ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υποστήριξη Σημασιολογικών Ερωτήσεων σε XML Βάσεις Δεδομένων

ΝΙΚΟΛΑΟΣ Ε. ΜΠΙΚΑΚΗΣ

Χανιά 2008

Περίληψη

Η ευρεία εξάπλωση και χρήση του **Διαδικτύου(Internet)**, η διάθεση και διακίνηση μεγάλου όγκου πληροφορίας μέσω αυτού συνδυασμένη με την ανάπτυξη πληροφοριακών συστημάτων, τεχνολογιών και προτύπων βασισμένων σε διαφορετικές ανάγκες και ιδιαιτερότητες, έχουν σαν αποτέλεσμα την εμφάνιση **ετερογένιας (Heterogeneity)** και τον περιορισμό των δυνατοτήτων του σημερινού **Παγκόσμιου Ιστού (Word Wide Web-WWW)**. Τα παραπάνω, καλείται να αντιμετωπίσει ο **Σημασιολογικός Ιστός (Semantic Web)** ο οποίος αποτελεί τη μεγαλύτερη προσπάθεια αυτόματης ενοποίησης συστημάτων, με σκοπό να συνεργάζονται διαλειτουργίκα σε παγκόσμιο επίπεδο.

Σήμερα, στο Διαδίκτυο το κυρίαρχο πρότυπο ανταλλαγής δεδομένων είναι η **XML (eXtensible Markup Language)**, η οποία βασίζεται στο ημιδομημένο μοντέλο δεδομένων με κυρίαρχη γλώσσα ερωτήσεων την **XQuery(XML Query Language)**. Η γλώσσα **XML Schema** χρησιμοποιείται για τον ορισμό της δομής των XML εγγραφών, υποστηρίζοντας πλούσιες δομές και τύπους δεδομένων. Έτσι, οι **XML** και **XML Schema** αποτελούν τη βάση της **συντακτικής** και **δομικής διαλειτουργικότητας (Structural and Syntactic Interoperability)** στο Διαδίκτυο.

Την ανάγκη για **σημασιολογική διαλειτουργικότητα (Semantic interoperability)** έρχεται να καλύψει ο Σημασιολογικός Ιστός, δίνοντας την δυνατότητα "μετασχηματισμού" του παγκόσμιου ιστού σε πλούσιες πλέον σημασιολογικές φόρμες με τη χρήση των σημασιολογικών τεχνολογιών. Η επικρατέστερη γλώσσα σημασιολογικών ερωτήσεων στο περιβάλλον αυτό είναι η γλώσσα **SPARQL (Simple Protocol and RDF Query Language)**, η οποία πρόσφατα αποτέλεσε σύσταση (Recommendation) του **W3C (The World Wide Web Consortium)**

Στα πλαίσια της παρούσας εργασίας, αντικείμενο έρευνας αποτέλεσε η επίτευξη διαλειτουργικότητας μεταξύ των ετερογενών Σημασιολογικού και XML περιβάλλοντος. Αποτέλεσμα αυτής, είναι η ανάπτυξη του πλαισίου (framework) **SPARQL2XQuery** το οποίο υποστηρίζει την **διαλειτουργικότητα** μεταξύ του **Σημασιολικού** και **XML περιβάλλοντος**, επιτρέποντας σημασιολογικές **SPARQL** ερωτήσεις να αποτιμώνται μέσω **XQuery** διεπαφών (interfaces) σε **XML Βάσεις Δεδομένων**. Της ανάπτυξης του πλαισίου προηγήθηκε θεωρητική τεκμηρίωση μεθόδων και αλγορίθμων, αυστηρή αναπαράσταση της σημασιολογίας και ανάλυση των σημασιολογικών ισοδυναμιών.

Η διαλειτουργικότητα μεταξύ SPARQL και XQuery, θα αποτελέσει βασικό δομικό συστατικό των αρχιτεκτονικών του Σημασιολογικού Ιστού.

Λέξεις Κλειδιά: SPARQL, XQuery, Query Translation, Semantic Interoperability, Semantic Data Integration, XML Databases/Repositories.

Στην Οικογένεια μου

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Σταύρο Χριστοδουλάκη για την επίβλεψη και την καθοδήγησή του κατά τη διάρκεια της εκπόνησης της παρούσας διπλωματικής εργασίας. Επιπλέον, θα ήθελα να τον ευχαριστήσω για τις σημαντικές εμπειρίες που μου προσέφερε κατά τη διάρκεια της εργασίας μου στο Εργαστήριο Διανεμημένων Πληροφοριακών Συστημάτων και Εφαρμογών.

Θα ήθελα επίσης να ευχαριστήσω τα υπόλοιπα μέλη της τριμελούς επιτροπής, τον επίκουρο καθηγητή κ. Αντώνη Δεληγιαννάκη και τον επίκουρο καθηγητή κ. Μιχάλη Λαγουδάκη.

Ιδιαίτερα θα ήθελα να ευχαριστήσω τον Νεκτάριο Γιολδάση για την επίβλεψη και την πολύτιμη βοήθειά του. Θα ήθελα ακόμα να ευχαριστήσω την Χρύσα Τσιναράκη που ήταν πάντα πρόθυμη να προσφέρει τη βοήθειά της.

Επίσης θα ήθελα να ευχαριστήσω τον Άκη, την Χρυσούλα και την Χρυσούλα για τις διορθώσεις και τις παρατηρήσεις στο παρόν κείμενο.

Τέλος, θα ήθελα να ευχαριστήσω όλο το προσωπικό, τους προπτυχιακούς και μεταπτυχιακούς φοιτητές του Εργαστηρίου Διανεμημένων Πληροφοριακών Συστημάτων και Εφαρμογών για τη συνεργασία τους.

Πίνακας Περιεχομένων

4 Το Πλαίσιο SPARQL2XQuery

5 Αντιστοιχήσεις

6 Ανακάλυψη Και Αποθήκευση Αντιστοιχήσεων

7 Κανονικοποίηση Σχηματομορφών Γράφων

12 Μετάφραση Σχηματομορφών Γράφων

13 Μετάφραση Τροποποιητών Ακολουθίας Λύσεων

14 Μετάφραση με Βάση την Μορφή των Ερωτήσεων

15 Μετάφραση των Ενσωματωμένων Τελεστών

16 Μετασχηματισμός Αποτελεσμάτων

17 Ανακεφαλαίωση, Αξιολόγηση & Μελλοντικές Επεκτάσεις

Κατάλογος Αλγορίθμων

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Γενικά

Η ευρεία εξάπλωση και χρήση του **διαδικτύου(Internet)**, η διάθεση και διακίνηση μεγάλου όγκου πληροφορίας μέσω αυτού, σε συνδυασμό με την ανάπτυξη πληροφοριακών συστημάτων, τεχνολογιών και προτύπων βασισμένων σε διαφορετικές ανάγκες και ιδιαιτερότητες, έχουν σαν αποτέλεσμα την εμφάνιση **ετερογένιας (heterogeneity)** και τον περιορισμό των δυνατοτήτων του σημερινού **παγκόσμιου ιστού (World Wide Web- WWW).** Τα παραπάνω καλείται να αντιμετωπίσει ο **Σημασιολογικός Ιστός (Semantic Web)**, ο οποίος αποτελεί τη μεγαλύτερη προσπάθεια αυτόματης ενοποίησης συστημάτων με σκοπό να συνεργάζονται διαλειτουργίκα σε παγκόσμιο επίπεδο.

Ο **Tim Berners-Lee**, ένας από τους "εφευρέτες" του **Παγκοσμίου Ιστού (World wide web- www)**, το 1998 μίλησε για το όραμα, (τον **Σημασιολογικό Ιστό**) ενός ιστού δεδομένων αυτόματα επεξεργάσιμων από τις εφαρμογές, βάσει του νοήματος και όχι της

μορφής της πληροφορίας[67]. Κάνοντας στην συνέχεια (Μάιος 2001) την πρώτη δημοσίευση για τον Σημασιολογικό Ιστό, στο **Scientific American Magazine** με το άρθρο "**The Semantic Web-A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities**" [69].

> ***The Semantic Web is a web of data, in some ways like a global database….***
>
> ***Tim Berners Lee -1998*** *[67]*

> ***The Semantic Web*** *is not about the meaning of English documents. It's not about marking up existing HTML documents to let a computer understand what they say. It's not about the artificial intelligence areas of machine learning or natural language understanding -- they use the word semantics with a different meaning.* ***It is about the data which currently is in*** *relational databases,* ***XML documents****, spreadsheets, and proprietary format data files,* ***and all of which would be useful to have access to as one huge database****.*
>
> ***Tim Berners Lee -June 2005*** *[68]*

Στο Διαδίκτυο σήμερα, το κυρίαρχο πρότυπο ανταλλαγής δεδομένων είναι η **XML (eXtensible Markup Language)** [39], η οποία βασίζεται στο ημιδομημένο μοντέλο δεδομένων, με κυρίαρχη γλώσσα ερωτήσεων την **XQuery(XML Query Language)**[6][7][9]. Η γλώσσα **XML Schema** [19] [20] χρησιμοποιείται για τον ορισμό της δομής των XML εγγράφων, υποστηρίζοντας πλούσιες δομές και τύπους δεδομένων. Λόγω των δυνατοτήτων δόμησης που παρέχει η γλώσσα XML Schema και του κεντρικού ρόλου που παίζει κατά την ανταλλαγή δεδομένων στο Διαδίκτυο, σημαντικά πρότυπα περιγραφής δεδομένων και μεταδεδομένων(metadata) για πολλές διαφορετικές περιοχές εφαρμογών έχουν εκφραστεί στη γλώσσα XML Schema, συμπεριλαμβανομένων προτύπων στην περιοχή των πολυμέσων, προτύπων ηλεκτρονικής εκπαίδευσης (e-learning), προτύπων για Ψηφιακές Βιβλιοθήκες (Digital Libraries) κ.α. Έτσι, οι **XML** και **XML Schema** αποτελούν τη βάση της **συντακτικής** και **δομικής διαλειτουργικότητας (Structural and Syntactic Interoperability)** στο Διαδίκτυο.

Την ανάγκη για **σημασιολογική διαλειτουργικότητα (Semantic interoperability)** έρχεται να καλύψει ο Σημασιολογικός Ιστός, δίνοντας την δυνατότητα "μετασχηματισμού" του παγκόσμιου ιστού σε πλούσιες πλέον σημασιολογικές φόρμες με τη χρήση των σημασιολογικών τεχνολογιών. Καθοριστικό ρολό στην σημασιολογική διαλειτουργικότητα έχουν οι **Οντολογίες (Ontologies)**. ***Οντολογία*** *είναι μια – πιθανώς μη πλήρης – αξιωματική διατύπωση (axiomatization) μιας σύλληψης εννοιών (conceptualization) που εκφράζει τη συναίνεση (consensus) μιας ή περισσότερων κοινοτήτων χρηστών για τη σημασιολογία (semantics) συγκεκριμένων εννοιών (concepts)* [70]. Οι οντολογίες αναπαρίστανται με τη χρήση γλωσσών περιγραφής οντολογιών, στις οποίες τα τελευταία χρόνια το κυρίαρχο πρότυπο είναι η **OWL (Web Ontology Language)** [1][21][22]

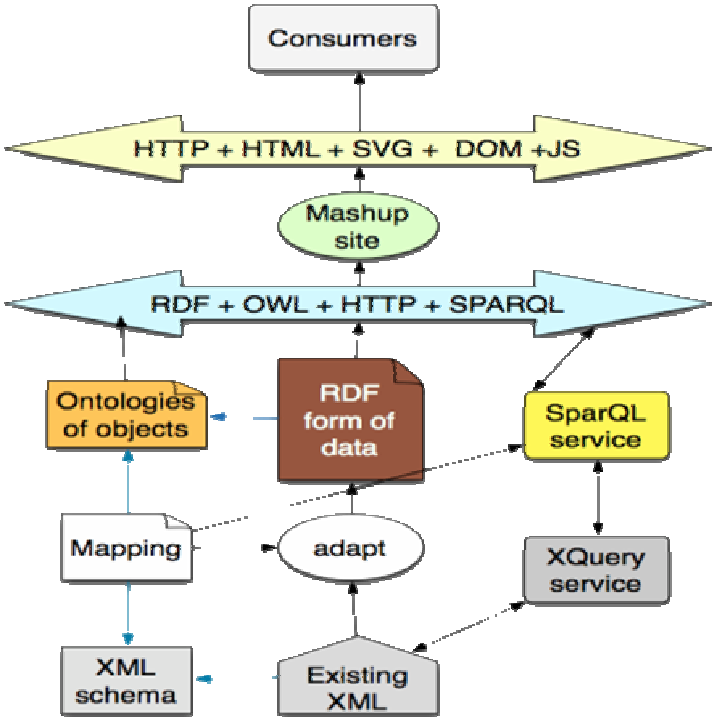
Τα δεδομένα στον Σημασιολογικό Ιστό περιγράφονται με την χρήση της **RDF (Resource Description Framework)** σύνταξης , ακολουθώντας τις έννοιες που ορίζουν οι οντολογίες. Για τα RDF δεδομένα έχουν αναπτυχθεί πολλές γλώσσες ερωτήσεων, με κυρίαρχη την **SPARQL(Simple Protocol and RDF Query Language)** [5] η οποία αποτελεί σύσταση **(Recommendation)** του **W3C (The World Wide Web Consortium)** [71] από τον Ιανουάριο του 2007.

Από τους βασικότερους στόχους του Σημασιολογικού Ιστού, είναι η ενοποίηση όλων των πηγών της πληροφορίας και η δυνατότητα διαχείριση της με έναν ενιαίο τρόπο. Για την πραγματοποίηση του είναι απαραίτητη η εξεύρεση λύσεων, για την διαχείριση της πληροφορίας που βρίσκεται αποθηκευμενή σε παραδοσιακές πηγές (legacy sources) όπως σχεσιακές βάσεις, αντικειμενοστραφείς βάσεις, XML βάσεις κ.α. και η ενοποίηση της με την πληροφορία του Σημασιολογικού Ιστού.

Η ανάγκη για διαλειτουργικότητα μεταξύ των σημασιολογικών τεχνολογιών (OWL , RDF/S , SPARQL) και των "παραδοσιακών τεχνολογιών" (XML Schema , XML , XQuery , SQL) είναι άμεση και καθοριστική για την επίτευξη του οράματος του Σημασιολογικού Ιστού.

Όπως έχει αναφερθεί, σημαντικά πρότυπα περιγραφής δεδομένων και μεταδεδομένων(metadata) για πολλές διαφορετικές περιοχές εφαρμογών έχουν εκφραστεί στη γλώσσα XML Schema, δημιουργώντας την ανάγκη δυνατότητας πρόσβασης και διαχείρισης των XML δεδομένων από τον Σημασιολογικό Ιστό. Η δυνατότητα αυτή επιτυγχάνεται, με την αντιστοίχηση των XML και σημασιολογικών πληροφοριών, με την επίτευξη διαλειτουργικότητα μεταξύ των γλωσσών ερωτήσεων SPARQL και XQuery, καθώς και με τον μετασχηματισμό της XML πληροφορίας.

Τα παραπάνω θα αποτελέσουν βασικά δομικά συστατικά των αρχιτεκτονικών του Σημασιολογικού Ιστού. Όπως είναι ευδιάκριτο και από τις τελευταίες σχεδιαστικές πρότασης του W3C και συγκεκριμένα του Tim Berners Lee, με την πρόταση του για το "**Double Bus Architecture**" Εικόνα 1.1 , η οποία δημοσιεύτηκε πρώτη φορά τον Μάρτιο του 2008 μέσω του προσωπικού του Blog.



Εικόνα 1.1 **The Double Bus Architecture**

1.2 Συνεισφορά Εργασίας

Η παρούσα εργασία αποτελεί μέρος της ευρύτερης έρευνας που διεξάγεται από το εργαστήριο Διανεμημένων πληροφοριακών Συστημάτων και Εφαρμογών Πολυμέσων – MUSIC στην περιοχή της Σημασιολογικής Διαλειτουργικότητα και Σημασιολογικής Ολοκλήρωσης Δεδομένων (Semantic Data Integration).

Στα πλαίσια της παρούσας εργασίας αντικείμενο έρευνας αποτέλεσε η επίτευξη διαλειτουργικότητας μεταξύ των ετερογενών Σημασιολογικού και XML περιβάλλοντος. Αποτέλεσμα αυτής, είναι η ανάπτυξη του πλαισίου (framework) **SPARQL2XQuery**, το οποίο υποστηρίζει την **διαλειτουργικότητα** μεταξύ του **Σημασιολικού** και **XML περιβάλλοντος**, επιτρέποντας σημασιολογικές **SPARQL** ερωτήσεις να αποτιμώνται μέσω **XQuery** διεπαφών (interfaces) σε **XML βάσεις δεδομένων**. Της ανάπτυξης του πλαισίου προηγήθηκε θεωρητική τεκμηρίωση μεθόδων και αλγορίθμων, αυστηρή αναπαράσταση της σημασιολογίας και απόδειξης των σημασιολογικών ισοδυναμιών.

Πιο συγκεκριμένα η συνεισφορά της παρούσα εργασίας είναι :

- Ανάπτυξη μια **γενικής μεθοδολογία** και **αλγορίθμων**, για την **μετάφραση SPARQL** ερωτήσεων, σε **σημασιολογικά ισοδύναμες XQuery**, κατά την οποία επιτεύχθηκαν τα εξής :

  α) Η διαδικασία μετάφρασης είναι ανεξάρτητη από τον **τρόπο ορισμού** και **αποθήκευσης** τον **αντιστοιχήσεων** (**mappings**) μεταξύ της οντολογίας και του XML σχήματος

  β) Μετάφραση **όλων των πιθανών ερωτήσεων**, καλύπτοντας όλους τους δυνατούς συνδυασμούς της γραμματικής της γλώσσας SPARQL.

  γ) **Αυστηρή** τήρηση της σημασιολογίας της γλώσσας κατά την διαδικασία της μετάφρασης.

  δ) Οι **ακολουθίες λύσεων** που προκύπτουν από το μεταφρασμένο XQuery ερώτημα **είναι οι επιθυμητές** και δεν απαιτούν κάποια περεταίρω επεξεργασία (από API ή Software), επομένως **ανεξαρτησία** από το query engine και το περιβάλλον εκτέλεσης της XQuery ερώτησης.

  ε) Ανάπτυξη **εύκολα κατανοητής** διαδικασίας και αλγορίθμων μετάφρασης.

  ζ) Παραγωγή όσο το δυνατόν **μικρότερων** και **λιγότερο πολύπλοκων** εκφράσεων(expressions) XQuery.

  η) Ανάπτυξη και σύνταξη των μεταφρασμένων ερωτήσεων με τρόπο ώστε οι "**αντιστοιχίες**" μεταξύ των δυο ερωτήσεων(SPARQL-XQuery) και ο **τρόπος μετάφρασης** να γίνονται **εύκολα αντιληπτά**.

  θ) Σε συνδυασμό με όλα τα παραπάνω όσο το δυνατόν **βελτιστοποίηση** (Optimization) των XQuery ερωτήσεων.

  ι) Η παραγωγή **μιας και μόνο XQuery** ερώτησης για κάθε μία SPARQL ερώτηση που δίνεται προς μετάφραση χωρίς την εκτέλεση ενδιάμεσων ερωτήσεων.

- **Συνεργασία** με το πλαίσιο **XS2OWL**[17], το οποίο παράγει OWL οντολογίες από XML σχήματα. Στην περίπτωση αυτή, το πλαίσιο **SPARQL2XQuery** πραγματοποιεί την **ανακάλυψη** (discover) και την **αυτόματη παραγωγή** και **αποθήκευση** των **αντιστοιχήσεων**. Με την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων επιτυγχάνεται μια **πλήρως αυτοματοποιημένη διαδικασία**, χωρίς να είναι απαραίτητη η παρέμβαση ανθρώπινου παράγοντα. Επίσης επιτυγχάνεται η **πλήρης αντιστοίχηση** όλων των στοιχείων τόσο της οντολογίας όσο και του XML σχήματος, **με αντιστοιχήσεις απολύτως σημασιολογία ορθές**, **χωρίς ύπαρξη αβεβαιότητα (Uncertainty)** και

πιθανότητας σφάλματος κατά την δημιουργία τους, προβλήματα που εμφανίζονται στην περίπτωση "χειροκίνητου"(manual) ορισμού των αντιστοιχήσεων από εξειδικευμένο χρήστη.

- **Μετασχηματισμός** των **αποτελεσμάτων** που προκύπτουν από τις XQuery ερωτήσεις, σύμφωνα με την XML μορφή των SPARQL αποτελεσμάτων(SPARQL Query Results XML Format), που προτείνεται από το W3C [56].

- Υπηρεσία Διαδικτύου (Web Service) η οποία παρέχει την δυνατότητα μετάφρασης SPARQL ερωτήσεων σε XQuery.

Η έμμεση συνεισφορά της παρούσας εργασίας με την υποστήριξη σημασιολογικών SPARQL ερωτήσεων σε XML βάσεις δεδομένων είναι:

- Έ**ρευνα** των **σημασιολογικών ισοδυναμιών** και της **διαδικασίας μετάφρασης** των δυο γλωσσών ερωτήσεων SPARQL και XQuery. Καθώς από όσο γνωρίζουμε δεν υπάρχει κάποια σχετική εργασία η οποία να έχει ασχοληθεί είτε με την αντιστοιχία, είτε με την μετάφραση, είτε με την σημασιολογική ισοδυναμία, μεταξύ των δυο γλωσσών ερωτήσεων, γεγονός το οποίο κάνει την παρούσα εργασία **καινοτόμα**.
- Επίτευξη **διαλειτουργικότητας των εφαρμογών** του Σημασιολογικού και XML περιβάλλοντος
- **Δημοσίευση** και **διαχείριση** αποθηκεμένων XML δεδομένων από τον Σημασιολογικό Ιστό μέσω της γλώσσας σημασιολογικών ερωτήσεων SPARQL.
- Η τελικοί χρήστες εκφράζουν σημασιολογικές ερωτήσεις (semantic-based queries), οι οποίες είναι **ποιο κατανοητές** και ποιο **κοντά στην διαίσθηση** τους, έναντι των βασισμένων σε δομή ερωτήσεων (structure-based queries), όπως η γλώσσα ερωτήσεων XQuery.
- Δυνατότητα ερωτήσεων **βασισμένων** σε **τύπους δεδομένων (data types)**, το οποίο δεν είναι εφικτό με την γλώσσα ερωτήσεων XQuery. Όπως για παράδειγμα, επέστεψε τα δεδομένα τα οποία είναι τύπου Person.
- Δυνατότητα ερωτήσεων βασισμένων σε **ιεραρχίες τύπων δεδομένων (data types hierarchies)**, το οποίο επίσης δεν εφικτό με την γλώσσα ερωτήσεων XQuery. Όπως για παράδειγμα, επέστεψε τα δεδομένα τα οποία ο τύπος τους είναι "πιο εξειδικευμένος" από τον τύπου Person. (Εκμεταλλευόμενοι τις IS-A σχέσεις οι οποίες προσφέρονται από τις

οντολογίες, παρόμοια σχέση υποστηρίζεται και από το xml schema μέσω της δήλωσης extension, όμως δεν μπορεί να γίνει "εκμετάλλευση" της από την γλώσσα XQuery)

1.3 Δομή Εργασίας

Στο **δεύτερο κεφάλαιο** θα γίνει η ανάλυση των πρότυπων, των προδιαγραφών και των τεχνολογιών που χρησιμοποιήθηκαν κατά την διάρκεια της παρούσας εργασίας.

Στο **τρίτο κεφάλαιο** θα γίνει μια αναφορά σε σχετικές εργασίες που έχουν πραγματοποιηθεί, όπως και μια εισαγωγή στον ευρύτερο ερευνητικό τομέα στον οποίο εντάσσονται, την Σημασιολογική Ολοκλήρωση Δεδομένων (Semantic Data Integration).

Στο **τέταρτο κεφάλαιο** του κειμένου, αναλύεται η ανάπτυξη του πλαισίου SPARQL2XQuery, παρουσιάζεται η γενική αρχιτεκτονική στην οποία εντάσσεται το πλαίσιο, αναλύεται ο τρόπος ανάπτυξης τον επιμέρους στοιχείων λογισμικού (software Component) του και οι μεταξύ τους διασυνδέσεις και αλληλεξαρτήσεις. Όπως επίσης και η διασύνδεση και αλληλεπίδραση του με άλλα συστήματα και χρήστες.

Στο **πέμπτο κεφάλαιο** του κειμένου, παρουσιάζεται ένας αφηρημένος τρόπος αναπαράστασης των αντιστοιχήσεων μεταξύ οντολογίας και Xml σχήματος, ορίζονται οι αντιστοιχίες στο επίπεδο των δυο γλωσσών. Τέλος ορίζεται ένας αριθμός από τελεστές οι οποίοι εφαρμόζονται σε σύνολα μονοπατιών.

Στο **έκτο κεφάλαιο** του κειμένου, παρουσιάζεται η διαδικασία της ανακάλυψης(discovery) και του αυτόματου τρόπου παραγωγής των αντιστοιχήσεων, καθώς και η μορφή με την οποία αυτές αποθηκεύονται.

Στο **έβδομο κεφάλαιο** του κειμένου, αναλύεται η διαδικασία "Κανονικοποίηση Σχηματομορφών Γράφων" η οποία πραγματοποιεί την κανονικοποίηση της σχηματομορφής γράφου της SPARQL ερώτησης. Ορίζονται οι κανόνες ισοδυναμίας οι οποίοι χρησιμοποιούνται για την κανονικοποίηση και η κανονικοποιημένη γραμματική η οποία προκύπτει με χρήση των κανόνων αυτών και re-writing τεχνικών.

Στο **όγδοο κεφάλαιο** του κειμένου, αναλύεται η διαδικασία "Προσδιορισμός τύπων μεταβλητών" η οποία προσδιορίζει τον τύπο των μεταβλητών που περιέχονται στην SPARQL ερώτηση. Ορίζονται οι τύποι των μεταβλητών, οι κανόνες προσδιορισμών τους και η συσχέτιση τους με την μορφής των αποτελεσμάτων.

Στο **ένατο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Επεξεργασία των Οντο τριπλέτων" στην οποία αναλύεται ο τρόπος χειρισμού των ερωτήσεων που περιέχουν Οντο τριπλέτες και στόχος της είναι η σύνδεση(bind) των μεταβλητών με μονοπάτια του XML εγγράφου, που προκύπτουν από τις Οντο τριπλέτες.

Στο **δέκατο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Σύνδεση Μεταβλητών" (Variables Binding), η οποία εφαρμόζεται σε βασικές σχηματομορφές γράφων (Basic graph pattern), δηλαδή σε ακολουθίες από τριπλέτες σχηματομορφών, ώστε να επιτευχθεί η σύνδεση των μεταβλητών που περιέχονται στο BGP με μονοπάτια του XML εγγράφου.

Στο **ενδέκατο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Μετάφραση βασικών σχηματομορφών γράφων" και ο αλγόριθμος BGP2XQuery ο οποίος την πραγματοποιεί. Η διαδικασία δέχεται ως είσοδο μια βασική σχηματομορφή γράφου και παράγει ως έξοδο XQuery εκφράσεις σημασιολογικά ισοδύναμες με την βασική σχηματομορφή γράφου της εισόδου

Στο **δωδέκατο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Μετάφρασης σχηματομορφών γράφων" και ο αλγόριθμος GP2XQuery ο οποίος την πραγματοποιεί. Χρησιμοποιώντας τον αλγόριθμο BGP2XQuery για την μετάφραση των βασικών σχηματομορφών γράφων η διαδικασία μετάφρασης των σχηματομορφών γράφων επιτυγχάνεται η μετάφραση του "βασικότερου" και πιο πολύπλοκου μέρους μιας SPARQL ερώτησης, καθώς αυτή αντιστοιχεί στην μετάφραση της where δομής της ερώτησης.

Στο **δέκατο τρίτο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Μετάφρασης των τροποποιητών της ακολουθίας λύσεων". Παρουσιάζεται η διαδικασία μετάφραση των τροποποιητών: DISTINCT, REDUCE, LIMIT, OFFSET και ORDERBY.

Στο **δέκατο τέταρτο κεφάλαιο** του κειμένου, περιγράφεται η διαδικασία "Μετάφρασης των ερωτήσεων με βάση την μορφή τους". Η διαδικασία δέχεται ως είσοδο τις XQuery εκφράσεις που έχουν προκύψει από την μετάφραση τόσο της σχηματομορφής γράφου της ερώτησης όσο και των τροποποιητών λύσεων, τις επεξεργάζεται και ανάλογα με την μορφή της SPARQL ερώτησης τις εμπλουτίζει ώστε η μορφή των λύσεων να προσαρμοστεί ανάλογα με την μορφή της SPARQL ερώτησης.

Στο **δέκατο πέμπτο κεφάλαιο** του κειμένου, αναλύεται ο τρόπος προσομοίωση-μετάφρασης των ενσωματωμένων (built-in) τελεστών/συναρτήσεων που περιέχονται στην παρούσα προδιαγραφή της γλώσσας SPARQL με χρήση της γλώσσας XQuery.

Στο **δέκατο έκτο κεφάλαιο** του κειμένου, περιγράφεται η XML μορφή των SPARQL αποτελεσμάτων, όπως αυτή προτείνεται από το W3C, καθώς και μια επέκταση της για την αναπαράσταση αποτελεσμάτων που προέρχονται από DESCRIBE ερωτήσεις. Τέλος περιγράφεται η διαδικασία μετασχηματισμού των αποτελεσμάτων που επιστρέφονται από τις XQuery ερωτήσεις, ώστε να ακολουθούν την XML μορφή που προτείνεται για τα SPARQL αποτελέσματα.

Τέλος, στο **δέκατο έβδομο κεφάλαιο** γίνεται μια ανακεφαλαίωση της εργασίας, περιγράφεται ο τρόπος αξιολόγησης της και επισημαίνονται οι μελλοντικές επεκτάσεις της.

2 Σχετικά Τεχνικά Πρότυπα, Προδιαγραφές και Τεχνολογίες

2.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα γίνει αρχικά η ανάλυση των πρότυπων (Ενότητες 2.2-2.8) και στην συνέχεια των τεχνολογιών (Ενότητες 2.9-2.11) που χρησιμοποιήθηκαν κατά την διάρκεια της παρούσας εργασίας. Ειδικότερα:

Στην Ενότητα 2.2 θα παρουσιάζεται η γλώσσα XML η οποία είναι μια γλώσσα περιγραφής δομημένων πληροφοριών. Στην Ενότητα 2.3 θα παρουσιάζεται η γλώσσα XML Schema. Στην Ενότητα 2.4 παρουσιάζεται η εκφράσεις μονοπατιών για XML δεδομένα Xpath. Στην Ενότητα 2.5 παρουσιάζεται η γλώσσα ερωτήσεων για XML δεδομένα XQuery.

Στην Ενότητα 2.6 παρουσιάζονται οι γλώσσες RDF και RDF Schema. Στην Ενότητα 2.7 παρουσιάζεται η γλώσσα περιγραφή οντολογιών OWL. Στην Ενότητα 2.8 παρουσιάζεται η γλώσσα ερωτήσεων για RDF δεδομένα SPARQL.

Στην Ενότητα 2.9 παρουσιάζεται το framework XML Beans. Στην Ενότητα 2.10 παρουσιάζεται η XML βάση δεδομένων Oracle Berkley DB XML.Και τέλος στην Ενότητα 2.11 παρουσιάζεται το Jena Framework.

2.2 XML eXtensible Markup Language

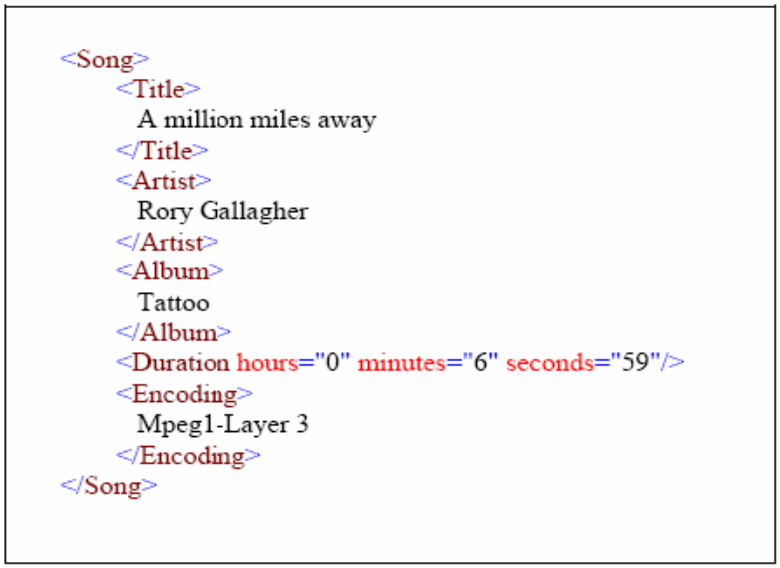
Το βασικό πρότυπο ανταλλαγής δομημένων δεδομένων στο Διαδίκτυο σήμερα είναι η **XML (eXtensible Markup Language)**[39] . Η XML υποστηρίζει δομημένα **XML Έγγραφα (XML Documents)**, τα οποία απαρτίζονται από **στοιχεία (elements)**. Η αρχή και το τέλος ενός XML εγγράφου καθορίζονται από την αρχή και το τέλος του **στοιχείου-ρίζας (root element)** του. Τα στοιχεία μπορεί να είναι εμφωλευμένα (nested) και να διαθέτουν χαρακτηριστικά, τα οποία αναπαρίστανται ως **γνωρίσματα (attributes)**.

Η XML θεωρείται επεκτάσιμη (extensible) επειδή επιτρέπει στους χρήστες να ορίσουν το δικό τους σχήμα, σε αντίθεση με την HTML η οποία είναι προκαθορισμένη γλώσσα σήμανσης (markup language) .Η XML αποτελεί μια μετα-γλώσσα επιτρέποντας, μέσω της δημιουργίας νέων ετικετών (tags), τον σχεδιασμό και την δημιουργία καινούριων γλωσσών, εφαρμογών – παράγωγων της XML. Οι προγραμματιστές μπορούν, ορίζοντας ένα δικό τους λεξιλόγιο, να προσδιορίσουν μια καινούρια γλώσσα σήμανσης προσαρμοσμένη στις εξειδικευμένες ανάγκες και απαιτήσεις της εκάστοτε εφαρμογής ή του συγκεκριμένου πεδίου εφαρμογής. Εξαιτίας της μεγάλης ευελιξίας της, η XML είναι ο πλέον διαδεδομένος τρόπος για την διανομή και παρουσίαση δομημένων και ημιδομημένων δεδομένων μέσω του διαδικτύου. Η XML είναι συμβατή με την πλειοψηφία των πρωτοκόλλων μετάδοσης του διαδικτύου και επιπλέον είναι ιδιαίτερα συμπιέσιμη για ταχύτερη μετάδοση. Η XML είναι πολύ φιλική προς τον χρήστη και έχει σχεδιαστεί να είναι ανεξάρτητη από προμηθευτές λογισμικού, λειτουργικά συστήματα και πρωτοκόλλα επικοινωνίας.

Η XML βασίζεται σε ένα ιεραρχικό μοντέλο δεδομένων το οποίο αποτελείται από δεδομένα και το σχήμα (schema) το οποίο τα περιγράφει.

