αντίστοιχες fluent τους, εδώ την __4DF_worksAt. Με αυτό τον τρόπο, σε αντίθεση με την πρώτη μας προσπάθεια, το αποτέλεσμα είναι πιο ευπαρουσίαστο, πιο ευκολονόητο και βοηθάει τον χρήστη να διακρίνει ποια time-slices ανήκουν που.
- Παράλληλα η όλη διαδικασία γίνεται σχεδόν αποκλειστικά από ένα και μόνο σημείο του προγράμματος με σαφή βήματα που ενώνονται εννοιολογικά μεταξύ τους αποτελώντας μια φυσική συνέχεια στις κινήσεις του χρήστη αποτρέποντας τον έτσι να αμελήσει κάποιο βήμα.
- Τέλος ο συνεχής έλεγχος που προσφέρει το plug-in και οι βοήθειες που δίνει στον χρήστη, όπως το να προσυμπληρώνει τα πεδία όταν έχει τα απαραίτητα δεδομένα, αποτελούν σημαντικό παράγοντα στο να αποφευχθούν δημιουργίες σχέσεων ανάμεσα σε αντικείμενα με διαφορετικές χρονικές περιόδους ή σε αντικείμενα που βάση των αρχικών σχέσεων δεν θα έπρεπε να σχετίζονται.

Στις επόμενες ενότητες θα δούμε αναλυτικά το plug-in. Τι προσφέρει σε γραφικό επίπεδο ποιες είναι οι λειτουργικότητες του αλλά και το πώς μπορεί ο χρήστης να το χρησιμοποιήσει.

3.1 Εγκατάσταση του plug-in

Το Protégé φτιάχτηκε με τρόπο που το καθιστά εύκολα επεκτάσιμο σε λειτουργικότητα μέσω plug-ins. Μάλιστα το ίδιο το Protégé-owl είναι από μόνο του ένα καλογραμμένο plug-in το οποίο και φορτώνεται κατά το άνοιγμα της εφαρμογής. Για τον λόγο αυτό οι δημιουργοί του Protégé σχεδίασαν τον τρόπο φόρτωσης των plug-ins έτσι ώστε να είναι απλός και γρήγορος.

Αρχικά ο δημιουργός του plug-in δημιουργεί ένα φάκελο με την ονομασία που επιθυμεί και μέσα σε αυτόν τοποθετεί όσα αρχεία χρειάζεται το plug-in του αρκεί δυο από αυτά να είναι: (α) η ίδια εφαρμογή που έγραψε σε μορφή jar και (β) ένα αρχείο κειμένου με συγκεκριμένη δομή μέσω του οποίου το Protégé συσχετίζει το κάθε plug-in με κάποια ονομασία και επιλύει τυχόν εξαρτήσεις που μπορεί να έχει από βιβλιοθήκες ή άλλα plug-ins. Το αρχείο αυτό ονομάζεται αναγκαστικά **plug-in.properties**, πρέπει να βρίσκετε στο πρώτο επίπεδο του φακέλου που δημιούργησε ο χρήστης για το plug-in του και η δομή του πρέπει να περιέχει μια σειρά συγκεκριμένων μεταβλητών τις οποίες αναγνωρίζει μόνο το Protégé.

Αφού δημιουργηθεί ο φάκελος αυτός τοποθετείται στον υποφάκελο plug-ins που υπάρχει στο σημείο εγκατάστασης του Protégé και μέσα στον οποίο τοποθετούνται όλα τα plug-ins. Το Protégé κατά την εκκίνηση του σκανάρει τον συγκεκριμένο φάκελο και για κάθε υποφάκελο που συναντά φορτώνει το .jar αρχείο που βρίσκει έχοντας πρώτα θέσει συγκεκριμένες τιμές

σε κάποιες εσωτερικές του μεταβλητές βασισμένο στο αρχείο plug-in.properties του υποφακέλου.

Για τις ανάγκες του δικού μας προγράμματος κινηθήκαμε ως εξής:

Για την ονομασία του φακέλου μας διαλέξαμε να γράψουμε το URL του εργαστηρίου ανάποδα καταλήγοντας στην ονομασία του plug-in: **gr.tuc.intelligence.fdf** .

Μέσα στον φάκελο τοποθετήσαμε:

- το αρχείο **4DFluents.jar** που είναι το plug-in
- το αρχείο **4DFluents.conf** που αποτελεί ένα βοηθητικό αρχείο της εφαρμογής μας για να κρατάει τις τιμές των βασικών παραμέτρων της. Με τον τρόπο αυτό ο χρήστης μπορεί να ορίσει μια συγκεκριμένη συμπεριφορά από το plug-in για όλα τα projects του.
- το αρχείο **about_fdf.html** που αποτελεί την σελίδα που παρουσιάζεται στο About του plug-in.
- το αρχείο **help.html** που αποτελεί τη σελίδα που παρουσιάζεται στο Help του plug-in.
- και φυσικά το αρχείο **plug-in.properties** ώστε να γνωρίζει το Protégé πόσα plug-ins θέλουμε να φορτώσει και από τι εξαρτούνται αυτά. Συγκεκριμένα, όσον αφορά το plug-in.properties οι μεταβλητές που χρησιμοποιήσαμε είναι οι εξής:

  1. **plug-in.component.count** : μέσω της μεταβλητής αυτής ο χρήστης δηλώνει πόσα plug-ins πρέπει να περιμένει το Protégé να βρει μέσα στον φάκελο. Μάλιστα για κάθε plug-in δίνεται η ευχέρεια να δηλωθούν και κάποιες λεπτομέρειες για αυτό όπως το όνομα του ή το documentation του. Ο τρόπος για να ξεχωρίσουν οι πολλαπλές τιμές για τις μεταβλητές αυτές είναι μέσω ενός αριθμού Ν που μπαίνει στο τέλος της κάθε μεταβλητής και ο οποίος ξεκινάει από το μηδέν και αυξάνει κατά ένα για κάθε plug-in που έχουμε . Για τη περίπτωση μας η μεταβλητή αυτή έχει την τιμή 1 και άρα το Ν παίρνει μόνο την τιμή 0. Έτσι στο αρχείο θα συναντήσουμε τα παρακάτω: (α) **plug-in.component.count=1** με το οποίο δηλώνουμε πως μέσα στον φάκελο υπάρχει μόνο ένα plug-in, (β) **plug-in.component.name.0=4D Fluents** με το οποίο δηλώνουμε την ονομασία του plug-in, (γ) **plug-in.component.doc.0=help.html** με το οποίο δηλώνουμε ποιο αρχείο αποτελεί το documentation του plug-in και (δ) **plug-in.component.about.0=about_fdf.html** με το οποίο δηλώνουμε ποιο αρχείο πρέπει να χρησιμοποιηθεί όταν ζητήσει κάποιος το About του plug-in.
  2. **plug-in.dependency.count** : με την οποία ο χρήστης δηλώνει πόσες εξαρτήσεις έχει το plug-in που δημιούργησε. Και η μεταβλητή αυτή ακολουθεί την σύνταξη πολλαπλών τιμών που είδαμε πριν εφόσον μπορεί να εξαρτάται από παραπάνω από ένα πράγματα (βιβλιοθήκες, plug-ins). Το δικό μας plug-in έχει μόνο μια εξάρτηση και αυτή δεν είναι άλλη από το ίδιο το Protégé-owl το οποίο φορτώνεται και αυτό ως plug-in. Έτσι στο αρχείο θα συναντήσουμε τα παρακάτω: (α) **plug-in.dependency.count=1** το οποίο δηλώνει ότι υπάρχει μια μόνο εξάρτηση και (β) **plug-in.dependency.0=edu.stanford.smi.Protégéx.owl** μέσω του οποίου δηλώνεται ο φάκελος που περιέχει το κατάλληλο plug-in.

3.2 Το μενού του plug-in

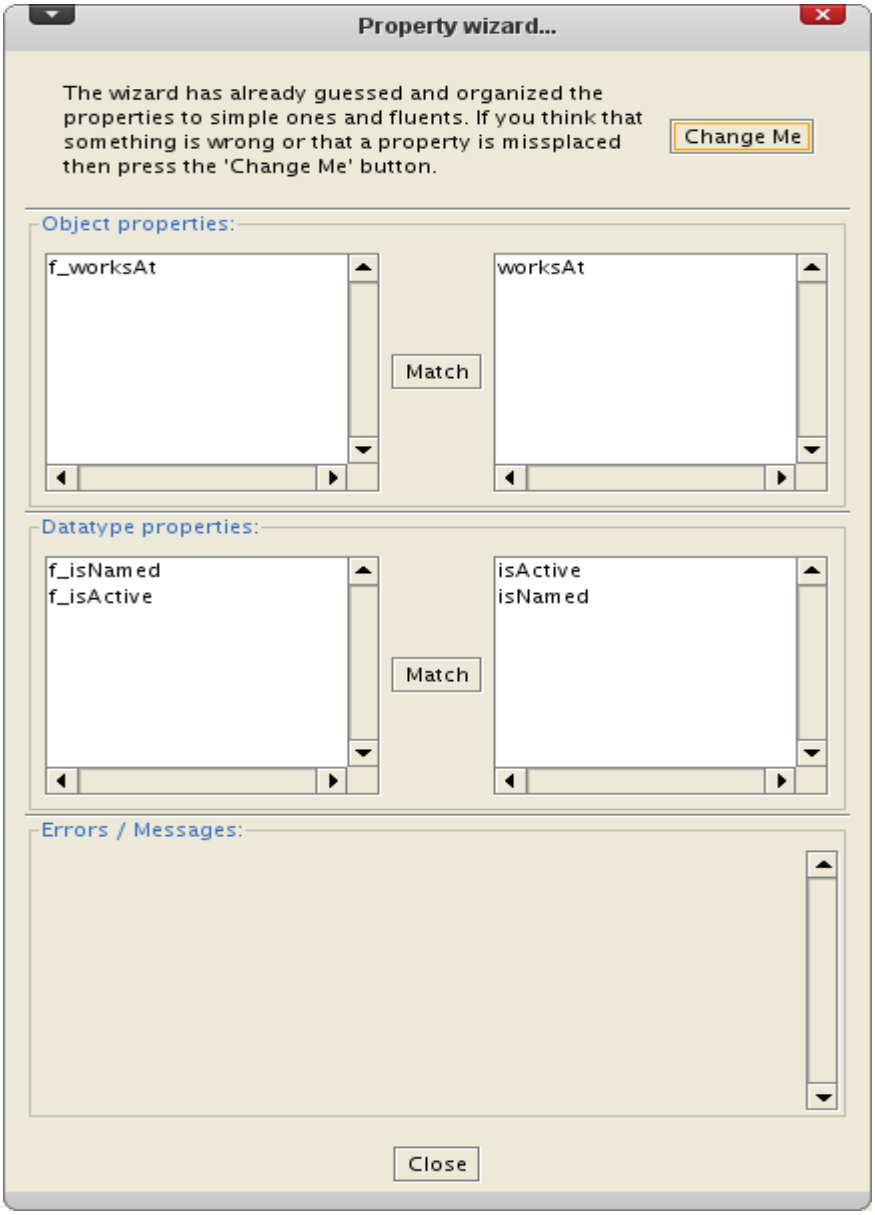
Το plug-in είναι σχεδιασμένο έτσι ώστε όταν ο χρήστης δημιουργήσει ή ανοίξει κάποιο project που είναι βασισμένο σε owl να προσθέτει στην menu-bar του Protégé το δικό μου μενού:

3.2.1 Menu item: *Status*

Μέσω του *Status* ο χρήστης ενεργοποιεί και απενεργοποιεί το plug-in. Δεν έχει σημασία σε πιο χρονικό σημείο της εργασίας του γίνεται η ενεργοποίηση εφόσον είτε γίνει στην αρχή ενός καινούργιου project είτε πολύ αργότερα σε μια ήδη γεμάτη από αντικείμενα οντολογία, το plug-in θα αναγνωρίσει τη κατάσταση του project και θα προσαρμοστεί στις ανάγκες και τις επιλογές του χρήστη. Θα δούμε αναλυτικότερα την λειτουργία ενεργοποίησης του plug-in σε επόμενη ενότητα.

Αξίζει να σημειώσουμε πως το plug-in επιτρέπει στον χρήστη να ενεργοποιήσει και απενεργοποιήσει το plug-in όσες φορές το επιθυμεί. Μάλιστα αν ο χρήστης έχει μικρό αριθμό αντικειμένων η επανενεργοποίηση δε θα δημιουργήσει κάποια δυσλειτουργία καθώς έχουν προβλεφθεί και υλοποιηθεί κάποιες διαδικασίες επαναφοράς του project στη προτέρα του κατάσταση. Ωστόσο από τη στιγμή που το πρόγραμμα μας δε σχεδιάστηκε για να υποστηρίξει extreme καταστάσεις χρήσης στη περίπτωση που το project χειρίζεται μεγάλο αριθμό κλάσεων, γνωρισμάτων, time slices κτλ, δεν εγγυόμαστε ότι η επανενεργοποίηση του plug-in θα επαναφέρει το project χωρίς προβλήματα ή απώλειες δεδομένων. Για το λόγο αυτό συνιστάται η απενεργοποίηση του 4d-Fluents plug-in να γίνεται μόνο στη περίπτωση που ο χρήστης είναι σίγουρος πως δε θα το χρειαστεί άλλο στο project που δουλεύει.

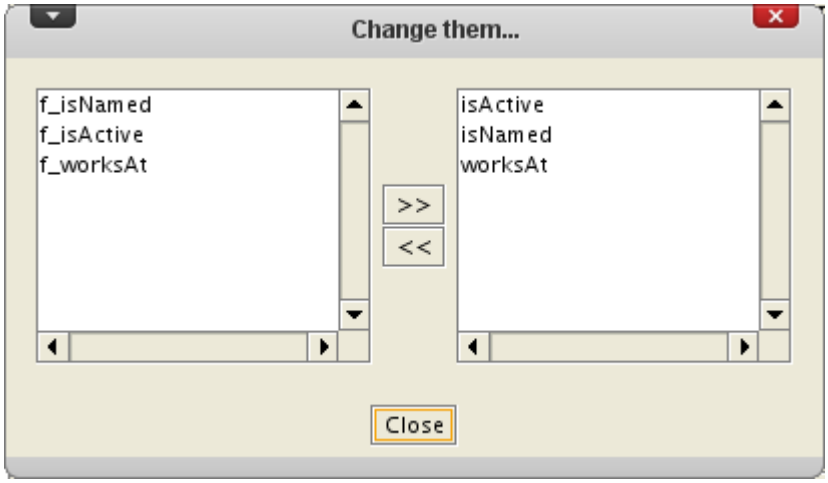
3.2.2 Menu item: *Property Wizard*



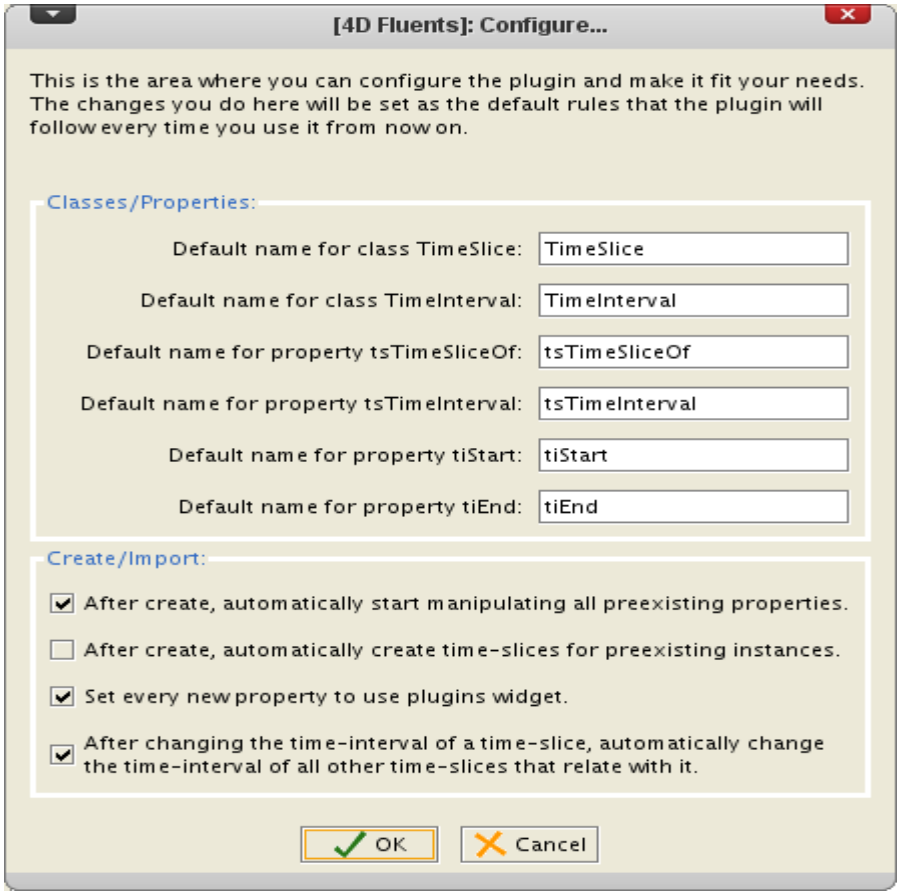
Το plug-in σχεδιάστηκε και υλοποιήθηκε με τρόπο που να εξυπηρετεί και τους χρήστες εκείνους που έχουν ήδη κατασκευάσει οντολογίες τεσσάρων διαστάσεων βασισμένοι στην πρόταση του paper της IBM. Αν και θα μιλήσουμε για την υλοποίηση αυτή πιο αναλυτικά σε άλλη ενότητα εδώ θα δούμε μια από τις ευκολίες που παρέχει το plug-in. Βάση του paper ο χρήστης που θέλει να σχεδιάσει μια οντολογία τεσσάρων διαστάσεων πρέπει για κάθε απλό γνώρισμα που δημιουργεί να δημιουργήσει και ένα fluent γνώρισμα μέσω του οποίου θα σχετίζει time-slices των domains που έχουν σχετιστεί από το πρώτο γνώρισμα. Με το menu-item *Property Wizard* δίνεται η δυνατότητα στον χρήστη να αντιστοιχίσει “απλά γνωρίσματα” με “fluent γνωρίσματα” ώστε το plug-in να αναλάβει την διαχείριση τους χωρίς να χαθούν δεδομένα. Μάλιστα κατά το άνοιγμα της παραπάνω φόρμας, το plug-in κάνει μια πρώτη

προσπάθεια να ξεχωρίσει αυτά που θεωρεί απλά γνωρίσματα από τα γνωρίσματα που συσχετίζουν time-slices.

Αν πάλι κάτι δεν πήγε καλά κατά τον υπολογισμό αυτό, με το κουμπί Change Me ανοίγει η παρακάτω φόρμα που βοηθάει τον χρήστη να διορθώσει τα λάθος ερμηνευμένα γνωρίσματα.