2.2.1 XML Στοιχεία, Γνωρίσματα και Δεδομένα

Τα XML έγγραφα περιέχουν δομημένο κείμενο. Οι συντάκτες αναπαριστούν την δομή τοποθετώντας ετικέτες (tags) γύρω από τα δεδομένα. Οι δομικοί χαρακτήρες αρχής, τέλους



Εικόνα 2.1 : Παράδειγμα XML εγγράφου [40]

(structural delimiters) είναι ετικέτες, οι οποίες ξεκινούν και τελειώνουν με γωνιακές αγκύλες <…>. Το κείμενο μεταξύ των γωνιακών αγκύλων περιέχει πληροφορία σχετικά με το XML στοιχείο (element) - κατ' ελάχιστων ονοματίζει το στοιχείο. Ένα στοιχείο (element) αποτελείται από ετικέτα (tag) ανοίγματος, περιεχόμενα, και ετικέτα κλεισίματος. Οι ετικέτες κλεισίματος έχουν το ίδιο όνομα με τις ετικέτες ανοίγματος όμως ξεκινούν με </. Τα περιεχόμενα ενός στοιχείου (elements) μπορεί να είναι κείμενο, άλλα στοιχεία ή συνδυασμός των δύο. Ένα XML έγγραφο πρέπει να έχει μοναδικό στοιχείο το οποίο περιλαμβάνει όλα τα υπόλοιπα στοιχεία (root element). Η XML είναι case and space sensitive και τα στοιχεία πρέπει πάντα να έχουν ετικέτα κλεισίματος, ενώ απαγορεύεται η ύπαρξη επικαλυπτόμενων στοιχείων. Ένα στοιχείο μπορεί να είναι κενό και να μην έχει περιεχόμενα. Τα XML σχόλια εμφανίζονται μεταξύ των χαρακτήρων <!--και -->.

Οι ετικέτες μπορεί να περιέχουν πρόσθετη πληροφορία η οποία καλείται γνώρισμα (attribute). Τα γνωρίσματα τοποθετούνται στην ετικέτα ανοίγματος ενός στοιχείου(element) και γράφονται με την μορφή όνομα="τιμή" . Δεν υπάρχει κενό μεταξύ = και ονόματος και η τιμή ενός γνωρίσματος πρέπει να περιβάλλεται από χαρακτήρες ' ή ". Τα γνωρίσματα μέσα σε ένα στοιχείο πρέπει να έχουν μοναδικά ονόματα.

Έχουμε παραθέσει δείγμα XML εγγράφου με πληροφορίες σχετικά με ένα αρχείο mp3 στην εικόνα 2.1. Το στοιχείο <Song> περιλαμβάνει πέντε στοιχεία: <Title>, <Artist>, <Album>, <Duration> και <Encoding>.Το στοιχείο <Title> εμπεριέχει το κείμενο "A million miles away" δίδοντας πληροφορίες σχετικά με τον τίτλο του τραγουδιού. Το στοιχείο <Artist> εμπεριέχει το κείμενο "Rory Gallagher" δίδοντας πληροφορίες σχετικά με το όνομα του καλλιτέχνη. Το <Album> περιέχει το κείμενο "Tattoo"

πληροφορώντας μας για το όνομα του album. Το στοιχείο <Duration> περιλαμβάνει τρία γνωρίσματα hours, minutes, seconds με τιμές που υποδεικνύουν την διάρκεια του τραγουδιού. Τέλος το πέμπτο στοιχείο <Encoding> περιέχει κείμενο "Mpeg1-Layer3" με πληροφορίες σχετικά με την κωδικοποίηση του αρχείου.

Τέλος ένας μηχανισμός που συμβάλλει στην επεκτασιμότητα της XML και βοηθάει στον προσδιορισμό των στοιχείων ενός εγγράφου XML είναι οι χώροι ονοματοδοσίας (XML Namespaces). Προσδιορίζοντας τους χώρους ονοματοδοσίας που χρησιμοποιεί ένα έγγραφο αποφεύγεται η σύγχυση από συνώνυμα στοιχεία. Παράλληλα δίνεται η δυνατότητα στον καθένα να επεκτείνει την γλώσσα καθορίζοντας δικές του ετικέτες, υπάγοντας τις κάτω από ένα καινούριο χώρο ονοματοδοσίας, χωρίς να υπάρχει πρόβλημα σύγκρουσης με ονόματα ετικετών που ήδη χρησιμοποιούνται. Για την δήλωση των χώρων ονοματοδοσίας χρησιμοποιούνται τα URIs(Universal Resource Identifier), που προσδιορίζουν μοναδικά ένα χώρο στον παγκόσμιο ιστό.

2.3 XML Schema

Η γλώσσα **XML Schema**[19][20], που παρουσιάζεται σε αυτή την υπό-ενότητα, είναι η γλώσσα που χρησιμοποιείται για τον ορισμό κλάσεων XML εγγράφων, όπου για κάθε κλάση καθορίζονται τα χαρακτηριστικά, το περιεχόμενο και η δομή των **στιγμιοτύπων (instances)** της. Η γλώσσα XML Schema χρησιμοποιεί XML σύνταξη και υποστηρίζει πλούσιες δομές και τύπους δεδομένων. Ο ορισμός μιας κλάσης εγγράφων στη γλώσσα XML Schema είναι ένα XML σχήμα (schema).

Η γλώσσα XML Schema επιτρέπει τον ορισμό απλών και σύνθετων στοιχείων, τα οποία ανήκουν σε XML Schema τύπους που καθορίζονται στο γνώρισμα "`type`" των στοιχείων. Τα **σύνθετα στοιχεία (complex elements)** ανήκουν σε **σύνθετους τύπους (complex types)** και μπορεί να διαθέτουν γνωρίσματα και να περιέχουν άλλα στοιχεία. Οι σύνθετοι τύποι μπορεί να είναι απτοί (concrete), οπότε επιτρέπεται να οριστούν στιγμιότυπά τους, ή αφηρημένοι (abstract), οπότε δεν επιτρέπεται ο ορισμός στιγμιοτύπων τους, όπως καθορίζεται στο γνώρισμα "`abstract`" των σύνθετων τύπων. Τα **απλά στοιχεία (simple elements)** ανήκουν σε **απλούς τύπους (simple types)**, οι οποίοι συνήθως ορίζονται ως περιορισμοί (restrictions) των βασικών τύπων που παρέχονται από την XML Schema, όπως είναι οι συμβολοσειρές (strings), οι αριθμοί (ακέραιοι και πραγματικοί), τα σύμβολα (tokens) κ.α. Τα απλά στοιχεία δε διαθέτουν γνωρίσματα και δεν επιτρέπεται να περιέχουν άλλα στοιχεία. Η γλώσσα XML Schema υποστηρίζει κληρονομικότητα (inheritance) και περιορισμούς (constraints) για όλους τους τύπους, απλούς και σύνθετους. Η κληρονομικότητα μπορεί να εφαρμόζεται τόσο με την επέκταση (extension) όσο και με τον

περιορισμό του τύπου που αποτελεί βάση του τρέχοντος τύπου. Ο τύπος που αποτελεί τη βάση του τρέχοντος τύπου δηλώνεται στο γνώρισμα "base" του ορισμού του τρέχοντος τύπου. Επιπλέον, μπορεί να δηλωθεί αν επιτρέπεται ή όχι ο τρέχον τύπος να αποτελέσει βάση για άλλους τύπους, μέσω της τιμής του γνωρίσματος "final" των σύνθετων τύπων. Υποστηρίζονται ακόμα επαναχρησιμοποιήσιμοι ορισμοί στοιχείων: Η παρουσία του γνωρίσματος "substitutionGroup" σε ορισμούς στοιχείων δηλώνει ότι το τρέχον στοιχείο αποτελεί εξειδίκευση κάποιου άλλου στοιχείου, το όνομα του οποίου καθορίζεται στην τιμή του γνωρίσματος "substitutionGroup". Η δήλωση αν επιτρέπεται ή όχι το τρέχον στοιχείο να εξειδικευτεί γίνεται μέσω της τιμής του γνωρίσματος "block" των στοιχείων.

Η γλώσσα XML Schema υποστηρίζει γνωρίσματα, που αναπαριστούν τα χαρακτηριστικά των σύνθετων τύπων. Γνωρίσματα τα οποία πρέπει να χρησιμοποιούνται ταυτόχρονα σχηματίζουν **ομάδες γνωρισμάτων (attribute groups)**.

Προκαθορισμένες (fixed) και εξ' ορισμού (default) τιμές υποστηρίζονται από την XML Schema τόσο για γνωρίσματα όσο και για απλού τύπου στοιχεία, μέσω των γνωρισμάτων "default" και "fixed" αντίστοιχα.

Τα στοιχεία που περιέχονται σε κάποιο άλλο στοιχείο ή στον ορισμό κάποιου τύπου μπορεί να είναι εναλλάξιμα, οπότε απαρτίζουν **επιλογές (choices)** και **σύνολα** (που υλοποιούνται από τις δομές "xs:choice" και "xs:all" αντίστοιχα) ή να έχουν προκαθορισμένη σειρά, οπότε απαρτίζουν **ακολουθίες (sequences)** (που υλοποιούνται από τη δομή "xs:sequence"). Τόσο οι ακολουθίες όσο και οι επιλογές μπορεί να είναι εμφωλευμένες. Ο ελάχιστος και ο μέγιστος επιτρεπτός αριθμός στοιχείων μέσα σε ακολουθίες και επιλογές ορίζονται από τα γνωρίσματα "minOccurs" και "maxOccurs" αντίστοιχα. Αν δεν προσδιορίζεται η τιμή του γνωρίσματος "minOccurs" και/ή του γνωρίσματος "maxOccurs", υπονοείται η τιμή 1. Επαναχρησιμοποιήσιμες σύνθετες δομές, που μπορεί να συνδυάζουν ακολουθίες και επιλογές, σχηματίζουν τα **μοντέλα ομάδων (model groups)**.

Τα μοντέλα ομάδων και οι ομάδες γνωρισμάτων ορίζονται σε κορυφαίο επίπεδο (top-level) μόνο, ενώ τα γνωρίσματα και τα στοιχεία μπορούν να οριστούν τόσο σε κορυφαίο επίπεδο όσο και μέσα στους ορισμούς τύπων. Οι τύποι (απλοί και σύνθετοι) μπορούν να οριστούν τόσο σε κορυφαίο επίπεδο όσο και μέσα στους ορισμούς στοιχείων και γνωρισμάτων.

Οι τύποι (απλοί και σύνθετοι), τα στοιχεία, τα γνωρίσματα, οι ομάδες γνωρισμάτων και τα μοντέλα ομάδας κορυφαίου επιπέδου διαθέτουν μοναδικά ονόματα, που αναπαρίστανται από το χαρακτηριστικό "name". Ονόματα διαθέτουν και τα εμφωλευμένα στοιχεία και γνωρίσματα, τα οποία πρέπει να είναι μοναδικά στα όρια των πιο κοντινών τύπων μέσα στους οποίους ορίζονται. Να σημειωθεί ότι οι εμφωλευμένοι τύποι (απλοί και σύνθετοι) είναι ανώνυμοι. Η XML Schema υποστηρίζει XML χώρους ονομάτων (namespaces), που επιτρέπουν τη διάκριση τύπων, στοιχείων και γνωρισμάτων από διαφορετικά σχήματα που έχουν το ίδιο όνομα και μπορεί να χρησιμοποιούνται στο ίδιο έγγραφο. Επιπλέον, όλες οι XML

Schema δομές (γνωρίσματα, στοιχεία, επιλογές, ακολουθίες, απλοί και σύνθετοι τύποι, μοντέλα ομάδες και ομάδων γνωρισμάτων) επιτρέπεται να διαθέτουν μοναδικές ταυτότητες (identifiers) που αναπαρίστανται από το γνώρισμα "id".

Τα γνωρίσματα, τα στοιχεία, οι ομάδες γνωρισμάτων και τα μοντέλα ομάδας που ορίζονται σε κορυφαίο επίπεδο μπορούν να χρησιμοποιηθούν σε ορισμούς τύπων μέσω αναφορών (references) που υλοποιούνται από το γνώρισμα "ref".

Λόγω των δυνατοτήτων δόμησης που παρέχει η γλώσσα XML Schema και του κεντρικού ρόλου που παίζει κατά την ανταλλαγή δεδομένων στο Διαδίκτυο, σημαντικά πρότυπα για πολλές διαφορετικές περιοχές εφαρμογών έχουν εκφραστεί στη γλώσσα XML Schema, συμπεριλαμβανομένων προτύπων στην περιοχή των πολυμέσων όπως τα MPEG-7[101] και MPEG-21[102], προτύπων ηλεκτρονικής εκπαίδευσης (e-learning) όπως τα IEEE LOM[103] και SCORM[104], προτύπων για Ψηφιακές Βιβλιοθήκες (Digital Libraries) όπως το METS[105] κ.α.

Το XML Schema για το XML έγγραφο της εικόνας 2.1 θα μπορούσε να είναι αυτό που δίνεται παρακάτω:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Song">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Title" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="Artist" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="Album" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="Duration" minOccurs="1" maxOccurs="1">
          <xsd:complexType>
            <xsd:attribute name="hours" type="xsd:string" use="required"/>
            <xsd:attribute name="minutes" type="xsd:string" use="required"/>
            <xsd:attribute name="seconds" type="xsd:string" use="required"/>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Encoding" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

</xsd:schema>
```

Εικόνα 2.3 : Πιθανό XML Scheme για το XML έγγραφο της Εικόνας 2.1 [40]

2.3.1 Well-Formedness και Validity

Η XML αναγνωρίζει δύο κατηγορίες εγγράφων: Τα σωστά μορφοποιημένα (well-formed) και τα έγκυρα (valid).

Για να μπορεί να χαρακτηριστεί ένα XML έγγραφο ως σωστά μορφοποιημένο πρέπει να ακολουθεί αυστηρά τους συντακτικούς κανόνες της XML τους οποίους συνοψίζουμε στα εξής:

- Μοναδικό ριζικό στοιχείο
- Συμφωνία ετικετών Ανοίγματος – Κλεισίματος
- Σωστά εμφωλευμένες ετικέτες

- Οι τιμές των ιδιοτήτων να εσωκλείονται σε εισαγωγικά
- Όχι γνωρίσματα με το ίδιο όνομα στο ίδιο στοιχείο

Για να χαρακτηριστεί ένα έγγραφο έγκυρο θα πρέπει να υπακούει στους κανόνες που έχουν δοθεί από το XML Schema στο οποίο υπόκειται και επιπλέον θα πρέπει να είναι και σωστά μορφοποιημένο.

2.4 Xpath XML Path Language

Η **XPath (XML Path Language)** [7] [8] αποτελεί μια γλώσσα έκφρασης που επιτρέπει τη διευθυνσιοδότηση των τμημάτων που απαρτίζουν ένα XML έγγραφο. Είναι ένα ακόμη πρότυπο του διεθνή οργανισμού Web Wide World Consortium (W3C). Η XPath παρέχει κοινής σύνταξη και σημασιολογία λειτουργικότητας τόσο για την XPointer[57] όσο και για την XSLT[107]. Εκτός από την υποστήριξη διευθυνσιοδότησης, η XPath παρέχει και ένα σύνολο βασικών δυνατοτήτων διαχείρισης συμβολοσειρών, αριθμών και δίτιμων μεταβλητών (booleans). Σήμερα, η XPath έχει υιοθετηθεί απο τους προγραμματιστές ως μια περιεκτική γλώσσα ερωτήσεων. Ειδικότερα, η σύνταξη της XPath είναι συμπαγής και δε βασίζεται στην XML, έτσι ώστε να είναι δυνατή η χρήση της σε URIs και σε τιμές XML χαρακτηριστικών. Η XPath λειτουργεί περισσότερο πάνω στην αφηρημένη λογική δομή των XML εγγράφων παρά στη σύνταξή τους. Οφείλει το όνομά της στη χρήση της ένδειξης του μονοπατιού ("/"), όπως συμβαίνει και στα URL's, που στόχο έχει τη πλοήγηση στην ιεραρχική δομή ενός XML εγγράφου. Η γλώσσα αυτή στηρίζεται στην δεντρική αναπαράσταση ενός XML έγγραφου και παρέχει την ικανότητα για πλοήγηση γύρω από το δέντρο, επιλέγοντας κόμβους με βάση κάποια κριτήρια. Ως ένα παράδειγμα ενός XML έγγραφου και των κόμβων από τους οποίους αποτελείται δίνεται στην Εικόνα 2.4:

```
<?xml version="1.0"?>
<Node0>
        <Node1 class="myValue1">Node1 text </Node1>
        <Node2>
                <Node3>Node3 text</Node3>
                <Node3>Node3 text 2</Node3>
                <Node3>Node3 text 3</Node3>
                <Node4>300</Node4>
        </Node2>
</Node0>
```

Εικόνα 2.4: Ένα απλό XML έγγραφο

Στο έγγραφο της Εικόνας 2.4 το <Node0> αποτελεί το κόμβο ρίζας ενώ το <Node1> αποτελεί κόμβο στοιχείου. Επιπρόσθετα, ο κόμβος στοιχείου <Node1> περιέχει ένα κόμβο γνωρίσματος του οποίου το όνομα είναι class και η τιμή του είναι "myValue1". Ακόμη, ο κόμβος Node1 περιέχει ένα κόμβου κειμένου του οποίου η τιμή είναι "Node1 text". Εκτός απο τη κύρια χρήση της για διευθυνσιοδότηση, η XPath σχεδιάστηκε για να έχει ένα φυσικό υποσύνολο που μπορεί να χρησιμοποιηθεί για τον έλεγχο ταιριάσματος ενός κόμβου με ένα πρότυπο. Η χρήση της αυτή περιγράφεται απο την XSLT. Η XPath μοντελοποιεί ένα XML έγγραφο σαν ένα δέντρο από κόμβους. Υπάρχουν διαφορετικά είδη κόμβων, όπως κόμβοι στοιχείων, γνωρισμάτων και κειμένου. Καθορίζει ένα τρόπο για τον υπολογισμό μιας string τιμής για κάθε τύπο κόμβου. Επιπλέον, κάποια είδη κόμβων έχουν ονόματα. Ακόμη, η XPath υποστηρίζει πλήρως XML namespaces. Ως αποτέλεσμα, το όνομα ενός κόμβου έχει μοντελοποιηθεί ως ένα ζευγάρι αποτελούμενο από ένα τοπικό τμήμα και ένα ενδεχόμενο null namespace URI και το οποίο καλείται expanded-name. Η βασική συντακτική δομή της XPath είναι η Έκφραση (Expression). Μια έκφραση υπολογίζεται και αποδίδει κάποιο αποτέλεσμα. Το αποτέλεσμα μιας έκφρασης είναι ένα αντικείμενο που μπορεί να ανήκει σε κάποιον από τους εξής τέσσερις βασικούς τύπους:

1. Σύνολο κόμβων (node set), που είναι μια μη διατεταγμένη συλλογή κόμβων που αναπαριστούν στοιχεία XML εγγράφων, όπου δεν υπάρχουν διπλές εμφανίσεις κόμβων.
2. Δίτιμη μεταβλητή (boolean), που μπορεί να πάρει μια από τις τιμές αληθής και ψευδής.
3. Αριθμός, που μπορεί να πάρει οποιαδήποτε τιμή αντιστοιχεί σε πραγματικό αριθμό.
4. Συμβολοσειρά, που είναι ένας συνδυασμός χαρακτήρων.

Ο υπολογισμός της τιμής μιας έκφρασης γίνεται με βάση τις παραμέτρους (context) της έκφρασης, οι οποίες καθορίζονται από την XPointer και την XSLT.

Οι παράμετροι μιας έκφρασης είναι οι εξής:

- Ο κόμβος των παραμέτρων της έκφρασης (context node).
- Το σύνολο κόμβων των παραμέτρων της έκφρασης (context list node), μέλος του οποίου αποτελεί ο κόμβος των παραμέτρων της έκφρασης.
- Ένα σύνολο αντιστοιχήσεων μεταβλητών (variable bindings), το οποίο αποτελείται από τις αντιστοιχίσεις των ονομάτων των μεταβλητών στις τιμές των μεταβλητών αυτών. Καθεμιά από τις μεταβλητές είναι ένα αντικείμενο είτε κάποιου από τους τέσσερις βασικούς τύπους είτε κάποιου άλλου τύπου.
- Μια βιβλιοθήκη λειτουργιών (function library), η οποία είναι ένα σύνολο επωνύμων συναρτήσεων. Κάθε συνάρτηση δέχεται 0 ή περισσότερα ορίσματα και επιστρέφει κάποιο αποτέλεσμα. Η XPath διαθέτει ένα πυρήνα βασικών συναρτήσεων, των οποίων τόσο τα

ορίσματα όσο και το αποτέλεσμα ανήκουν στους βασικούς τύπους. Τόσο η XPointer όσο και η XSLT ορίζουν επιπρόσθετες συναρτήσεις, κάποιες από τις οποίες λειτουργούν με τους τέσσερις βασικούς τύπους ενώ κάποιες άλλες ενεργούν σε επιπρόσθετους τύπους που ορίζονται από την XPointer και την XSLT.

- Το σύνολο δηλώσεων των namespaces που εφαρμόζονται στην έκφραση.

Η XPath περιλαμβάνει πάνω απο εκατό έτοιμες και ενσωματωμένες συναρτήσεις. Υπάρχουν συναρτήσεις για string τιμές, αριθμητικές τιμές, σύγκριση ημερομηνίας και χρόνου, διαχείριση κόμβου και QName (έτσι καλούνται τα ονόματα στην XPath), διαχείριση αλληλουχίας, λογικές τιμές και άλλες. Η XPath χρησιμοποιεί εκφράσεις μονοπατιού (path expressions) για την επιλογή κόμβων σε ένα XML έγγραφο. Ο κόμβος επιλέγεται ακολουθώντας ένα μονοπάτι ή κάποια βήματα. Οι πιο χρήσιμες εκφράσεις παρουσιάζονται στον Πίνακα 2.1:

Όνομα κόμβου	Επιλογή όλων των κόμβων-παιδιών του κόμβου
/	Επιλογή από το κόμβο ρίζας
//	Επιλογή κόμβων του εγγράφου από το τρέχον κόμβο που ικανοποιούν τα κριτήρια σε οποιοδήποτε βάθος
.	Επιλογή του τρέχοντος κόμβου
..	Επιλογή του κόμβου-πατέρα του τρέχοντος κόμβου
@	Επιλογή γνωρισμάτων

Πίνακας 2.1: Χρήσιμες εκφράσεις μονοπατιού

Όταν εκτελείται μια XPath έκφραση, αυτο που επιστρέφεται είναι ένα σύνολο αποτελεσμάτων. Υπάρχει δυνατότητα φιλτραρίσματος του συνόλου των αποτελεσμάτων με τη χρήση των κατηγορημάτων (predicates). Αυτά περιέχονται πάντα μέσα σε brackets και είναι δύο ειδών: αριθμητικά και boolean. Τα αριθμητικά predicates επιτρέπουν την επιλογή κόμβου με βάση τη θέση σχετικά με ένα άλλο κόμβο του εγγράφου. Ως παράδειγμα ενός αριθμητικού predicate, αναφερόμενοι στο έγγράφο της Εικόνας 2.4, η έκφραση: /Node1/Node2/Node3[2] θα επιστρέψει το δεύτερο κόμβο <Node3> του εγγράφου.

Τα Boolean predicates φιλτράρουν τα αποτελέσματα του query ώστε να επιλέγονται μόνο τα συγκεκριμένα elements του αποτελέσματος έαν η έκφραση υπολογίζεται ως true. Ως παράδειγμα ενός Boolean predicate, η έκφραση /Node1/Node2[Node3="Node3 text 3"], θα επιστρέψει ένα κόμβο του οποίου ο κόμβος κειμένου είναι ίσος με την παραπάνω τιμή.

Ένα άλλο σημαντικό στοιχείο στην χρήση της XPath είναι η παρουσία των επονομαζόμενων wildcards. Η χρήση αυτών θεωρείται απαραίτητη για την επιλογή στοιχείων του εγγράφου που είναι άγνωστα. Στον πίνακα 2.2 παρουσιάζονται τα βασικότερα wildcards:

*	Αντιστοίχηση σε κάθε κόμβο στοιχείου
*@	Αντιστοίχηση σε κάθε κόμβο γνωρίσματος
node()	Αντιστοίχηση σε κάθε κόμβο ανεξαρτήτως τύπου

Πίνακας 2.2 Χρήσιμα wildcards

2.5 XQuery an XML Query Language

Η **XQuery (XML Query Language)**[6][7] είναι γλώσσα ερωτήσεων σε βάσεις XML δεδομένων και σε XML έγγραφα η οποία σχεδιάστηκε για την ανάκτηση πληροφοριών από XML συλλογές δεδομένων. Σημασιολογικά παρουσιάζει ομοιότητες με την SQL.

2.5.1 Χαρακτηριστικά

Η XQuery παρέχει την δυνατότητα ανάκτησης και διαχείρισης δεδομένων που περιέχονται σε XML έγγραφα ή σε οποιαδήποτε πηγή μπορεί να αντιμετωπιστεί ως XML.

Η XQuery είναι μια λειτουργική γλώσσα στην οποία κάθε ερώτημα είναι μια έκφραση. Οι εκφράσεις XQuery εμπίπτουν σε επτά ευρείς τύπους: εκφράσεις διαδρομής (path expressions), κατασκευαστές στοιχείων (element constructor), εκφράσεις FLWR (FLWR expressions), εκφράσεις που περιέχουν χειριστές και συναρτήσεις, εκφράσεις συνθήκης (conditional expressions), ποσοτικές εκφράσεις ή εκφράσεις που δοκιμάζουν ή μετατρέπουν τύπους δεδομένων.

Η XQuery είναι ένα περίπλοκο σύστημα τύπων βασισμένο στους τύπους δεδομένων των XML Schemas και υποστηρίζει την διαχείριση των κόμβων του εγγράφου.

Θα παραθέσουμε XML έγγραφο (books.xml) που θα αποτελέσει την βάση στην οποία θα εφαρμοστούν κάποια XQuery ερωτήματα:

```
<bib>
  <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
  </book>
  <book year="1992">
        <title>Advanced Programming in the UNIX Environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
  </book>
  <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>65.95</price>
  </book>
</bib>
```

Εικόνα 2.5 : Το έγγραφο book.xml πάνω στο οποίο θα εφαρμοστούν XQuery ερωτήματα

2.5.2 Εντοπισμός κόμβων με χρήση της XPath

Η XQuery, διαχειρίζεται τα XML έγγραφα ως δένδρα που αποτελούνται από κόμβους. Τα είδη των κόμβων που προκύπτουν είναι τα εξής: έγγραφο, στοιχείο, κείμενο, γνώρισμα, χώρος ονοματοδοσίας, οδηγίες επεξεργασίας, και σχόλιο. Η σύνταξη των δηλώσεων που χρησιμοποιούνται για τον εντοπισμό των κόμβων (δηλώσεις διαδρομής, path expression) προέρχεται από τη σύνταξη της γλώσσας XPath. Ένα απλό παράδειγμα χρήσης της XQuery για τον εντοπισμό κόμβων είναι το εξής:

doc (“books.xml”)/bib/book

Η παραπάνω δήλωση ανοίγει το έγγραφο books.xml χρησιμοποιώντας την συνάρτηση doc (), η οποία επιστρέφει το έγγραφο. Στο επόμενο βήμα δηλώνεται το /bib που επιστρέφει το root element και στη συνέχεια δηλώνεται το /book που επιστρέφει όλα τα στοιχεία book που βρίσκονται στο στοιχείο (element) bib. Το αποτέλεσμα αυτού είναι να επιστρέψει τα τρία στοιχεία book που περιέχονται στο ριζικό στοιχείο bib και φαίνονται στην Εικόνα 2.5.

2.5.3 Δημιουργία Κόμβων

Μέσω της XQuery παρέχεται η δυνατότητα δημιουργίας νέων κόμβων. Συγκεκριμένα, για την δημιουργία των νέων στοιχείων, χρησιμοποιούνται οι αγκύλες οι οποίες περιέχουν δηλώσεις που εκτελούνται για την δημιουργία νέου περιεχομένου. Για παράδειγμα:

```
<example>
      <p> Here is a query </p>
          <eg> doc("books.xml")/bib/book[1]/title </eg>
          <p> Here is the result of the above query </p>
          <eg> { doc("books.xml")/bib/book[1]/title } </eg>
</example>
```

Το αποτέλεσμα του παραπάνω ερωτήματος είναι το εξής :

```
<example>
      <p> Here is a query </p>
          <eg> doc("books.xml")/bib/book[1]/title </eg>
          <p> Here is the result of the above query </p>
          <eg><title>TCP/IP Illustrated</title></eg>
</example>
```

Οι δηλώσεις που περιέχονται στις αγκύλες επιτρέπουν την δημιουργία νέων XML τιμών αναδομώντας υπάρχουσες XML τιμές. Στο προηγούμενο παράδειγμα πήραμε τον τίτλο του πρώτου βιβλίου. Καθώς οι τιμές μέσα στις αγκύλες αποτιμούνται (γίνονται evaluate) πρώτα και μετά επιστέφεται η τιμή που προκύπτει από την αποτίμηση.

Το επόμενο παράδειγμα δημιουργεί ένα νέο XML έγγραφο το οποίο περιέχει τους τίτλους των βιβλίων που περιγράφονται από το έγγραφο της εικόνας 2.5:

```
<titles count="{ count(doc("books.xml")//title) }">

{

      doc("books.xml")//title

}

</titles>
```

Το αποτέλεσμα αυτού του παραπάνω ερωτήματος είναι το εξής :

```
<titles count = "3">

       <title>TCP/IP Illustrated</title>

       <title>Advanced Programming in the Unix Environment</title>

       <title>Data on the web</title>

</titles>
```

2.5.4 Συνδυασμός και αναδόμηση κόμβων

Οι δηλώσεις FLWR, είναι οι πιο συνηθισμένες και πιο ισχυρές δηλώσεις της γλώσσας XQuery. Οι δηλώσεις αυτές είναι παρόμοιες με τις δηλώσεις SELECT – FROM – WHERE που χρησιμοποιούνται στην γλώσσα SQL για την διαχείριση των βάσεων δεδομένων. Αντίθετα, όμως με την SQL, όπου οι δηλώσεις καθορίζονται με βάση τους πίνακες μιας βάσης δεδομένων, τις εγγραφές και τα χαρακτηριστικά τους, οι FLWR δηλώσεις δεσμεύουν μεταβλητές με τιμές μέσω των for και των let συντακτικών μονάδων (clause), και χρησιμοποιούν αυτές τις αντιστοιχίες για την δημιουργία νέων αποτελεσμάτων. Ένας συνδυασμός τέτοιων αποτελεσμάτων ονομάζεται συστοιχία (tuple). Μια απλή FLWR δήλωση που επιστρέφει τον τίτλο και την τιμή κάθε βιβλίου που εκδόθηκε τη χρονιά 2000 είναι η εξής:

```
for $b in doc("books.xml")/bib/book

where $b/@year = "2000"

return $b/title
```

Η δήλωση αυτή δεσμεύει την μεταβλητή $b σε ένα βιβλίο κάθε φορά, δημιουργώντας μια σειρά από συστοιχίες. Κάθε συστοιχία περιέχει μια δέσμευση κατά την οποία η μεταβλητή $b αντιστοιχεί σε ένα συγκεκριμένο βιβλίο. Η συντακτική μονάδα where ελέγχει κάθε συστοιχία για το αν ισχύει η ισότητα $b/@year = "2000" (με @ κάνουμε αναφορά στα γνωρίσματα (attributes) που περιέχει ένα στοιχείο), ενώ η συντακτική μονάδα return υπολογίζεται για κάθε συστοιχία που ικανοποιεί τη συνθήκη που εκφράστηκε με το where clause. Στην περίπτωσή μας το αποτέλεσμα του ερωτήματος είναι :

```
<title> Data on the web </title>
```

Τα αρχικά FLWR προέρχονται από τα πρώτα γράμματα των συντακτικών μονάδων που χρησιμοποιούνται στις εκφράσεις FLWR και είναι οι εξής :

- for: δεσμεύει μια ή περισσότερες μεταβλητές σε σειρά από συστοιχίες.
- let: δεσμεύουν μεταβλητές σε ολόκληρο το αποτέλεσμα μιας δήλωσης, είτε προσθέτοντας αυτές τις δεσμεύσεις στις συστοιχίες που δημιουργούνται από τη μονάδα for, είτε δημιουργώντας μια απλή συστοιχία που να περιέχει αυτές τις δεσμεύσεις σε περίπτωση που δεν έχει οριστεί μια for συντακτική μονάδα.
- where: φιλτράρει συστοιχίες, διατηρώντας μόνο αυτές που ικανοποιούν μια συνθήκη
- order by: ταξινομεί τις συστοιχίες

- return: δημιουργούν αποτελέσματα μιας FLWR δήλωσης για μια δοσμένη συστοιχία

Μια FLWR δήλωση ξεκινάει με μια ή περισσότερες for, let μονάδες τοποθετημένες σε οποιαδήποτε σειρά, ακολουθούμενες από τις συντακτικές μονάδες where, order by και return. Η χρήση των where και order by είναι προαιρετική ενώ η χρήση της return είναι υποχρεωτική.

2.5.4.1 Οι συντακτικές μονάδες for και let

Οι FLWR δηλώσεις καθορίζονται από τις συστοιχίες οι οποίες δημιουργούνται μέσω των μονάδων for και let, συνεπώς κάθε FLWR δήλωση πρέπει να αποτελείται από τουλάχιστον μια for ή let μονάδα. Τα παραδείγματα που ακολουθούν βοηθούν ώστε να γίνει κατανοητός ο τρόπος με τον οποίο δημιουργούνται οι συστοιχίες στις FLWR δηλώσεις.

Στο επόμενο παράδειγμα ορίζεται ένα ερώτημα που δημιουργεί (μέσω της return) ένα στοιχείο που ονομάζεται <tuple> προκειμένου να εμφανίσει τις συστοιχίες που δημιουργούνται από ένα τέτοιο ερώτημα:

```
for $i in (1,2,3)

return

      <tuple><i>{ $i }</i></tuple>
```

Στο παράδειγμα αυτό, δεσμεύεται η μεταβλητή $i στη δήλωση (1,2,3), που δημιουργεί μια αλληλουχία ακεραίων. Η XQuery παρέχει μια γενική σύνταξη, με αποτέλεσμα οι μονάδες for και let να δεσμεύουν οποιαδήποτε XQuery δήλωση. Ακολουθεί το αποτέλεσμα του παραπάνω ερωτήματος, που δείχνει τον τρόπο με τον οποίο η μεταβλητή $i δεσμεύεται σε κάθε συστοιχία:

```
<tuple><i>1</i></tuple>
<tuple><i>2</i></tuple>
<tuple><i>3</i></tuple>
```

Η συντακτική μονάδα let δεσμεύει μια μεταβλητή σε ολόκληρο το αποτέλεσμα μιας δήλωσης. Αν δεν έχουν οριστεί for μονάδες, τότε δημιουργείται μια απλή συστοιχία, που περιέχει τις δεσμεύσεις μιας μεταβλητής όπως ορίστηκε μέσω της let μονάδας. Το ακόλουθο ερώτημα είναι ίδιο με το προηγούμενο, με τη διαφορά ότι χρησιμοποιείται η συντακτική μονάδα let:

```
let $i  := (1,2,3)

return

      <tuple><i>{ $i }</i></tuple>
```

Ακολουθεί το αποτέλεσμα του παραπάνω ερωτήματος κατά το οποίο η μεταβλητή $i δεσμεύεται σε ολόκληρη την αλληλουχία των ακεραίων:

```
<tuple><i>1 2 3</i></tuple>
```

Στο ερώτημα που ακολουθεί οι δηλώσεις που δεσμεύτηκαν μέσω της let μονάδας προστίθεται στις συστοιχίες που δημιουργήθηκαν μέσω της μονάδας for.

```
for $i in (1 , 2 , 3)
let $j := (1, 2, 3)
return
    <tuple><i>{ $i }</i><j>{$j}</j></tuple>
```

Το αποτέλεσμα του παραπάνω ερωτήματος είναι το εξής:

```
<tuple><i>1</i><j>1 2 3</j></tuple>
<tuple><i>2</i><j>1 2 3</j></tuple>
<tuple><i>3</i><j>1 2 3</j></tuple>
```

2.5.4.2 Η συντακτική μονάδα where

Η συντακτική μονάδα where επιλέγει τις συστοιχίες αυτές που ικανοποιούν συγκεκριμένες συνθήκες. Η μονάδα return εκτελείται μόνο για τις συστοιχίες που επιστρέφονται από τη μονάδα where. Το ακόλουθο ερώτημα επιστρέφει τα βιβλία των οποίων η τιμή ξεπερνάει τα 50 € :

```
for $b in doc("books.xml")/bib/book
where $b/price > 50.00
return $b/title
```

Το αποτέλεσμα του ερωτήματος αυτού είναι:

```
<title>TCP/IP Illustrated</title>

<title>Advanced Programming in the Unix Environment</title>

<title>Data on the web</title>
```

Η μονάδα for μπορεί να περιέχει οποιαδήποτε δήλωση το αποτέλεσμα της οποίας, είναι μια Boolean τιμή. Το ακόλουθο ερώτημα επιστρέφει τον τίτλο των βιβλίων των οποίων οι συγγραφείς είναι παραπάνω από δύο:

```
for $b in doc("books.xml")//book

let $c := $b//author

where count ($c) >2

return $b/title
```

Το αποτέλεσμα της παραπάνω ερώτησης είναι το εξής:

```
<title>Data on the web</title>
```

2.5.4.3 Η συντακτική μονάδα order by

Η συντακτική μονάδα order by, ταξινομεί τις συστοιχίες πριν την εκτέλεση της μονάδας return προκειμένου να αλλάξει η σειρά των αποτελεσμάτων. Για παράδειγμα το ακόλουθο ερώτημα εμφανίζει τους τίτλους των βιβλίων με αλφαβητική σειρά:

```
for $t in doc("books.xml")//title

order by $t

return $t
```

Η μονάδα for παράγει μια αλληλουχία από συστοιχίες, όπου κάθε μια περιέχει τον κόμβο title. Η μονάδα order by ταξινομεί αυτές τις συστοιχίες με βάση την τιμή που περιέχει το στοιχείο <title> σε κάθε συστοιχία, ενώ η μονάδα return επιστρέφει τα στοιχεία <title> με την ίδια σειρά στην οποία βρίσκονται ταξινομημένες οι συστοιχίες.

Το αποτέλεσμα αυτού του ερωτήματος είναι το εξής:

```
<title>Advanced Programming in the Unix Environment</title>

<title>Data on the web</title>

<title>TCP/IP Illustrated</title>
```

2.5.4.4 Η συντακτική μονάδα return

Όμοια με τις συντακτικές μονάδες for και let, όπου επιτρέπουν τη δέσμευση μεταβλητών σε οποιαδήποτε δήλωση και με την μονάδα where η οποία μπορεί να περιέχει οποιαδήποτε Boolean δήλωση, η μονάδα return μπορεί να περιέχει οποιαδήποτε δήλωση. Έχουν ήδη παρατεθεί αρκετά παραδείγματα σχετικά με τον τρόπο χρησιμοποίησης της μονάδας return.

2.5.5 Δήλωση Συναρτήσεων

Η XQuery επιτρέπει και την δήλωση συναρτήσεων. Για την εκτέλεση συνάρτησης η έκφραση στο σώμα της συνάρτησης υπολογίζεται και η τιμή επιστρέφεται.

Παράδειγμα δήλωσης συνάρτησης

```
declare function local:doubler($x) { $x * 2 }
```

2.6 RDF/S Resource Description Framework/Schema

Το πρώτο βήμα στην κατεύθυνση της ανάπτυξης γλωσσών που υποστηρίζουν περιγραφές βάσει σημαντικής ήταν η ανάπτυξη της **RDF (Resource Description Framework)** [2], που είναι μια γενικού σκοπού και ανεξάρτητη από περιοχές εφαρμογών σύνταξη για την περιγραφή πηγών στο Διαδίκτυο.

Αν και η XML είναι μία πολύ ευέλικτη γλώσσα δεν προσφέρει κάποια πληροφορία για τη σημασιολογία των δεδομένων που περιγράφει. Επιτρέπει την μοντελοποίηση της πληροφορίας με πολλούς διαφορετικούς τρόπους και έτσι είναι πολύ δύσκολο για τα προγράμματα να καθορίσουν αυτόματα την ερμηνεία της εκάστοτε XML περιγραφής. Υποθέστε για παράδειγμα ότι θέλουμε να περιγράψουμε την πληροφορία "Ο κ. Χριστοδουλάκης διδάσκει το μάθημα Συστήματα Βάσεων Δεδομένων". Στην XML κάποιοι πιθανοί τρόποι αναπαράστασης αυτής της πληροφορίας θα ήταν:

```
<Course name="Database Systems">
      <Teacher>Christodoulakis Stavros</Teacher>
</Course>

<Teacher name="Christodoulakis Stavros">
      <Teaches>Database Systems</ Teaches>
</Teacher>

<Courses>
      <Teacher>Christodoulakis Stavros</Teacher>
      <Course>Database Systems</Course>
</Courses>
```

Η **RDF (Resource Description Framework)** [2] έρχεται να λύσει το παραπάνω πρόβλημα. Αποτελεί ένα μοντέλο δεδομένων με σύνταξη παρόμοια με την XML, που αναπαριστά την πληροφορία με τέτοιο τρόπο ώστε να επιτρέπει στα προγράμματα να καταλάβουν το νόημα της εκάστοτε περιγραφής. Η αναπαράσταση των δεδομένων στηρίζεται στην έννοια των statements, που αποτελούν τριπλέτες της μορφής κατηγόρημα – θέμα – αντικείμενο (predicate – subject – object). Το θέμα και το αντικείμενο μια τριπλέτας είναι τα στοιχεία στα οποία αναφέρεται το κάθε statement, ενώ το κατηγόρημα περιγράφει τη σχέση που έχει το θέμα πάνω στο αντικείμενο. Οι θεμελιώδεις έννοιες που υπάρχουν στο RDF είναι οι εξής:

- **Resources**: Αποτελούν τα στοιχεία για τα οποία θέλουμε να μιλήσουμε. Κάθε resource έχει ένα URI (Universal Resource Identifier). Τα URIs μπορούν να είναι ηλεκτρονικές διευθύνσεις (Unified Resource Locators) ή κάποιου άλλου είδους αναγνωριστικά, όπως αριθμοί, γεωγραφικές τοποθεσίες κλπ., τα οποία δεν είναι απαραίτητο να περιγράφουν την πρόσβαση σε ένα resource.

- **Properties**: Τα properties είναι κάποια ειδικά resources που περιγράφουν τις σχέσεις μεταξύ των στοιχείων για τα οποία θέλουμε να μιλήσουμε. Όπως και τα resources, έτσι και τα properties προσδιορίζονται με τη χρήση URIs.

- **Statements**: Για τα statements, αναφέραμε κάποια πράγματα παραπάνω. Είναι τριπλέτες κατηγορήματος – θέματος – αντικειμένου, οι τιμές των οποίων μπορούν να είναι είτε resources είτε literals (τα literals είναι ατομικές τιμές, κυρίως strings). Μπορούμε να φανταστούμε την κάθε τριπλέτα (x, P, y) ως μία λογική συνάρτηση P(x, y), όπου το δυαδικό κατηγόρημα P συσχετίζει τα στοιχεία x και y.

Ένα RDF έγγραφο αναπαρίσταται μέσω του XML στοιχείου rdf:RDF. Το περιεχόμενο αυτού του στοιχείου είναι ένας αριθμός περιγραφών (descriptions), για τις οποίες χρησιμοποιείται το στοιχείο rdf:Description. Κάθε περιγραφή δηλώνει κάποιο statement για ένα resource. Η αναφορά στο εκάστοτε resource μπορεί να γίνει χρησιμοποιώντας:

- το χαρακτηριστικό **about**, για την αναφορά σε κάποιο ήδη υπάρχον resource,
- το χαρακτηριστικό **ID**, για τη δημιουργία ενός νέου resource και
- χωρίς κάποιο όνομα δημιουργώντας ένα ανώνυμο resource

Σύμφωνα με όλα τα παραπάνω, αν θέλαμε να περιγράψουμε την πληροφορία " Ο κ. Χριστοδουλάκης διδάσκει το μάθημα Συστήματα Βάσεων Δεδομένων " σε RDF/S, θα είχαμε:

Την RDF κλάση Teacher η οποία επιτρέπει την αναπαράσταση δασκάλων :

```
<rdfs:Class rdf:ID= "Teacher"/>
```

Την RDF κλάση Course η οποία επιτρέπει την αναπαράσταση μαθημάτων :

```
<rdfs:Class rdf:ID= "Course"/>
```

Οι δάσκαλοι έχουν την ιδιότητα "teaches", η οποία συσχετίζει τους δασκάλους με τα μαθήματα τα ποια διδάσκουν. Όπως φαίνεται η ιδιότητα "teaches" έχει ως **πεδίο ορισμού (domain)** την κλάση "Teacher" και ως **πεδίο τιμών (range)** την κλάση Course.

```
<rdfs:Property rdf:ID="teaches">
        <rdfs:domain rdf:resource="#Person" />
        <rdfs:range rdf:resource= "#Course"/>
</rdfs:Property>
```

Για παράδειγμα, έστω οι RDF περιγραφές που παρουσιάζονται στην συνέχεια για τις πηγές http://www.music.tuc.gr/Person#StavrosChristodulakis και http://www.music.tuc.gr/Course#DatabaseSystems , που περιγράφουν των δάσκαλο StavroChristodoulaki και το μάθημα DatabaseSystems και έχουν δομηθεί με βάση το RDF Schema που απεικονίζεται παραπάνω.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
         xmlns:Courses="http://www.music.tuc.gr/Courses#">

  <schema: Teacher  rdf:ID=" http://www.music.tuc.gr/Person#StavrosChristodulakis">
        <schema:teaches>rdf:about =" http://www.music.tuc.gr/Course#DatabaseSystems"
       </schema:teaches>
  </schema: Teacher>

  <schema: Course  rdf:ID=" http://www.music.tuc.gr/Course#DatabaseSystems" />

<rdf:RDF>
```

Η RDF είναι μία καθολική γλώσσα που επιτρέπει στους χρήστες να περιγράψουν resources χρησιμοποιώντας ο καθένας το δικό του λεξιλόγιο. Δεν κάνει υποθέσεις σχετικά με τις έννοιες κάποιας περιοχής, ούτε καθορίζει τη σημασιολογία της. Αυτό μπορεί να γίνει από το χρήστη με τη χρήση της RDF Schema (RDFS) [2][3].

Το RDFS είναι ένα λεξιλόγιο που επιτρέπει στους χρήστες να ορίσουν κλάσεις και ιδιότητες για τα διάφορα RDF resources, καθώς και να τα οργανώσει σε ιεραρχίες. Αυτές οι κλάσεις και ιδιότητες καθορίζουν τη σημασιολογία των RDF statements για κάποια συγκεκριμένη περιοχή. Οι κλάσεις μοντελοποιούν τις οντότητες κάποιας περιοχής και τους περιορισμούς που τις διέπουν, ενώ οι ιδιότητες (properties) καθορίζουν τις σχέσεις που μπορεί να υπάρξουν μεταξύ αυτών των οντοτήτων, για την κάθε ιδιότητα ορίζεται κάποιο domain και κάποιο range. Παρ'όλα αυτά, υπάρχουν κάποιες ελλείψεις στην RDFS :

- Το range κάποιας ιδιότητας εφαρμόζεται για όλες τις κλάσεις. Έτσι δεν μπορούμε να δηλώσουμε τον περιορισμό ότι ορισμένες ιδιότητες μπορούν να εφαρμοστούν σε ορισμένες μόνο κλάσεις.

- Δεν υπάρχει η δυνατότητα δυαδικών συνδυασμών, όπως ένωση (union), τομή (intersection) και συμπλήρωμα (complement).

- Δεν μπορούμε να δηλώσουμε τον αριθμό των διακριτών τιμών που μπορεί να πάρει κάποια ιδιότητα (cardinality constraints). Για παράδειγμα, δεν μπορούμε να καθορίσουμε ότι η ιδιότητα "has birth-mother" μπορεί να αναφερθεί μόνο σε ένα πρόσωπο.

- Δεν μπορούμε να δηλώσουμε συγκεκριμένα χαρακτηριστικά των ιδιοτήτων, όπως μεταβατικότητα (transitive), μοναδικότητα (unique) και αντιθετικότητα (inverse).

Στα παραπάνω προβλήματα έρχονται να δώσουν λύσεις οι γλώσσες οντολογιών.

2.7 OWL Web Ontology Language

Η **OWL (Web Ontology Language)** [1] είναι μια πρότυπη γλώσσα περιγραφής οντολογιών στο Διαδίκτυο και αποτελεί συνέχεια και επέκταση της DAML+OIL. Η OWL στοχεύει στην παροχή πλήρους υποστήριξης των χαρακτηριστικών των γλωσσών αναπαράστασης γνώσης, επεκτείνοντας με αυτό τον τρόπο την εκφραστικότητα (expressiveness) της DAML+OIL. Καθώς η πολυπλοκότητα (complexity) της εξαγωγής συμπερασμάτων αυξάνεται όσο αυξάνεται η εκφραστικότητα της γλώσσας, αναπτύχθηκαν τρία είδη

(species) της OWL, που μπορούν να χρησιμοποιηθούν κατά περίπτωση, ανάλογα με τις ανάγκες σε εκφραστικότητα και τους περιορισμούς σε πολυπλοκότητα. Τα τρία είδη της OWL είναι τα εξής:

a) Η **OWL Lite**, που στοχεύει σε χαμηλής πολυπλοκότητας εξαγωγή συμπερασμάτων, αλλά έχει περιορισμένη εκφραστικότητα. Η εξαγωγή συμπερασμάτων από OWL Lite οντολογίες είναι εγγυημένο ότι δίνει συμπεράσματα, τα οποία παράγονται, στη χειρότερη περίπτωση, σε εκθετικό χρόνο. Για να είναι αυτό δυνατό, η OWL Lite δεν υποστηρίζει τις λειτουργίες ένωση (union) και συμπλήρωμα (complement), ενώ οι περιορισμοί πολλαπλότητας των ιδιοτήτων μπορεί να έχουν μόνο τις τιμές 0 και 1.

b) Η **OWL DL (OWL Description Logics)**, που επεκτείνει την εκφραστικότητα της OWL Lite με λειτουργικότητα περιγραφικής λογικής και τύπους δεδομένων. Η OWL DL είναι το πιο εκφραστικό από τα είδη της OWL που εγγυώνται περατότητα και αποτελεσματικότητα της εξαγωγής συμπεασμάτων.

c) Η **OWL Full**, που υποστηρίζει όλα τα χαρακτηριστικά της περιγραφικής λογικής και της RDF. Η OWL Full είναι το πιο εκφραστικό από τα είδη της OWL, αλλά δεν εγγυάται περατότητα και αποτελεσματικότητα της εξαγωγής συμπερασμάτων.

Κάθε ένα από τα τρία είδη αποτελεί επέκταση του προηγούμενου απλούστερου είδους σε ότι αφορά τόσο στο τι μπορεί να εκφραστεί έγκυρα όσο και στο τι μπορεί αξιόπιστα να εξαχθεί ως συμπέρασμα. Έτσι, ισχύουν οι παρακάτω σχέσεις:

- Κάθε έγκυρη(legal) OWL Lite οντολογία είναι και έγκυρη OWL-DL οντολογία.
- Κάθε έγκυρη OWL-DL οντολογία είναι και έγκυρη OWL Full οντολογία.
- Κάθε αξιόπιστο(valid) OWL Lite συμπέρασμα είναι και αξιόπιστο OWL-DL συμπέρασμα.
- Κάθε αξιόπιστο OWL-DL συμπέρασμα είναι και αξιόπιστο OWL Full συμπέρασμα.

Η OWL αναπτύχθηκε με βάση το παράδειγμα (paradigm) της περιγραφικής λογικής(description logic)[106] και χρησιμοποιεί RDF(S) σύνταξη. Τα δομικά στοιχεία των OWL οντολογιών είναι οι κλάσεις, οι ιδιότητες και τα άτομα. Οι OWL κλάσεις, οι OWL ιδιότητες, και τα OWL άτομα ταυτοποιούνται από μοναδικές ταυτότητες, που καθορίζονται στο "rdf:ID" γνώρισμά τους. Επιπλέον, μπορεί να διαθέτουν ετικέτες και σχόλια σε γλώσσα κατανοητή από άνθρωπο, τα οποία δεν επηρεάζουν τη λογική ερμηνεία τους από τα εργαλεία εξαγωγής συμπερασμάτων και ορίζονται μέσω των δομών "rdfs:label" και "rdfs:comment" αντίστοιχα.

Οι **OWL κλάσεις** αναπαριστούν σύνολα ατόμων που διαθέτουν κάποιες κοινές ιδιότητες και ανήκουν στην ίδια κατηγορία. Κάθε OWL άτομο είναι μέλος της κλάσης "owl:Thing" και συνεπώς κάθε νέα OWL κλάση που ορίζεται είναι υποκλάση της κλάσης "owl:Thing". Οι OWL κλάσεις ορίζονται μέσω της δομής "owl:Class" και μπορεί να οριστούν εξ' αρχής, μέσω των λειτουργιών συνόλων τομή (intersection), ένω-

ση και συμπλήρωμα (με τη χρήση των δομών "owl:intersectionOf", "owl:unionOf" και "owl:complementOf" αντίστοιχα) και με απαρίθμηση (enumeration) των ατόμων που ανήκουν σε αυτές (μέσω της δομής "owl:oneOf"). Ιεραρχίες κλάσεων σχηματίζονται με τη χρήση της δομής "rdfs:subClassOf", που δηλώνει ότι μια κλάση "A1" αποτελεί υποκλάση της κλάσης "A", την οποία εξειδικεύει. Μια OWL κλάση μπορεί να είναι υποκλάση περισσότερων από μιας κλάσεων. Επιπλέον, δυο OWL κλάσεις μπορούν να χαρακτηριστούν ως ισοδύναμες (equivalent) ή ξένες (disjoint), μέσω των δομών αντιστοίχησης (mapping) "owl:equivalentClass" και "owl:disjointWith" αντίστοιχα. Παράδειγμα ορισμού OWL κλάσεων αποτελούν οι κλάσεις "Animal" και "Person", ορίζονται σε OWL σύνταξη στην Εικόνα2.6.

```
<owl:Class rdf:ID="Animal"/>

<owl:Class rdf:ID="Person">

        <rdfs:subClassOf rdf:resource="#Animal"/>

</owl:Class>
```

Εικόνα 2.6: Ορισμός των Κλάσεων "Animal" και "Person" σε OWL σύνταξη

Τα **OWL άτομα**(OWL individual) αποτελούν τα μέλη των OWL κλάσεων. Το σύνολο των ατόμων που ανήκουν σε μια συγκεκριμένη κλάση ονομάζεται επέκταση κλάσης (class extension). Ένα άτομο μπορεί να ανήκει σε μία ή περισσότερες κλάσεις. Ένα άτομο συνδέεται με τις κλάσεις στις οποίες ανήκει μέσω της δομής "rdf:type". Τα άτομα διαθέτουν τις ιδιότητες των OWL κλάσεων όπου ανήκουν και υπακούν στους περιορισμούς που υπάρχουν γι' αυτές.

Οι **OWL ιδιότητες** (OWL properties) επιτρέπουν τον ισχυρισμό γενικευμένων κοινά αποδεκτών δεδομένων (facts) για τις κλάσεις και συγκεκριμένων κοινά αποδεκτών δεδομένων για τα άτομα των κλάσεων. Οι ιδιότητες είναι δυαδικές σχέσεις και διακρίνονται σε 2 κατηγορίες:

- Τις **Ιδιότητες Τύπων Δεδομένων (Datatype Properties)**, που συσχετίζουν άτομα που ανήκουν στην OWL κλάση που αποτελεί πεδίο ορισμού της ιδιότητας, με τιμές ενός συγκεκριμένου τύπου δεδομένων, που αποτελεί πεδίο τιμών της ιδιότητας. Το πεδίο τιμών μπορεί να είναι κάποιος XML Schema τύπος ή συγκεκριμένες κυριολεκτικές τιμές.
- Τις **Ιδιότητες Αντικειμένων (Object Properties)**, που συσχετίζουν άτομα που ανήκουν στην OWL κλάση που αποτελεί πεδίο ορισμού της ιδιότητας, με άτομα που ανήκουν στην OWL κλάση που αποτελεί πεδίο τιμών της ιδιότητας. Οι Ιδιότητες Αντικειμένων ορίζονται μέσω της δομής "owl:ObjectProperty".

```
<rdf DatatypeProperty rdf:ID=”hasSurname”>
        <rdfs:domain rdf:resource=”#Person”/>
        <rdfs:range rdf:resource=”http://www.w3.org/2001/XMLSchema#string”/>
</owl:DatatypeProperty>
```

Εικόνα 2.7 : Ορισμός της Ιδιότητας Τύπου Δεδομένων “hasSurname” σε OWL σύνταξη

Οι OWL ιδιότητες μπορεί να καθοριστούν ως συμμετρικές (symmetric), μεταβατικές, μονοσήμαντες και επί (inverse functional). Επιπλέον, υποστηρίζονται οι σχέσεις αντιστοίχησης μεταξύ ιδιοτήτων ισοδυναμία και αντιστροφή (inversion), μέσω των δομών “owl:equivalentProperty” και “owl:inverseOf” αντίστοιχα. Ιεραρχίες ιδιοτήτων μπορούν να σχηματιστούν με τη χρήση της δομής “rdfs:subPropertyOf”, που δηλώνει ότι μια ιδιότητα “I1” αποτελεί υποιδιότητα της ιδιότητας “I”, την οποία εξειδικεύει. Μια OWL ιδιότητα μπορεί να είναι υποιδιότητα περισσότερων από μιας ιδιοτήτων.

Υποστηρίζονται επίσης OWL περιορισμοί μέσω της δομής “owl:Restriction”, που συμπεριλαμβάνουν περιορισμούς τύπου (μέσω των δομών “owl:allValuesFrom” και “owl:someValuesFrom”), περιορισμούς πολλαπλότητας (μέσω των δομών “owl:cardinality”, “owl:minCardinality” και “owl:maxCardinality”) και περιορισμούς τιμής (μέσω της δομής “owl:hasValue”). Ως παράδειγμα, για να εκφραστεί ότι τα πρόσωπα πρέπει να διαθέτουν τουλάχιστον ένα επώνυμο, ο ορισμός της κλάσης “Person” πρέπει να τροποποιηθεί όπως φαίνεται στην Εικόνα 2.8, με την προσθήκη του περιορισμού η πολλαπλότητα της ιδιότητας “hasSurname” να είναι τουλάχιστον 1. Ο περιορισμός αυτός δε μπορεί να εκφραστεί σε RDFS.

```
<owl:Class rdf:ID=”Person”>
    <rdfs:subClassOf rdf:resource=”#Animal”/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource=”#hasSurname”/>
        <owl:minCardinality
rdf:datatype=”http://www.w3.org/2001/XMLSchema#integer”>1</owl:minCardinality>
        </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Εικόνα 2.8: Τροποποίηση του ορισμού της Κλάσης “Person”, ώστε όλα τα Πρόσωπα να έχουν τουλάχιστον ένα Επώνυμο

2.8 SPARQL Query Language for RDF

Η SPARQL [5] είναι μία γλώσσα ερωτήσεων (query language) η οποία απευθύνεται σε RDF δεδομένα. Το όνομα της προέρχεται από τα αρχικά Simple Protocol and RDF Query Language. Η SPARQL αποτελεί W3C Recommendation από τον Ιανουάριο του 2008.

2.8.1 Σχηματομορφές Γράφων (Graph Patterns)

Η γλώσσα ερωτήσεων SPARQL βασίζεται στο ταίριασμα σχηματατομορφών γράφων (graph pattern matching) για να εκφράσει τις ερωτήσεις. Συνδυάζοντας απλούστερες σχηματομορφές μπορεί να δημιουργήσει πολύ πιο πολύπλοκες, οι οποίες διαχωρίζονται ως :

Τριπλέτες Σχηματομορφών (Triples Patterns)

Οι Τριπλέτες σχηματομορφών (triples patterns) αποτελούν την απλούστερη μορφή σχηματομορφής γράφου. Αποτελούνται από Υποκείμενο (Subject) , Κατηγόρημα(Predicate) και Αντικείμενο(Object) όπως και οι RDF Τριπλέτες , με την διαφοροποίηση ότι έχουν την δυνατότητα να περιέχουν και μεταβλητές στις θέσεις του υποκειμένου ,του κατηγορήματος ή του αντικειμένου. Οι μεταβλητές στην SPARQL συντάσσονται με χρήση των χαρακτήρων ? ή $ ως πρόθεμα.

Τα παρακάτω παραδείγματα αποτελούν Τριπλέτες Σχηματομορφών :

- <http://example.org/book/book1> dc:title ?title
  Η τριπλέτα αυτή έχει ως υποκείμενο το Uri http://example.org/book/book1, ως κατηγόρημα την ιδιότητα title που ορίζεται στο namespace dc και αντικείμενο την μεταβλητή title.
- ?x ns:Firstname "john"
  Η τριπλέτα αυτή έχει ως υποκείμενο την μεταβλητή x, ως κατηγόρημα την ιδιότητα Firstname που ορίζεται στο namespace ns και αντικείμενο την σταθερά (literal) John.
- ?x ?p "John"
  Η τριπλέτα αυτή έχει ως υποκείμενο την μεταβλητή x, ως κατηγόρημα την μεταβλητή p και αντικείμενο την σταθερά (literal) John.
- ?x ?p ? o
  Η τριπλέτα αυτή έχει ως υποκείμενο την μεταβλητή x, ως κατηγόρημα την μεταβλητή p και ως αντικείμενο την μεταβλητή o.

Βασικές Σχηματομορφές Γράφων (Basic Graph Patterns)

Οι βασικές Σχηματομορφές γράφων είναι ένα σύνολο από Τριπλέτες σχηματομορφών στις οποίες εφαρμόζεται η σύζευξη ανάμεσα τους. Οι βασικές Σχηματομορφές γράφων έχουν την δυνατότητα να περιέχουν και φίλτρα (FILTER). Τα φίλτρα χρησιμοποιούνται για να περιορίσουν τις λύσεις, σε αυτές για τις οποίες η αποτίμηση της έκφρασης του φίλτρου είναι true. Τα φίλτρα περιέχουν εκφράσεις οι οποίες περιέχουν τελεστές (πχ <, > , = , isliteral, isbound κτλ) σταθερές (πχ 10, 3.14 , "john" , true , false κλτ) και μεταβλητές.

Το παρακάτω παράδειγμα αποτελεί μια Βασική Σχηματομορφή Γράφων:

```
?X ns:FirstName "John" .
?X ns:LastName ?lastName.
?X ns:Address ?addr.
```

Το παρακάτω παράδειγμα αποτελεί μια Βασική Σχηματομορφή Γράφων η οποία περιέχει και φίλτρο :

```
?X ns:FirstName "John" .
?X ns:LastName ?lastName.
?X ns:Address ?addr.
FILTER ( ?lastName = "Turing" )
```

Ομάδα Σχηματομορφών Γράφων (Group Graph Patterns)

Στις ερωτήσεις SPARQL, οι ομάδες σχηματομορφών γράφων είναι η πιο γενική κατηγορία σχηματομορφών, καθώς μπορεί να περιέχει όλες τις άλλες. Οι ομάδες σχηματομορφών γράφων περικλείονται από αγκύλες ({ }).

Τα παρακάτω παραδείγματα αποτελεί μια Ομάδα Σχηματομορφών Γράφων:

```
{ ?X ns:FirstName "John" .
  ?X ns:LastName ?lastName. }
```

Η παραπάνω σχηματομορφή γράφου εφόσον αποτιμηθεί σε RDF δεδομένα, στην μεταβλητή X θα αντιστοιχηθούν τα στιγμιότυπα κλάσεων οι οποίες έχουν στην ιδιότητα ns:FirstName την τιμή John και επίσης έχουν κάποια τιμή στην ιδιότητα ns:LastName. Ενώ στην μεταβλητή lastName θα αντιστοιχηθούν οι τιμές τις ιδιότητας ns:LastName για τα στιγμιότυπα που αντιστοιχηθήκαν στην μεταβλητή X.

Η παραπάνω σχηματομορφή γράφου είναι ισοδύναμη με:

```
{ {?X ns:FirstName "John" .}
 {?X ns:LastName ?lastName.} }
```

Προαιρετικές Σχηματομορφές Γράφων (Optional Graph Patterns)

Είναι πολλές φορές χρήσιμο να υποστηρίζονται ερωτήσεις οι οποίες να επιτρέπουν κάποια πληροφορία να προσδεθεί στην λύση όταν αυτή υπάρχει, αλλά χωρίς να απορρίπτονται οι λύσεις όταν αυτή δεν είναι διαθέσιμη. Οι Προαιρετικές Σχηματομορφές Γράφων παρέχουν αυτή την δυνατότητα, καθώς το προαιρετικό μέρος του γράφου εφόσον επαληθευτεί επιστέφει πληροφορίες, ενώ στην περίπτωση που δεν επαληθεύεται δεν απορρίπτονται λύσεις. Ένα παράδειγμα αυτού είναι η ερώτηση : "Επέστεψε το μικρό όνομα των ατόμων και το επώνυμο τους και στην περίπτωση που τα άτομα έχουν ψευδώνυμο επέστεψε το και αυτό". Οι Προαιρετικές Σχηματομορφές Γράφων δηλώνονται με την χρήση της λέξης OPTIONAL και περιέχουν μια Ομάδα Σχηματομορφών Γράφων.

Τα παρακάτω παράδειγμα αποτελεί μια Προαιρετική Σχηματομορφή Γράφων:

```
OPTIONAL{  ?X ns:FirstName "John" .
           ?X ns:LastName ?lastName. }
```

Η παραπάνω σχηματομορφή θα συμπεριληφθεί στην λύση εφόσον είναι δυνατόν να αποτιμηθεί, αλλιώς θα αγνοηθεί. Δηλαδή εφόσον υπάρχει κάποιο στιγμιότυπο με μικρό όνομα(FirstName) John και να έχει και κάποια τιμή στο επίθετο(LastName).

Εναλλακτικές Σχηματομορφές Γράφων (Alternative Graph Patterns)

Αν παραπάνω από μια σχηματομορφή γράφου μπορεί να ταιριάζει, η γλώσσα SPARQL προσφέρει την δυνατότητα δήλωσης εναλλακτικών σχηματομορφών γράφων, με την χρήση της λέξης UNION ανάμεσα σε Ομάδες Σχηματομορφών Γράφων. Οι λύσεις των εναλλακτικών σχηματομορφών γράφων αποτελούνται από την ένωση όλων των λύσεων των σχηματατομορφών που εμφανίζονται ανάμεσα στον τελεστή UNION.

Τα παρακάτω παράδειγμα αποτελεί μια Εναλλακτική Σχηματομορφή Γράφων:

```
{ {?x ns:FirstName ?firstName .} UNION {?x ns:LastName ?lastName. } }
```

2.8.2 Μορφές Ερωτήσεων (Query Forms)

Η γλώσσα SPARQL έχει τέσσερεις μορφές ερωτήσεων, αυτές οι μορφές χρησιμοποιούν τις τιμές που έχουν ανατεθεί στις μεταβλητές και δημιουργούν σύνολα αποτελεσμάτων (result sets) ή RDF γράφους.

- **SELECT** Οι ερωτήσεις αυτής της μορφής, επιστρέφουν για όλες ή μέρος των μεταβλητών, τις τιμές που έχουν ανατεθεί σε αυτές. Η σύνταξη SELECT * επιστρέφει τις τιμές για όλες τις μεταβλητές, σε αντίθετη περίπτωση μετά το SELECT αναφέρονται οι μεταβλητές για τις οποίες θα επιστραφούν αποτελέσματα.

  Έστω η SELECT SPARQL ερώτηση η οποία απευθύνεται στα παρακάτω RDF δεδομένα, και επιστρέφει τα ονόματα των ατόμων τα οποία το ένα γνωρίζει το άλλο.

  RDF Data

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:name   "Alice" .
_:a    foaf:knows  _:b .
_:a    foaf:knows  _:c .

_:b    foaf:name   "Bob" .
_:c    foaf:name   "Clare" .
```

  Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY
WHERE
  { ?x foaf:knows ?y .
    ?x foaf:name ?nameX .
    ?y foaf:name ?nameY .
  }
```

  Result

nameX	nameY
"Alice"	"Bob"
"Alice"	"Clare"

  Σημείωση : *Ως κενός κόμβος στο RDF μοντέλο δεδομένων ορίζεται ένας κόμβος που δεν ούτε κόμβος IRI , ούτε κόμβος Literal. Ένας κενός κόμβος είναι ένας μοναδικός (unique) κόμβος που όμως δεν έχει κάποιο "φυσικό" (intrinsic) όνομα. Και συμβολίζεται με χρήση των χαρακτήρων _: ως πρόθεμα. Με την δήλωση @prefix ορίζεται το namespace στο οποίο βρίσκεται το rdf dataset..*

- **CONSTRUCT** Οι ερωτήσεις αυτής της μορφής, επιστρέφουν έναν RDF γράφο του οποίου η μορφή έχει προσδιοριστεί από έναν γράφο πρότυπο (graph pattern). Ο γράφος δημιουργείται περνώντας κάθε λύση από την ακολουθία λύσεων και αντικαθιστώντας τις τιμές των μεταβλητών, στις αντίστοιχες μεταβλητές των σχηματομορφών τριπλέτων του RDF γράφου. Ο τελικός γράφος παράγεται από την ένωση των σχηματομορφών τριπλέτων που έχουν δημιουργηθεί για όλες τις λύσεις της ακολουθίας των λύσεων. Στην περίπτωση που κάποια μεταβλητή μιας σχηματομορφής τριπλέτας δεν έχει ανατεθεί τιμή (δηλαδή η μεταβλητή είναι Unbound), τότε η τριπλέτα αυτή δεν περιλαμβάνεται στον τελικό γράφο. Επίσης ο γράφος πρότυπο μπορεί να περιέχει τριπλέτες σχηματομορφών που δεν περιέχουν μεταβλητές (γνωστές ως ground or explicit triples) και οι οποίες θα περιέχονται στον τελικό γράφο που θα επιστραφεί. Υπάρχει η δυνατότητα ο γράφος πρότυπο να περιέχει και κενούς κόμβους (blank nodes), σε αυτή την περίπτωση για κάθε λύση από την ακολουθία λύσεων, θα δημιουργείται και ένας διαφορετικός κενός κόμβος. Αν ένας κενός κόμβος εμφανίζεται σε δυο σημεία στον πρότυπο γράφο, τότε για κάθε λύση θα είναι ο ίδιος κενός κόμβος και στις δυο θέσεις, αλλά θα είναι διαφορετικοί για διαφορετικές λύσεις.

Έστω η CONSTRUCT SPARQL ερώτηση η οποία δημιουργεί έναν RDF γράφο ο οποίος "βασίζεται" στο vcard vocabulary [54], ενώ τα RDF δεδομένα είναι "βασισμένα" στο foaf vocabulary [55].

RDF Data

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:name   "Alice" .
_:a    foaf:mbox   <mailto:alice@example.org> .
```

Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT   { <http://example.org/person#Alice> vcard:FN ?name
}
WHERE       { ?x foaf:name ?name }
```

Result

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://example.org/person#Alice> vcard:FN "Alice" .
```

Έστω η CONSTRUCT SPARQL ερώτηση η οποία περιέχει τον κενό κόμβο _:v σε τρεις θέσεις στο πρότυπο γράφο.