3.2.3 Menu item: *Configure*



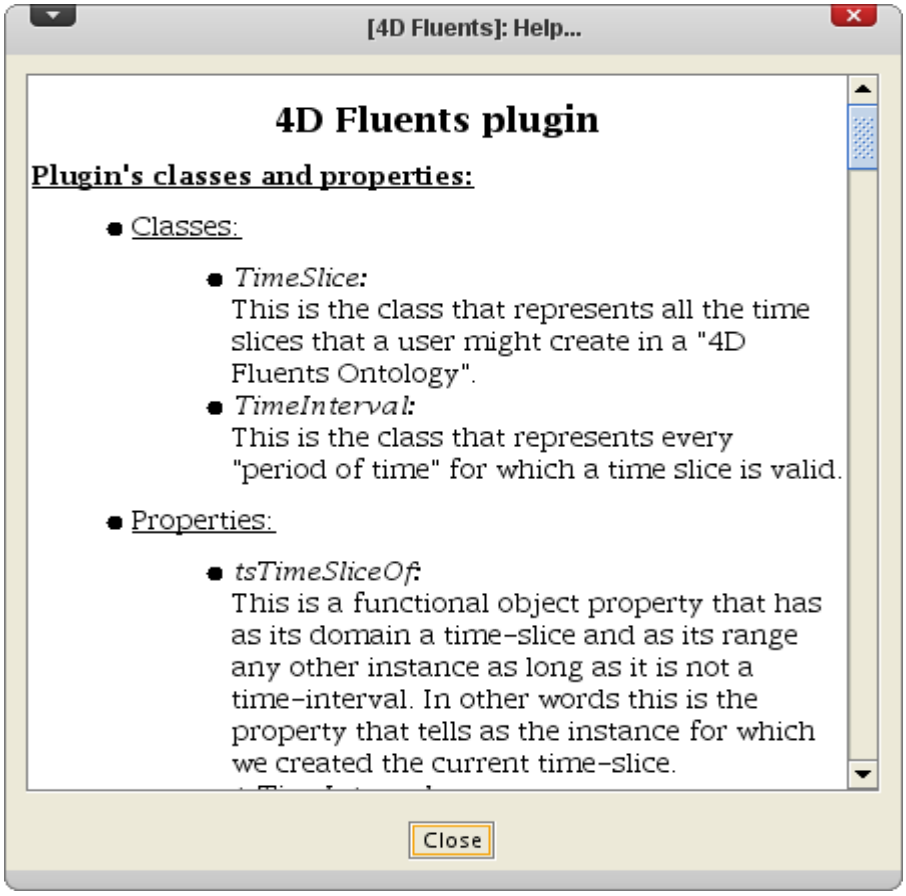
Το *Configure* δεν είναι τίποτα παραπάνω από ένα απλό dialog-box μέσω του οποίου ο χρήστης παραμετροποιεί το plug-in κατάλληλα ώστε να το προσαρμόσει στις ανάγκες και τις προτιμήσεις του.

Οι παράμετροι που παρέχει το plug-in μας χωρίζονται σε δυο μέρη:

- Το πρώτο αφορά τις κλάσεις και τα γνωρίσματα που αντιπροσωπεύουν τη θεωρία των οντολογιών τεσσάρων διαστάσεων και τα οποία το plug-in καλείται να προσθέσει στο project κατά την ενεργοποίηση του. Εξ ορισμού επιλέξαμε οι ονομασίες να είναι ίδιες με αυτές που προτείνονται στο paper της IBM αλλά επειδή ο κάθε χρήστης έχει τους δικούς του κώδικες ονοματοδωσίας, θεωρήσαμε πρέπον να δίνουμε τη δυνατότητα αλλαγής τους.
- Στο δεύτερο μέρος έχουμε τοποθετήσει μερικά check-boxes μέσω των οποίων ο χρήστης καθορίζει τη συμπεριφορά του plug-in σε συγκεκριμένες ενέργειες του.
  Το πρώτο check-box επιτρέπει ή αποτρέπει το plug-in από το να διατρέξει όλα τα γνωρίσματα του project που προυπήρχαν της ενεργοποίησης του και να τα μετατρέψει σε fluents ώστε να τα διαχειρίζεται όπως αρμόζει σε μια οντολογία τεσσάρων διαστάσεων.
  Το δεύτερο check-box επιτρέπει ή αποτρέπει το plug-in από το να δημιουργήσει time-slices για κάθε αντικείμενο που είχε δημιουργηθεί πριν την ενεργοποίηση του.
  Το τρίτο check-box επιτρέπει ή αποτρέπει το plug-in από το να θέτει τη διαχείριση κάθε καινούργιου γνωρίσματος στο γραφικό component που φτιάξαμε εμείς για το plug-in και το οποίο είναι κατάλληλο για τη παρουσίαση των συνδέσεων των time-slices.
  Το τέταρτο και τελευταίο check-box επιτρέπει ή αποτρέπει το plug-in από το να αλλάζει την χρονική περίοδο ισχύος ενός time-slice στη περίπτωση που αυτή άλλαξε σε time-slice που συνδέεται με το πρώτο μέσω fluent σχέσης.

Να σημειώσουμε εδώ πως σε περίπτωση που το plug-in είναι ήδη ενεργοποιημένο, οποιαδήποτε αλλαγή δε θα το επηρεάσει. Γενικά καλό είναι ο χρήστης να έχει παραμετροποιήσει το plug-in πριν αρχίσει να το χρησιμοποιεί στα projects του.

3.2.4 Menu item: *Help*



Με την επιλογή του menu-item *Help* το plug-in παρουσιάζει στον χρήστη μια απλή σελίδα η οποία δίνει τις εξής πληροφορίες:

- Plug-in's classes and properties: λίγα λόγια για τις κλάσεις και τα γνωρίσματα που προσθέτει το plug-in στο project, βασισμένα στο paper της IBM, ώστε ο χρήστης να έρθει σε μια πρώτη επαφή με τις οντολογίες τεσσάρων διαστάσεων.
- Enable the plug-in: λίγα λόγια για την ενεργοποίηση του plug-in η οποία και διαφέρει ανάλογα την κατάσταση του project (αν υπάρχουν ήδη ή όχι αντικείμενα κτλ).
- Disable the plug-in: λίγα λόγια για τη διαδικασία απενεργοποίησης του plug-in και τους κινδύνους της συνεχούς ενεργοποίησης - απενεργοποίησης.
- Load / Save a project with "4D Fluents ontology": λίγα λόγια για τη συμπεριφορά του plug-in κατά την αποθήκευση και φόρτωση ενός project στο οποίο έχει χρησιμοποιηθεί το plug-in μας.
- Configure the plug-in: επεξήγηση του menu-item *Configure* όπως είδαμε παραπάνω.

3.3 Ενεργοποίηση του plug-in

Η ενεργοποίηση του plug-in βασίζεται κατά κύριο λόγο στο στάδιο στο οποίο βρίσκεται το project αλλά και το είδος του project. Συγκεκριμένα τρία μπορεί να είναι τα στάδια: (α) ένα κενό project, (β) ένα project στο οποίο έχουν ήδη δημιουργηθεί κλάσεις, γνωρίσματα, αντικείμενα και σχέσεις αλλά χωρίς να μιλάμε για οντολογία τεσσάρων διαστάσεων και (γ) ένα project στο οποίο έχει υλοποιηθεί μια οντολογία τεσσάρων διαστάσεων βασισμένη στη θεωρία που παρουσιάσαμε και εμείς. Ανάλογα λοιπόν το στάδιο το plug-in ενεργεί διαφορετικά όσον αφορά τις προσθήκες που θα κάνει στο project.

3.3.1 Ενεργοποίηση σε άδειο project

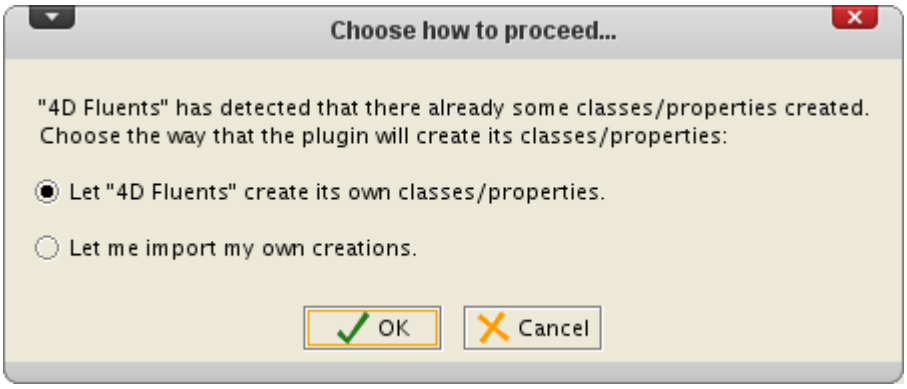
Η πιο απλή των περιπτώσεων. Μετά την επιλογή του Enabled το plug-in διατρέχει το project και αφού δει ότι είναι άδειο ακολουθεί τα εξής βήματα:

- Δημιουργεί τις δύο βασικές κλάσεις που θα χειρίζονται τις χρονικές περιόδους και τα time-slices. Επίσης για κάθε μια από τις κλάσεις αυτές δημιουργούνται και τα απαραίτητα γνωρίσματα μέσω των οποίων ο χρήστης θα μπορεί να συσχετίζει time-slices με αντικείμενα αλλων κλάσεων, time-slices με χρονικές περιόδους αλλά και να θέτει την αρχή και το τέλος μιας χρονικής περιόδου. Οι ονομασίες και των κλάσεων και των γνωρισμάτων είναι αυτές που ο χρήστης έθεσε μέσω του μενού που είδαμε στη προηγούμενη ενότητα (βλ: 3.2.2).
- Αλλάζει την κατάσταση των προσθηκών που μόλις έκανε σε μη editable ώστε ο χρήστης να μη μπορεί να τις σβήσει ή να τις αλλάξει όσο το plug-in είναι ενεργοποιημένο.
- Ενεργοποιεί τους μηχανισμούς του plug-in που σε κάθε νέα προσθήκη ή αλλαγή του χρήστη, στην οντολογία, τρέχουν αυτόματα ώστε να διατηρήσουν την οντολογία ως μια τεσσάρων διαστάσεων μειώνοντας έτσι τις ενέργειες που θα έπρεπε να εκτελέσει ο χρήστης αν δεν είχε το plug-in.