RDF Data

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:givenname   "Alice" .
_:a    foaf:family_name "Hacker" .

_:b    foaf:firstname   "Bob" .
_:b    foaf:surname     "Hacker" .
```

Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x  vcard:N _:v .
            _:v vcard:givenName ?gname .
            _:v vcard:familyName ?fname }
WHERE
 {
    { ?x foaf:firstname ?gname } UNION  { ?x foaf:givenname
?gname } .
    { ?x foaf:surname   ?fname } UNION  { ?x foaf:family_name
?fname } .
 }
```

Result

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:v1 vcard:N         _:x .
_:x vcard:givenName  "Alice" .
_:x vcard:familyName "Hacker" .

_:v2 vcard:N         _:z .
_:z vcard:givenName  "Bob" .
_:z vcard:familyName "Hacker" .
```

Όπως παρατηρείται ο κενός κόμβος :_v που εμφανίζεται στον πρότυπο γράφο, εμφανίζεται ως _:x και ως _:z στον γράφο των αποτελεσμάτων. Ο κενός κόμβος έχει κοινό όνομα για κάθε λύση(Εφόσον όπως έχει αναφερθεί υπάρχει η δυνατότητα ο γράφος πρότυπο να περιέχει και κενούς κόμβους (blank nodes), όμως σε αυτή την περίπτωση για κάθε λύση από την ακολουθία λύσεων, θα δημιουργείται και ένας διαφορετικός

κενός κόμβος). Επίσης παρατηρείται να έχει αλλάξει το όνομα των κενών κόμβων _:a και _:b που εμφανίζονται στα RDF δεδομένα σε _:v1 και _:v2 (Καθώς το όνομα δεν έχει κάποια σημασία) .

- **ASK** Οι ερωτήσεις αυτής της μορφής, μπορούν να χρησιμοποιηθούν από της εφαρμογές για να εκλεχθεί αν μια ερώτηση έχει λύση. Δεν γίνεται επιστροφή πληροφορίας για πιθανές λύσεις της ερώτησης, παρά μόνο αν η ερώτηση έχει λύση ή όχι, επιστρέφοντας yes ή no .

RDF Data

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .


_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

Result

```
yes
```

- **DESCRIBE** Οι ερωτήσεις αυτής της μορφής, επιστέφουν έναν RDF γράφο ο οποίος περιλαμβάνει RDF δεδομένα τα οποία "περιγράφουν" τις πηγές (Resources). Αντίθετα με τις CONSTRUCT ερωτήσεις η μορφή του γράφου δεν προσδιορίζεται από την ερώτηση αλλά ορίζεται από το SPARQL query engine. Η σχηματομορφή (Pattern) του ερωτήματος χρησιμοποιείται για να δημιουργηθεί ένα σύνολο αποτελεσμάτων. Οι ερωτήσεις αυτής της μορφής για κάθε πηγή που εμφανίζεται στα αποτελέσματα ή για κάθε πυγή που προσδιορίζεται από IRI, δημιουργούν μια "Περιγραφή"(Description) βασισμένη στα RDF δεδομένα. Τα αποτέλεσμα που παράγει μια ερώτηση αυτής της μορφής δεν είναι καθορισμένα από την προδιαγραφή της γλώσσας, αλλά καθορίζονται από την υλοποίηση του SPARQL query engine που εκτελεί την ερώτηση.

Query

```
PREFIX ent:  <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

Result

([5] Πιθανόν να επιστρέφει μια "περιγραφή" του εργαζόμενου και μερικές άλλες χρήσιμες πληροφορίες)

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg:  <http://org.example.com/employees#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#>

_:a     exOrg:employeeId    "1234" ;

        foaf:mbox_sha1sum   "ABCD1234" ;
        vcard:N
         [ vcard:Family       "Smith" ;
           vcard:Given        "John"  ] .

foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```

2.8.3 Τροποποιητές Ακολουθίας Λύσεων (Solution Sequence Modifiers)

Οι σχηματομορφές (Patterns) των ερωτήσεων δημιουργούν μη ταξινομημένες ακολουθίες λύσεων, υπάρχει η δυνατότητα εφαρμογής τροποποιητών ώστε να δημιουργηθεί μια νέα ακολουθία, η οποία είναι αυτή που θα επιστραφεί από την ερώτηση. Οι τροποποιητές εφαρμόζονται σε ερωτήσεις μορφής SELECT, CONSTRUCT ή DESCRIBE και όχι σε ερωτήσεις μορφής ASK, επίσης οι τροποποιητές DISTINCT και REDUCE εφαρμόζονται μόνο σε SELECT ερωτήσεις. Οι τροποποιητές ακολουθιών λύσεων είναι οι παρακάτω :

- **ORDER BY** Ο τροποποιητής ORDER BY προσδιορίζει την σειρά της ακολουθίας των λύσεων. Η δομή ORDER BY, ακολουθείται από ένα σύνολο μεταβλητών με βάση των οποίων θα πραγματοποιηθεί η ταξινόμηση της ακολουθίας των λύσεων Προαιρετικά υπάρχει η δυνατότητα, να προσδιοριστεί η φορά της ταξινόμησης, δηλώνοντας ASC ή DESC για αύξουσα ή φθίνουσα ταξινόμηση αντίστοιχα, στην περίπτωση που δεν προσδιορίζεται η φορά ταξινόμησης, εφαρμόζεται αύξουσα.

  Η παρακάτω ερώτηση θα ταξινομήσει την ακολουθία λύσεων, αρχικά με αύξουσα φορά με βάση το όνομα και στην περίπτωση που υπάρχουν παραπάνω από μια λύση με το ίδιο όνομα ,θα τις ταξινομήσει με βάση την τιμή της μεταβλητής ?emp με φθίνουσα σειρά.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC(?emp)
```

- **DISTINCT** Ο τροποποιητής DISTINCT εξαλείφει την ύπαρξη διπλών λύσεων (duplicate). Ο τροποποιητής DISTINCT εφαρμόζεται μόνο στις SELECT ερωτήσεις.

Έστω η ερώτηση χωρίς χρήση του **DISTINCT**

RDF Data

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:x    foaf:name   "Alice" .
_:x    foaf:mbox   <mailto:alice@example.com> .

_:y    foaf:name   "Alice" .
_:y    foaf:mbox   <mailto:asmith@example.com> .

_:z    foaf:name   "Alice" .
_:z    foaf:mbox   <mailto:alice.smith@example.com> .
```

Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE { ?x foaf:name ?name }
```

Result

name
"Alice"
"Alice"
"Alice"

Η ερώτηση με χρήση του **DISTINCT**

RDF Data

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:x    foaf:name   "Alice" .
_:x    foaf:mbox   <mailto:alice@example.com> .

_:y    foaf:name   "Alice" .
_:y    foaf:mbox   <mailto:asmith@example.com> .

_:z    foaf:name   "Alice" .
_:z    foaf:mbox   <mailto:alice.smith@example.com> .
```

Query

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name WHERE { ?x foaf:name ?name }
```

Result

name
"Alice"

- **REDUCED** Ο τροποποιητής REDUCED χρησιμοποιείται για τον χειρισμό των διπλών λύσεων, όμως σε αντίθεση με τον DISTINCT δεν εξαλείφει τελείως τις διπλές λύσεις, αλλά επιτρέπει να εμφανίζονται με πολλαπλότητα (cardinality) μεταξύ του ένα και της πολλαπλότητας της λύσης πριν την εφαρμογή των REDUCED και DISTINCT τροποποιητών. Η επιλογή της πολλαπλότητας καθορίζεται από το SPARQL query engine που εκτελεί την ερώτηση. Ο τροποποιητης REDUCE εφαρμόζεται μόνο στις SELECT ερωτήσεις. Η εφαρμογή του τροποποιητή REDUCED έναντι του DISTINCT στο προηγούμενο ερώτημα θα είχε ως αποτέλεσμα μια , δυο ή τρεις λύσεις .

- **OFFSET** Ο τροποποιητής OFFSET έχει ως αποτέλεσμα, η ακολουθία λύσεων να δημιουργείται στην περίπτωση που περιέχει περισσότερες λύσεις από αυτές που προσδιορίζει ο OFFSET .

  Η παρακάτω ερώτηση θα επιστρέψει αποτελέσματα στην περίπτωση που ανατεθούν πάνω από δέκα τιμές στην μεταβλητή name .

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE   { ?x foaf:name ?name }
OFFSET  10
```

- **LIMIT** Ο τροποποιητής LIMIT εφαρμόζει ένα άνω όριο στον αριθμοί των λύσεων που επιστρέφονται. Στην περίπτωση που ο αριθμός των λύσεων είναι μεγαλύτερος από αυτόν που προσδιορίζεται από τον LIMIT , επιστρέφονται τόσες λύσεις όσες ορίζει ο LIMIT . Εάν ο LIMIT έχει οριστεί μηδέν (0) δεν έχει καμιά επίπτωση .

  Η παρακάτω ερώτηση θα επιστρέψει το πολύ είκοσι (20) αποτελέσματα .

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE   { ?x foaf:name ?name }
LIMIT 20
```

2.8.4 Ανάλυση της Σημασιολογίας της Γλώσσας Ερωτήσεων SPARQL (SPARQL Semantics)

Στην παρούσα παράγραφο γίνεται ανάλυση της σημασιολογίας, των βασικών εννοιών της γλώσσας ερωτήσεων SPARQL. Η κατανόηση και η διατύπωση των βασικών ορισμών είναι απαραίτητη για τις παρακάτω διαδικασίες και αποδείξεις. Οι ορισμοί που ακολουθούν έχουν οριστεί στην προδιαγραφή(specification) της γλώσσας SPARQL [5]. Λόγω ελλιπούς διατύπωσης ορισμών από την προδιαγραφή της γλώσσας, χρησιμοποιηθήκαν και ορισμοί από τις δημοσιεύσεις [11][12] οι οποίες έχουν υιοθετηθεί από την προδιαγραφή της γλώσσας SPARQL.

Ορισμός 2.1 RDF Όρος (RDF Term / RDF-T) Ορίζεται το σύνολο **I** που περιέχει όλα τα IRI (Internationalized Resource Identifier). Ορίζεται το σύνολο **RDF-L** που περιέχει όλες τις RDF σταθερές (RDF Literals). Ορίζεται το σύνολο **RDF-B** που περιέχει όλους τους κενούς κόμβους (blank nodes) των RDF γράφων. Το σύνολο των RDF όρων (**RDF-T**) ισούται με : **I** union **RDF-L** union **RDF-B.**

Τα σύνολα **I**, **RDF-L**, **RDF-B** είναι αποσυνδεδεμένα-μη επικαλυπτόμενα (disjoin)

Σημείωση : *Ως κενός κόμβος στο RDF μοντέλο δεδομένων ορίζεται ένας κόμβος που δεν είναι IRI κόμβος, ούτε κόμβος Literal. Ένας κενός κόμβος είναι ένας μοναδικός (unique) κόμβος που όμως δεν έχει κάποιο "φυσικό" (intrinsic) όνομα.*

Ορισμός 2.2 RDF Τριπλέτα (RDF Triple) Μια τριπλέτα για την οποία ισχύει : **(s,p,o) ϵ (I** U **B** U **L)** x **I** x **(I** U **B** U **L)** ονομάζετε RDF Τριπλέτα (RDF Triple)

Με βάση το παραπάνω καρτεσιανό γινόμενο θα ισχύει :

Η RDF τριπλέτα αποτελείται :

- Υποκείμενο (Subject) : μπορεί να είναι IRI (I) ή κενός κόμβος (B) ή σταθερά (L)
- Κατηγόρημα (Predicate or Property) : μπορεί να είναι IRI(I)
- Αντικείμενο (Object) : μπορεί να είναι IRI (I) ή κενός κόμβος (B) ή σταθερά (L)

Ορισμός 2.3 RDF Γράφος (RDF Graph) RDF γράφος είναι ένα σύνολο από RDF τριπλέτες .

Ορισμός 2.4 RDF Σύνολο Δεδομένων (RDF Dataset) RDF Σύνολο Δεδομένων είναι ένα σύνολο { G , (<U_1>,G_1) , (<U_2>,G_2),…(<U_n>,G_n) } όπου το G και G_i είναι RDF Γράφοι και κάθε <Ui> είναι IRI. Κάθε IRI είναι διαφορετικό.

G ονομάζετε ο default γράφος και (<U_i>,G_i) ονομάζονται επώνυμοι γράφοι (named graph)

Ορισμός 2.5 Μεταβλητή (Query Variable) Μεταβλητή είναι ένα στοιχείο του συνόλου **V**, όπου V είναι ένα σύνολο αποσυνδεδεμένο-μη επικαλυπτόμενο(δηλαδή είναι δυο σύνολα ανεξάρτητα, χωρίς κοινά στοιχεία- dsjoin) με το σύνολο RDF-T και έχει ως στοιχεία τις μεταβλητές. Η μεταβλητή σύμφωνα με την γραμματική της γλώσσας συντάσσεται με το πρόθεμα '$' ή '?' ακολουθούμενο από το όνομα της μεταβλητής.

Ορισμός 2.6 Τριπλέτα Σχηματομορφής (Triple Pattern) Τριπλέτα Σχηματομορφής είναι στοιχείο του συνόλου (**RDF-T** union **V**) x (**I** union **V)** x (**RDF-T** union **V**).Όπως φαίνεται και από την παραπάνω σχέση, η Τριπλέτα Σχηματομορφής είναι οι RDF τριπλέτες με δυνατότητα ύπαρξης και μεταβλητών (για παραδείγματα βλέπε υπό-ενότητα 2.8.1).

Σημείωση: *[5] Όπως παρατηρείτε από τον ορισμό, στην θέση του υποκειμένου επιτρέπεται η ύπαρξη RDF σταθεράς (Literal). Ωστόσο στην γλώσσα SPARQL δεν επιτρέπετε στην θέση του υποκειμένου η ύπαρξη RDF σταθεράς.*

Ορισμός 2.7 Βασικές Σχηματομορφές Γράφων (Basic Graph Patterns – BGP) Ως Βασικές Σχηματομορφές Γράφων ονομάζεται ένα σύνολο από τριπλέτες σχηματομορφής. Η κενή σχηματομοφή γράφου (empty graph pattern) είναι μια Βασική Σχηματομορφή Γράφου η οποία είναι κενό σύνολο . Μια βασική σχηματομορφή γράφου μπορεί επίσης να περιέχει και φίλτρα (filters) (για παραδείγματα βλέπε υπό-ενότητα 2.8.1).

Σημείωση: *Τα φίλτρα (FILTERS) χρησιμοποιούνται για να περιορίσουν τις λύσεις, σε αυτές για τις οποίες η αποτίμηση της έκφρασης του φίλτρου είναι true. Οι εκφράσεις των φίλτρων δημιουργείται από στοιχεία του συνόλου V ∪ T , σταθερές (constants) , τα λογικά συνδετικά ($\neg, \wedge, \vee$) , τα σύμβολα ($\leq, <, \geq, >, =$) και τέλος τους ενσωματωμένους (built-in) τελεστές όπως bound, isBlank κτλ (βλέπε υπό-ενότητα 4.14).*

Ορισμός 2.8 SPARQL Σχηματομορφή Γράφου(Graph Pattern) ορίζεται αναδρομικά ως :

a) Μια Βασική Σχηματομορφή Γράφου είναι Σχηματομορφή Γράφου .

b) Αν P_1 και P_2 είναι σχηματομορφές γράφων τότε (P_1 AND P_2) , (P_1 UNION P_2) και (P_1 OPT P_2) είναι σχηματομορφές γράφων, οπού AND , UNION και OPT είναι τελεστές της γλώσσας SPARQL.

c) Αν P_1 είναι σχηματομορφή γράφων και R μια έκφραση φίλτρου, τότε και το (P_1 FILTER R) είναι σχηματομορφή γράφων.

Σημείωση : *Οι μεταβλητές που εμφανίζονται σε μια έκφραση φίλτρου R, δεν είναι απαραίτητο να περιέχονται στην σχηματομορφή P. Λόγω ότι δεν είναι ξεκάθαρο από την παρούσα προδιαγραφή της γλώσσας SPARQL, τέτοιων περιπτώσεων, στην παρούσα προσέγγιση θεωρείται ότι οι μεταβλητές του φίλτρου περιέχονται στην σχηματομορφή P.*

Σημείωση *: [5] Ο τελεστής AND έχει μεγαλύτερη προτεραιότητα έναντι του τελεστή OPT, επίσης για τον OPT ισχύει η αριστερή προσεταιριστική ιδιότητα.*

Ορισμός 2.9 Ταίριασμα Υπό-Γράφου (Subgraph Matching) Έστω ο RDF γράφος G και P μια βασική σχηματομορφή γράφου. Η αποτίμηση (evaluation) του P στον G συμβολίζεται ως $\mathbf{[[P]]_G}$ και ορίζεται ως ένα σύνολο από αντιστοιχίες.

$$\mathbf{[[P]]_G = \{\ \mu : V \to T \mid dom(\mu) = var(P) \text{ και } \mu(P) \subseteq G\}}$$

Αν το $\mu \in [[P]]_G$ τότε θεωρείται ότι το μ είναι λύση του P στον G.

Όπου var(P) είναι το σύνολο των μεταβλητών που εμφανίζονται στην βασική σχηματομορφή γράφου P.

Σημείωση: *Για κάθε RDF γράφο G ισχύει ότι* $[[\emptyset]]_G = \{\mu_\emptyset\}$ *.Που σημαίνει ότι η αποτίμηση μιας κενής βασικής σχηματομορφή γράφου(empty basic graph pattern) σε οποιοδήποτε γράφο, έχει ως αποτέλεσμα πάντα ένα σύνολο, που περιέχει μόνο την κενή αντιστοιχία.*

Ορισμός 2.10 Αντιστοίχηση Λύσεων (Solution Mapping) Αντιστοίχηση λύσεων, είναι η αντιστοίχηση ενός συνόλου από μεταβλητές, σε ένα σύνολο από RDF όρους . Αντιστοίχηση λύσεων **μ** , είναι μια μερική συνάρτηση $\mathbf{\mu : V \to T}$. Το πεδίο ορισμού (**dom**(μ)) της συνάρτησης είναι υπό-σύνολο του V .

Ορισμός 2.11 Συμβατές Αντιστοιχήσεις (Compatible Mappings) Δυο αντιστοιχήσεις $\mathbf{\mu_1 : V \to T}$ και $\mathbf{\mu_2 : V \to T}$ είναι συμβατές αν για κάθε $?X \in dom(\mu_1) \cap dom(\mu_2)$ ισχύει $\mu_1(?X) = \mu_2(?X)$.

Σημείωση: *Δυο αντιστοιχήσεις* μ_1, μ_2 *με αποσυνδεμένα -μη επικαλυπτόμενα (disjoin) πεδία ορισμού* ($dom(\mu_1) \cap dom(\mu_2)=\emptyset$) *είναι πάντα συμβατές. Επίσης η κενή αντιστοίχηση (empty mapping)* $\mu_\emptyset$ *είναι συμβατή με όλες τις αντιστοιχήσεις.*

Ορισμός 2.12 Σύνολα Αντιστοιχήσεων και Τελεστές (Set of Mappings and Operations) Θεωρούμε ότι τα Ω_1 Ω_2 είναι σύνολα από αντιστοιχίες. Ορίζονται οι τελεστές σύζευξης (**Join**) , ένωσης (**Union**) και διαφοράς (**Difference**) μεταξύ Ω_1 , Ω_2 ως :

$$\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ είναι συμβατές αντιστοιχήσεις}\}$$

$$\Omega_1 \bigcup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \ or \ \mu \in \Omega_2 \ \}$$

$$\Omega_1 \setminus \Omega_2 = \{\mu \in \Omega_1 \mid \text{για ολα τα } \mu' \in \Omega_2 \text{ , } \mu \text{ και } \mu' \text{ δεν είναι συμβατές αντιστοιχήσεις } \}$$

Με βάση τους παραπάνω ορισμούς ορίζεται η αριστερή εξωτερική σύζευξη (**Left Outer Join**) ως :

$$\Omega_1 \mathbin{⟕} \Omega_2 = (\Omega_1 \bowtie \Omega_2) \bigcup (\Omega_1 \setminus \Omega_2)$$

Ορισμός 2.13 Αποτίμηση Σχηματομορφής Γράφου (Graph Pattern) Έστω D ένα RDF Σύνολο Δεδομένων και G ένας RDF γράφος στο D .Η αποτίμηση μιας Σχηματομορφής Γράφου (Graph Pattern) σε ένα γράφο G και σε ένα D RDF Σύνολο Δεδομένων, συμβολίζεται ως $[[.]]_G^D$ και ορίζεται αναδρομικά ως :

$$[[\ (P_1 \ \text{AND} \ P_2)\ ]]_G^D = [[P_1]]_G^D \bowtie [[P_1]]_G^D$$

$$[[\ (P_1 \ \text{UNION} \ P_2)\ ]]_G^D = [[P_1]]_G^D \bigcup [[P_1]]_G^D$$

$$[[\ (P_1 \ \text{OPT} \ P_2)\ ]]_G^D = [[P_1]]_G^D \mathbin{⟕} [[P_1]]_G^D$$

$$[[\ (P \ \text{FILTER} \ R)\ ]]_G^D = \{\mu \in [[P]]_G^D \mid \mu \models R\}$$

Σημείωση: *Με το* $\mu \models R$ *συμβολίζεται ότι το* μ *ικανοποιεί την R έκφραση φίλτρου.*

Ορισμός 2.14 Ισοδύναμες (equivalent) Σχηματομορφές Γράφων Δυο σχηματομορφές γράφων P_1 και P_2 είναι ισοδύναμες (equivalent) , $\mathbf{P_1 \equiv P_2}$, αν $\mathbf{[[P_1]]^D = [[P_2]]^D}$ για κάθε RDF Σύνολο Δεδομένων D .

Ορισμός 2.15 Έστω $\{t_1,t_2...t_n\}$ είναι μια βασική σχηματομορφή γράφων, όπου $n \geq 1$ και κάθε t_i είναι τριπλέτα σχηματομορφής ($1 \leq i \leq n$) . Τότε για κάθε για κάθε RDF Σύνολο Δεδομένων D ισχύει :

$[[\ \{t_1,t_2...t_n\}\ ]]^D = [[\ \{t_1\} \text{ AND } \{t_2\} \text{ AND } ...\text{AND } \{t_n\}\ ]]^D$

Ορισμός 2.16 [11] **Καλά Σχεδιασμένες (Well Designed) Σχηματομορφές Γράφων** Μια σχηματομορφή γράφου P, ονομάζετε "καλά σχεδιασμένη" (Well Designed) αν για κάθε εμφάνιση του υπόγράφου P' =(P_1 OPT P_2) στον P και για κάθε μεταβλητή ?X που εμφανίζεται στον P , ισχύει η εξής συνθήκη:

Αν ?X εμφανίζεται στον P_2 και έξω από τον P' ,τότε πρέπει να εμφανίζετε και στον P_1

Σημείωση: *Τις σχηματομορφές γράφων που δεν αποτελούν καλά σχεδιασμένες σχηματομορφές γράφων , θα τις ονομάζουμε "όχι καλά σχεδιασμένες"(no-well designed) σχηματομορφές γράφων.*

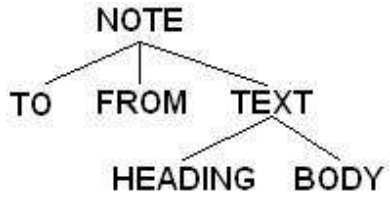
2.9 XML Beans

Ο όρος "XML data binding" αναφέρεται στην αναπαράσταση XML εγγράφων με αντικείμενα. Τα αντικείμενα αυτά χρησιμοποιούν ένα σχήμα (classes), σχεδιασμένο ειδικά για τα δεδομένα που υπάρχουν σε αυτά τα έγγραφα. Αυτό επιτρέπει στις εφαρμογές να διαχειρίζονται δεδομένα τα οποία υπάρχουν σε XML μορφή. Ας θεωρήσουμε για παράδειγμα το εξής τμήμα ενός XML εγγράφου :