3.3.2 Ενεργοποίηση σε project με υπάρχουσες κλάσεις / γνωρίσματα / αντικείμενα

Υπάρχουν περιπτώσεις που ο χρήστης μπορεί να αποφασίσει να μετατρέψει μια υπάρχουσα οντολογία σε οντολογία τεσσάρων διαστάσεων. Για να μη χρειαστεί να χαθεί η μέχρι τώρα δουλειά του, δημιουργήσαμε έναν μηχανισμό που θα τον βοηθήσει να κάνει τις απαραίτητες προσθήκες στην οντολογία χωρίς να χάσει δεδομένα. Μετά την ενεργοποίηση του το plug-in

αναγνωρίζει ότι η οντολογία δεν είναι κενή και ξεκινάει την διαδικασία μετατροπής της οντολογίας με τον παρακάτω διάλογο:



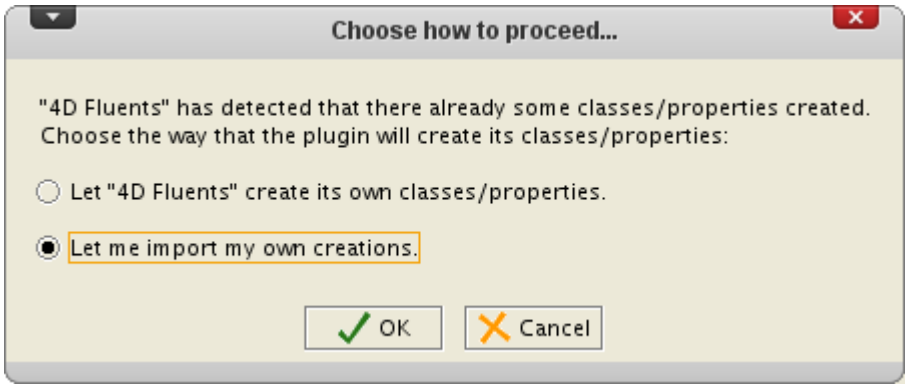
εδώ ο χρήστης επιλέγει το πρώτο radio-button το οποίο θα έχει ως αποτέλεσμα να εκτελεστούν τα τρία βήματα που είδαμε στην ενότητα 3.3.1 ακολουθούμενα από τα εξής:

- Το plug-in διατρέχει όλες τις κλάσεις που προυπήρχαν της ενεργοποίησης του και τις θέτει ως disjointed με τις κλάσεις που μόλις πρόσθεσε ώστε να αποτρέψουμε το ενδεχόμενο ένα αντικείμενο να ανήκει σε μια από τις κλάσεις του plug-in αλλά και σε μια από τις κλάσεις του χρήστη.
- Αν ο χρήστης έχει ενεργοποιημένο το πρώτο από τα checkbox που είδαμε στην ενότητα 3.2.3 τότε το plug-in για κάθε ένα από τα γνωρίσματα που προυπήρχαν θα φροντίσει να δημιουργήσει το αντίστοιχο fluent γνώρισμα που θα χειρίζεται τα timeslices του domain και του range του αρχικού γνωρίσματος.
- Αν ο χρήστης έχει ενεργοποιημένο το δεύτερο από τα checkbox που είδαμε στην ενότητα 3.2.3 τότε το plug-in θα αναλάβει την δημιουργία ενός timeslice για κάθε ένα από τα αντικείμενα που θα συναντήσει. Μάλιστα αν δυο αντικείμενα διαφορετικών κλάσεων έχουν συνδιαστεί μέσω γνωρίσματος και το γνώρισμα αυτό μετατραπεί σε fluent από το παραπάνω βήμα, το plug-in θα δημιουργήσει ένα timeslice για κάθε ένα από τα δυο αντικείμενα και θα τα “ενώσει” με το fluent γνώρισμα. Να σημειώσουμε εδώ πως τα timeslices που δημιουργούνται δεν έχουν κάποια χρονική περίοδο που τα χαρακτηρίζει. Αυτό αφήνεται στον χρήστη.

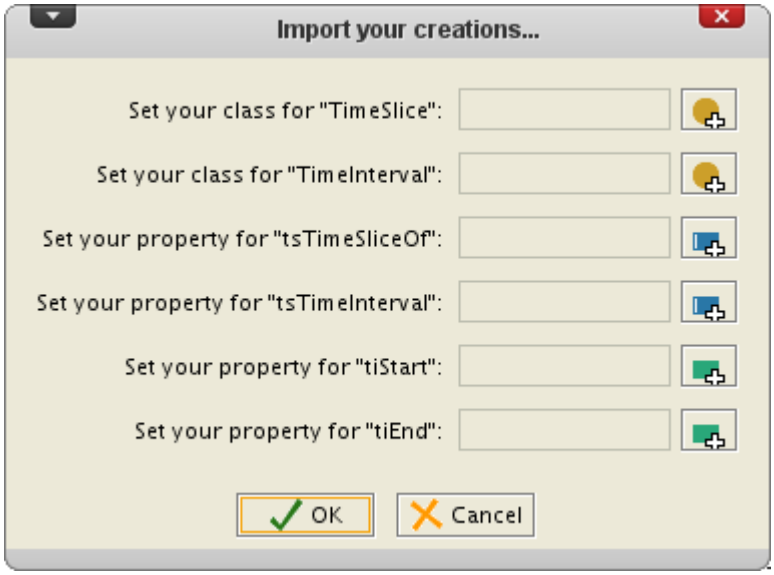
3.3.3 Ενεργοποίηση σε project με οντολογία τεσσάρων διαστάσεων

Η τρίτη των περιπτώσεων ενεργοποίησης του plug-in είναι και η πιο περίπλοκη γιατί η οντολογία πάνω στην οποία επιθυμούμε να κάνουμε αλλαγές έχει ήδη κλάσεις και γνωρίσματα που ο χρήστης χρησιμοποιεί για τη δημιουργία timeslices και fluent-σχέσεων. Για τον λόγο αυτό η ενεργοποίηση του plug-in γίνεται μέσα από μια σειρά διαλόγων ώστε (α) να μην χαθούν δεδομένα, (β) οι ρόλοι των κλάσεων και των γνωρισμάτων να διατηρηθούν και (γ) ο έλεγχος της οντολογίας να περάσει στο plug-in.

Όπως και παραπάνω κατά την ενεργοποίηση το plug-in θα δει ότι η οντολογία δεν είναι άδεια και θα ρωτήσει τον χρήστη πως θέλει να συνεχίσει. Στη περίπτωση μας όμως ο χρήστης θα επιλέξει το δεύτερο radio-button



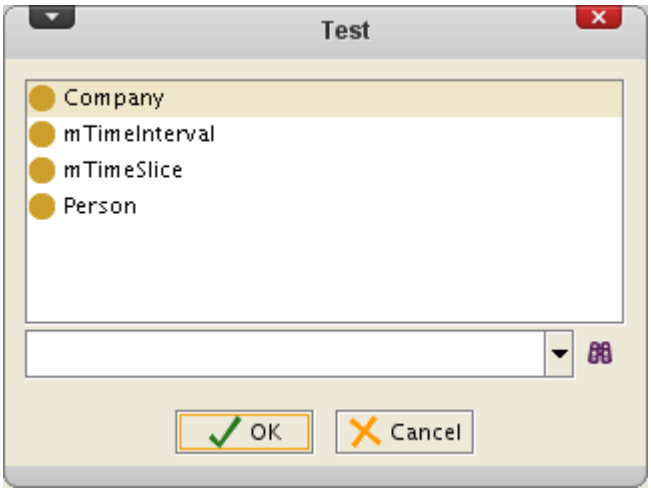
το οποίο με τη σειρά του θα ανοίξει την παρακάτω φόρμα και θα ξεκινήσει μια διαδικασία δυο σταδίων κατά την οποία ο χρήστης θα βοηθήσει το plug-in να αναγνωρίσει και ταξινομήσει κατάλληλα τα στοιχεία της οντολογίας.



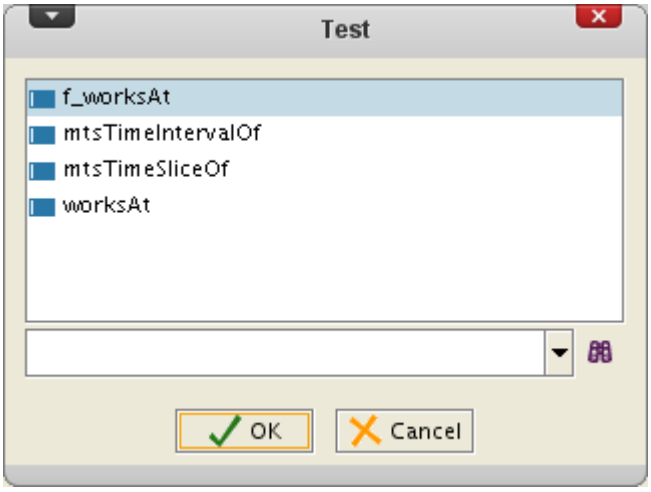
στάδιο 1: Δήλωση βασικών κλάσεων θεωρίας και των γνωρισμάτων τους

Με την παραπάνω φόρμα ο χρήστης θα δηλώσει στο plug-in ποιες από τις υπάρχουσες κλάσεις της οντολογίας χρησιμοποιήθηκαν για τη δημιουργία time-slices και χρονικών περιόδων. Ταυτόχρονα ο χρήστης καλείται να δηλώσει και ποια από τα γνωρίσματα που δημιούργησε αντιπροσωπεύουν τα γνωρίσματα που έχουμε ήδη αναφέρει και που ζητάει η θεωρία μας για να χαρακτηρίσει και συσχετίσει τα αντικείμενα των δυο βασικών της κλάσεων.

Φυσικά για κάθε μια από τις έξι επιλογές ανοίγει και ξεχωριστή φόρμα στην οποία δίνονται ως επιλογές μόνο οι δημιουργίες του χρήστη που ταιριάζουν στην κάθε αντιστοιχία. Έτσι, για παράδειγμα, στις δύο πρώτες επιλογές θα ανοίξει η παρακάτω φόρμα:



η οποία φέρνει μόνο τις κλάσεις της οντολογίας. Αντίστοιχα για τις δυο επόμενες η φόρμα θα περιέχει μόνο τα γνωρίσματα της οντολογίας που είναι τύπου object - property:



Στη περίπτωση που ο χρήστης δεν συμπληρώσει και τα έξι πεδία η διαδικασία του import θα σταματήσει. Επίσης η διαδικασία θα διακοπεί αν κάποιο από τα γνωρίσματα που δηλώθηκαν δεν πληρεί τις απαραίτητες προϋποθέσεις που έχει θέσει η θεωρία. Έτσι αν επιλεχθεί κάποιο γνώρισμα που το domain ή το range του δεν είναι η κατάλληλη κλάση, το plug-in θα ενημερώσει τον χρήστη και θα σταματήσει τη λειτουργία του.

Στην αντίθετη περίπτωση που ο έλεγχος ολοκληρωθεί επιτυχώς το plug-in θα συνεχίσει εκτελώντας τα τρία βήματα που είδαμε και στην ενότητα 3.3.2.

στάδιο 2: Διαχώριση και συσχέτιση απλών και fluent γνωρισμάτων

Το στάδιο αυτό είναι ουσιαστικά η διαδικασία που αναφέραμε στην ενότητα 3.2.2. Αρχικά το plug-in διαχωρίζει όλα τα γνωρίσματα της οντολογίας σε αυτά που έχουν domain και range απλές κλάσεις και σε αυτά που θεωρεί πλέον fluent εφόσον για domain / range βλέπει μια από τις δυο βασικές κλάσεις της θεωρίας.

Μετά τον πρώτο διαχωρισμό συνεχίζει ξεχωρίζοντας αυτά που είναι object-properties από τα datatype-properties και παρουσιάζει τις τέσσερις λίστες που έφτιαξε στον χρήστη με τον διάλογο που είδαμε παραπάνω. Εδώ ο χρήστης καλείται να δημιουργήσει ζεύγη “απλού γνωρίσματος” - “γνωρίσματος fluent” διαλέγοντας ένα γνώρισμα από την μια λίστα, ένα από τη διπλανή της και πατώντας το κουμπί Match. Το plug-in στο σημείο αυτό αναλαμβάνει να ελέγξει ότι το fluent γνώρισμα χρησιμοποιήθηκε ορθώς μέσα στην οντολογία, ότι δηλαδή (α) οι σχέσεις που δημιούργησε ήταν ανάμεσα σε time-slices των domain και range του άλλου επιλεγμένου γνωρίσματος και (β) ότι τα time-slices αυτά έχουν την ίδια χρονική περίοδο.

Αν όλοι οι έλεγχοι αποδειχθούν σωστοί τότε το fluent γνώρισμα μετονομάζεται με βάση τα πρότυπα του plug-in μας και παράλληλα αλλάζει και το εξ ορισμού γραφικό component που το παρουσιάζει στον χρήστη σε αυτό που εμείς κατασκευάσαμε.

Σε κάθε περίπτωση ο χρήστης ενημερώνεται με κατάλληλα μηνύματα στην περιοχή Error / Messages του διαλόγου που είδαμε.

Μετά και την ολοκλήρωση του σταδίου αυτού ο έλεγχος της οντολογίας έχει περάσει στο plug-in μας.

3.4 Λειτουργικότητα του plug-in

Στην προηγούμενη ενότητα είδαμε τους διαφορετικούς τρόπους με τους οποίους ενεργοποιείται το plug-in μας. Η λειτουργία του ωστόσο δε τελειώνει εκεί. Ο βασικός σκοπός του plug-in είναι να επιτρέψει στον χρήστη να δημιουργεί οντολογίες τεσσάρων διαστάσεων χωρίς να χρειαστεί να προσθέσει ή να αλλάξει ο ίδιος κάτι άμεσα στο επίπεδο του κώδικα και χωρίς να πρέπει να κάνει επιπλέον προσθήκες μέσω του προγράμματος. Η ιδέα είναι ο χρήστης να συνεχίσει να δημιουργεί τις κλάσεις και τα γνωρίσματα που επιθυμεί καθώς και να συσχετίζει τα αντικείμενα του όπως θα το έκανε σε μια απλή οντολογία. Για τον λόγο αυτό το plug-in παρακολουθεί τις κινήσεις του χρήστη και ανάλογα το είδος της αλλαγής χειρίζεται την οντολογία κατάλληλα δημιουργώντας τις απαραίτητες κλάσεις και γνωρίσματα καθώς και προσαρμόζοντας το γραφικό περιβάλλον του προγράμματος ώστε τα καινούργια δεδομένα να παρουσιάζονται σωστά και να μην παρερμηνεύονται. Έτσι η οντολογία διατηρείται καθαρή και συνεπής στα δεδομένα της. Αναλυτικότερα:

3.4.1 Δημιουργία κλάσεων

Από την στιγμή που το plug-in είναι ενεργό κάθε καινούργια κλάση που εισάγει ο χρήστης πρέπει αναγκαστικά να γίνει disjointed με τις δυο κλάσεις της θεωρίας ώστε να μην υπάρξει κάποια στιγμή αντικείμενο που προήλθε από τον συνδυασμό μιας εξ αυτών με κάποια τρίτη. Για τον λόγο αυτό μετά την δημιουργία μιας οποιαδήποτε κλάσης το plug-in αναλαμβάνει να προσθέσει στη λίστα των κλάσεων με τις οποίες είναι disjointed τις δυο κλάσεις της θεωρίας. Φυσικά η καινούργια κλάση προστίθενται κατόπιν και στις αντίστοιχες λίστες των κλάσεων της θεωρίας.

3.4.2 Δημιουργία γνωρισμάτων

Η απλή δημιουργία ενός γνωρίσματος, είτε object είτε datatype, δεν πυροδοτεί αρχικά κάποια ενέργεια. Όταν όμως ο χρήστης αλλάξει το domain του γνωρίσματος τότε το plug-in αλλάζει το project τόσο σε επίπεδο OWL όσο και σε επίπεδο γραφικού περιβάλλοντος:

Αρχικά αναλαμβάνει να δημιουργήσει για το γνώρισμα αυτό ένα αντίστοιχο fluent γνώρισμα μέσω του οποίου θα συνδέονται timeslices των αντικειμένων του domain και του range του αρχικού γνωρίσματος. Η ονομασία του δεύτερου γνωρίσματος είναι ο συνδυασμός του προθέματος *__4DF_* και του ονόματος του αρχικού γνωρίσματος. Έτσι αν για παράδειγμα δημιουργούσαμε ένα γνώρισμα ονόματι DrivesA, μετά τον ορισμό του domain του το plug-in θα δημιουργήσει ένα γνώρισμα με την ονομασία *__4DF_DrivesA* το οποίο για domain θα έχει την κλάση της θεωρίας που αντιπροσωπεύει τις χρονικές στιγμές των αντικειμένων (εξ ορισμού τιμή: TimeSlice) και για range την ίδια κλάση αν πρόκειται για object-property ή τον ίδιο τύπο δεδομένων με το αρχικό γνώρισμα αν πρόκειται για datatype-property.

Κατόπιν αλλάζει τα εξ' ορισμού widgets που έχουν ανατεθεί στα γνωρίσματα και συνδέει όλα τα object properties με το FDFObjectPropertyWidget - widget και όλα τα datatype properties με το FDFDatatypePropertyWidget - widget. Τα δύο widgets είναι μέρος του plug-in μας και ο ρόλος τους είναι να παρουσιάζουν στον χρήστη καλύτερα τις σχέσεις και τις τιμές που έχουν τα timeslices των αντικειμένων. Όσον αφορά τα καινούργια fluent γνωρίσματα που δημιουργήσαμε, σε αυτά δεν αντιστοιχείται κάποιο widget εφόσον δεν θέλουμε ο χρήστης να έχει άμεση επαφή με τον ορισμό τους ή να τους αλλάζει τις τιμές με τρόπο που δεν μπορεί το plug-in να ελέγξει.

3.4.3 Μετονομασία

Από τη στιγμή που στην οντολογία μας έχουμε, εκτός των κλάσεων του χρήστη, κλάσεις και γνωρίσματα που αντιπροσωπεύουν μια θεωρία και που πάνω σε αυτά έχουμε στηρίξει ολόκληρη την λειτουργία του plug-in θεωρήσαμε σοφό το plug-in να παρακολουθεί όλες τις μετονομασίες που κάνει ο χρήστης και σε μερικές περιπτώσεις να τις απαγορεύει. Συγκεκριμένα:

- αν ο χρήστης προσπαθήσει να μετονομάσει ένα fluent γνώρισμα τότε το plug-in θα επαναφέρει την αρχική του ονομασία.
- αν ο χρήστης μετονομάσει ένα γνώρισμα για το οποίο έχει ήδη δημιουργηθεί ένα αντίστοιχο fluent τότε το plug-in θα μετονομάσει κατάλληλα και το δεύτερο γνώρισμα για να υπάρχει συνέπεια ανάμεσα τους αλλά και για συνεχίσει το plug-in να λειτουργεί σωστά.
- οποιαδήποτε άλλη αλλαγή επιτρέπεται και μάλιστα οι τελικές ονομασίες των κλάσεων και των γνωρισμάτων “κλειδιών” φυλάσσονται σε συγκεκριμένο αρχείο όπως θα δούμε παρακάτω.

3.5 Το plug-in σε επίπεδο γραφικών

Κατά τη δημιουργία του plug-in δυο ήταν οι στόχοι μας. Τον πρώτο τον αναλύσαμε παραπάνω και αφορούσε τις αλλαγές του plug-in σε επίπεδο κώδικα. Ο δεύτερος αφορά την οπτική παρουσίαση του plug-in στον τελικό χρήστη η οποία θέλαμε και να παρουσιάζει σωστά τα δεδομένα της οντολογίας αλλά και να μη ξενίζει τον χρήστη ή να φαίνεται παράταιρη από το υπόλοιπο Protégé.