```
<Note>
        <To>George</To>
        <From>Alexandros</From>
        <Text>
                <Heading>Reminder</Heading>
                <Body>Don't forget to send the e-mail tomorrow</Body>
        </Text>
</Note>
```

Εικόνα 2.9 : Τμήμα XML εγγράφου

Το τμήμα αυτό του XML εγγράφου, μπορεί να αναπαρασταθεί με τις κλάσεις "Note", "From", "To", "Text", "Heading" και "Body", η ιεραρχία των οποίων (η οποία φαίνεται στο παρακάτω σχήμα) είναι σύμφωνα με το XML Schema στο οποίο υπακούει το παραπάνω XML έγγράφο. Έτσι, κάθε φορά που μεταφέρονται δεδομένα από το XML έγγράφο, το αποτέλεσμα είναι ένα δένδρο από αντικείμενα.



Εικόνα 2.10 : Ιεραρχία των κλάσεων του XML εγγράφου

Υπάρχουν αρκετά εργαλεία στη γλώσσα προγραμματισμού Java, που χρησιμοποιούνται για XML data binding. Αυτό που χρησιμοποιήθηκε για την υλοποίηση της εφαρμογής είναι το XMLBeans[52]. Το εργαλείο αυτό μας επιτρέπει να εκμεταλλευθούμε την αφθονία και τα χαρακτηριστικά γνωρίσματα της XML και του XML Schema και να τα χαρτογραφήσουμε, όσο το δυνατόν φυσικότερα, στα ισοδύναμα

κατασκευάσματα της Java. Το XMLBeans χρησιμοποιεί το XML Schema, στο οποίο υπακούουν κάποια XML έγγραφα, για να δημιουργήσει τις κατάλληλες Java διεπαφές και κλάσεις, οι οποίες μπορούν να χρησιμοποιηθούν για την πρόσβαση και τροποποίηση των XML δεδομένων.

Αν και υπάρχουν διάφορα Java εργαλεία για τη διαχείριση των XML εγγράφων, όπως το DOM, το JAXB και το SAX, επιλέξαμε να χρησιμοποιήσουμε το XMLBeans λόγω των εξής πλεονεκτημάτων :

- Το XMLBeans μπορεί να χειριστεί οποιοδήποτε έγγραφο, όσο μεγάλο και αν είναι, χωρίς να υπερφορτώνεται η μνήμη.
- Ένα από τα μεγαλύτερα πλεονεκτήματα του XMLBeans είναι ότι παρέχει πλήρη υποστήριξη στην XMLSchema. Επίσης επιτρέπει την πρόσβαση σε ολόκληρο το XML Infoset. Αυτό είναι πολύ χρήσιμο, γιατί σε αρκετές περιπτώσεις η διάταξη των στοιχείων ενός XML εγγράφου και τα σχόλια μπορεί να είναι πολύ κρίσιμα για κάποια εφαρμογή.
- Μας δίνει τη δυνατότητα πιστοποίησης της εγκυρότητας ενός XML εγγράφου, τη στιγμή που αυτό φορτώνεται στη μνήμη ως δενδρική δομή στιγμιοτύπων των κλάσεων που έχουν παραχθεί από το αντίστοιχο XML Schema.
- Τέλος, μας προσφέρει καινοτόμα χαρακτηριστικά, όπως η χρήση των XML Cursors και η υποστήριξη εκφράσεων σε XQuery.

2.10 Oracle Berkley DB XML

Για την αποθήκευση των XML εγγράφων χρησιμοποιήθηκε μία **Native Xml Database** και συγκεκριμένα η **Berkeley DB XML**[41] ή οποία είναι ένα εργαλείο **ανοιχτού λογισμικού (open source)**. Η Berkeley DB XML αποτελεί μια υψηλών δυνατοτήτων XML βάση δεδομένων που παρέχει υποστήριξη για XQuery γλώσσα αναζήτησης. Χαρακτηρίζεται ως embeddable, δηλαδή έχει δυνατότητες ενσωμάτωσης στην ίδια την εφαρμογή. Ως αποτέλεσμα, τρέχει απευθείας με την εφαρμογή που την χρησιμοποιεί, χωρίς να απαιτείται ανεξάρτητος database server και ανθρώπινη διαχείριση. Διαχειρίζεται τα XML έγγραφα χρησιμοποιώντας XQuery και προσφέρει προχωρημένες υπηρεσίες διαχείρισης δεδομένων, που περιλαμβάνουν ταυτόχρονη πρόσβαση, συνδιαλλαγές και δημιουργία αντιγράφων (replication) για υψηλή διαθεσιμότητα και αντοχή στα λάθη.

Η βασική δομή αποθήκευσης της Berkeley Dbxml είναι τα XML έγγραφα. Η συγκεκριμένη βάση δεδομένων οργανώνει τα XML έγγραφα σε **συλλογές (containers)** και κάθε συλλογή μπορεί να περιέχει έγγραφα τα οποία βασίζονται σε κάποιο συγκεκριμένο σχήμα (σε έναν container, τα αρχεία που αποθηκεύονται, δεν χρειάζεται να υπακούουν όλα στο ίδιο σχήμα).

Σε αντίθεση με τις σχεσιακές βάσεις δεδομένων, οι οποίες αποθηκεύουν δεδομένα σε σχεσιακούς πίνακες , η Berkeley DB Xml έχει ως στόχο να αποθηκεύσει αυθαίρετα δέντρα XML δεδομένων. Στην συνέχεια αυτά μπορούν να αντιστοιχηθούν και να ανακτηθούν, είτε ως πλήρη έγγραφα είτε ως μεμονωμένα τμήματα εγγράφου μέσω μιας XML γλώσσας αναζήτησης, όπως η XQuery.

Πλεονεκτήματα της XML βάσης Δεδομένων

Μια XML βάση δεδομένων έχει διάφορα πλεονεκτήματα σε σχέση με τις σχεσιακές (relational) και αντικειμενοστραφής (object-oriented) βάσεις δεδομένων .

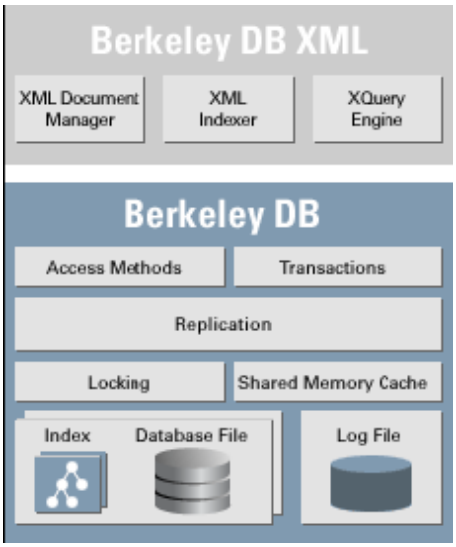
1. Xml δεδομένα αποθηκεύονται κατευθείαν στην βάση δεδομένων χωρίς να χρειάζεται περαιτέρω επεξεργασία ή απόσπαση των δεδομένων από κάποιο έγγραφο.

2. Τα περισσότερα στοιχεία ενός εγγράφου ,όπως κενά διαστήματα, παραμένουν ανέπαφα κατά την εισαγωγή του στην XML βάση δεδομένων .

3. Ερωτήματα (Queries) επιστρέφουν τα έγγραφα XML ή τμήματα τους, το οποίο σημαίνει ότι η ιεραρχική δομή των πληροφοριών XML διατηρείται.

2.10.1 Αρχιτεκτονική της Oracle Berkeley DB XML

Η Oracle Berkeley DB XML, είναι χτισμένη στη κορυφή της Oracle Berkeley DB και ως αποτέλεσμα κληρονομεί σημαντικά χαρακτηριστικά και στοιχεία. Ειδικότερα, η ίδια προσθέτει στη κορυφή της Oracle Berkeley DB έναν document parser, έναν XML indexer και μια μηχανή XQuery με στόχο την επίτευξη ταχύτερης και αποδοτικότερης ανάκτησης δεδομένων. Στην εικόνα παρουσιάζεται η αρχιτεκτονική της Oracle Berkeley DB XML



Εικόνα 2.11 : Αρχιτεκτονική Oracle Berkeley DB XML

Χάρη στην Oracle Berkeley DB ως την υποκείμενη μηχανή αποθήκευσης, η Oracle Berkeley DB XML κληρονομεί πλήρεις ACID (Atomicity, Consistency, Isolation, Durability) συνδιαλλαγές, αυτόματη αποκατάσταση, κρυπτογράφηση δεδομένων στο δίσκο με AES (Advanced Encryption Standard)και δημιουργία αντιγράφων (replication) για υψηλή διαθεσιμότητα. Επιπρόσθετα, τόσο XML όσο και μη-XML δεδομένα μπορούν να αποθηκευτούν στην Oracle Berkeley DB XML, κάτι που αποτελεί πλεονέκτημα σε κάποιες εφαρμογές.

Η Oracle Berkeley DB XML υποστηρίζει την XQuery 1.0 (Η Xquery αποτελεί πλέον τη κατεξοχήν γλώσσα ερωτήσεων για πρόσβαση σε XML δεδομένα) και την XPath 2.0, XML ονοματοδοσία, έλεγχο εγκυρότητας XML εγγράφων. Πιο συγκεκριμένα, η XQuery μηχανή χρησιμοποιεί ένα εξελιγμένο, βασισμένο στο κόστος, βελτιστοποιητή ερώτησης και υποστηρίζει προ-μεταγλωττισμένη εκτέλεση ερώτησης με ενσωματωμένες μεταβλητές. Μεγάλα έγγραφα μπορούν να αποθηκευτούν ολόκληρα ή κατατμημένα σε κόμβους, επιτυγχάνοντας έτσι αποδοτικότερη ανάκτηση και μερική ενημέρωση των εγγράφων. Επίσης, υποστηρίζει ευέλικτη δεικτοδότηση XML κόμβων, στοιχείων, χαρακτηριστικών, μεταδεδομένων για πιο γρήγορη και αποδοτική ανάκτηση.

2.11 Jena

Το Jena είναι ένα open source Semantic Web framework για Java. Παρέχει ένα API για την εξαγωγή και εγγραφή δεδομένων σε RDF γράφους. Οι γράφοι αναπαρίστανται σαν ένα αφηρημένο μοντέλο. Στο μοντέλο αυτό μπορούν να γίνουν ερωτήσεις (queries) μέσω μίας συγκεκριμένης γλώσσας, της SPARQL. Το μεγάλο πλεονέκτημα του Jena API σε σχέση με άλλα παρεμφερή frameworks (Sesame) είναι η δυνατότητα του για υποστήριξη της γλώσσας OWL

Το Jena1 αρχικά εκδόθηκε το 2000 ενώ το Jena2 εκδόθηκε τον Αύγουστο του 2003. Η κύρια συνεισφορά του Jena1[44] είναι το εμπλουτισμένο Model API για τη διαχείριση RDF γραφημάτων. Γύρω από αυτό το API το Jena1 παρέχει πολλά εργαλεία, όπως: ένα RDF/XML parser[45], μία γλώσσα αναζήτησης (query language)[46], επιπλέον I/O modules για N3[47] και N-triple[48] και RDF/XML[49] εξόδους. Το Jena1 παρέχει επιπρόσθετο API για το χειρισμό της DAML+OIL[50].

Το Jena2 παρέχει επιπλέον λειτουργικότητα με την υποστήριξη OWL[1] και RDFS[3]. Υπάρχουν νέα APIs για την προσπέλαση των οντολογιών, και επίσης προσφέρονται 2 νέα σημεία επέκτασης . Το πρώτο επιτρέπει την ανάπτυξη νέων APIs για νέα λειτουργικότητα στους σχεδιαστές εφαρμογών. Το δεύτερο επιτρέπει την ανάπτυξη νέου μηχανισμού τριάδων, όπως εικονικές τριάδες που δημιουργούνται δυναμικά και είναι προϊόντα ορισμένης επεξεργασίας.

Οι δύο κύριοι στόχοι του Jena2 είναι :

- Πολλαπλές ευέλικτες αναπαραστάσεις RDF γραφημάτων. Αυτό επιτρέπει εύκολη πρόσβαση και διαχείριση των δεδομένων των γραφημάτων επιτρέποντας στον χρήστη – προγραμματιστή την προσπέλαση – πλοήγηση μέσω του μηχανισμού των τριάδων. Πιο συγκεκριμένα, το model API παρουσιάζει τα γραφήματα χρησιμοποιώντας συνθήκες και περιορισμούς από τα RDF recommendations, και το Ontology API το οποίο παρουσιάζει τα γραφήματα με τη χρήση της OWL και της RDFS.
- Μία απλουστευμένη προβολή του RDF γραφήματος στο χρήστη με σκοπό την αναπαράσταση των δεδομένων με τη μορφή τριάδων. Αυτό είναι ιδιαίτερα χρήσιμο για τις προσεγγίσεις RDFS και OWL.

2.12 Περίληψη

Το παρόν κεφάλαιο αποτέλεσε μια εισαγωγή σε όλα τα πρότυπα και τις τεχνολογίες που χρησιμοποιηθήκαν κατά την υλοποίησης της παρούσας εργασίας. Στην συνέχεια του κειμένου και όπου αυτό κρίνεται αναγκαίο γίνεται σε μεγαλύτερο βάθος η ανάλυση των παραπάνω προτύπων και τεχνολογιών.

Στο επόμενο κεφάλαιο (Κεφάλαιο 3) γίνεται αναφορά σε σχετικές εργασίες που έχουν πραγματοποιηθεί, όπως και μια εισαγωγή στον ευρύτερο ερευνητικό τομέα στον οποίο εντάσσονται, την Σημασιολογική Ολοκλήρωση Δεδομένων (Semantic Data Integration).

3 Σχετικές Εργασίες

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο γίνεται αναφορά σε σχετικές εργασίες που έχουν πραγματοποιηθεί, όπως και μια εισαγωγή στον ευρύτερο ερευνητικό τομέα στον οποίο εντάσσονται, την **Σημασιολογική Ολοκλήρωση Δεδομένων** (**Semantic Data Integration**).

Στην ενότητα 3.2 γίνεται η εισαγωγή στον ερευνητικό τομέα της σημασιολογικής ολοκλήρωσης δεδομένων. Στην ενότητα 3.3 γίνεται η καταγραφή και η ανάλυση των σχετικών εργασιών που έχουν πραγματοποιηθεί και τέλος στην ενότητα 3.4 η περίληψη του κεφαλαίου.

3.2 Σημασιολογική Ολοκλήρωση Δεδομένων

Ένας από τους βασικότερους στόχους του Σημασιολογικού Ιστού (Semantic Web) είναι η επίτευξης της διαλειτουργικτότητας των δεδομένων που δημοσιεύονται στον παγκόσμιο ιστό και η πρόσβαση τους με έναν ενιαίο τρόπο, σαν να αποτελούσαν μια τεράστια βάση δεδομένων.

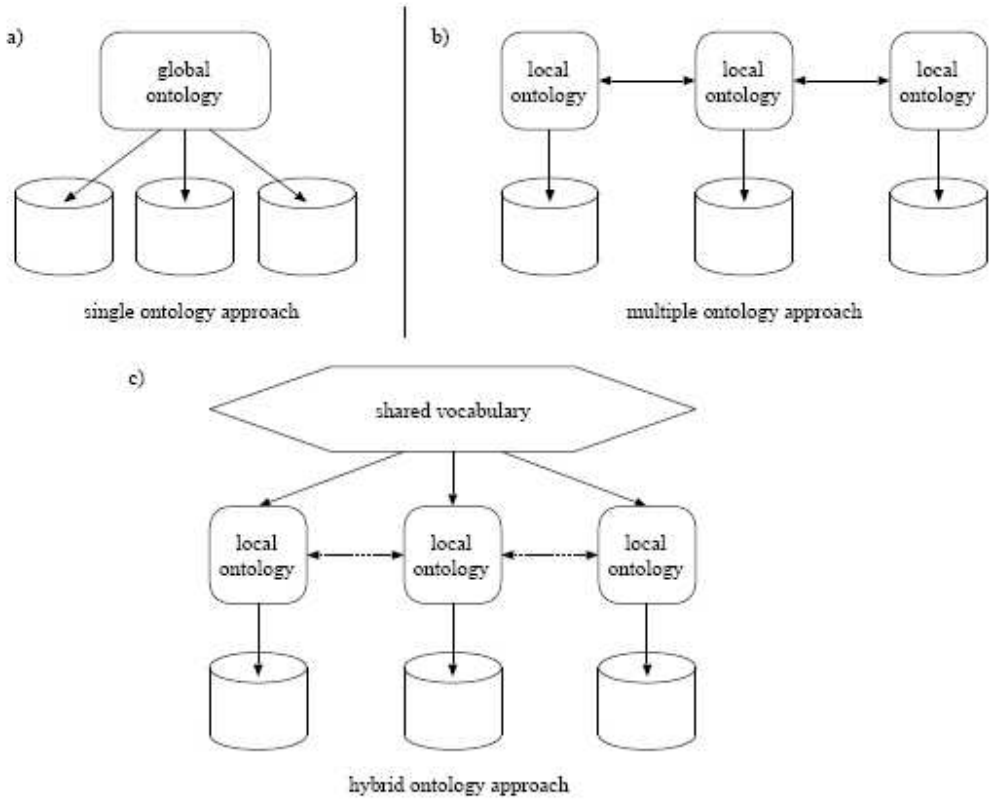
Η ανάγκη ολοκλήρωσης της πληροφορίας εμφανίστηκε από την αρχή της ανάπτυξης των πληροφοριακών συστημάτων, η παραδοσιακή Ολοκλήρωση Δεδομένων αποτελεί μία ανοιχτή ερευνητική περιοχή στο πεδίο των Βάσεων Δεδομένων, αλλά πρόσφατα, το ερευνητικό ενδιαφέρον μεταφέρεται από την ολοκλήρωση δεδομένων στη **σημασιολογική ολοκλήρωση δεδομένων**. Τη συγκεκριμένη τάση επηρέασε το όραμα του Σημασιολογικού Ιστού και η χρήση των σημασιολογικών τεχνολογιών.

Η σημασιολογική ολοκλήρωση δεδομένων καλείται να αντιμετωπίσει και προβλήματα σημασιολογικής ετερογένειας (semantic heterogeneity) τα οποία προκύπτουν από τη χρήση διαφορετικών όρων για την αποτύπωση της ίδιας έννοιας τόσο σε επίπεδο σχημάτων κωδικοποίησης δεδομένων (schema level) όσο και στο επίπεδο των δεδομένων (data level).

Η σημασιολογική ολοκλήρωση δεδομένων είναι η διαδικασία της χρήσης εννοιολογικών αναπαραστάσεων των δεδομένων και των σχέσεών τους με στόχο την εξάλειψη της ετερογένειας. Εφόσον οι οντολογίες επιτρέπουν την πολύπλοκη έκφραση εννοιών και των σχέσεών τους, ο ρόλος τους στη σημασιολογική ολοκλήρωση δεδομένων είναι ιδιαίτερα ενεργός.

Μία οντολογία επιλέγεται ως καθολικό σχήμα διότι μπορεί να ορίσει πολύπλοκες σημασιολογικές σχέσεις ενός θεματικού χώρου. Παράλληλα, είναι διατυπωμένη σε αυστηρό μαθηματικό φορμαλισμό, ο οποίος επιτρέπει την εξαγωγή συμπερασμάτων, για παράδειγμα τον ορισμό επιπρόσθετων σχέσεων ανάμεσα στις έννοιες. Στο πλαίσιο αυτό έχουν αναπτυχθεί από ερευνητικές ομάδες συγκεκριμένες οντολογίες οι οποίες ορίζουν έννοιες και σχέσεις διαφόρων τομέων, π.χ. πολιτιστική κληρονομιά, υπολογιστική γλωσσολογία και γνωστική επιστήμη, οικονομικά κ.α.

Σε ένα σενάριο ολοκλήρωσης δεδομένων, υπάρχουν τρεις προσεγγίσεις σχετικές με το ρόλο των οντολογιών (οι αρχιτεκτονικές αυτές φαίνονται στην Εικόνα 3.1) [62]:



Εικόνα 3.1 : Αρχιτεκτονικές Χρήσης Οντολογιών

a) **Προσέγγιση απλής οντολογίας (single ontology approach)**: Μία καθολική οντολογία (global ontology) παρέχει ένα διαμοιρασμένο λεξιλόγιο (shared vocabulary) για τον ορισμό των σημασιών αυτόνομων πηγών δεδομένων οι οποίες σχετίζονται με αυτήν. Η καθολική οντολογία περιγράφει ένα συγκεκριμένο τομέα, οπότε η συγκεκριμένη προσέγγιση εφαρμόζεται ιδιαίτερα σε πηγές δεδομένων που παρουσιάζουν διαφορετικές «όψεις» του ίδιου τομέα.

b) **Προσέγγιση πολλαπλών οντολογιών (multiple ontology approach)**: Στην προσέγγιση αυτή κάθε τοπικό σύστημα δεδομένων περιγράφεται από μία ξεχωριστή τοπική οντολογία (local ontology). Αντί να χρησιμοποιείται μία κοινή οντολογία, οι τοπικές οντολογίες συσχετίζονται μεταξύ τους. Η απουσία μίας καθολικής οντολογίας διευκολύνει την αυτόνομη ανάπτυξη τοπικών οντολογιών, οι οποίες εκφράζουν αναλυτικά και με συνέπεια τις έννοιες και τις σχέσεις κάθε τοπικού συστήματος δεδομένων. Εν τούτοις, το θετικό αυτό χαρακτηριστικό αποτελεί παράλληλα και ανοικτό ερευνητικό πρόβλημα κυρίως αναφορικά με τον ορισμό των κανόνων αντιστοίχησης μεταξύ των οντολογιών (ontology mapping)

c) **Προσέγγιση υβριδικής οντολογίας (hybrid ontology approach)**: Η συγκεκριμένη προσέγγιση συνδυάζει χαρακτηριστικά από τις δύο προαναφερθείσες προσεγγίσεις. Κάθε τοπική πηγή περιγράφεται από μία ξεχωριστή τοπική οντολογία, η οποία δεν είναι συσχετισμένη με άλλες τοπικές οντολογίες, αλλά με μία καθολική διαμοιραζόμενη οντολογία, και της οποίας η δημιουργία έχει βασιστεί είτε στις πρωτογενείς έννοιες (λεξιλόγιο) μίας καθολικής οντολογίας είτε στη μετατροπή της τοπικής πηγής σε οντολογία. Το γεγονός αυτό διευκολύνει την αντιστοίχηση των τοπικών οντολογιών και κατά συνέπεια των τοπικών πηγών δεδομένων. Νέες πηγές μπορούν εύκολα να προστίθενται χωρίς την ανάγκη αλλαγής των υπαρχουσών αντιστοιχήσεων.

3.3 Εργασίες

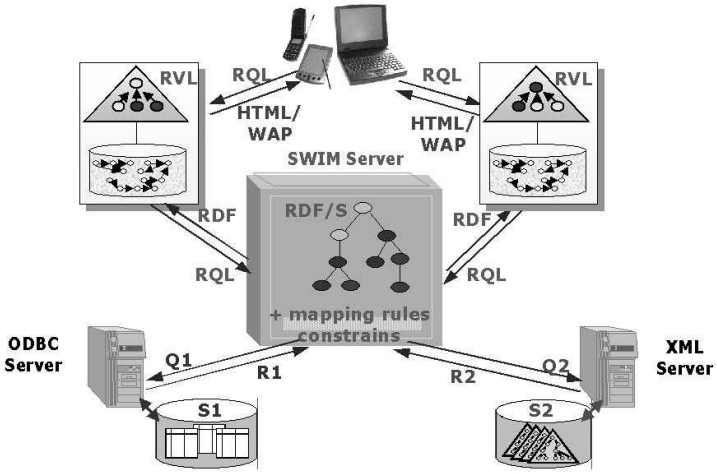
Υπάρχει ένας πολύ μεγάλους αριθμός εργασιών στον ευρύτερο τομέα της ολοκλήρωσης δεδομένων, ένας πλήρης κατάλογος εργασιών είναι ο [63]. Σε αυτή την ενότητα παρουσιάζονται οι εργασίες που είναι πιο σχετικές με την παρούσα προσέγγιση.

3.3.1 The ICS-FORTH Semantic Web Integration Middleware (SWIM) [32][33][34]

Σύστημα ολοκλήρωσης XML και σχεσιακών πηγών με χρήση **RDF/S** σχημάτων. Έχει αναπτυχθεί από Ινστιτούτου Πληροφορικής (Ι.Π.) του Ιδρύματος Τεχνολογίας και Έρευνας (ΙΤΕ). Η αρχιτεκτονική του συστήματος φαίνεται στην Εικόνα 3.2. Ο διαμεσολαβητής ολοκλήρωσης σημασιολογικού ιστού (**ICS-FORTH SWIM**) ολοκληρώνει σχεσιακές πηγές και πηγές XML δεδομένων χρησιμοποιώντας RDF/S σχήματα που περιγράφουν συγκεκριμένες εφαρμογές ή επιστημονικά πεδία. Το SWIM διευκολύνει τους χρήστες να επερωτήσουν το ενδιάμεσο ιδεατό RDF/S σχήμα με την χρήση της δηλωτικής γλώσσας **RQL**[64], επιπλέον προσφέρει μηχανισμούς αφαίρεσης με την δυνατότητα δήλωσης όψεων(views), με την βοήθεια της γλώσσας ορισμού όψεων **RVL**[65]. Η χρήση ενός καλά θεμελιωμένου υποσυνόλου της πρωτοβάθμιας λογικής(first order logic) για τον ορισμό αντιστοιχήσεων μεταξύ του RDF/S σχήματος και των XML δεδομένων, εκμεταλλεύεται προηγούμενα αποτελέσματα στην αναδιατύπωση και βελτιστοποίηση σχεσιακών ερωτήσεων.

Το πλαίσιο προσφέρει τις εξής λειτουργίες :

- Δήλωση αντιστοιχήσεων μεταξύ **XML** και **RDF/S** και μεταξύ **Σχεσιακών Βάσεων** και **RDF/S**
- Εξακρίβωση ότι οι αντιστοιχήσεις είναι σύμφωνες με την σημασιολογία του RDF σχήματος
- Μετάφραση των **RQL** ερωτήσεων σε **XQuery** και **SQL**
- Δυνατότητα **δήλωσης όψεων**(views) του RDF/S σχήματος, με χρήση της γλωσσών ορισμού όψεων RVL
- **Μετασχηματισμός** (reformulation) των **RQL** ερωτήσεων με βάση τις **RVL** όψεις
- **Βελτιστοποίηση ερωτήσεων**



Εικόνα 3.2 : Αρχιτεκτονική του SWIM

Μετασχηματισμός - Μετάφραση RQL ερωτήσεων

- Οι RQL ερωτήσεις αρχικά μετατρέπονται και αναπαρίστανται με πρωτοβάθμια λογική (first order logic)
- Επίσης με πρωτοβάθμια λογική αναπαρίσταται η σημασιολογία της RDF οντολογίας. Με βάση αυτή την σημασιολογία ελέγχεται η πιθανή βελτιστοποίηση της πρωτοβάθμια λογικής αναπαράστασης της RQL ερώτησης.
- Πραγματοποιείται βελτιστοποίηση της ερώτησης με βάσει τις αντιστοιχήσεις που έχουν οριστεί μεταξύ οντολογίας και XML πηγής.
- Ο μετασχηματισμός της παραπάνω διαδικασία παράγει την βέλτιστη – ελάχιστη ερώτηση. Αυτή η (μετασχηματισμένη) ερώτηση θα μεταφραστή στην συνέχεια.

- Η μετασχηματισμένη-βέλτιστη πρωτοβάθμια λογική αναπαράσταση της ερώτησης παρέχει όλες τις απαραίτητες πληροφορίες για την μετάφραση της.
- Η ερώτηση μεταφράζεται είτε σε XPath, για τις περιπτώσεις που η αρχική ερώτηση είναι απλοϊκή, είτε σε XQuery για τις πιο πολύπλοκες ερωτήσεις.

3.3.2 PEPSINT - An Ontology-based Framework for XML Semantic Integration [27][29]

Σύστημα ολοκλήρωσης XML πηγών με χρήση οντολογιών. Έχει αναπτυχθεί από το πανεπιστήμιο του Illinois. Η αρχιτεκτονική του συστήματος φαίνεται στην Εικόνα 3.4 Κάθε XML πηγή ακολουθεί ένα συγκεκριμένο XML σχήμα, από κάθε πηγή παράγεται μια τοπική RDF οντολογία(**local RDF ontology**). Η παραγωγή της τοπικής οντολογίας ακολουθεί τις αντιστοιχίες που φαίνονται στην Εικόνα 3.3. Για την προσομοίωση τις ιεραρχικής δομής των XML στοιχείων(element), η προσέγγιση ορίζει μια την ιδιότητα αντικειμένου(Object property) **rdfx:contain**.

Αντιστοιχήσεις μεταξύ των στοιχείων των τοπικών οντολογιών και μονοπατιών των xml πηγών αποθηκεύονται στους πίνακες αντιστοιχήσεων (**mappings tables**).

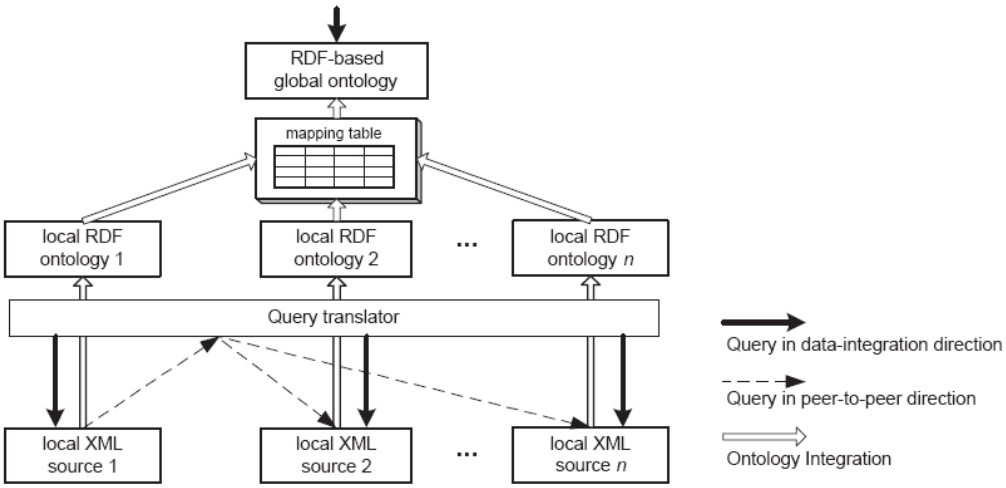
XML Schema concepts	RDF Schema concepts
Attribute	Property
Simple-type element	Property
Complex-type element	Class

Εικόνα 3.3 : Αντιστοιχίες Μετασχηματισμού

Το καθολική RDF/S οντολογία (**Global RDF ontology**), προκύπτει από την ένωση των τοπικών RDF/S οντολογιών με μη-αυτοματοποιημένο τρόπο, κάνοντας χρήση των παρακάτω κανόνων. Ένωση των κλάσεων που είναι σημασιολογικά όμοιες. Ένωση των ιδιοτήτων τιμών δεδομένων(datatype properties) που είναι σημασιολογικά όμοια. Ένωση των ιδιοτήτων αντικειμένων(σχέσεις δηλαδή μεταξύ των classes) που είναι σημασιολογικά όμοια. Δημιουργία υπέρκλάσεων(superclass) για τις κλάσεις από τις οποίες μπορούν να προκύψουν γενικότερες. Επίσης οι αντιστοιχήσεις μεταξύ της καθολικής οντολογίας και των τοπικών οντολογιών αποθηκεύονται στους πίνακες αντιστοιχήσεων.

Το πλαίσιο πραγματοποιεί την **αναδιατύπωση** (**Reformulation**) **RDQL**[66] ερωτήσεων, τόσο μεταξύ της καθολικής οντολογίας και των τοπικών οντολογιών, όσο και μεταξύ διαφορετικών τοπικών οντολογιών. Επίσης πραγματοποιεί την **μετάφραση** (**translation**) των **RDQL**

ερωτήσεων που αναφέρονται στις τοπικές οντολογίες, σε **XQuery** που αναφέρονται στην αντίστοιχη XML πηγή.



Εικόνα 3.4 : Αρχιτεκτονική του πλαισίου ολοκλήρωσης

Μετάφραση RDQL ερωτήσεων

Οι **RDQL** ερωτήσεις που αναφέρονται στις τοπικές οντολογίες μεταφράζονται σε **XQuery** που αναφέρονται στην αντίστοιχη XML πηγή. Μεταξύ των τοπικών οντολογιών και των XML πηγών δημιουργούνται μόνο 1-1 αντιστοιχήσεις. Λόγω του δοκιμαστικού σταδίου ανάπτυξης της γλώσσας RDQL και στην συνέχεια την αντικατάστασης της από την γλώσσα SPARQL, οι μορφές των ερωτήσεων που υποστηρίζει είναι ιδιαιτέρα απλοϊκές. Ο αλγόριθμος μετάφρασης είναι υπερβολικά απλοϊκός και πραγματοποιεί την μετάφραση μιας ακολουθίας τριπλέτων σχηματομορφών(triples patterns), χωρίς την υποστήριξη διαμοιραζόμενων μεταβλητών (shared variables) μεταξύ των τριπλετών, χωρίς την υποστήριξη χρήσης εννοιών από το λεξιλόγιο(Vocabulary) του RDF/S (όπως πχ rdfs:subClassOf) και τέλος χωρίς την τήρηση της σημασιολογίας για την ακολουθία λύσεων που παράγει.

3.1 Περίληψη

Στο κεφάλαιο που προηγήθηκε έγινε μια εισαγωγή στον ερευνητικό τομέα της σημασιολογικής ολοκλήρωσης δεδομένων, καθώς και μια αναφορά σε σχετικές με την παρούσα εργασίες.

Από όσο γνωρίζουμε δεν υπάρχει κάποια σχετική δουλεία η οποία να έχει ασχοληθεί είτε με την αντιστοιχία , είτε με την μετάφραση, είτε με την σημασιολογική ισοδυναμία, μεταξύ των δυο γλωσσών ερωτήσεων SPARQL και XQuery, γεγονός το οποίο κάνει την παρούσα εργασία καινοτόμα.

Στο επόμενο κεφάλαιο του κειμένου (Κεφάλαιο 4) αναλύεται η ανάπτυξη του πλαισίου SPARQL2XQuery, παρουσιάζεται η γενική αρχιτεκτονική στην οποία εντάσσεται το πλαίσιο, αναλύεται ο τρόπος ανάπτυξης τον επιμέρους στοιχείων λογισμικού (software Component) του και οι μεταξύ τους διασυνδέσεις και αλληλεξαρτήσεις.

4 Το Πλαίσιο SPARQL2XQuery

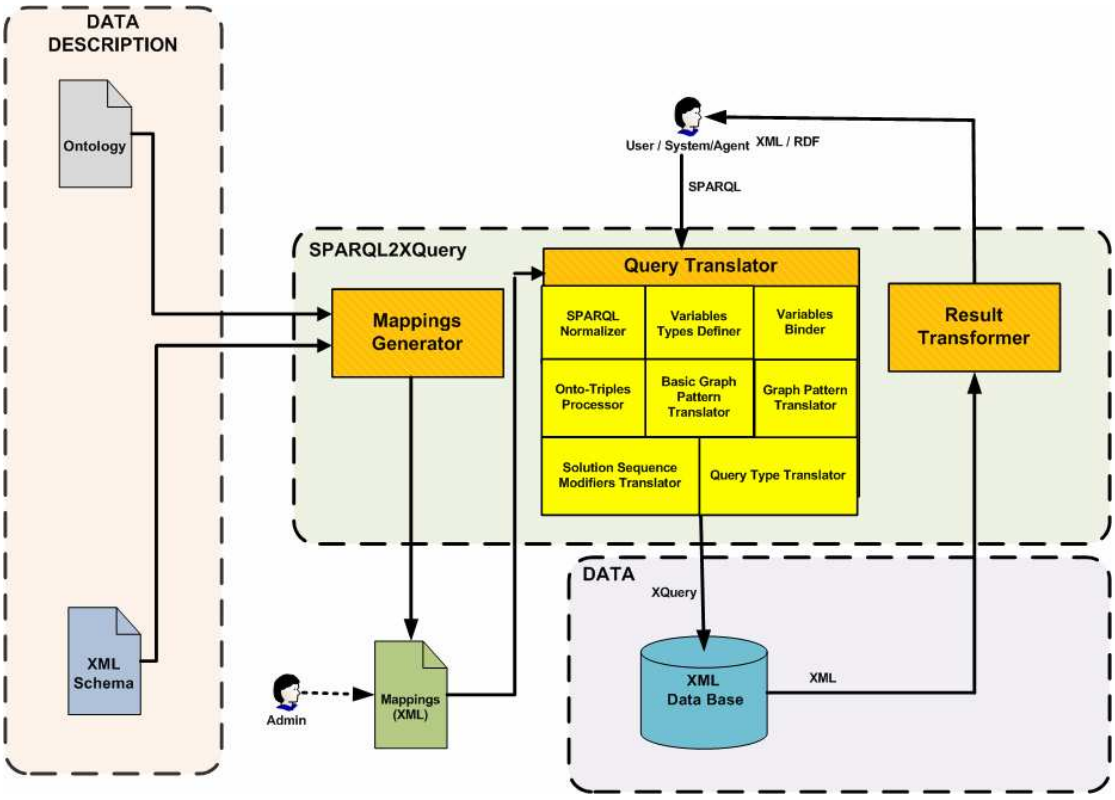
4.1 Εισαγωγή

Στο παρόν κεφάλαιο αναλύεται η ανάπτυξη του πλαισίου **SPARQL2XQuery**, παρουσιάζεται η γενική αρχιτεκτονική στην οποία εντάσσεται το πλαίσιο, αναλύεται ο τρόπος ανάπτυξης τον επιμέρους στοιχείων λογισμικού (software Component) του και οι μεταξύ τους διασυνδέσεις και αλληλεξαρτήσεις. Όπως επίσης και η διασύνδεση και αλληλεπίδραση του με άλλα συστήματα και χρήστες. Τέλος, αναλύεται λεπτομερώς και μοντελοποιείται με UML διαγράμματα δραστηριοτήτων η διαδικασία μετάφρασης των σημασιολογικών SPARQL ερωτήσεων.

Στην ενότητα 4.2 γίνεται η ανάλυση των επιμέρους τμημάτων της γενικής αρχιτεκτονικής στην οποία εντάσσεται το πλαίσιο **SPARQL2XQuery**. Ενώ στην ενότητα 4.3 αναλύεται η αρχιτεκτονική του πλαισίου **SPARQL2XQuery**.Τέλος στην ενότητα 4.4 αναλύεται η διαδικασία μετάφρασης των σημασιολογικών SPARQL ερωτήσεων.

4.2 Γενική Αρχιτεκτονική

Στην Εικόνα 4.1 φαίνεται η γενική αρχιτεκτονική στην οποία εντάσσεται το πλαίσιο **SPARQL2XQuery**, η αλληλεπίδραση και διασύνδεση του με άλλα συστήματα και χρήστες. Σε αυτή την εικόνα φαίνονται τα εξής τμήματα : **User/System/Agent, Admin, Data, Data Description, Mappings** και **SPARQL2XQuery**.



Εικόνα 4.1 Γενική Αρχιτεκτονική

- **User/System/Agent**

  Ο χρήστης, το σύστημα ή ο πράκτορας με τον οποίο αλληλεπιδρά το πλαίσιο, έχοντας γνώση μόνο των εννοιών που περιγράφονται από την οντολογία και αγνοώντας την δομή και την μορφή των υποθηκευμένων δεδομένων, δίνει ως είσοδο μια SPARQL ερώτηση και δέχεται τα αποτελέσματα είτε σε μορφή RDF γράφου είτε σε XML μορφή, σύμφωνα με την XML μορφή των SPARQL αποτελεσμάτων [56].

- **Admin**

  Εξειδικευμένος χρήστης, γνώστης των εννοιών που περιγράφονται από την οντολογίας και από το XML σχήμα, ορίζει τις μεταξύ τους αντιστοιχήσεις. Οι αντιστοιχήσεις ορίζονται μέσω του εξειδικευμένο χρήστη, στην περίπτωση όπου η οντολογία δεν έχει παραχθεί με χρήση της μεθόδου **XS2OWL** και επομένως οι αντιστοιχήσεις δεν παράγονται αυτόματα.

- **Data**

  XML Data Base Τα δεδομένα είναι αποθηκευμένα σε μορφή **XML**, ακολουθώντας ένα **XML Schema**. Το πλαίσιο **SPARQL2XQuery** είναι ανεξάρτητο από τον τρόπο με τον οποίο είναι υποθηκευμένα τα XML δεδομένα, όπως επίσης και από το XQuery engine και το περιβάλλον εκτέλεσης των XQuery ερωτήσεων. Παρόλα αυτά κατά τον έλεγχο και αξιολόγηση του πλαισίου, για την αποθήκευση των XML δεδομένων έγινε χρήση της XML βάσης δεδομένων **Oracle Berkley DB XML**[41] η οποία αναλύεται στην ενότητα 2.10.

- **Data Description**

  Γενική περίπτωση :

  Ontology Οντολογία σε γλώσσα OWL ή RDFS. Η οντολογία περιγράφει είτε κάποια περιοχή γνώσης (domain ontology) είτε κάποιες πιο αφηγημένες έννοιες (upper ontology) είτε και τα δυο. Η οντολογία πρέπει να έχει κάποια σημασιολογική συσχέτιση με τα δεδομένα(Data).

  XML Schema Το XML Schema είναι το σχήμα το οποίο ακολουθούν τα XML δεδομένα και έχει κάποια σημασιολογική συσχέτιση με την οντολογία.

  Η οντολογία έχει προκύψει με χρήση της μεθόδου XS2OWL:

  Ontology Οντολογία σε γλώσσα OWL-DL. Η οντολογία έχει παραχθεί με βάση την μεθοδολογία XS2OWL από το XML Schema.

  XML Schema Το XML Schema είναι το σχήμα το οποίο ακολουθούν τα XML δεδομένα, βάση το οποίο έχει παραχθεί η οντολογία

- **Mappings**

  Οι αντιστοιχήσεις μεταξύ της Οντολογίας και του XML σχήματος. Αποτελούν ένα XML έγγραφο το οποίο ακολουθεί το σχήμα που περιγράφεται στο Κεφάλαιο 6. Όπως έχει αναφερθεί οι αντιστοιχήσεις είτε δημιουργούνται αυτόματα από το πλαίσιο **SPARQL2XQuery**, στην περίπτωση την οποία η οντολογία έχει προκύψει με βάση την μεθοδολογία **XS2OWL**, είτε ορίζονται από κάποιον εξειδικευμένο χρήστη (**Admin**).

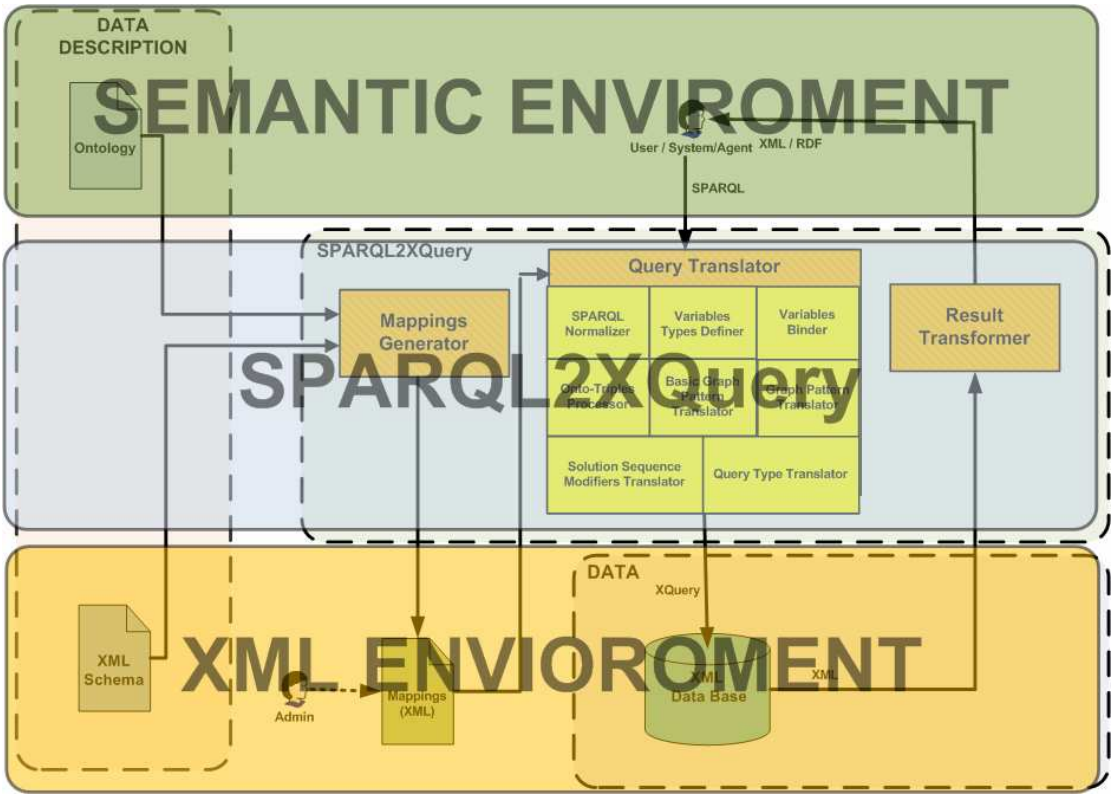
- **SPARQL2XQuery**

  Το πλαίσιο **SPARQL2XQuery** πραγματοποιεί την μετάφραση των ερωτήσεων, την δημιουργία των αντιστοιχήσεων και τον μετασχηματισμό των αποτελεσμάτων. Αποτελείται από τα στοιχεία

λογισμικού **Mapping Generator**, **Query Translator** και **Result Transformer**. Τα οποία θα αναλυθούν στην επόμενη ενότητα.

4.2.1 Semantic Environment vs. XML Environment

Στην Εικόνα 4.2 φαίνεται ο διαχωρισμός σε δυο διαφορετικά, ετερογενή περιβάλλοντα (**heterogeneous environments**), στα οποία ανάμεσα τους εμφανίζεται το πλαίσιο SPARQL2XQuery παίζοντας τον ρόλο του **ενδιαμέσου λογισμικού** (**Middleware**). Τα ετερογενή περιβάλλοντα είναι το **Σημασιολογικό Περιβάλλον** (**Semantic Environment**) στο οποίο γίνεται χρήση των σημασιολογικών τεχνολογιών όπως OWL, RDF/S , SPARQL κτλ. Και το **XML Περιβάλλον** (**XML Environment**) στο οποίο γίνεται χρήση των XML τεχνολογιών όπως XML, XML Schema, XQuery κτλ. Με την χρήση του πλαισίου SPARQL2XQuery επιτυγχάνεται η **διαλειτουργικότητα** (**Interoperability**) μεταξύ των δυο αυτών ετερογενή περιβαλλόντων, καθώς και **ενοποίηση πληροφορίας** (**data integration**).



Εικόνα 4.2 Ετερογενή Περιβάλλοντα

4.3 Αρχιτεκτονική του Πλαισίου SPARQL2XQuery

Όπως αναφέρθηκε και στην προηγούμενη ενότητα το πλαίσιο **SPARQL2XQuery** πραγματοποιεί την μετάφραση των ερωτήσεων, την δημιουργία των αντιστοιχήσεων και τον μετασχηματισμό των αποτελεσμάτων. Όπως φαίνεται και στην Εικόνα 4.1 το πλαίσιο αποτελείται από τα στοιχεία λογισμικού **Mapping Generator**, **Query Translator** και **Result Transformer**.

- **Mapping Generator**

  Το στοιχείο λογισμικού **Mapping Generator** αποτελεί μέρος του πλαισίου **SPARQL2XQuery**. Χρησιμοποιείται στην περίπτωση στην οποία η οντολογία έχει προκύψει με χρήση της μεθόδου **XS2OWL**. Δέχεται ως εισόδους την οντολογία και το XML σχήμα, για τα οποία ανακαλύπτει(discover) και παράγει αυτόματα τις μεταξύ τους αντιστοιχήσεις, αποθηκεύοντας τις σε μορφή XML . Σύμφωνα με την μεθοδολογία που αναλύεται στο Κεφάλαιο 6.

  Με την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων επιτυγχάνεται μια πλήρως αυτοματοποιημένη διαδικασία, χωρίς να είναι απαραίτητη η παρέμβαση ανθρωπίνου παράγοντα. Επίσης επιτυγχάνεται η πλήρης αντιστοίχηση όλων των στοιχείων τόσο της οντολογίας όσο και του XML σχήματος, με αντιστοιχήσεις απολύτως σημασιολογία ορθές, χωρίς πιθανότητα σφάλματος ή παρερμηνείας κατά την δημιουργία τους κάτι που μπορεί να συμβεί στην περίπτωση "χειροκινήτου"(manual) ορισμού των αντιστοιχήσεων από εξειδικευμένο χρήστη.