3.5.1 Widgets

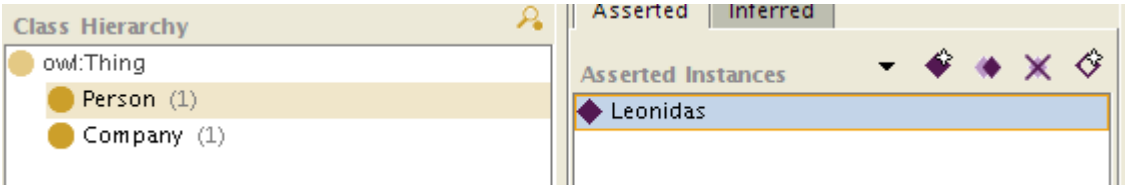
Στο Protégé κάθε γνώρισμα αντιπροσωπεύεται από ένα γραφικό component που ονομάζεται widget και μέσω του οποίου παρουσιάζεται η σχέση ή η τιμή ενός αντικειμένου για το συγκεκριμένο γνώρισμα. Σε μια οντολογία τεσσάρων διαστάσεων ένα αντικείμενο μπορεί για ένα γνώρισμα να έχει παραπάνω από μια σχέσεις ή τιμές. Τα default widgets του προγράμματος δεν ήταν σε θέση να παρουσιάσουν στον χρήστη μια 1 προς N σχέση. Για τον λόγο αυτό χρησιμοποιώντας το API και τις δυνατότητες που μας έδινε δημιουργήσαμε δικά μας widgets τα οποία βοηθούν τον χρήστη να δημιουργήσει και παρουσιάσει τα δεδομένα που επιθυμεί.

Έστω ότι έχουμε μια απλή οντολογία στην οποία μια κλάση ονόματι Person σχετίζει τα αντικείμενα της, μέσω του γνωρίσματος worksAt, με αντικείμενα της κλάσης Company. Στην περίπτωση που δεν υπάρχει το plug-in μας ο χρήστης θα έπρεπε να φτιάξει

- ένα αντικείμενο για την Person (πχ: Leonidas),
- ένα αντικείμενο για την Company (πχ: aCompany),
- ένα αντικείμενο για την κλάση TimeInterval (πχ 1/1/2009 με 1/1/2011) ώστε να οριοθετήσει τη χρονική περίοδο για τον οποία ισχύει μια σχέση και
- από δυο αντικείμενα για την κλάση TimeSlice τα οποία θα αντιπροσωπεύσουν τα αντικείμενα Leonidas (πχ: timeslice με ονομασία leonidas20092011) και aCompany (πχ: timeslice με ονομασία acompany20092011) για το διάστημα που φτιάξαμε.

Φυσικά για να συσχετιστούν όλα τα παραπάνω αντικείμενα ο χρήστης πρέπει να πάει σε κάθε ένα από τα γνωρίσματα τους και να θέσει τις αντίστοιχες τιμές για να καταλήξει στο τέλος με μια οντολογία η οποία, εκτός του ότι μπορεί να έχει κάποιο λάθος, δεν παρουσιάζει και σωστά την αρχική σχέση που ήθελε να δημιουργήσει:

ότι ένα timeslice του αντικειμένου Leonidas



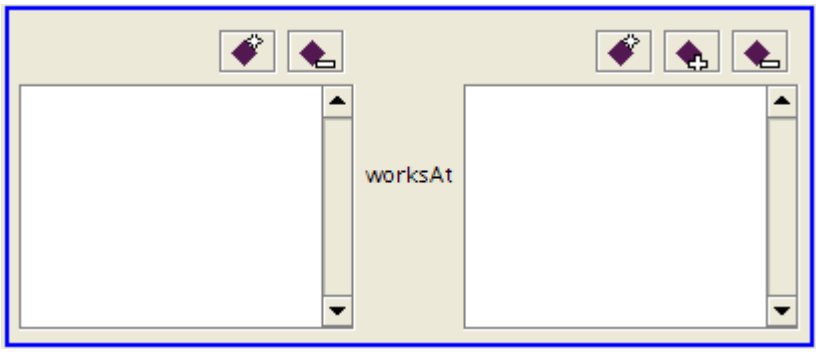
(εικόνα 11)

σχετίζεται με ένα timeslice του αντικειμένου aCompany

(εικόνα 12)

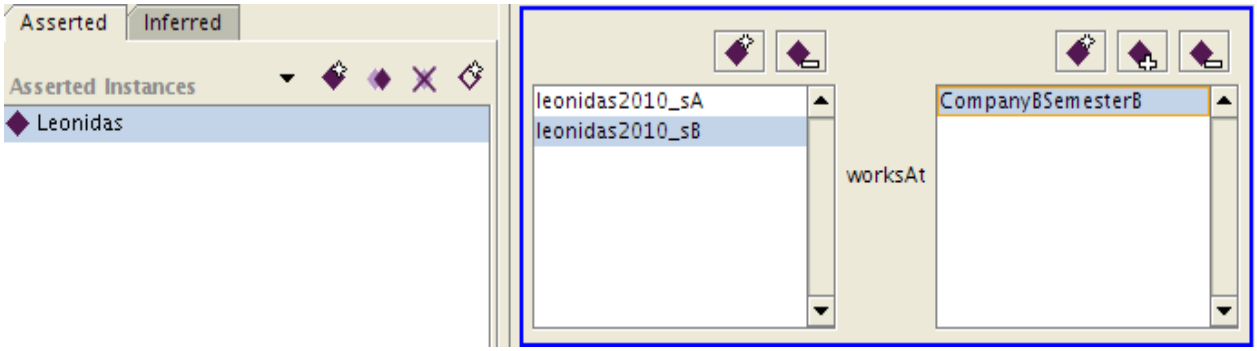
Στη περίπτωση όμως που στο Protégé έχει φορτωθεί το plug-in μας τότε οι κινήσεις που αναφέραμε παραπάνω εκτός του ότι περιορίζονται γίνονται και με τρόπο που αποτελεί μια συνεχή και φυσική ροή αποτρέποντας τον χρήστη από το να μπερδευτεί και να κάνει κάποιο λάθος.

Όπως και πριν έτσι και στη δικιά μας περίπτωση ο χρήστης έχει δημιουργήσει τα δύο αρχικά αντικείμενα των κλάσεων. Αυτή τη φορά όμως όταν πηγαίνει στο tab των αντικειμένων βλέπει το δικό μας widget



(εικόνα 13)

το οποίο (α) τον καθοδηγεί στο να δημιουργήσει τα timeslices που χρειάζονται και (β) κατασκευάζει αυτόματα οι απαραίτητες συσχετίσεις ώστε η οντολογία να γνωρίζει πια είναι τα timeslices ενός αντικειμένου καθώς και σε ποια χρονική περίοδο αναφερόμαστε κάθε φορά:



(εικόνα 14)

Το API του προγράμματος επιτρέπει στον προγραμματιστή να κατασκευάσει widgets περιορίζοντας τον μόνο όσον αφορά διαδικασίες που αφορούν την εκκίνηση του widget καθώς και της αλληλεπίδρασης με τον κώδικα σε OWL. Σε επίπεδο γραφικών ο προγραμματιστής έχει το ελεύθερο να χρησιμοποιήσει όποιο γραφικό component προσφέρει η java και του αρέσει.

Επιλέξαμε την λύση των λιστών γιατί είναι μια λύση καθαρή, ευκολονόητη και η πλέον κατάλληλη για την περίπτωση παρουσίασης “1 προς Ν” σχέσεων. Συγκεκριμένα δημιουργήσαμε ένα component με:

- δύο λίστες. Η αριστερή λίστα θα περιέχει τα timeslices του αντικειμένου της domain κλάσης που επιλέχθηκε ενώ για κάθε timeslice που επιλέγει ο χρήστης θα φορτώνονται στην δεξιά λίστα τα timeslices της range κλάσης με τα οποία έχει γίνει η σύνδεση.
- ένα σετ κουμπιών, για την κάθε λίστα, ώστε ο χρήστης να δημιουργεί ( ), προσθέτει ( ) και αφαιρεί ( ) timeslices από το ίδιο σημείο. Τα κουμπιά που επιλέξαμε είναι ακριβώς ίδια με αυτά που παρέχει το Protégé ώστε να υπάρχει ομοιομορφία με το υπόλοιπο πρόγραμμα ενώ διατηρήσαμε ίδιες και τις φόρμες που σηκώνει το πρόγραμμα για τις αντίστοιχες ενέργειες. Αξίζει να σημειώσουμε πως οποιαδήποτε και

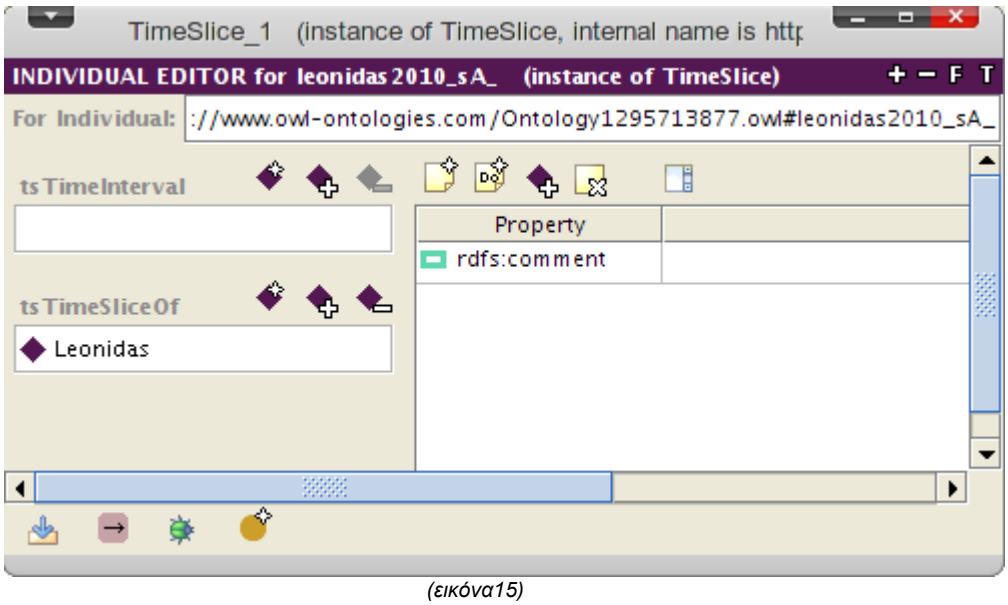
αν είναι η ενέργεια του χρήστη το plug-in, εκτός από την παρουσίαση στο component, αναλαμβάνει να προσθέσει / αφαιρέσει και τον κώδικα στο owl αρχείο της οντολογίας.
- την ονομασία της σχέσης στο κέντρο του, ανάμεσα στις δυο λίστες, ώστε ο χρήστης να νιώθει πως διαβάζει κάποιο κείμενο:

[timeslice] leonidas2010_sB **worksAt** *[timeslice]* CompanyBSemesterB

Η χρήση του widget του plug-in είναι πάρα πολύ απλή και για να το αποδείξουμε θα δούμε πως μπορεί ένας χρήστης να δημιουργήσει μερικά timeslices για το αντικείμενο Leonidas (βλ: εικόνα 11) και να καταλήξει σε μια σχέση με την μορφή που παρουσιάζεται στην εικόνα 14.