  Για την ανάπτυξη του στοιχείο λογισμικού **Mapping Generator** έγινε χρήση των τεχνολογιών : **JAVA**, **XML Beans** [52] που αναλύεται στην ενότητα 2.9 και **Jena API** [42] που αναλύεται στην ενότητα 2.11.

- **Query Translator**

  Το στοιχείο λογισμικού **Query Translator** αποτελεί το κύριο μέρος του πλαισίου **SPARQL2XQuery**. Πραγματοποιεί την μετάφραση των SPARQL ερωτήσεων σε σημασιολογικά ισοδύναμες XQuery, κάνοντας χρήση των αντιστοιχήσεων. Το στοιχείο δέχεται ως εισόδους μια ερώτηση SPARQL και τις αντιστοιχήσεις και παράγει ως έξοδο, την σημασιολογικά ισοδύναμη XQuery. Η διαδικασία μετάφρασης των σημασιολογικών ερωτήσεων καθώς και ο τρόπος αλληλεπίδρασης των υπό-στοιχείων(sub components) αναλύεται στην ενότητα 4.4.

  Το στοιχείο **Query Translator** αποτελείται από τα υπό-στοιχεία :

  - **SPARQL Normalizer**

    Το υπό-στοιχείο λογισμικού **SPARQL Normalizer** αποτελεί μέρος του στοιχείου **Query Translator**. Πραγματοποιεί την κανονικοποίηση της σχηματομορφής γράφου

των SPARQL ερωτήσεων. Η κανονικοποίηση έχει ως σκοπό την βελτιστοποίηση και απλοποίηση της διαδικασίας της μετάφρασης. Αναδρομική εφαρμογή κανόνων ισοδυναμίας , ιδιοτήτων των τελεστών που εμφανίζονται μεταξύ των σχηματομορφών γράφων και re-writing τεχνικών , έχουν ως αποτέλεσμα τη δημιουργία μιας κανονικοποιημένης γραμματικής για τη σχηματομορφή γράφου της ερώτησης. Αναλύεται στο **Κεφάλαιο 7**.

- **Variables Type Determinator**

  Το υπό-στοιχείο λογισμικού **Variables Type Determinator** αποτελεί μέρος του στοιχείου **Query Translator**. Πραγματοποιεί τον προσδιορισμό των τύπων των μεταβλητών που περιέχονται στην SPARQL ερώτηση. Δέχεται ως είσοδο μια Union-Free σχηματομορφή γράφου και προσδιορίζει του τύπους των μεταβλητών που περιέχονται σε αυτή. Αναλύεται στο **Κεφάλαιο 8**.

- **Onto Triples Processor**

  Το υπό-στοιχείο λογισμικού **Onto Triples Processor** αποτελεί μέρος του στοιχείου **Query Translator**. Δέχεται ως είσοδο είσοδο μια Union-Free σχηματομορφή γράφου επεξεργάζεται τις Onto Τριπλέτες, ώστε να προσδιοριστούν οι συνδέσεις(bindings) των μεταβλητών με μονοπάτια του XML εγγράφου. Αυτές οι συσχετίσεις πρόκειται να χρησιμοποιηθούν ως αρχικές συνδέσεις (Initial bindings) στο υπό-στοιχείο λογισμικού **Variables Binder**. Αναλύεται στο **Κεφάλαιο 9**.

- **Variables Binder**

  Το υπό-στοιχείο λογισμικού **Variables Binder** αποτελεί μέρος του στοιχείου **Query Translator**. Δέχεται ως είσοδο βασικές σχηματομορφές γράφων (Basic graph pattern-BGP), δηλαδή σε ακολουθίες από τριπλέτες σχηματομορφών, ώστε να προσδιορίσει την συνδέσεις των μεταβλητών που περιέχονται στο BGP με μονοπάτια του XML εγγράφου. Η συνδέσεις των μεταβλητών της SPARQL ερώτησης με μονοπάτια του XML εγγράφου είναι απαραίτητο για να μπορέσει να πραγματοποιηθεί η μετάφρασης Βασικών Σχηματομορφών Γράφων σε XQuery από υπό-στοιχείο **Basic Graph Pattern Translator**. Αναλύεται στο **Κεφάλαιο 10**.

- **Basic Graph Pattern Translator**

  Το υπό-στοιχείο λογισμικού **Basic Graph Pattern Translator** αποτελεί μέρος του στοιχείου **Query Translator**. Ένα από τα βασικότερα και πιο πολύπλοκα κομμάτια της μετάφρασης. πραγματοποιεί την μετάφραση των βασικών σχηματομορφών γράφων, δέχεται ως είσοδο βασικές σχηματομορφές γράφων και παράγει τις αυστηρά σημασιολογικά ισοδύναμες XQuery εκφράσεις. Αναλύεται στο **Κεφάλαιο 11**.

- **Graph Pattern Translator**

  Το υπό-στοιχείο λογισμικού **Graph Pattern Translator** αποτελεί μέρος του στοιχείου **Query Translator**. Πραγματοποιεί την μετάφραση των σχηματομορφών γράφων, δέχεται ως είσοδο μια σχηματομορφή γράφου και τις μεταφράσεις των βασικών σχηματομορφών γράφων(που πραγματοποιήθηκε από το υπό-στοιχείο **Basic Graph Pattern Translator**) που την αποτελούν και πραγματοποιεί την μετάφραση της. Αναλύεται στο **Κεφάλαιο 12**.

- **Solution Sequence Modifier Translator**

  Το υπό-στοιχείο λογισμικού **Solution Sequence Modifier Translator** αποτελεί μέρος του στοιχείου **Query Translator**. Πραγματοποιεί την μετάφραση των Τροποποιητών Ακολουθίας Λύσεων που προσφέρονται από την γλώσσα SPARQL. Αναλύεται στο **Κεφάλαιο 13**.

- **Query Type Translator**

  Το υπό-στοιχείο λογισμικού **Query Type Translator** αποτελεί μέρος του στοιχείου **Query Translator**. Πραγματοποιεί την μετάφραση της μορφής της ερώτησης, όπως έχει αναλυθεί και στην Παράγραφο 2.7.3 η γλώσσα SPARQL έχει τέσσερις μορφές ερωτήσεων, ανάλογα με την μορφή της ερώτησης είναι και η μορφή των αποτελεσμάτων, για τον λόγο αυτόν διαφοροποιείται και ο τρόπος μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους. Αναλύεται στο **Κεφάλαιο 14**.

Για την ανάπτυξη των παραπάνω στοιχείων λογισμικού έγινε χρήση των τεχνολογιών : **JAVA** και **Jena API** [42] που αναλύεται στην ενότητα 2.11.

- **Result Transformer**

  Το στοιχείο λογισμικού **Result Transformer** αποτελεί μέρος του πλαισίου **SPARQL2XQuery**. Πραγματοποιεί τον μετασχηματισμό των αποτελεσμάτων των XQuery ερωτήσεων. Δέχεται ως είσοδο τα XML δεδομένα και τα μετασχηματίζει, σύμφωνα με την XML μορφή των SPARQL αποτελεσμάτων, που προτείνεται από το W3C [56]. Ο τρόπος μετασχηματισμού των αποτελεσμάτων αναλύεται στο **Κεφάλαιο 16**.

  Για την ανάπτυξη του στοιχείο λογισμικού **Result Transformer** έγινε χρήση των τεχνολογιών : **JAVA**, **XML Beans** [52] που αναλύεται στην ενότητα 2.9 και **Jena API** [42] που αναλύεται στην ενότητα 2.11.

4.4 Ανάλυση του Query Translator Component – Διαδικασία Μετάφρασης Σημασιολογικών Ερωτήσεων

Στην παρούσα ενότητα αναλύεται η διαδικασία μετάφρασης σημασιολογικών SPARQL ερωτήσεων σε σημασιολογικά ισοδύναμες XQuery ερωτήσεις. Η διαδικασία της μετάφρασης πραγματοποιείται από το **Query Translator** component της αρχιτεκτονικής της Εικόνας 4.1. Η διαδικασία της μετάφρασης προσομοιώνεται με UML διαγράμματα δραστηριοτήτων (activity diagrams) ώστε να είναι ξεκάθαρη η σειρά και ο τρόπος εκτελέσεις των δραστηριοτήτων που εκτελούνται ώστε να πραγματοποιηθεί η μετάφραση των ερωτήσεων. Οι δραστηριότητες έχουν χωριστεί σε κεφάλαια του κειμένου ώστε να διευκολύνεται η ανάγνωση και η κατανόηση τους.

Κατά την ανάπτυξη των δραστηριοτήτων έγινε προσπάθεια να αποτελούν μια όσο το δυνατό πιο γενική δραστηριότητα, η οποία ανεξαρτητοποιείται από τον τρόπο ορισμού, αποθήκευσης και ελέγχου συνέπειας των αντιστοιχήσεων (mappings). Η ανάπτυξη και ο ορισμός των δραστηριοτήτων έγινε με βάση τους παρακάτω άξονες :

α) Δυνατότητα μετάφρασης **όλων των πιθανών ερωτήσεων**, καλύπτοντας όλους τους δυνατούς συνδυασμούς της γραμματικής της γλώσσας SPARQL.

β) **Αυστηρή** τήρηση της σημασιολογίας της γλώσσας κατά την διαδικασία της μετάφρασης.

γ) Οι **ακολουθίες λύσεων** που προκύπτουν από το μεταφρασμένο XQuery ερώτημα **είναι οι επιθυμητές** και δεν απαιτούν κάποια περεταίρω επεξεργασία (από API, Software), επομένως η διαδικασία **ανεξαρτητοπείται** από το query engine και το περιβάλλον εκτέλεσης της XQuery ερώτησης.

δ) Ανάπτυξη μιας **γενικής** και **κατανοητής** διαδικασίας και αλγορίθμων μετάφρασης. ε) Παραγωγή όσο το δυνατόν **μικρότερων** και **λιγότερο πολύπλοκων** εκφράσεων XQuery

ζ) Ανάπτυξη και σύνταξη των μεταφρασμένων ερωτήσεων με τρόπο ώστε οι "**αντιστοιχίες**" μεταξύ των δυο ερωτήσεων(SPARQL-XQuery) και ο **τρόπος μετάφρασης** να γίνονται **εύκολα αντιληπτά**.

η) Σε συνδυασμό με τους άλλους στόχους όσο το δυνατόν **βελτιστοποίηση** (Optimization) των XQuery.

θ) Η παραγωγή μιας και μόνο XQuery ερώτησης για κάθε μία SPARQL ερώτηση που δίνεται προς μετάφραση χωρίς την εκτέλεση ενδιάμεσων ερωτήσεων

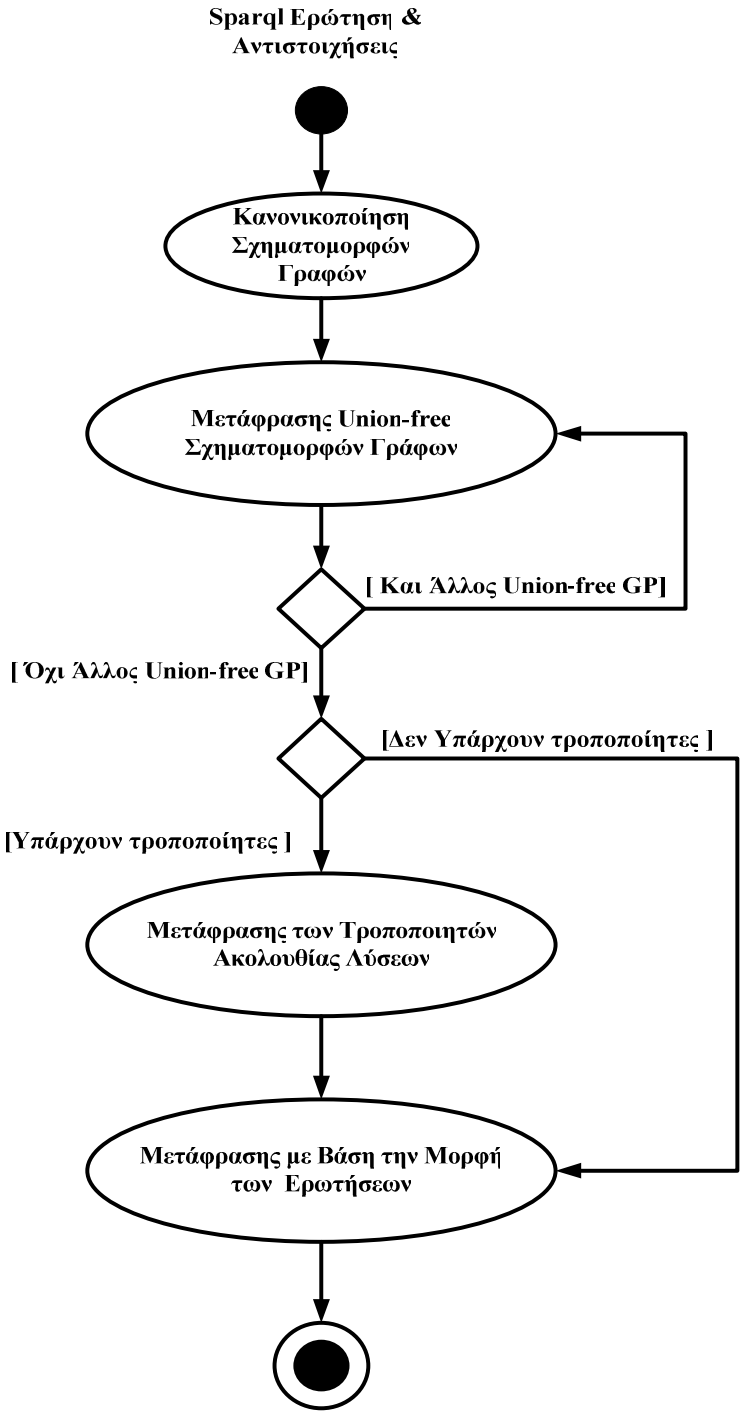
4.4.1 Περιγραφή της Διαδικασίας

Η διαδικασία μετάφρασης σημασιολογικών ερωτήσεων έχει ως αποτέλεσμα την μετάφραση σημασιολογικών SPARQL ερωτήσεων που αναφέρονται σε RDF δεδομένα που περιγράφονται από μια OWL οντολογία , σε XQuery ερωτήσεις οι οποίες στην συνέχεια θα αποτιμηθούν σε XML δεδομένα. Για την καλύτερη κατανόηση της διαδικασίας στο σύνολο της , επιλέχτηκε να γίνει ανάλυση και μοντελοποίηση της με UML διαγράμματα δραστηριοτήτων

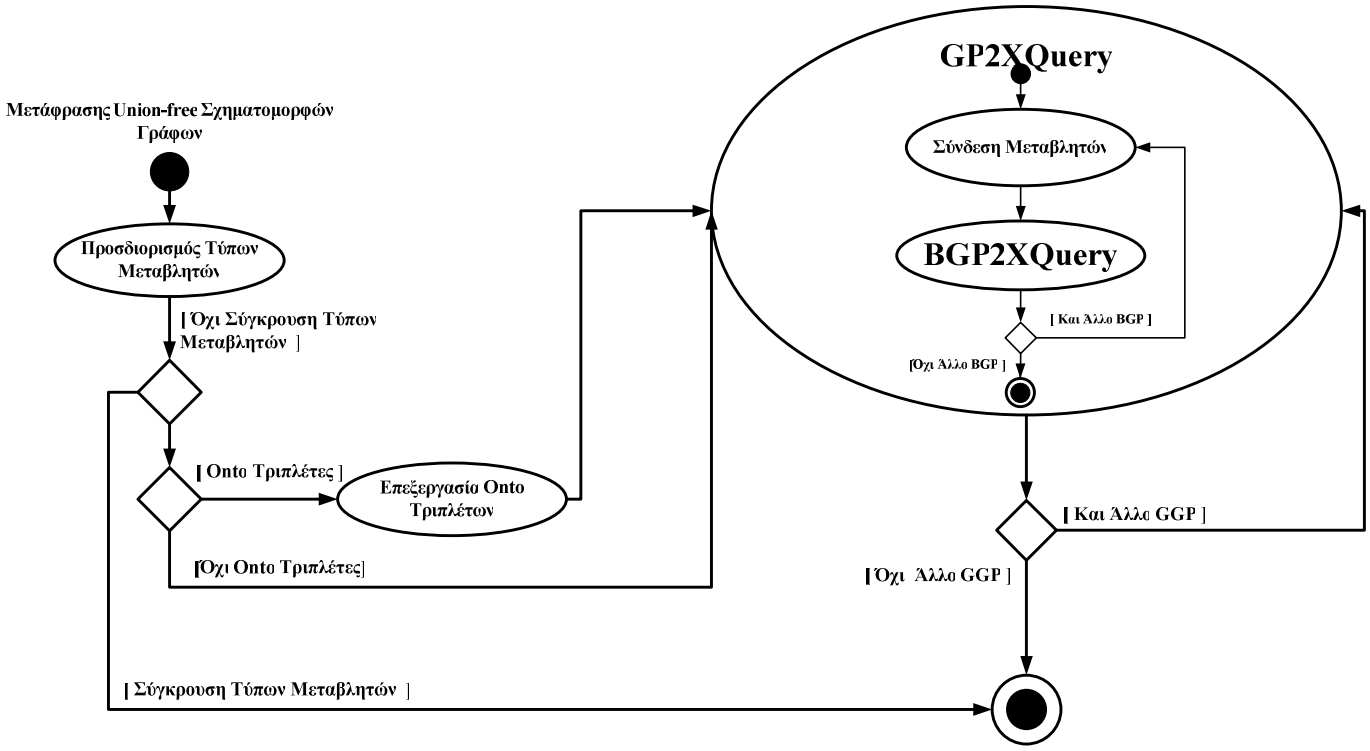
Τα UML διαγράμματα δραστηριοτήτων που μοντελοποιούν την διαδικασία μετάφρασης των ερωτήσεων, φαίνονται στις Εικόνες 4.3 και 4.4 . Στο πρώτο διάγραμμα Εικόνα 4.3 αναλύεται η διαδικασία μετάφρασης των σημασιολογικών ερωτήσεων, ενώ στο δεύτερο διάγραμμα Εικόνα 4.4 αναλύεται η σύνθετη δραστηριότητα "**Μετάφρασης Union- free Σχηματομορφών Γράφων**" που εμφανίζεται στο πρώτο διάγραμμα.

Η διαδικασία μετάφρασης ξεκινά παίρνοντας ως παραμέτρους την SPARQL ερώτηση που πρόκειται να μεταφραστεί και τις αντιστοιχήσεις μεταξύ οντολογίας και XML σχήματος. Η πρώτη δραστηριότητα είναι η "**Κανονικοποίηση Σχηματομορφών Γράφων**" η οποία κάνοντας χρήση κανόνων ισοδυναμίας μετασχηματίζει την σχηματομορφή γράφου της ερώτησης και αναλύεται στο **Κεφάλαιο 7**. Στην συνέχεια φαίνεται η σύνθετη δραστηριότητα "**Μετάφρασης Union-free Σχηματομορφών Γράφων**" η οποία μεταφράζει τις Union-free Σχηματομορφών Γράφων σε XQuery εκφράσεις, αναλύεται στην Εικόνα 4.4. Ακολουθεί η δραστηριότητα "**Μετάφρασης των Τροποποιητών Ακολουθίας Λύσεων"** η οποία μεταφράζει τους τροποποιητές(όπως DISTINCT, ORDER BY κτλ) που πιθανά περιέχει η SPARQL ερώτηση σε XQuery εκφράσεις, αναλύεται στο **Κεφάλαιο 13**. Τέλος η δραστηριότητα "**Μετάφρασης με Βάση την Μορφή των Ερωτήσεων**" η οποία ανάλογα με τον τύπο της SPARQL ερώτησης διαφοροποιεί τις XQuery εκφράσεις ώστε να παράγουν την επιθυμητή ακολουθία λύσεων και αναλύεται στο **Κεφάλαιο 14**.

Στο δεύτερο διάγραμμα (Εικόνα 4.4) αναλύεται η συνθέτη δραστηριότητα "**Μετάφρασης Union-free Σχηματομορφών Γράφων**" που εμφανίζεται στο πρώτο διάγραμμα. Η πρώτη δραστηριότητα είναι η "**Προσδιορισμός Τύπων Μεταβλητών**" η οποία προσδιορίζει τον τύπο των μεταβλητών που περιέχονται στην σχηματομορφή γράφου της ερώτησης και αναλύεται στο **Κεφάλαιο 8**. Στην συνέχεια φαίνεται η δραστηριότητα "**Επεξεργασία Onto Τριπλετών**" η οποία πραγματοποιεί την επεξεργασία των Onto Τριπλετών και αναλύεται στο **Κεφάλαιο 9**. Ακολουθεί η σύνθετη δραστηριότητα με το όνομα "**GP2XQuery**" η οποία πραγματοποιεί την μετάφραση Σχηματομορφών Γράφων (Graph Patterns) σε XQuery εκφράσεις και περιέχει στο εσωτερικό τις δραστηριότητες "**Σύνδεση Μεταβλητών**" η οποία πραγματοποιεί την σύνδεση των μεταβλητών με μονοπάτια του XML εγγράφου και αναλύεται στο **Κεφάλαιο 10** και " **BGP2XQuery**" η οποία πραγματοποιεί την μετάφραση Βασικών Σχηματομορφών Γράφων (Basic Graph Patterns) σε XQuery εκφράσεις και αναλύεται στο **Κεφάλαιο 11**. Τέλος η "**GP2XQuery**" αναλύεται στο **Κεφάλαιο 12**.



Εικόνα 4.3 : Διαδικασία μετάφρασης σημασιολογικών ερωτήσεων



Εικόνα 4.4 : Διαδικασία μετάφρασης σημασιολογικών ερωτήσεων

4.5 Περίληψη

Στο κεφάλαιο αυτό, έγινε η περιγραφή της γενικής αρχιτεκτονικής στην οποία εντάσσεται το πλαίσιο SPARQL2XQuery, όπως επίσης και η ανάλυση της αρχιτεκτονικής του ιδίου του πλαισίου και των επιμέρους στοιχείων και υπό-στοιχείων λογισμικού. Επίσης έγινε μια γενική επισκόπηση της διαδικασίας Μετάφρασης Σημασιολογικών Ερωτήσεων και η προσομοίωση της με UML διαγράμματα δραστηριοτήτων.

Στα επόμενα κεφάλαια του κειμένου θα γίνει η ανάλυση των αντιστοιχήσεων (Κεφάλαιο 5), της διαδικασίας ανακάλυψης(discovery) και του αυτόματου τρόπου παραγωγής των αντιστοιχήσεων (Κεφάλαιο 6) και τέλος θα γίνει η ανάλυση της διαδικασία μετάφρασης των σημασιολογικών ερωτήσεων (Κκεφάλαια 7- 12).

5 Αντιστοιχήσεις

5.1 Εισαγωγή

Όπως έχει αναφερθεί στο κεφάλαιο 1 και αναλύεται στο κεφάλαιο 6, το πλαίσιο **SPARQL2XQuery** παρέχει την δυνατότητα αυτόματης ανακάλυψης(discover) και παραγωγής των αντιστοιχήσεων μεταξύ Οντολογίας και XML σχήματος, στην περίπτωση στην οποία η οντολογία έχει παραχθεί με χρήση της μεθοδολογίας **XS2OWL** [15][16][17][18]. Σε αντίθετη περίπτωση οι αντιστοιχήσεις ορίζονται "χειροκίνητα" από κάποιον εξειδικευμένο χρήστη γνώστης των εννοιών που περιγράφονται από την οντολογίας και από το XML σχήμα.

Για την ανεξαρτητοποίηση της διαδικασίας μετάφρασης από τη μορφή ,την σύνταξη και τον τρόπο δημιουργίας και αποθήκευσης των αντιστοιχήσεων όπως επίσης και από το σύστημα XS2OWL ορίζεται ένας αφηρημένος τρόπος αναπαράστασης των αντιστοιχήσεων, καθώς και ένα σύνολο από "κανόνες" που πρέπει να τηρούνται κατά την δημιουργία των αντιστοιχήσεων. Συνεπώς η οντολογία δεν απαιτείται να

έχει προκύψει(παραχθεί) από ένα XML σχήμα. Το μόνο που απαιτείται είναι οι αντιστοιχήσεις που ορίζονται να τηρούν τους "κανόνες" που θα περιγραφούν παρακάτω. Το γεγονός αυτό κάνει τη μεθοδολογία μετατροπής των ερωτήσεων ανεξάρτητη από συγκεκριμένα εργαλεία και μεθοδολογίες αντιστοιχήσεων. Έτσι, καθίσταται δυνατή η μετάφραση ερωτήσεων που αφορούν σε προϋπάρχουσες οντολογίες σε ερωτήσεις XQuery για XML σχήματα ανεξάρτητα προς τις οντολογίες αυτές. Σχήματα για τα οποία ωστόσο υπάρχουν διαθέσιμες αντιστοιχίσεις με την προϋπάρχουσα οντολογία.

Αρχικά θα πρέπει να γίνει αντιληπτό, ότι σε αυτό το σημείο δεν παρουσιάζεται ένας αυστηρός τρόπος ορισμού αντιστοιχήσεων μεταξύ οντολογιών και XML σχήματος , ούτε ορίζονται προϋποθέσεις για τον έλεγχο της σημασιολογικής ορθότητας των αντιστοιχήσεων, καθώς δεν αποτελεί μέρος της παρούσας εργασίας. Ορίζεται μια σειρά από "περιορισμούς" που πρέπει να τηρούνται κατά την δήλωση των αντιστοιχήσεων ώστε να είναι δυνατή η μετάφραση των ερωτήσεων.

5.2 Παράδειγμα

Για την καλύτερη κατανόηση της διαδικασίας στην συνέχεια του κεφαλαίου συμπεριλαμβάνονται μια σειρά από παραδείγματα, βασισμένα στο XML σχήμα και την Οντολογία που περιγράφονται στις παρακάτω υπό-ενότητες . Στο συγκεκριμένο παράδειγμα η οντολογία έχει παραχθεί με χρήση της μεθοδολογίας XS2OWL [17].

5.2.1 XML σχήμα

Το XML σχήμα που φαίνεται στην Εικόνα 5.1 καθώς και η δενδρική του αναπαράσταση στην Εικόνα 5.2, επιτρέπει την αναπαράσταση περιγραφών προσώπων και απαρτίζεται από:

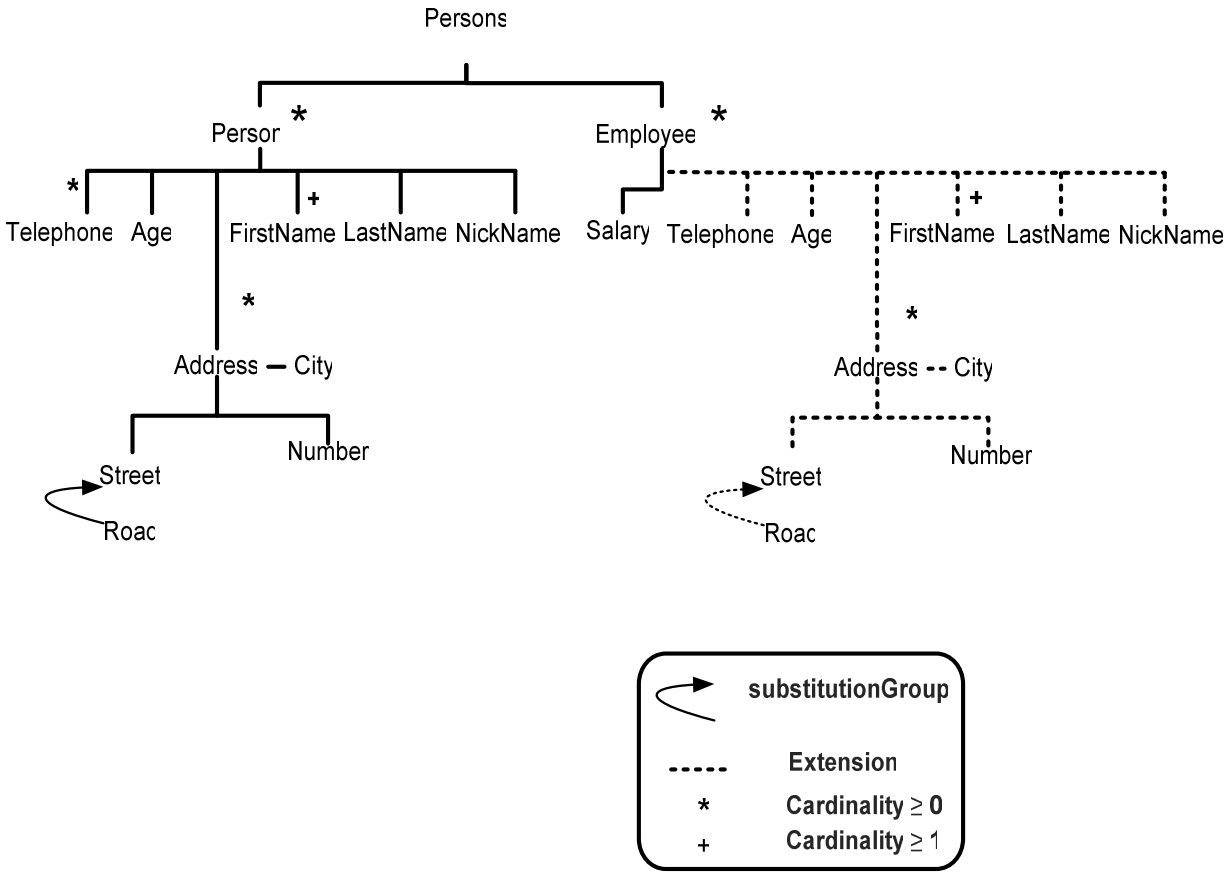
- Τα XML Schema στοιχεία σύνθετου τύπου :"Persons" (τύπου "Persons_Type") που αναπαριστά λίστες προσώπων και εργαζομένων ,"Person" (τύπου "Person_Type") που αναπαριστά πρόσωπα , "Employee" (τύπου " Employee_Type ") που αναπαριστά εργαζομένους και το "Address" (τύπου " Address_Type ") που αναπαριστά την διεύθυνση. Το στοιχείο "Persons" είναι το στοιχείο-ρίζα του σχήματος.
- Τα XML Schema στοιχεία απλού τύπου :"FirstName" (τύπου "xs:String") που αναπαριστά το μικρό όνομα , "NickName" (τύπου "xs:String") που αναπαριστά το ψευδώνυμο, "LastName" (τύπου "xs:String") που αναπαριστά το επίθετο, "Telephone" (τύπου "xs:String") που αναπαριστά τους πιθανούς τηλεφωνικούς αριθμούς , "Age" (τύπου "validAgeType ") που αναπαριστά την ηλικία, "Salary" (τύπου "xs:integer") που αναπαριστά τον μισθό, "Street" (τύπου

"xs:string") που αναπαριστά την οδό ,"Road" (τύπου "xs:string") που αναπαριστά τον δρόμο, "Number" (τύπου "xs:integer") που αναπαριστά τον αριθμό διεύθυνσης.

- Το XML Schema γνώρισμα "city", που αναπαριστά την πόλη.
- Τους σύνθετους XML Schema τύπους, "Persons_Type" (που αναπαριστά λίστες προσώπων), "Person_Type" (που αναπαριστά πρόσωπα), "Employee_Type" (που εξειδικεύει τον τύπο "Person_Type" και αναπαριστά εργαζόμενους), "Address_Type" (που αναπαριστά διευθύνσεις).
- Τον απλό XML Schema τύπο "validAgeType", που ορίζει τις επιτρεπτές τιμές ηλικίας.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:complexType name="Person_Type">
  <xs:sequence>
   <xs:element name="LastName" type="xs:string"/>
   <xs:element name="FirstName" type="xs:string" minOccurs="1" maxOccurs="unbounded"/>
   <xs:element name="NickName" type="xs:string" minOccurs="0"/>
   <xs:element name="Age" type="validAgeType"/>
   <xs:element name="Telephone" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="Address" type="Adress_Type" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="Employee_Type">
  <xs:complexContent>
   <xs:extension base="Person_Type">
    <xs:sequence>
     <xs:element name="Salary" type="xs:integer"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
 </xs:complexType>

 <xs:complexType name="Adress_Type">
  <xs:sequence>
   <xs:element ref="Street"/>
   <xs:element name="Number" type="xs:integer"/>
  </xs:sequence>
  <xs:attribute name="City" type="xs:string"/>
 </xs:complexType>

 <xs:simpleType name="validAgeType">
  <xs:restriction base="xs:float">
   <xs:minInclusive value="0.0"/>
   <xs:maxInclusive value="150.0"/>
  </xs:restriction>
 </xs:simpleType>


 <xs:element name="Street" type="xs:string"/>
 <xs:element name="Road" substitutionGroup="Street"/>

 <xs:element name="Persons">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Person" type="Person_Type" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="Employee" type="Employee_Type" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Εικόνα 5.1 : XML Σχήμα για την περιγραφή Προσώπων



Εικόνα 5.2 : Δενδρική αναπαράσταση του XML Σχήμα για την περιγραφή Προσώπων

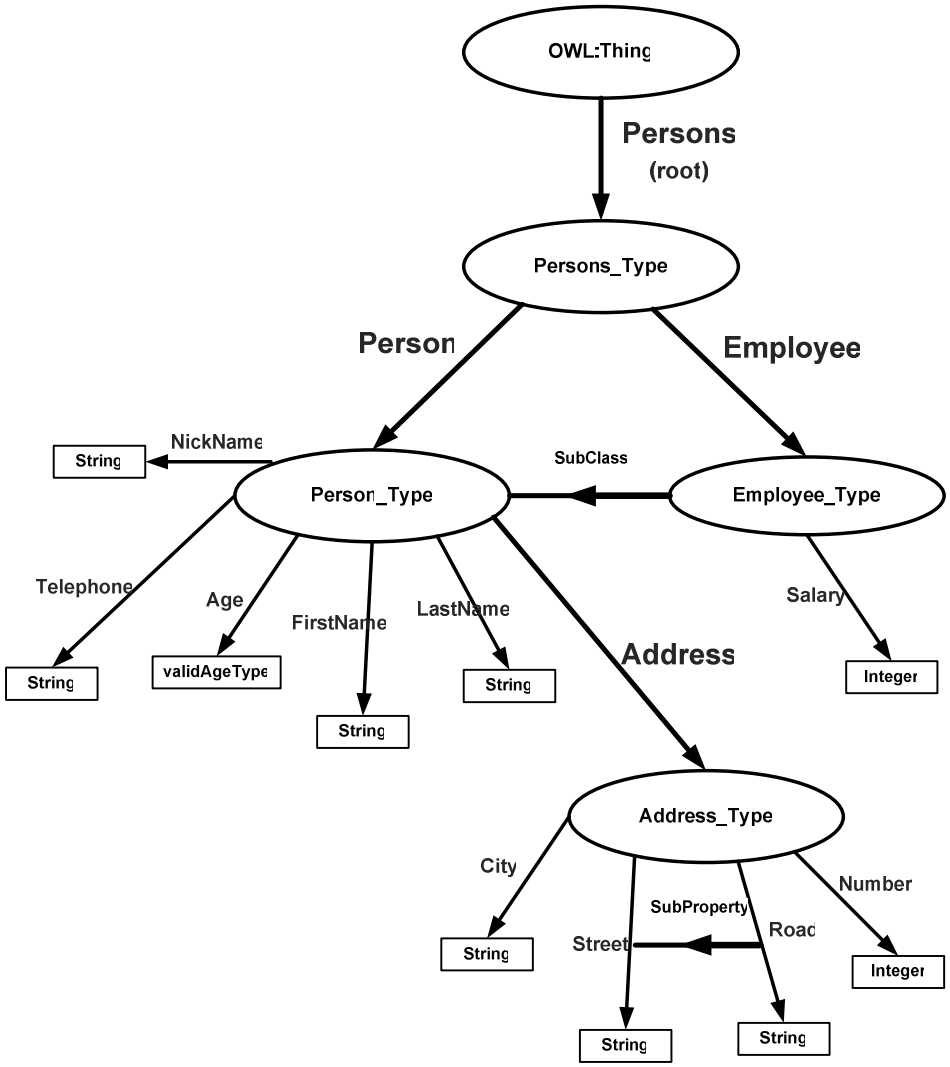
5.2.2 Οντολογία

Επίσης ο γράφος της οντολογίας που έχει παραχθεί με βάση το παραπάνω XML σχήμα από την εφαρμογή της μεθοδολογίας XS2OWL[17] , φαίνεται στην Εικόνα 5.3 και 5.4 απαρτίζεται από:

- **Τις Κλάσεις** : "Persons_Type", "Person_Type", "Employee_Type", "Address_Type"
  - Η κλάση "Employee_Type" είναι υπό-κλάση της κλάσης "Person_Type"
- **Τις Ιδιότητες Αντικειμένων** (Object Properties) : "Persons", "Person" ,"Employee" , "Address"

.

- **Τις Ιδιότητες Τύπων Δεδομένων** (Data Type Properties) : "Telephone", "Age", "FirstName" , NickName" , "LastName" , "Salary" ,"City" ,"Street" ,"Road" ,"Number".
    - Η ιδιότητα τύπου δεδομένου "Road" είναι υπό-ιδιότητα της ιδιότητας "Street".



Εικόνα 5.3 : Γράφος Οντολογίας για την περιγραφή Προσώπων

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
        <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
        <!ENTITY owl "http://www.w3.org/2002/07/owl#">
        <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
        <!ENTITY datatypes "datatypes.xsd#">
        <!ENTITY   xml "http://www.w3.org/XML/1998/namespace#">
        <!ENTITY   xs "http://www.w3.org/2001/XMLSchema#">
]>
```

```
<rdf:RDF  xmlns:xs="http://www.w3.org/2001/XMLSchema#" xmlns:owl="&owl;" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;" xmlns:datatypes="&datatypes;">
        <owl:Ontology rdf:about="">
        </owl:Ontology>

        <rdfs:Datatype rdf:about="&datatypes;validAgeType">
                <rdfs:isDefinedBy rdf:resource="&datatypes;"/>
                <rdfs:label>validAgeType</rdfs:label>
        </rdfs:Datatype>
        <!--        Class Definitions           -->
        <owl:DatatypeProperty rdf:ID="City">
                <rdfs:domain rdf:resource="#Adress_Type"/>
                <rdfs:range rdf:resource="&xs;string"/>
                <rdfs:label>City</rdfs:label>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="LastName">
                <rdfs:domain rdf:resource="#Person_Type"/>
                <rdfs:range rdf:resource="&xs;string"/>
                <rdfs:label>LastName</rdfs:label>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="FirstName">
                <rdfs:domain rdf:resource="#Person_Type"/>
                <rdfs:range rdf:resource="&xs;string"/>
                <rdfs:label>FirstName</rdfs:label>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="NickName">
                <rdfs:domain rdf:resource="#Person_Type"/>
                <rdfs:range rdf:resource="&xs;string"/>
                <rdfs:label>NickName</rdfs:label>
        </owl:DatatypeProperty>
        <owl:ObjectProperty rdf:ID="Age">
                <rdfs:domain rdf:resource="#Person_Type"/>
                <rdfs:range rdf:resource="#validAgeType"/>
                <rdfs:label>Age</rdfs:label>
        </owl:ObjectProperty>
        <owl:DatatypeProperty rdf:ID="Telephone">
                <rdfs:domain rdf:resource="#Person_Type"/>
                <rdfs:range rdf:resource="&xs;string"/>
                <rdfs:label>Telephone</rdfs:label>
        </owl:DatatypeProperty>
        <owl:ObjectProperty rdf:ID="Address">
                <rdfs:domain rdf:resource="#Adress_Type"/>
                <rdfs:range rdf:resource="#Adress_Type"/>
                <rdfs:label>Address</rdfs:label>
        </owl:ObjectProperty>
        <owl:DatatypeProperty rdf:ID="Salary">
                <rdfs:domain rdf:resource="#Empoloyee_Type"/>
                <rdfs:range rdf:resource="&xs;integer"/>
                <rdfs:label>Salary</rdfs:label>
        </owl:DatatypeProperty>
        <owl:DatatypeProperty rdf:ID="Number">
                <rdfs:domain rdf:resource="#Adress_Type"/>
                <rdfs:range rdf:resource="&xs;integer"/>
                <rdfs:label>Number</rdfs:label>
        </owl:DatatypeProperty>
        <owl:ObjectProperty rdf:ID="Person">
                <rdfs:domain rdf:resource="#Persons_Type"/>
                <rdfs:range rdf:resource="#Person_Type"/>
                <rdfs:label>Person</rdfs:label>
        </owl:ObjectProperty>
        <owl:ObjectProperty rdf:ID="Employee">
                <rdfs:domain rdf:resource="#Persons_Type"/>
                <rdfs:range rdf:resource="#Employee_Type"/>
                <rdfs:label>Employee</rdfs:label>
        </owl:ObjectProperty>
        <owl:Class rdf:ID="Person_Type">
                <rdfs:subClassOf>
                        <owl:Class>
                                <owl:intersectionOf rdf:parseType="Collection">
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#LastName__xs_string"/>
                                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                                </owl:Restriction>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#FirstName"/>
                                        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
                                </owl:Restriction>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#NickName"/>
                                        <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
                                </owl:Restriction>
                                <owl:Restriction>
                                        <owl:onProperty rdf:resource="#Age"/>
```

```
                                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                                    </owl:Restriction>
                                    <owl:Restriction>
                                        <owl:onProperty rdf:resource="#Address"/>
                                        <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
                                    </owl:Restriction>
                                    </owl:intersectionOf>
                                </owl:Class>
                            </rdfs:subClassOf>
                            <rdfs:label>Person_Type</rdfs:label>
                        </owl:Class>

                        <owl:Class rdf:ID="Employee_Type">
                            <rdfs:subClassOf rdf:resource="#Person_Type"/>
                            <rdfs:subClassOf>
                                <owl:Class>
                                    <owl:intersectionOf rdf:parseType="Collection">
                                    <owl:Restriction>
                                    <owl:onProperty rdf:resource="#Salary"/>
                                    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                                </owl:Restriction>
                            </owl:intersectionOf>
                        </owl:Class>
                        </rdfs:subClassOf>
                            <rdfs:label>Employee_Type</rdfs:label>
                        </owl:Class>

                        <owl:Class rdf:ID="Adress_Type">
                            <rdfs:subClassOf>
                                <owl:Restriction>
                                    <owl:onProperty rdf:resource="#City"/>
                                    <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
                                </owl:Restriction>
                            </rdfs:subClassOf>
                            <rdfs:subClassOf>
                                <owl:Class>
                                    <owl:intersectionOf rdf:parseType="Collection">
                                    <owl:Restriction>
                                        <owl:onProperty rdf:resource="#Street"/>
                                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                                    </owl:Restriction>
                                    <owl:Restriction>
                                        <owl:onProperty rdf:resource="#Number"/>
                                        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
                                    </owl:Restriction>
                                    </owl:intersectionOf>
                                </owl:Class>
                            </rdfs:subClassOf>
                            <rdfs:label>Adress_Type</rdfs:label>
                        </owl:Class>

                        <owl:DatatypeProperty rdf:ID="Street">
                            <rdfs:domain rdf:resource="#Adress_Type"/>
                            <rdfs:range rdf:resource="&xs;string"/>
                            <rdfs:label>Street</rdfs:label>
                        </owl:DatatypeProperty>
                        <owl:ObjectProperty rdf:ID="Road">
                            <rdfs:domain rdf:resource="#Adress_Type"/>
                            <rdfs:subPropertyOf rdf:resource="#Street"/>
                            <rdfs:label>Road</rdfs:label>
                        </owl:ObjectProperty>

                        <owl:Class rdf:ID="Persons_Type">
                            <rdfs:label>Persons_Type</rdfs:label>
                        </owl:Class>
                        <owl:ObjectProperty rdf:ID="Persons">
                            <rdfs:domain rdf:resource="#Persons_Type"/>
                            <rdfs:range rdf:resource="#Persons_Type"/>
                            <rdfs:label>Persons</rdfs:label>
                        </owl:ObjectProperty>
</rdf:RDF>
```

Εικόνα 5.4 : OWL Περιγραφή της οντολογίας για την περιγραφή Προσώπων

5.3 Αντιστοιχίες σε Επίπεδο Γλωσσών (Language Level Correspondences)

Σε αυτή την ενότητα παρουσιάζονται οι αντιστοιχίες(correspondences) σε επίπεδο γλωσσών(Πίνακας 5.1), μεταξύ της γλώσσας XML Schema και της γλώσσας OWL. Αυτές οι αντιστοιχίες πρέπει να ακολουθούνται κατά την διαδικασία δημιουργίας των αντιστοιχήσεων(mappings), είτε αυτές προκύπτουν χειροκίνητα (manually) είτε αυτόματα (automatically) όπως στην περίπτωση του XS2OWL[17]. Οι αντιστοιχίες αυτές είναι κοινώς αποδεκτές, ακολουθούνται από το XS2OWL[17] αλλά και από ένα μεγάλο αριθμό προσεγγίσεων [26][28][32][35][61] για ενοποίηση πληροφορίας (data integration) μεταξύ οντολογιών και XML σχήμα, αλλά και από πολλά συστήματα παραγωγής οντολογιών από XML σχήμα [36][37][38] .

OWL Construct	***XML Schema Construct***
Class	Complex Type
Data Type Property	Simple Elements or Attributes
Object Property	Complex Element

Πίνακας 5.1 : OWL και XML Schema αντιστοιχίες

5.4 Διαδικασία Δημιουργία Αντιστοιχήσεων

Στη διαδικασία της αντιστοίχησης το XML σχήμα θεωρηθείται ως ένα σύνολο **XMLS= {CT , ST , CE , SE , AT}** ,οπού οι CT (Complex Types) είναι οι σύνθετοι τύποι δεδομένων ,ST (Simple Types) είναι απλοί τύποι δεδομένων, SE (Simple Elements) είναι απλού τύπου στοιχεία , CE (Complex Elements) είναι σύνθετου τύπου στοιχεία και AT (Attributes) είναι τα γνωρίσματα. Τα στοιχεία του συνόλου είναι αυτά τα οποία μπορούν να συμμετέχουν στην διαδικασία της αντιστοίχησης ,καθώς είναι αυτά τα οποία μπορούν να συνδεθούν άμεσα με τα δεδομένα. Από το σύνολο παραλείπονται άλλες έννοιες που μπορεί να περιέχονται στο XML σχήμα (πχ Σύνολα (xs:all) , Ακολουθίες (xs:sequence), Επιλογές (xs:choice) κτλ) δεδομένου ότι δεν χαρακτηρίζουν τα δεδομένα αυτά καθαυτά, αλλά τον τρόπο οργάνωσης τους.

Στη διαδικασία της αντιστοίχησης η οντολογία μπορεί να θεωρηθεί ως ένα σύνολο **O={ CL , DTP , OP }** ,οπού CL (Classes) είναι οι κλάσεις , DTP (Data Type Properties) είναι οι ιδιότητες τύπων δεδομένων και OP (Object Properties) οι ιδιότητες αντικειμένων. Στο σύνολο δεν περιέχονται έννοιες όπως Subclass

,Subproperty κτλ οι οποίες δεν συμμετέχουν άμεσα στην διαδικασία της αντιστοίχησης ,όμως υπολογίζονται κατά τον ορισμό τον αντιστοιχήσεων.

Η αντιστοίχηση μιας **Οντολογίας O** σε ένα **XML σχήμα XMLS** είναι ένα σύνολο από αντιστοιχήσεις (οι οποίες είναι σύμφωνες με τις αντιστοιχίες σε επίπεδο γλωσσών) και αναπαριστούν τις σημασιολογικές συσχετίσεις μεταξύ των στοιχείων των O και XMLS.

$$\mathrm{M}_{[O,XMLS]} = \{\mu(o_i, xmls_i) | \mathrm{o}_i \in O \ \text{και} \ \mathrm{xmls}_i \in XMLS\}$$

Όπου η αντιστοίχηση $\mu(o_i, xmls_i)$ μεταξύ δυο στοιχείων των συνόλων O και XMLS αναπαριστά μια συσχέτιση σημασιολογικής ισοδυναμίας μεταξύ τους. Είναι εφικτό για κάθε στοιχείο των δυο συνόλων να ορίζονται παραπάνω από μια αντιστοιχίσεις.

5.5 Αναπαράσταση Αντιστοιχήσεων

Η αντιστοίχηση ενός στοιχείου o_i του συνόλου O με ένα στοιχείο χ_i του συνόλου XMLS , δημιουργεί την αντιστοιχία σε "επίπεδο δεδομένων" του στοιχείου o_i με τα πιθανά μονοπάτια (Xpaths) που μπορεί να εμφανιστεί το στοιχείο χ_i στα XML δεδομένα, όπως φαίνεται και στην Εικόνα 5.5.

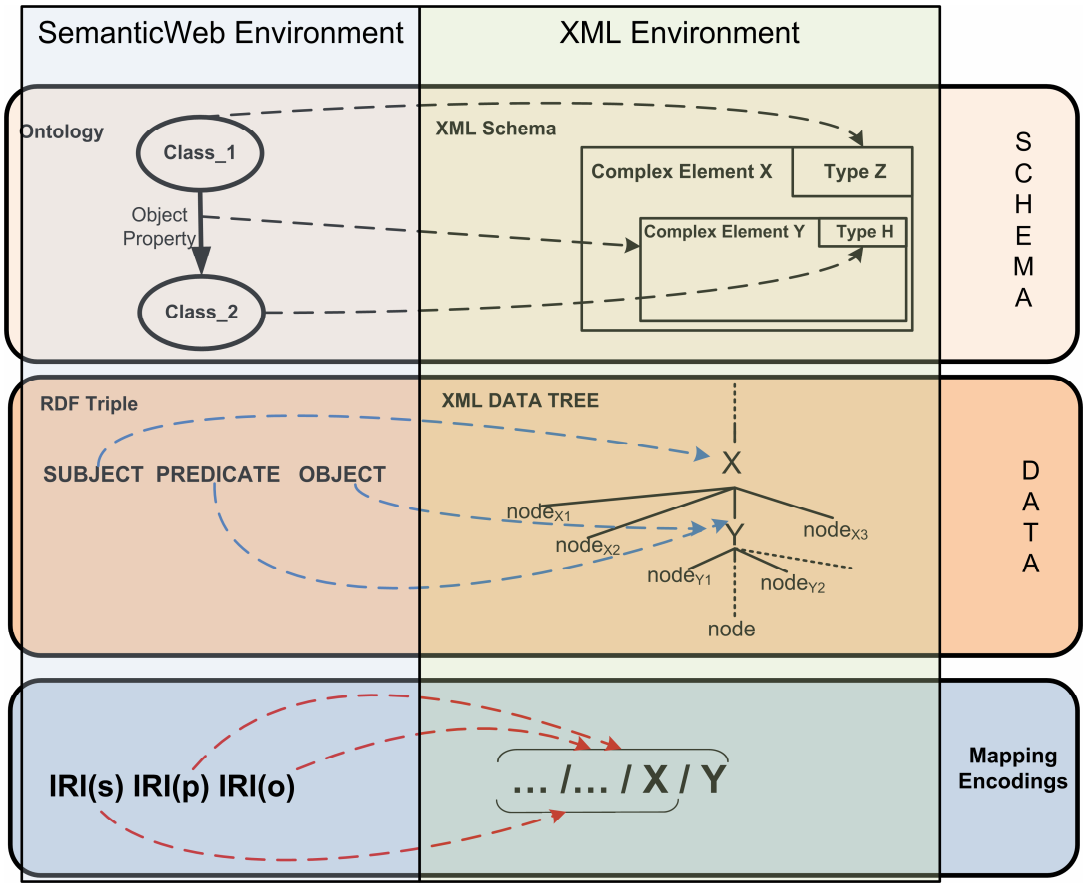
Με αυτό τον τρόπο επιτυγχάνεται η "άμεση" σύνδεση των στοιχείων της οντολογίας με τα XML δεδομένα. Οι αντιστοιχήσεις αυτές θα χρησιμοποιηθούν κατά τις διαδικασίες της μεταφράσεις (Κεφάλαια 9-12) για την ανάπτυξη των XQuery εκφράσεων, καθώς τα στοιχεία της οντολογίας που περιέχονται στις SPARQL ερωτήσεις θα αντικατασταθούν με μονοπάτια στις XQuery εκφράσεις.

> Σημείωση: *Ο όρος μονοπάτι αναφέρεται σε μονοπάτια της δενδρικής XML αναπαράστασης, γνωστά ως Xpaths. Στην παρούσα προσέγγιση τα μονοπάτια είναι της μορφής …/x/y/z/… , δεν περιέχουν κατηγορήματα(predicates) ή Xpaths συναρτήσεις. Τέλος τα μονοπάτια εμφανίζουν τον χαρακτήρα * (Wildcard) μόνο στο τελευταίο μέρος τους(δηλαδή ως κόμβο φύλο-leaf node). Επίσης δεν εμφανίζονται μονοπάτια της μορφής //*.*

Έστω το στοιχείο χ_i είναι ένας σύνθετος τύπος δεδομένων, τότε η αντιστοιχία που δημιουργείτε περιέχει όλα τα μονοπάτια των σύνθετων στοιχείων που είναι τύπου χ_i. Έστω ότι το χ_i είναι ο σύνθετος τύπος Person_Type από το XML σχήμα της ενότητας 5.2.1, τα μονοπάτια στο XML έγγραφο στα οποία βρίσκονται στοιχεία (elements) με αυτό τον τύπο είναι τα : /Persons/Person και /Persons/Employee(Σε αυτή την περίπτωση περιέχονται και τα στοιχεία τύπου Employee_Type, καθώς ο τύπος αυτός αποτελεί επέκταση (extension) του τύπου Person_Type). Στην περίπτωση που το χ_i είναι ο σύνθετος τύπος Employee_Type από το XML σχήμα της ενότητας 5.2.1, τα μονοπάτια στο XML έγγραφο στα οποία βρίσκονται στοιχεία (elements) με αυτό τον τύπο είναι τα : /Persons/Employee.

Μια άλλη περίπτωση είναι το χ_i το οποίο αποτελεί ένα στοιχείο (σύνθετο η απλό) που έχει οριστεί στο εσωτερικό ενός σύνθετου τύπου κορυφαίου επιπέδου (Top Level), η αντιστοιχία που δημιουργείται περιέχει όλα τα μονοπάτια των στοιχείων, που θα εμφανίζονται στο εσωτερικό σύνθετων στοιχείων με τύπο, τον σύνθετο τύπο στον οποίο περιέχεται το στοιχείο. Έστω ότι το χ_i είναι το απλό στοιχείο FirstName που ορίζεται στο εσωτερικό του σύνθετου τύπου(κορυφαίου επιπέδου) Person_Type από το XML σχήμα της ενότητας 5.2.1, τα μονοπάτια στο XML έγγραφο στα οποία βρίσκεται το στοιχείο (elements) αυτό είναι: /Persons/Person/FirstName και /Persons/Employee/FirstName

Συμπέρασμα 5.1 Όταν γίνεται αντιστοίχηση ενός στοιχείου o_i του συνόλου O με ένα στοιχείο χ_i του συνόλου XMLS και το χ_i είναι απλός ή σύνθετος τύπος , η αντιστοιχία που δημιουργείται σε "επίπεδο δεδομένων" είναι μεταξύ o_i και των πιθανών μονοπατιών των στοιχείων η γνωρισμάτων που είναι τύπου χ_i .



Εικόνα 5.5 : Αντιστοιχήσεις μεταξύ των Semantic και XML περιβαλλόντων

Οι αντιστοιχήσεις που προκύπτουν είναι μεταξύ των εννοιών και συσχετίσεων της οντολογίας (Κλάσεις και Ιδιότητες) και μονοπατιών των XML δεδομένων. Για την ανεξαρτητοποίηση της διαδικασίας μετάφρασης από τη μορφή ,την σύνταξη και τον τρόπο δημιουργίας και αποθήκευσης των αντιστοιχήσεων ορίζεται ένας αφηρημένος τρόπος αναπαράστασης των αντιστοιχήσεων. Για τον λόγο αυτό οι αντιστοιχήσεις θα αναπαρίστανται με σύνορα μονοπατιών, οι αντιστοιχήσεις αποτελούνται από τις αντιστοιχήσεις των κλάσεων και των ιδιοτήτων της οντολογίας. Επόμενος για τις αντιστοιχήσεις ισχύει :

- Για κάθε κλάση C ϵ CT που αντιστοιχίζεται, δημιουργείται ένα σύνολο από μονοπάτια. Το σύνολο αυτών των μονοπατιών συμβολίζεται με $\mathbf{X_c}$ **(Class Xpaths Set).** Το σύνολο των μονοπατιών $\mathbf{X_c}$ όπως έχει αναφερθεί περιέχει όλα τα πιθανά μονοπάτια στα XML δεδομένα στα οποία εμφανίζονται στιγμιότυπα με τύπους(complex types), τους τύπους με τους οποίους έχει αντιστοιχηθεί η κλάσης C. Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1, το σύνολο $\mathbf{X_c}$ για την κλάση Persons_Type είναι το { /Persons }, ενώ για την κλάση Person_Type είναι το { /Persons/Person , /Persons/Employee}.
- Για κάθε ιδιότητα Pr ϵ (DTP U OP) είτε αντικειμένων είτε τιμών δεδομένων που αντιστοιχίζεται, δημιουργείται ένα σύνολο από μονοπάτια $\mathbf{X_{Pr}}$ (**Property Xpaths Set).** Το σύνολο των μονοπατιών $\mathbf{X_{Pr}}$ όπως έχει αναφερθεί περιέχει όλα τα πιθανά μονοπάτια στα XML δεδομένα στα οποία εμφανίζονται στιγμιότυπα των στοιχείων ή χαρακτηριστικών με τα οποία έχει αντιστοιχηθεί η ιδιότητας Pr. Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1, το σύνολο $\mathbf{X_{Pr}}$ για την Ιδιότητα Age είναι το { /Persons/Person/Age , /Persons/Employee/Age }.
    - Από το σύνολο $\mathbf{X_{Pr}}$ προκύπτουν δυο σύνολα, ένα σύνολο για τις αντιστοιχήσεις των πεδίων ορισμού (Property Domains) και ένα σύνολο για τις αντιστοιχήσεις των πεδίων τιμών (Property Ranges). Τα σύνολα αυτά συμβολίζονται με $\mathbf{X_{PrD}}$ **(Property Domains Xpaths Set)** και $\mathbf{X_{PrR}}$ **(Property Range Xpaths Set)** αντίστοιχα. Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1, το σύνολο $\mathbf{X_{PrD}}$ για την Ιδιότητα τιμών Age είναι το { /Persons/Person, /Persons/Employee }, ενώ το είναι το $\mathbf{X_{PrR}}$ { /Persons/Person/Age , /Persons/Employee/Age }. Επίσης σύνολο $\mathbf{X_{PrD}}$ για την ιδιότητα αντικειμένων Employee είναι το { /Persons }, ενώ το είναι το $\mathbf{X_{PrR}}$ { /Persons/Employee }.

Τα παραπάνω σύνολα μονοπατιών είτε δημιουργούνται αυτόματα με την διαδικασία που αναλύεται στο Κεφάλαιο 6, για την περίπτωση την οποία η οντολογία έχει προκύψει με χρήση της μεθοδολογίας XS2OWL, είτε ορίζονται από κάποιον εξειδικευμένο χρήστη ο οποίος ορίζει τις αντιστοιχήσεις μεταξύ της οντολογίας και του XML σχήματος.

Τα παραπάνω αναπαρίστανται στην Εικόνα 5.6 :

$$\forall \text{Mapped Class C} \Rightarrow \text{Class Xpaths Set} = X_c = \{\text{xpath}_1, \text{xpath}_2 ... \text{xpath}_n\}$$

$$\forall \text{Mapped Property Pr} \Rightarrow \text{Property Xpaths Set} = X_{Pr} = \{\text{xpath}_1, \text{xpath}_2 ... \text{xpath}_n\}$$

$$\forall X_{Pr} \Rightarrow \begin{cases} \text{Property Ranges Xpaths Set} & X_{PrR} \\ \text{Property Domains Xpaths Set} & X_{PrD} \end{cases}$$

Εικόνα 5.6 Αναπαράσταση Αντιστοιχήσεων

5.5.1 Παράδειγμα Αντιστοιχήσεων μεταξύ οντολογίας και μονοπατιών

Οι παρακάτω αντιστοιχήσεις ορίζονται για την οντολογίας της ενότητας 5.2.1 και το XML σχήματος της ενότητας 5.2.2. Οι αντιστοιχίες που δημιουργούνται μεταξύ κλάσεων, ιδιοτήτων και πιθανών μονοπατιών στο xml έγγραφο είναι :

- **Κλάσεις**
  - $\mathbf{X_{Persons\_Type}}$ = { /Persons }
  - $\mathbf{X_{Person\_Type}}$ = { /Persons/Person , /Persons/Employee }
  - $\mathbf{X_{Employee\_Type}}$ = { /Persons/Employee }
  - $\mathbf{X_{Address\_Type}}$ = { /Persons/Person/Address , /Persons/Employee/Address }

- **Ιδιότητες Αντικειμένων**
  - $\mathbf{X_{Persons}}$ **=** { /Persons }
  - $\mathbf{X_{Person}}$ **=** { /Persons/Person }
  - $\mathbf{X_{Employee}}$ **=** { /Persons/Employee }
  - $\mathbf{X_{Address}}$ **=** { /Persons/Person/Address , /Persons/Employee/Address }

- **Ιδιότητες Τύπων Δεδομένων**
  - $\mathbf{X_{Telephone}}$ **=** { /Persons/Person/Telephone , /Persons/Employee/Telephone }
  - $\mathbf{X_{Age}}$ **=** { /Persons/Person/Age , /Persons/Employee/Age }
  - $\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

- $\mathbf{X_{LastName}}$ = { /Persons/Person/LastName , /Persons/Employee/LastName }
- $\mathbf{X_{NickName}}$ = { /Persons/Person/NickName, /Persons/Employee/NickName }
- $\mathbf{X_{Salary}}$ = { /Persons/Employee/Salary }
- $\mathbf{X_{City}}$ = { /Persons/Person/Address/@city , /Persons/Employee/Address/@city }
- $\mathbf{X_{Street}}$ = { /Persons/Person/Address/Street , /Persons/Employee/Address/Street
  /Persons/Person/Address/Road , /Persons/Employee/Address/Road }
- $\mathbf{X_{Road}}$ = { /Persons/Person/Address/Road , /Persons/Employee/Address/Road }
- $\mathbf{X_{Number}}$={/Persons/Person/Address/Number, /Persons/Employee/Address/Number}

5.6 Τελεστές Συνόλων Μονοπατιών

Σε αυτή την ενότητα ορίζονται μια σειρά από τελεστές οι οποίοι μπορούν να εφαρμοστούν σε σύνολα μονοπατιών. Οι τελεστές χρησιμοποιούνται για την δήλωση των υποθέσεων των αντιστοιχήσεων καθώς και κατά την διατύπωση της διαδικασίας μετάφρασης των ερωτήσεων στο επόμενο κεφάλαιο.

Ορισμός 5.1 Ορίζεται ο δυαδικός **τελεστής Ένωσης** – **U** (**Union**), για σύνολα μονοπατιών. Ο συγκεκριμένος τελεστής εμφανίζει ειδική συμπεριφορά σε σύγκριση με την ένωσης που ορίζεται από την θεωρία συνόλων για τα μονοπάτια που περιέχουν τον χαρακτήρα * (Wildcard). Το αποτέλεσμα της ένωσης μονοπατιών που περιέχουν τον χαρακτήρα * είναι το "πιο γενικό" μονοπάτι. (Βλέπε Παράδειγμα 5.1 b),c),d)).

Παράδειγμα 5.1

a) Έστω τα σύνολα X={ /a , /a/b , /c/d } και Y={ /a/c , /a } . Η ένωση τους ορίζετε ως X U Y = { /a , /a/b , /c/d , /a/c }.

b) Έστω τα σύνολα X={ /* , /a , /c } και Y={ /κ , /a } . Η ένωση τους ορίζετε ως X U Y = { /* }

c) Έστω τα σύνολα X={ a/* , /a /b } και Y={ a/c , /d } . Η ένωση τους ορίζετε ως X U Y = { a/* , /d }

d) Έστω τα σύνολα X={ a/@* , /a /b } και Y={ a/@c , /d } . Η ένωση τους ορίζετε ως X U Y = { a/@* , /a /b , /d }

Ορισμός 5.2 Ορίζεται ο δυαδικός **τελεστής Τομής –** ∩ (**Intersection**) για σύνολα μονοπατιών. Ο συγκεκριμένος τελεστής διαφοροποιείται από την τομή που ορίζεται από την θεωρία συνόλων για τα μονοπάτια που περιέχουν τον χαρακτήρα * (Wildcard). Το αποτέλεσμα της τομής μονοπατιών που περιέχουν τον χαρακτήρα * είναι το "πιο συγκεκριμένο" μονοπάτι (Βλέπε Παράδειγμα 5.2 b),c),d),e)).

Παράδειγμα 5.2

a) Έστω τα σύνολα μονοπατιών X={ /a , /a/b , /c/d } και Y={ /a/c , /a } . Η τομή τους ορίζετε ως X ∩ Y = { /a }.

b) Έστω τα σύνολα μονοπατιών X={ /* , /a , /c } και Y={ /κ , /a } . Η τομή του ορίζετε ως X ∩ Y = { /κ , /a } .

c) Έστω τα σύνολα μονοπατιών X={ a/* } και Y={ a/b , a/d } . Η τομή του ορίζετε ως X ∩ Y = { a/b , a/d } .

d) Έστω τα σύνολα μονοπατιών X={ a/* , a/b } και Y={ a/b , a/d } . Η τομή του ορίζετε ως X ∩ Y = { a/b , a/d } .

e) Έστω τα σύνολα μονοπατιών X={ a/@* } και Y={ a/@b , a/@d } . Η τομή του ορίζετε ως X ∩ Y = { a/@b , a/@d } .

Ορισμός 5.3 Ορίζεται ο δυαδικός **τελεστής Αριστερού Παιδιού (Right Child)** ⊕ για σύνολα μονοπατιών. Όταν εφαρμοστεί σε δυο σύνολα επιστέφει τα στοιχεία του δεξιού συνόλου για τα οποία οι πατέρες τους περιέχονται στο αριστερό σύνολο. $\mathbf{X \oplus Y = \{\ xpaths : xpaths\ \epsilon\ Y\ and\ Y^P\ \epsilon\ X\ \}}$.

Για την περίπτωση ύπαρξης χαρακτήρα * (Wildcard) σε στοιχεία x του συνόλου X, το τελικό σύνολο(για τα στοιχεία x), θα περιέχει τα στοιχεία y του συνόλου Y, για τα οποία ισχύει $\mathbf{x^P = (y^P)^P}$, επίσης θα περιέχει όλα τα άλλα στοιχεία που προέρχονται από την εφαρμογή του τελεστή ⊕ στο σύνολο των στοιχείων του X εκτός των στοιχείων x. (Βλέπε Παράδειγμα 5.3 b)).

Παράδειγμα 5.3

a) Έστω τα σύνολα μονοπατιών X={ /a/b , /c/d } και Y={ /a/b/x , /c/d/p , /c/d/p , /b/s } .
X⊕Y = {/a/b/x , /c/d/p , /c/d/p }

b) Έστω τα σύνολα μονοπατιών X={ /a/b , /c/* } και Y={ /a/b/x , /c/y/p , /c/z/p , /b/s } .
X⊕Y = {/a/b/x , /c/y/p , /c/z/p }

Ορισμός 5.4 Ορίζεται ο μοναδιαίος **τελεστής Πατέρας (Parent)** ^{P} ο οποίος όταν εφαρμοστεί σε ένα σύνολο από μονοπάτια επιστέφει τους πατέρες των μονοπατιών(δηλαδή τα μονοπάτια που αντιστοιχούν στους κόμβους πατέρες(parents nodes)) του συνόλου αυτού. Κοινώς επιστρέφει τα αρχικά μονοπάτια με το βάθος τους μειωμένο κατά ένα .