Αρχικά λοιπόν ο χρήστης επιλέγει το αντικείμενο για το οποίο θέλει να δημιουργήσει timeslices και σχέσεις με αυτά. Για το παράδειγμα μας επιλέγουμε το αντικείμενο Leonidas της κλάσης Person.

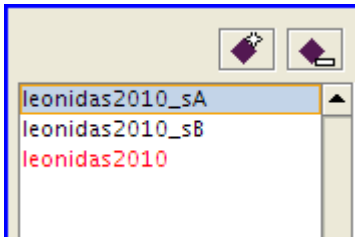
Αφού επιλέξει το αντικείμενο περνάει στο widget που αντιπροσωπεύει την σχέση που επιθυμεί, εδώ την worksAt, και πατάει το κουμπί δημιουργίας που βρίσκεται πάνω από την αριστερή λίστα. Με τον τρόπο αυτό θα δημιουργήσει ένα timeslice για το αντικείμενο συμπληρώνοντας την παρακάτω φόρμα



(εικόνα15)

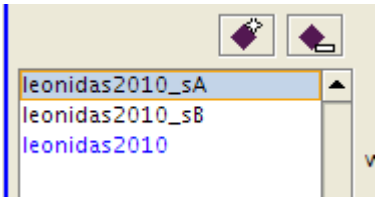
η οποία είναι η ίδια που χρησιμοποιείται σε όλο το Protégé για την εργασία αυτή. Η μόνη διαφορά στη περίπτωση μας είναι ότι έρχεται συμπληρωμένη με το πεδίο της σχέσης *tsTimeSliceOf* με τιμή το αντικείμενο για το οποίο φτιάχνεται το timeslice.

Στο σημείο αυτό αξίζει να σημειώσουμε πως ο χρήστης μπορεί να φτιάξει όσα timeslices επιθυμεί. Με την σειρά τους όλα θα φορτώνονται αυτόματα στην πρώτη λίστα του widget όπου ανάλογα την τιμή της σχέσης *tsTimeInterval* θα έχουν και διαφορετικό χρώμα ως ένδειξη της κατάστασης τους. Συγκεκριμένα, στη περίπτωση που ο χρήστης δεν θέσει κάποια χρονική περίοδο το timeslice παρουσιάζεται με κόκκινο χρώμα:



(εικόνα 16)

αν η χρονική περίοδος είναι ελλιπής, δηλαδή δεν έχει ένα από τα δυο χρονικά όρια, τότε παρουσιάζεται με μπλε Dexρώμα:



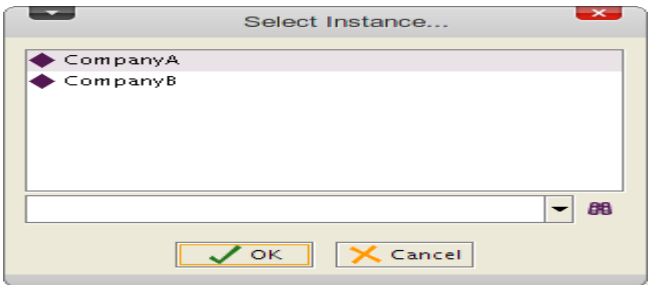
(εικόνα 17)

ενώ τέλος αν το timeslice είναι πλήρως συμπληρωμένο θα παρουσιαστεί στον χρήστη με μαύρο χρώμα.

Για το παράδειγμα μας ο χρήστης δημιουργεί δυο timeslices, ένα για κάθε εξάμηνο:

- το **leonidas2010_sA** αντιπροσωπεύει το πρώτο εξάμηνο και έχει για χρονική περίοδο το αντικείμενο **2010SemesterA** με αρχή την 1/1/2010 και τέλος την 30/6/2010.
- το **leonidas2010_sB** αντιπροσωπεύει το δεύτερο εξάμηνο και έχει για χρονική περίοδο το αντικείμενο **2010SemesterB** με αρχή την 1/7/2010 και τέλος την 31/12/2010.

Επόμενο βήμα είναι η δημιουργία timeslices για κάποιο από τα αντικείμενα που αντιπροσωπεύει την range-κλάση. Για το παράδειγμα μας η κλάση αυτή είναι η Company και σε αντίθεση με την περίπτωση δημιουργίας timeslice για την domain κλάση εδώ ο χρήστης δεν έχει την δυνατότητα να επιλέξει άμεσα με ποιο από τα αντικείμενα της θα ασχοληθεί. Για τον λόγο αυτό όταν πατήσει το κουμπί δημιουργίας timeslice της δεξιάς λίστας, ακολουθεί η παρακάτω φόρμα στην οποία έχει φορτωθεί μια λίστα με όλα τα αντικείμενα της range κλάσης που υπάρχουν στην οντολογία:



(εικόνα 17)

Ο χρήστης κάνει την επιλογή του, έστω ότι επέλεξε το CompanyB, και αμέσως μετά ανοίγει η φόρμα της εικόνας 15 για να συμπληρώσει τα στοιχεία του καινούργιου timeslice. Όπως και πριν η φόρμα έρχεται με συμπληρωμένα τα πεδία για τα οποία έχει τις τιμές. Έτσι το γνώρισμα tsTimeSliceOf παίρνει την τιμή CompanyB (αυτό επέλεγε ο χρήστης) και το πεδίο tsTimeInterval την τιμή 2010SemesterB (αυτή είναι η χρονική περίοδος του αντικείμενου leonidas2010_sB).

Έτσι με αυτή την απλή διαδικασία ο χρήστης κατάφερε να φτιάξει την σχέση που ήθελε και παρουσιάσαμε στην εικόνα 14 χωρίς να χρειαστεί να αλλάξει tab ή να πρέπει να συμπληρώσει πολλά πεδία γνωρισμάτων μόνος του.

Τέλος, όσον αφορά το widget μας, αξίζει να αναφέρουμε:.

την λειτουργία των δυο κουμπιών που υπάρχουν δίπλα στο κουμπί δημιουργίας:

- Το κουμπί προσθήκης αντικειμένου με το οποίο ο χρήστης μπορεί να συμπληρώσει την λίστα με ήδη κατασκευασμένα αντικείμενα. Συγκεκριμένα το plug-in ανοίγει μια λίστα με timeslices αντικειμένων της range κλάσης τα οποία έχουν ίδια χρονική περίοδο με το timeslice του αντικειμένου της domain κλάσης.
- Το κουμπί διαγραφής μέσω του οποίο ο χρήστης καθαρίζει τόσο την λίστα του widget όσο και την οντολογία σε επίπεδο κώδικα.

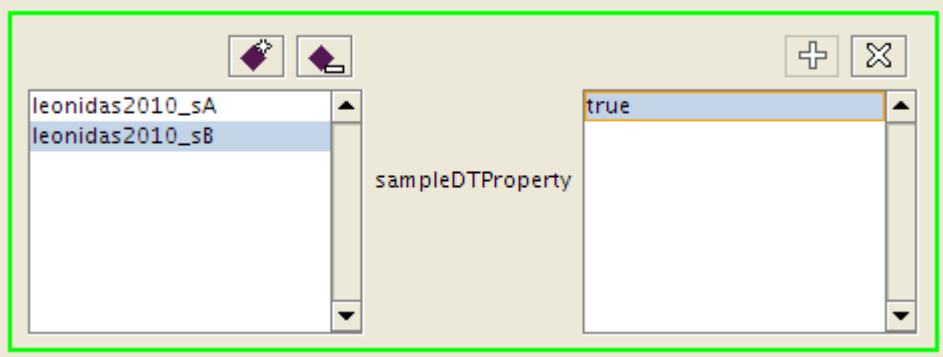
ότι με διπλό κλικ πάνω σε κάποιο στοιχείο της λίστας ανοίγει και η φόρμα της εικόνας 15 για να πραγματοποιήσει ο χρήστης τυχόν αλλαγές.

Ως αυτό το σημείο είδαμε μόνο την περίπτωση που ένα γνώρισμα είναι object property. Πρέπει να αναφέρουμε ότι στη περίπτωση που το γνώρισμα είναι datatype έχουμε φτιάξει ένα widget με ακριβώς την ίδια μορφή και λειτουργικότητα με το προαναφερθέν μόνο που η δεξιά του λίστα χειρίζεται τιμές τύπων δεδομένων και όχι αντικείμενα.

Έτσι, για παράδειγμα, στην περίπτωση που έχουμε ένα γνώρισμα boolean τύπου

(εικόνα 18)

το widget που θα παρουσιαστεί στον χρήστη θα είναι το παρακάτω

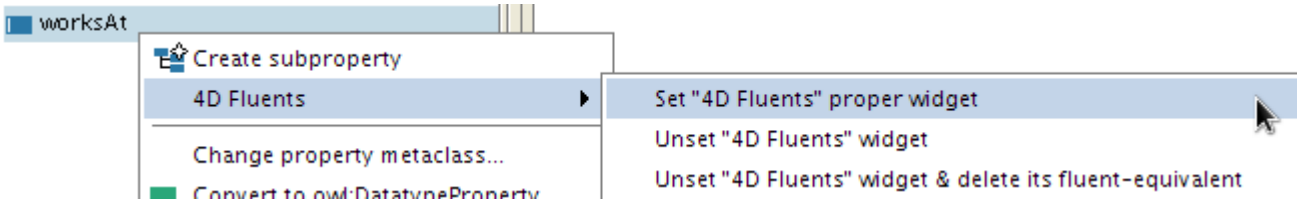


(εικόνα 19)

3.5.2 Pop-up μενού

Όπως είδαμε σε προηγούμενες ενότητες το plug-in προσφέρει στον χρήστη την δυνατότητα να δημιουργεί ή όχι αυτόματα fluent γνωρίσματα για κάθε καινούργιο γνώρισμα που δημιουργείται είτε άμεσα είτε μέσω της διαδικασίας import οντολογίας. Με τον ίδιο αυτόματο τρόπο σε κάθε fluent γνώρισμα το plug-in αναθέτει ένα από τα widgets που παρουσιάσαμε παραπάνω. Τι γίνεται όμως αν ο χρήστης δεν επιθυμεί να χειριστεί κάποιο από τα γνωρίσματα με το δικό μας widget;

Για την περίπτωση αυτή δημιουργήσαμε ένα pop-up μενού το οποίο εμφανίζεται με δεξί κλικ πάνω στα γνωρίσματα και δίνει στον χρήστη τις εξής δυνατότητες:



(εικόνα 20)

- **Set “4D Fluents” proper widget:** η εργασία αυτή αναλαμβάνει να δημιουργήσει ένα fluent γνώρισμα για το αρχικό γνώρισμα αλλά και να θέσει τη διαχείριση του τελευταίου στο widget του plug-in μας.
- **Unset “4D Fluents” widget:** η εργασία αυτή αναλαμβάνει να αλλάξει την διαχείριση του γνωρίσματος από το δικό μας widget και να την επαναφέρει στο εξ ορισμού που έρχεται μαζί με το Protégé.
- **Unset “4D Fluents” widget & delete its fluent-equivalent:** η εργασία αυτή κάνει ακριβώς ότι και η προηγούμενη συν ότι εντοπίζει το αντίστοιχο fluent γνώρισμα και το διαγράφει από την οντολογία.

Με το μενού αυτό ενισχύουμε την ελευθερία του χρήστη στο να ορίσει ακριβώς ποια γνωρίσματα θέλει να θεωρεί και να χειρίζεται ως fluent.

3.5.3 Project plug-in

Η ενότητα αυτή είναι αφιερωμένη στις γραφικές αλλαγές που κάνει το plug-in μας στο Protégé ωστόσο στην ενότητα 3.2 αναλύσαμε τη δομή ενός ολόκληρου μενού. Το μενού αυτό αποτελεί και αυτό μια από τις γραφικές προσθήκες που γίνονται στο πρόγραμμα με τη διαφορά ότι η δικιά του προσθήκη γίνεται αυτόματα και χωρίς την άμεση συγκατάθεση του χρήστη. Συγκεκριμένα, κάθε φορά που το Protégé ανοίγει τρέχει όλα του τα plug-ins και τους επιτρέπει να εκτελέσουν κάποιες αρχικές εργασίες. Η εργασία του δικού μας plug-in είναι να ελέγξει ότι το project που ανοίχθηκε είναι OWL και να

1. προσθέσει το μενού που είδαμε στη menu bar του προγράμματος
2. φορτώσει κάποια απαραίτητα δεδομένα για την ορθή λειτουργία του όσον αφορά το project αν αυτό δεν είναι καινούργιο.

3.6 Η διαδικασία φόρτωσης και αποθήκευσης των projects

Στην ενότητα 3.2.3 αναφερθήκαμε στη δυνατότητα που δίνει το plug-in στον χρήστη να ορίσει ποια θα είναι τα ονόματα των κλάσεων και των γνωρισμάτων που αντιπροσωπεύουν την θεωρία μας. Οι επιλογές του θα κρατιούνται σε ένα κεντρικό αρχείο, με την ονομασία 4dfluents.conf, μέσα στον φάκελο του plug-in (βλ: ενότητα 3.1) και αποτελούν την εξορισμού επιλογή σε κάθε νέο project που ανοίγεται από τον χρήστη.

Οι τιμές αυτές όμως είναι απλά προτεινόμενες. Το plug-in δεν θέτει κάποιον περιορισμό και έτσι ο χρήστης μπορεί ανά πάσα στιγμή να μετονομάσει τις κλάσεις και τα γνωρίσματα. Για τον λόγο αυτό για κάθε project κρατάε τις συγκεκριμένες τιμές σε ένα αρχείο ώστε το plug-in να γνωρίζει ποια από τις κλάσεις και τα γνωρίσματα πρέπει να χειρίζεται σαν αυτά της θεωρίας.

Συγκεκριμένα, κάθε φορά που ο χρήστης αποφασίζει να σώσει την δουλειά του ακολουθούμε τα εξής βήματα:

1. μέσα στον ίδιο φάκελο που το Protégé αποθηκεύει το αρχείο του project (με κατάληξη *.pprj*) δημιουργούμε ένα αρχείο με ονομασία ίδια με αυτή που έδωσε ο χρήστης για το project αλλά με κατάληξη *.fdfconfig*. Για παράδειγμα η ονομασία του project είναι **demo_ont.pprj** και ο χρήστης το αποθηκεύσει στη διαδρομή **/home/username/my_projects/** τότε στην ίδια διαδρομή το plug-in θα δημιουργήσει το **demo_ont.fdfconfig**.
2. στο καινούργιο αρχείο αποθηκεύουμε όλες τις τιμές που έχει δώσει ο χρήστης στα αντικείμενα της θεωρίας. Η δομή του κειμένου ακολουθεί τα πρότυπα που θέτει ένα

xml αρχείο ώστε να είναι ευκολοδιάβαστο τόσο από τον χρήστη όσο και από εφαρμογές τρίτων αν χρειαστεί.

Από τη στιγμή λοιπόν που το αρχείο υπάρχει στη διαδρομή του project, το plug-in αναλαμβάνει με το άνοιγμα του να φορτώσει τα δεδομένα του και έτσι να γνωρίζει κάθε φορά ποια από τα στοιχεία της οντολογίας πρέπει να αναγνωρίσει ως αυτά που αντιπροσωπεύουν την θεωρία και να συνεχίζει να ανταποκρίνεται κατάλληλα στις αλλαγές / προσθήκες της οντολογίας.

Κεφάλαιο 4

Επεκτάσεις

Ο βασικός σκοπός της εργασίας μας ήταν να βοηθήσει τον χρήστη να διαχειριστεί καλύτερα την χρονικά εξελισσόμενη πληροφορία είτε σε επίπεδο δημιουργίας αντικείμενων είτε σε επίπεδο παρουσίασης τους. Για τον λόγο αυτό όποιες τυχόν επεκτάσεις μπορούσαν να γίνουν θα ήταν κυρίως σε αυτούς τους δυο τομείς.

Για παράδειγμα θα μπορούσε να προστεθεί η δυνατότητα ο χρήστης να μπορεί να βλέπει, για μια σχέση, τα χρονικά στιγμιότυπα που δημιούργησε υπό τη μορφή δέντρου στο οποίο:

- η ρίζα θα είναι ένα αντικείμενο κλάσης
- το πρώτο επίπεδο παιδιών θα είναι τα χρονικά στιγμιότυπα του αντικειμένου, οργανωμένα κατά χρόνο
- ενώ το δεύτερο επίπεδο παιδιών θα αποτελείται από τα χρονικά στιγμιότυπα των αντικειμένων της range κλάσης της σχέσης.

Άλλη πιθανή επέκταση είναι η καλύτερη διαχείριση των χρονικών στιγμιότυπων ενός αντικειμένου με την προσθήκη μιας εργασίας που θα τα μελετάει και θα προτείνει στον χρήστη την δημιουργία καινούργιων ή την ένωση μερικών. Αν για παράδειγμα ένα στιγμιότυπο έχει χρησιμοποιηθεί σε μία μόνο σχέση τότε θα μπορούσε να αφαιρεθεί και στη θέση του να χρησιμοποιηθεί κάποιο άλλο που η χρονική περίοδος ισχύς του περιλαμβάνει και αυτό που αφαιρέσαμε. Επίσης θα ήταν αρκετά βολικό αν προσφέραμε στον χρήστη διεργασίες για την αυτόματη δημιουργία διαδοχικών χρονικών στιγμιότυπων ορίζοντας ημερομηνία αρχής, τέλος και βήμα. Για παράδειγμα ο χρήστης θα μπορούσε για το 2011 να δημιουργήσει αυτόματα 12 στιγμιότυπα, ένα για κάθε μήνα.

Τέλος μια ακόμα επέκταση θα ήταν η αυτόματη ονοματοδοσία ανάλογα τόσο την χρονική περίοδο όσο και το όνομα του αρχικού αντικειμένου. Φυσικά στις ρυθμίσεις του plug-in θα υπάρχουν πεδία στα οποία ο χρήστης θα ορίζει μερικώς την μορφή του ονόματος. Για παράδειγμα μπορεί να θέλει πρώτα το όνομα, μετά την χρονιά της περιόδου, μετά τον μήνα κτλ.

Βιβλιογραφία

[1] Semantic Web, http://en.wikipedia.org/wiki/Semantic_Web

[2] A reusable ontology for Fluents in OWL, 2005, Standford Univeristy, Welty, Fikes, Makarios

[3] Ontology in information science, http://en.wikipedia.org/wiki/Ontology_%28information_science%29

[4] Simple HTML Ontology Extensions, http://en.wikipedia.org/wiki/Simple_HTML_Ontology_Extensions

[5] Ontology Inference Layer, http://en.wikipedia.org/wiki/Ontology_Inference_Layer

[6] Outline Markup Language, http://en.wikipedia.org/wiki/OML

[7] RDF Schema, http://en.wikipedia.org/wiki/RDFS

[8] Defining N-ary Relations on the Semantics Web, 2006, http://www.w3.org/TR/swbp-n-aryRelations/

[9] Ontology editor, http://en.wikipedia.org/wiki/Ontology_editor

[10] Anzo Express, http://www.cambridgesemantics.com/products/anzo-express

[11] Νεologism, http://neologism.deri.ie/

[12] emftext, http://www.emftext.org/index.php/EMFText_Concrete_Syntax_Zoo_OWL2_Manchester

[13] OWLGrEd, http://owlgred.lumii.lv/

[14] Protégé overview, http://owlgred.lumii.lv/