Σημείωση : *Στην περίπτωση που ο τελεστής εφαρμοστεί στην διαδρομή της ρίζας (root xpath) δεν επιφέρει καμία αλλαγή.*

Παράδειγμα 5.4

Για το αρχικό σύνολο μονοπατιών X={ /a , /a/b , /c/d , / e/f/g} ορίζεται ως X^{P}={ /a , /a , /c , /e/f }

Ορισμός 5.5 Ορίζεται ο μοναδιαίος **τελεστής** ^{LN} **(leaf node)** ο οποίος όταν εφαρμοστεί σε ένα σύνολο από μονοπάτια επιστέφει ένα σύνολο (από ονόματα κόμβων) που περιέχει τα διαφορετικά όνομα των τελευταίων κόμβων(κόμβος φίλο-leaf node) για κάθε μονοπάτι του συνόλου .

Παράδειγμα 5.5

Για το αρχικό σύνολο μονοπατιών X={ /a/b , /c/d , / e/f/g} ορίζετε ως X^{LN}={ b , d , g } .

Για το αρχικό σύνολο μονοπατιών X={ /a/b , /c/b , / e/f/g} ορίζετε ως X^{LN}={ b , g } .

Ορισμός 5.6 Ορίζεται ο δυαδικός **τελεστής προσάρτησης (append) /** για σύνολα μονοπατιών και σύνολα από ονόματα κόμβων. Ως δεξιό όρισμα δέχεται ένα σύνολο μονοπατιών και ως αριστερό ένα σύνολο από ονόματα κόμβων. Τα στοιχεία του παραγόμενου συνόλου έχουν ως πατέρες (^{P}) τα στοιχεία του αρχικού συνόλου μονοπατιών και ως (^{LN}) τα στοιχεία του συνόλου με τα ονόματα κόμβων. Το παραγόμενο σύνολο δημιουργείται από την προσάρτηση κάθε στοιχείου του δεξιό συνόλου σε όλα τα στοιχεία του αριστερού.

$X / nodes = \{ xpaths : xpaths^{P} \in X \text{ and } xpaths^{LN} \in nodes \}$.

Σημείωση: *Κατά την χρήση του τελεστή* προσάρτησης δεν εμφανίζονται μονοπάτια που περιχέουν των χαρακτήρα * (wildcard).

Παράδειγμα 5.6

Για το αρχικό σύνολο μονοπατιών X={ /a/b , /c/d , / e/f/g} και nodes= { o , w }, τοτε X/nodes = { /a/b/o , /a/b/w , /c/d/o , /c/d/w, /e/f/g/o , /e/f/g/w }

5.7 Υποθέσεις Αντιστοιχήσεων

Αν και η διαδικασία μετάφρασης των ερωτήσεων, είναι ανεξάρτητη από τον τρόπο ορισμού, δημιουργίας και αναπαράστασης των αντιστοιχήσεων, έχουν γίνει οι παρακάτω υποθέσεις :

1. **Υπόθεση πλήρους ορισμού των αντιστοιχήσεων :** Για κάθε κλάση και ιδιότητα που περιέχει η ερώτηση θα πρέπει να έχουν οριστεί οι αντιστοιχήσεις .
2. **Υπόθεση συνέπειας των αντιστοιχήσεων :** Με βάση τις αντιστοιχίες του Πίνακα 5.1, για κάθε ιδιότητα Pr της οντολογίας :
    a. Η κλάση πεδίου ορισμού (domain class) της ιδιότητας Pr έχει αντιστοιχηθεί με σύνθετους τύπους που περιέχουν το/α στοιχείο/α και/ή το/α γνώρισμα/τα με τα οποίο/α έχει αντιστοιχηθεί η ιδιότητα Pr.
    b. Αν η Pr είναι ιδιότητα αντικειμένων (Object Property), η κλάση πεδίο τιμών(range class) της Pr, έχει αντιστοιχηθεί με σύνθετους τύπους, στους οποίους περιέχεται/ωνται ο/οι σύνθετος/οι τύπος/οι του/ων σύνθετου/ων στοιχείου/ων που έχει αντιστοιχηθεί η Pr.

Με βάση την Υπόθεση 2 "Υπόθεση συνέπειας των αντιστοιχήσεων" για τα σύνολα μονοπατιών προκύπτουν :

Συμπέρασμα 5.2

$$\forall class => X_c \subseteq X_{pr}^P \text{ ή } X_c \supseteq X_{pr}^P \text{ , όπου pr ειναι ιδιοτητα με domain την class}$$

Το σύνολο μονοπατιών $\mathbf{X_c}$ μιας κλάσης C, είναι υπό-σύνολο ή υπέρ-σύνολο του συνόλου που προκύπτει από την εφαρμογή του τελεστή $^{\mathbf{P}}$ (parent) στο σύνολο της ιδιότητας pr ($\mathbf{X_{Pr}}$) η οποία έχει πεδίο ορισμού (domain) την κλάση C.

Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1 το σύνολο $\mathbf{X_c}$ για την κλάση Person_Type είναι το { /Persons/Person , /Persons/Employee}. Επίσης το σύνολο $\mathbf{X_{Pr}}$ για την Ιδιότητα Age είναι το { /Persons/Person/Age , /Persons/Employee/Age}, εφαρμόζοντας τον τελεστή $^{\mathbf{P}}$ (parent) $\mathbf{X_{Pr}}^{\mathbf{P}}$ ={ /Persons/Person , /Persons/Employee}, σε αυτή την περίπτωση παρατηρείται ότι τα δυο σύνολα $\mathbf{X_{Pr}}^{\mathbf{P}}$ και $\mathbf{X_c}$ ταυτίζονται. Όμως στην γενική περίπτωση(στην οποία η οντολογία δεν έχει προκύψει με χρήση της μεθοδολογίας XS2OWL, όπως το παράδειγμα της ενότητας 5.5.1) μπορούν να εμφανίζονται σχέσεις υπό η υπέρ συνόλων .

Συμπέρασμα 5.3

$$\forall class => X_c \subseteq X_{pr} \text{ ή } X_c \supseteq X_{pr} \text{ , όπου pr ειναι ιδιοτητα με range την class}$$

Το σύνολο μονοπατιών $\mathbf{X_c}$ μιας κλάσης C, είναι υπό-σύνολο ή υπέρ-σύνολο της ιδιότητας pr ($\mathbf{X_{Pr}}$) η οποία έχει πεδίο τιμών(range) την κλάση C.

Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1 το σύνολο $\mathbf{X_c}$ για την κλάση Person_Type είναι το { /Persons/Person , /Persons/Employee}. Επίσης το σύνολο $\mathbf{X_{Pr}}$ για την Ιδιότητα Person είναι το { /Persons/Person }, επομένως σε αυτή την περίπτωση παρατηρείται ότι για τα δυο σύνολα ισχύει : $\mathbf{X_c \supseteq X_{Pr}}$.

Συμπέρασμα 5.4

$$\forall \text{Property pr} => X_{prR} = X_{pr} \quad \text{και} \quad X_{prD} = X_{pr}^P = X_{prR}^P$$

Το σύνολο μονοπατιών $\mathbf{X_{pr}}$ μιας ιδιότητας pr ισούται με το σύνολο μονοπατιών που αντιστοιχεί στο πεδίου τιμών της ($\mathbf{X_{prR}}$), όπως επίσης και με το σύνολο που αντιστοιχεί στο πεδίο ορισμού της ($\mathbf{X_{prD}}$) εφόσον εφαρμοστεί ο τελεστή $^{\mathbf{P}}$ (parent).

Όπως φαίνεται και στο παράδειγμα της ενότητας 5.5.1, για την Ιδιότητα Age το σύνολο που αντιστοιχεί στο πεδίο τιμών της ($\mathbf{X_{PrR}}$) είναι το { /Persons/Person/Age , /Persons/Employee/Age }, επίσης τα πεδία ορισμού της ιδιότητας είναι η κλάση Person, επομένως { /Persons/Person, /Persons/Employee } . Όπως παρατηρείται το σύνολο του πεδίου ορισμού έχει προκύψει από το πεδίο τιμών ($\mathbf{X_{prR}}$) με την εφαρμογή του τελεστή $^{\mathbf{P}}$ (parent). Εφόσον :

{ /Persons/Person/Age , /Persons/Employee/Age }^{P} = { /Persons/Person, /Persons/Employee }

5.8 Περίληψη

Στο κεφάλαιο αυτό, παρουσιάστηκε ένας αφηρημένος τρόπος αναπαράστασης των αντιστοιχήσεων, οριστήκαν οι αντιστοιχίες στο επίπεδο των δυο γλωσσών. Επίσης ορίστηκε ένας αριθμός από τελεστές οι οποίοι εφαρμόζονται σε σύνολα μονοπατιών. Τέλος οριστήκαν οι υποθέσεις που γίνονται για τις αντιστοιχήσεις.

Ο αφηρημένος τρόπος αναπαράστασης των αντιστοιχήσεων καθώς και οι τελεστές των μονοπατιών χρησιμοποιούνται στην συνέχεια του κειμένου κατά την περιγραφή των αλγορίθμων και των διαδικασιών μετάφρασης.

Στο επόμενο κεφάλαιο (Κεφάλαιο 6) γίνεται η περιγραφή της μορφής με την οποία αποθηκεύονται οι αντιστοιχήσεις, γίνεται μια εισαγωγή στο σύστημα XS2OWL και τέλος γίνεται η λεπτομερής παρουσίαση της διαδικασίας ανακάλυψης(discovery) και του αυτόματου τρόπου παραγωγής των αντιστοιχήσεων.

6 Ανακάλυψη Και Αποθήκευση Αντιστοιχήσεων

6.1 Εισαγωγή

Όπως έχει αναφερθεί, στην παρούσα προσέγγιση οι αντιστοιχήσεις μεταξύ οντολογιών και XML σχήματος, μπορούν να οριστούν είτε χειροκίνητα από έναν εξειδικευμένο χρήστη, είτε να παραχθούν αυτόματα στην περίπτωση την οποία η οντολογία έχει παραχθεί με χρήση της μεθοδολογίας XS2OWL[17].

Οι αντιστοιχήσεις ορίζονται μεταξύ στοιχείων της οντολογίας (κλάσεις και ιδιότητες) και μονοπατιών στα XML δεδομένα. Με αυτό τον τρόπο επιτυγχάνεται η "άμεση" σύνδεση των στοιχείων της οντολογίας με τα XML δεδομένα. Οι αντιστοιχήσεις αυτές θα χρησιμοποιηθούν κατά τις διαδικασίες της μεταφράσεις (Κεφάλαια 9-12) για την ανάπτυξη των XQuery εκφράσεων, καθώς τα στοιχεία της οντολογίας που περιέχονται στις SPARQL ερωτήσεις θα αντικατασταθούν με μονοπάτια στις XQuery εκφράσεις.

Στο παρόν κεφάλαιο γίνεται η περιγραφή της μορφής με την οποία αποθηκεύονται οι αντιστοιχήσεις (υπό-ενότητα 6.2). Επίσης γίνεται μια εισαγωγή στο σύστημα XS2OWL(υπό-ενότητα 6.3) και τέλος γίνεται η λεπτομερής παρουσίαση της διαδικασίας ανακάλυψης(discovery) και του αυτόματου τρόπου παραγωγής των αντιστοιχήσεων. Η διαδικασία δέχεται ως είσοδο το XML σχήμα και την OWL οντολογία που έχει παραχθεί από το σύστημα XS2OWL και παράγει ως έξοδο ένα XML έγγραφο που περιέχει τις αντιστοιχήσεις μεταξύ οντολογίας και XML σχήματος. Οι αντιστοιχήσεις αυτές όπως γίνεται αντιληπτό είναι απαραίτητες κατά την διαδικασία της μετάφρασης.

Με την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων επιτυγχάνεται μια πλήρως αυτοματοποιημένη διαδικασία, χωρίς να είναι απαραίτητη η παρέμβαση ανθρώπινου παράγοντα. Επίσης επιτυγχάνεται η πλήρης αντιστοίχηση όλων των στοιχείων τόσο της οντολογίας όσο και του XML σχήματος, με αντιστοιχήσεις απολύτως σημασιολογία ορθές, χωρίς ύπαρξη αβεβαιότητα (Uncertainty) και πιθανότητας σφάλματος κατά την δημιουργία τους, προβλήματα που εμφανίζονται στην περίπτωση "χειροκίνητου"(manual) ορισμού των αντιστοιχήσεων από εξειδικευμένο χρήστη.

6.2 Αποθήκευση Αντιστοιχήσεων

Οι αντιστοιχήσεις αποθηκεύονται σε μορφή XML ακολουθώντας το XML σχήμα που ορίζεται στην συνέχεια(υπό-ενότητα 6.2.1), το XML αυτό έγγραφο περιέχει τις πληροφορίες των αντιστοιχήσεων που είναι απαραίτητες για την μετάφραση των ερωτήσεων. Στην περίπτωση χρήσης της μεθοδολογίας XS2OWL, το XML έγγραφο με τις αντιστοιχήσεις παράγεται αυτόματα, όπως αναλύεται σε παρακάτω ενότητα(υπό-ενότητα 6.4).

Όπως αναλύθηκε στις ενότητες 5.4 και 5.5 :

> "Η αντιστοίχηση μιας Οντολογίας O σε κενά XML σχήμα XMLS είναι ένα σύνολο από αντιστοιχήσεις (οι οποίες είναι σύμφωνες με τις αντιστοιχίες σε επίπεδο γλωσσών) και αναπαριστούν τις σημασιολογικές συσχετίσεις μεταξύ των στοιχείων των O και XMLS."

$$\forall \text{Mapped Class C} \Rightarrow \text{Class Xpaths Set} = X_c = \{\text{xpath}_1, \text{xpath}_2 \ldots \text{xpath}_n\}$$

$$\forall \text{Mapped Property Pr} \Rightarrow \text{Property Xpaths Set} = X_{Pr} = \{\text{xpath}_1, \text{xpath}_2 \ldots \text{xpath}_n\}$$

$$\forall X_{Pr} \Rightarrow \begin{cases} \text{Property Ranges Xpaths Set } X_{PrR} \\ \text{Property Domains Xpaths Set } X_{PrD} \end{cases}$$

6.2.1 XML Σχήμα Αποθήκευσης Αντιστοιχήσεων

Σε αυτή την υπό-ενότητα θα γίνει η περιγραφή του XML σχήματος (Εικόνα 6.1 και 6.2) που ακολουθούν τα XML έγγραφα τα οποία περιέχουν τις αντιστοιχήσεις. Το στοιχείο **MappingFile** είναι το ριζικό στοιχείο το οποίο περιέχει τα στοιχεία **MappingInforamtion**, **Class**, **ObjectPropery** και **DataTypeProperty**.

- Το στοιχείο **MappingInforamtion** περιέχει πληροφορίες για την οντολογία και το Xml σχήμα για το οποίο ορίζονται οι αντιστοιχήσεις. Πιο συγκεκριμένα περιέχει δυο στοιχεία, το **OntologyURI** και το **XMLSchemaURI** τα οποία περιέχουν τις URI διευθύνσεις που βρίσκεται η οντολογία και το XML σχήμα αντίστοιχα. Το στοιχείο MappingInforamtion εμφανίζεται υποχρεωτικά.

- Το στοιχείο **Class** περιέχει πληροφορίες για τις αντιστοιχήσεις κλάσεων. Για κάθε κλάση για την οποία ορίζονται αντιστοιχήσεις, δημιουργείται και ένα στοιχείο Class (maxOccurs="unbounded"). Πιο συγκεκριμένα το στοιχείο Class περιέχει ένα χαρακτηριστικό **IRI** το οποίο περιέχει το IRI της κλάσης, καθώς και μια ακολουθία (sequence) από στοιχεία **Xpath**. Κάθε στοιχείο Xpath περιέχει ένα μονοπάτι Xpath που αντιστοιχεί στην κλάση. Το στοιχείο Class εμφανίζεται προαιρετικά (minOccurs="0").

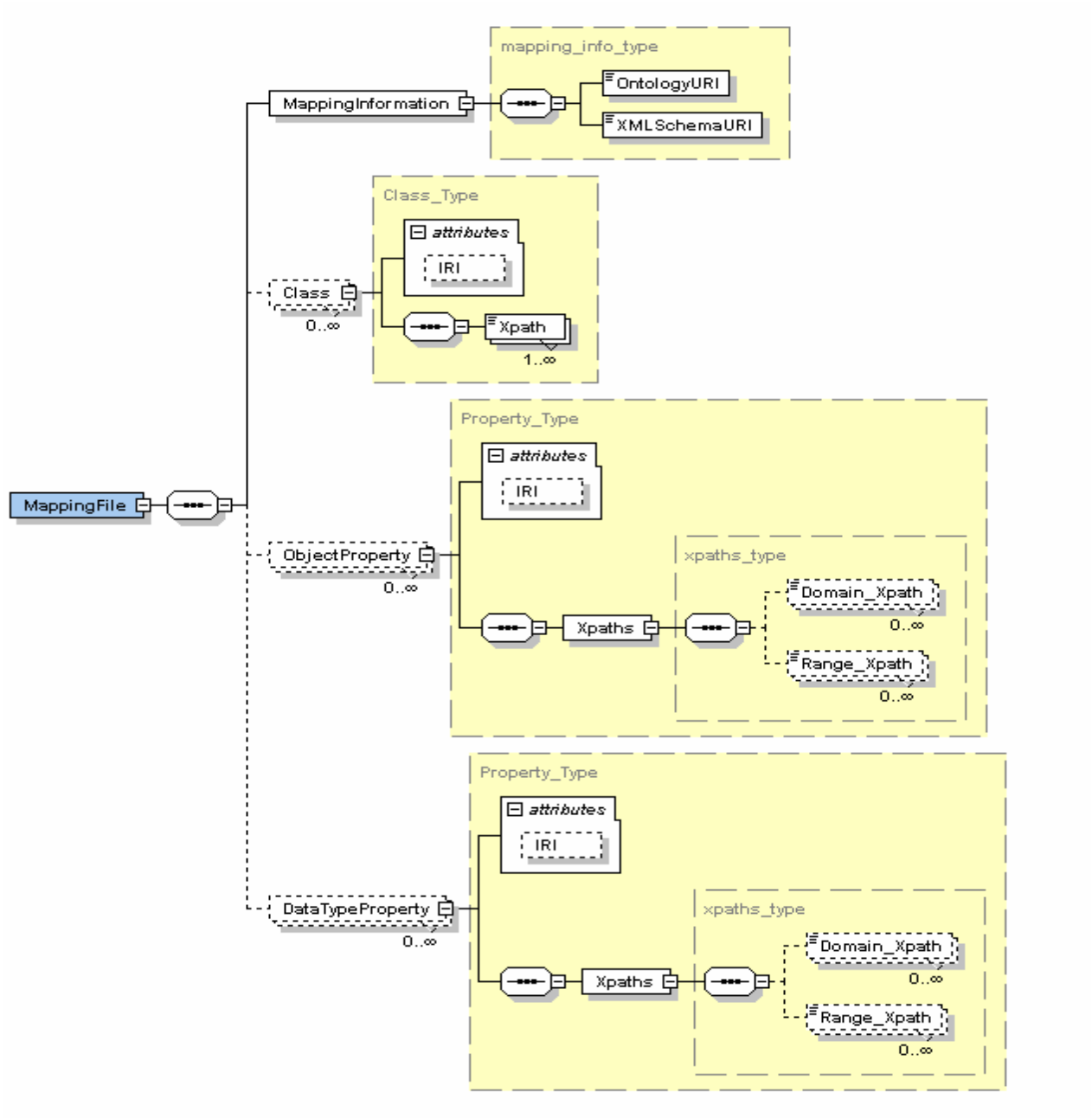
- Το στοιχείο **ObjectProperty** περιέχει πληροφορίες για τις αντιστοιχήσεις των ιδιοτήτων αντικειμένων. Για κάθε ιδιοτητα αντικειμένων για την οποία ορίζονται αντιστοιχήσεις, δημιουργείται και ένα στοιχείο **ObjectProperty** (maxOccurs="unbounded"). Πιο συγκεκριμένα το στοιχείο ObjectProperty περιέχει ένα χαρακτηριστικό **IRI** το οποίο περιέχει το IRI της ιδιότητας αντικειμένων, καθώς και μια ακολουθία (sequence) από στοιχεία **Xpaths**. Κάθε στοιχείο Xpaths περιέχει μια ακολουθία από στοιχεία **Domain_Xpath** τα οποία περιέχουν μονοπάτια Xpath που αντιστοιχούν στα πεδία ορισμού(domains) της ιδιότητας και μια ακολουθία από στοιχεία **Range_Xpath** τα οποία περιέχουν μονοπάτια Xpath που αντιστοιχούν στα πεδία τιμών(ranges) της ιδιότητας. Τα στοιχεία Range_Xpath και Domain_Xpath εμφανίζεται προαιρετικά (minOccurs="0", maxOccurs="unbounded").

- Το στοιχείο **DataTypeProperty** περιέχει πληροφορίες για τις αντιστοιχήσεις των ιδιοτήτων τιμών. Για κάθε ιδιότητα τιμών για την οποία ορίζονται αντιστοιχήσεις, δημιουργείται και ένα στοιχείο DataTypeProperty (maxOccurs="unbounded"). Πιο συγκεκριμένα το στοιχείο DataTypeProperty περιέχει ένα χαρακτηριστικό **IRI** το οποίο περιέχει το IRI της ιδιότητας τιμών, καθώς και μια ακολουθία (sequence) από στοιχεία **Xpaths**. Κάθε στοιχείο Xpaths περιέχει μια ακολουθία από στοιχεία **Domain_Xpath** τα οποία περιέχουν μονοπάτια Xpath που αντιστοιχούν στα πεδία ορισμού(domains) της ιδιότητας και μια ακολουθία από στοιχεία **Range_Xpath** τα οποία περιέχουν μονοπάτια Xpath που αντιστοιχούν στα πεδία τι-

μών(ranges) της ιδιότητας. Τα στοιχεία Range_Xpath και Domain_Xpath εμφανίζεται προαιρετικά (minOccurs="0", maxOccurs="unbounded").

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://www.music.tuc.gr/SPARQL2XQuery/mappingfile" targetNamespace="http://www.music.tuc.gr/SPARQL2XQuery/mappingfile" elementFormDefault="qualified">


<xs:complexType name="xpaths_type">
        <xs:sequence>
                <xs:element name="Domain_Xpath" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="Range_Xpath" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
</xs:complexType>


<xs:complexType name="Property_Type">
        <xs:sequence>
                <xs:element name="Xpaths" type="xpaths_type"/>
                </xs:sequence>
                <xs:attribute name="IRI" type="xs:anyURI"/>
</xs:complexType>


<xs:complexType name="Class_Type">
        <xs:sequence>
                <xs:element name="Xpath" type="xs:string" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="IRI" type="xs:anyURI"/>
</xs:complexType>


<xs:complexType name="mapping_info_type">
        <xs:sequence>
                <xs:element name="OntologyURI" type="xs:anyURI"/>
                <xs:element name="XMLSchemaURI" type="xs:anyURI"/>
                </xs:sequence>
</xs:complexType>



<xs:element name="MappingFile">
        <xs:complexType>
                <xs:sequence>
                        <xs:element name="MappingInformation" type="mapping_info_type"/>
                        <xs:element name="Class" type="Class_Type" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="ObjectProperty" type="Property_Type" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="DataTypeProperty" type="Property_Type" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
</xs:element>

</xs:schema>
```

Εικόνα 6.1 XML Σχήμα Αποθήκευσης Αντιστοιχήσεων



Εικόνα 6.2 XML Σχήμα Αποθήκευσης Αντιστοιχήσεων

6.2.2 Παράδειγμα - XML Έγγραφο Αντιστοιχήσεων

Στην επόμενη εικόνα (Εικόνα 6.3) εμφανίζεται μέρος του XML εγγράφου που περιέχει τις αντιστοιχήσεις της υπό-ενότητας 5.5.1 που προκύπτουν για την οντολογία και το XML σχήμα της υπό-ενότητας 5.2. Στην εικόνα εμφανίζονται οι πληροφορίες για την οντολογία και το XML σχήμα για το οποίο ορίζονται οι αντιστοιχήσεις. Οι αντιστοιχήσεις για τις κλάσεις Employee_Type , Persons_Type και Person_Type. Οι

αντιστοιχήσεις για τις ιδιότητες δεδομένων FirstName και Salary. Τέλος οι αντιστοιχήσεις για τις ιδιότητες αντικειμένων Employee και Person. Στην συνέχεια του κεφαλαίου περιγράφεται αναλυτικά με παραδείγματα ο τρόπος παραγωγής του εγγράφου αντιστοιχήσεων.

```
<map:MappingInformation>
        <map:OntologyURI>http://www.music.tuc.gr/ontology.owl#</map:OntologyURI>
        <map:XMLSchemaURI> http://www.music.tuc.gr/XMLSchema.xsd</map:XMLSchemaURI>
</map:MappingInformation>

<map:Class IRI="Employee_Type">
        <map:Xpath>/Persons/Employee</map:Xpath>
</map:Class>

<map:Class IRI ="Persons_Type">
        <map:Xpath>/Persons </map:Xpath>
</map:Class>

<map:Class IRI ="Person_Type">
        <map:Xpath>/Persons/Person</map:Xpath>
        <map:Xpath>/Persons/Employee</map:Xpath>
</map:Class>

<map:ObjectProperty IRI ="Employee ">
        <map:Xpaths>
                <map:Domain_Xpath>/Persons</map:Domain_Xpath>
                <map:Range_Xpath>/Persons/Employee</map:Range_Xpath>
        </map:Xpaths>
</map:ObjectProperty>

<map:ObjectProperty IRI ="Person">
        <map:Xpaths>
                <map:Domain_Xpath>/Persons </map:Domain_Xpath>
                <map:Range_Xpath>/Persons/Person</map:Range_Xpath>
        </map:Xpaths>
</map:ObjectProperty>

…..

<map:DataTypeProperty IRI ="FirstName">
        <map:Xpaths>
                <map:Domain_Xpath>/Persons/Person</map:Domain_Xpath>
                <map:Domain_Xpath>/Persons/Employee</map:Domain_Xpath>
                <map:Range_Xpath>/Persons/Person/FirstName </map:Range_Xpath>
                <map:Range_Xpath>/Persons/Employee/FirstName </map:Range_Xpath>
        </map:Xpaths>
</map:DataTypeProperty>


…..
…..
```

```
…..
…..

<map:DataTypeProperty IRI ="Salary">
        <map:Xpaths>
                <map:Domain_Xpath>/ Persons/Employee</map:Domain_Xpath>
                <map:Range_Xpath>/Persons/Employee/Salary</map:Range_Xpath>
        </map:Xpaths>
</map:DataTypeProperty>
```

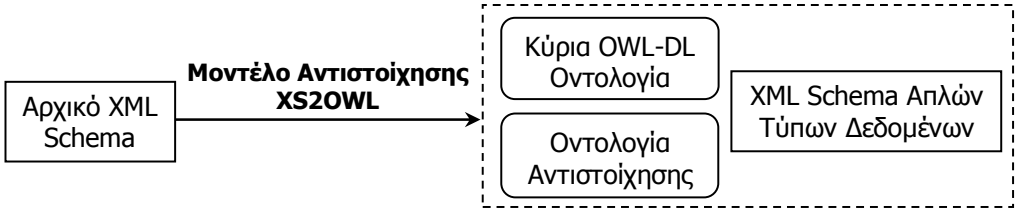
Εικόνα 6.3 Τμήμα XML Εγγράφου Αντιστοιχήσεων

6.3 Εισαγωγή στο Σύστημα XS2OWL

Στην ενότητα αυτή γίνεται μια εισαγωγή στο σύστημα XS2OWL[15][16][17][18], το οποίο παρέχει διαλειτουργικότητα μεταξύ OWL και XML. Το σύστημα XS2OWL θεμελιώνει τη διαλειτουργικότητα σε ένα σύνολο OWL οντολογιών που κωδικοποιούν σε OWL σύνταξη τη σημαντική των XML σχημάτων. Οι οντολογίες αυτές παράγονται αυτόματα από το σύστημα XS2OWL, το οποίο υλοποιεί το μοντέλο αντιστοίχησης(Πίνακας 6.1) XS2OWL και απεικονίζεται στην Εικόνα 6.4. Το σύστημα XS2OWL έχει υλοποιηθεί με τη χρήση της γλώσσας XML Stylesheet Transformation Language (XSLT) [107] .

Όπως φαίνεται στην Εικόνα 6.4, το σύστημα XS2OWL δέχεται ως είσοδο (input) ένα XML σχήμα και παράγει ως έξοδο (output) μέσω της εφαρμογής του μοντέλου αντιστοίχησης XS2OWL:

- Μια OWL-DL **Κύρια Οντολογία**, όπου αποτυπώνεται άμεσα τη σημαντική των δομών του XML σχήματος στις αντίστοιχες OWL-DL δομές. Αν το αρχικό XML σχήμα αναπαριστά κάποιο πρότυπο (ή τμήμα προτύπου), η οντολογία αυτή είναι ανώτερη οντολογία ή τμήμα ανώτερης οντολογίας, που αναπαριστά γενικού σκοπού δομές και η σημαντική της μπορεί να επεκταθεί στη συνέχεια από οντολογίες περιοχής και εφαρμογών.
- Ένα **XML Schema Απλών Τύπων Δεδομένων**, που περιέχει τους απλούς XML Schema τύπους δεδομένων που ορίζονται στο αρχικό XML σχήμα και χρησιμοποιούνται στην κύρια οντολογία.
- Μια OWL-DL **Οντολογία Αντιστοίχησης**, που αντιστοιχίζει τα ονόματα των δομών του αρχικού XML σχήματος με τις ταυτότητες των αντίστοιχων δομών της OWL-DL κύριας οντολογίας και αποτυπώνει συστηματικά τη σημαντική της XML Schema που δε μπορεί να αναπαρασταθεί άμεσα σε OWL.



Εικόνα 6.4 Το Σύστημα XS2OWL

XML Schema Δομή	OWL-DL Αναπαράσταση		
	Κύρια Οντολογία	**Οντολογία Αντιστοίχησης**	**XML Schema Απλών Τύπων Δεδομένων**
Σύνθετος Τύπος (xs:complexType)	Κλάση (owl:Class)	Άτομο ComplexTypeInfoType	
Απλός Τύπος (xs:simpleType)	Δήλωση Τύπου Δεδομένων (rdfs:Datatype)		Απλός Τύπος (xs:simpleType)
Στοιχείο (xs:element)	Ιδιότητα – Αντικειμένων ή Τύπου Δεδομένων) (owl:DatatypeProperty ή owl:ObjectProperty)	Άτομο ElementInfoType	
Γνώρισμα (xs:attribute)	Ιδιότητα Τύπου Δεδομένων (owl:DatatypeProperty)	Άτομο DatatypePropertyInfo-Type	
Σύνολο (xs:all)	Ανώνυμη Κλάση – Τομή (owl:Class – owl:intersectionOf)		
Ακολουθία (xs:sequence)	Ανώνυμη Κλάση – Τομή (owl:Class – owl:intersectionOf)	Άτομο SequenceInfoType	
Επιλογή (xs:choice)	Ανώνυμη Κλάση – Ένωση (owl:Class – owl:unionOf)	Άτομο ChoiceInfoType	
Υπόμνημα (xs:annotation)	Σχόλιο (rdfs:comment)		

Πίνακας 6.1: Σύνοψη του Μοντέλου Αντιστοίχησης XS2OWL

Παράδειγμα 6.1

Όπως φαίνεται και στο παράδειγμα της υπό-ενότητας 5.2 :

- Ο σύνθετος τύπος Person_Type του XML Σχήματος αντιστοιχεί στην οντολογία στην OWL κλάση Person_Type.
- Ο απλός τύπος validAgeType του XML Σχήματος αντιστοιχεί στην οντολογία στην δήλωση τύπου δεδομένων validAgeType.
- Το σύνθετο στοιχείο Person του XML Σχήματος αντιστοιχεί στην οντολογία στην OWL ιδιότητα αντικειμένων Person.

- Το απλό στοιχείο FirstName του XML Σχήματος αντιστοιχεί στην οντολογία στην OWL ιδιότητα τιμών δεδομένων FirstName.
- Το χαρακτηριστικό City του XML Σχήματος αντιστοιχεί στην οντολογία στην OWL ιδιότητα τιμών δεδομένων City.

6.4 Ανάλυση Διαδικασίας Ανακάλυψης και Αυτόματης Παραγωγής Αντιστοιχήσεων

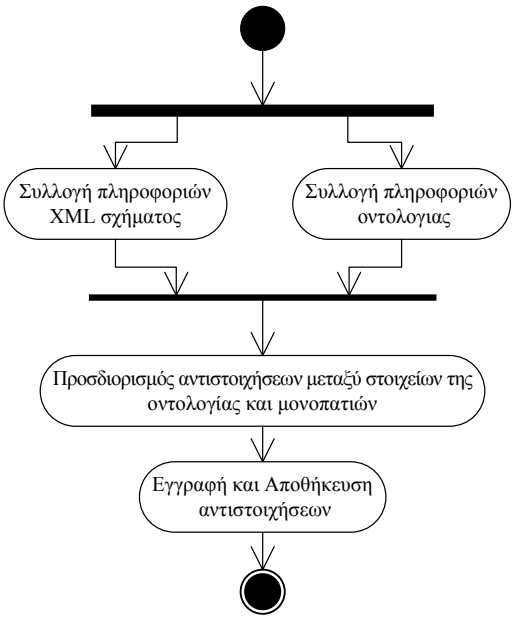
Στην περίπτωση όπου η οντολογία για την οποία ορίζονται οι αντιστοιχήσεις, έχει παραχθεί με την χρήση του συστήματος XS2OWL, υπάρχει η δυνατότητα οι αντιστοιχήσεις μεταξύ της οντολογίας και του XML σχήματος να ανακαλυφθούν(discover) και να γινει αυτόματα η παραγωγή τους. Για την ανακάλυψη και την αυτόματη παραγωγή των αντιστοιχήσεων λαμβάνονται υπόψη οι αντιστοιχήσεις του XS2OWL (Πίνακας 6.1) καθώς και οι άλλες μεθοδολογίες που χρησιμοποιούνται από το XS2OWL για την παραγωγή της οντολογίας.

Θα πρέπει να επισημανθεί ότι η ανακάλυψη των αντιστοιχήσεων δεν περιέχει κάποια αβεβαιότητα (Uncertainty) , δεν γίνεται κατά προσέγγιση, δεν περιέχει σφάλματα και εφαρμόζεται σε όλο το σύνολο των κλάσεων και ιδιοτήτων της οντολογίας.

Κατά την διαδικασία ανακάλυψης και αυτόματης παραγωγής των αντιστοιχιών έχουμε σαν εισόδους το αρχείο όπου περιέχει το XML Schema και το αρχείο της οντολογίας που έχει παραχθεί από αυτό το XML Schema με το σύστημα XS2OWL.

Στο πρώτο στάδιο της ανάλυσης της διαδικασίας ανακάλυψης και αυτόματης παραγωγής των αντιστοιχήσεων η αναπαράσταση γίνεται σε υψηλό επίπεδο αφαίρεσης, όπως φαίνεται και στο UML διάγραμμα δραστηριοτήτων στην Εικόνα 6.5. Στην συνέχεια θα εκβαθύνουμε στην λεπτομερή ανάλυση και περιγραφή της κάθε δραστηριότητας και την διάσπαση της σε περαιτέρω υπό-δραστηριότητες .

Όπως φαίνεται και από το διάγραμμα δραστηριοτήτων στην Εικόνα 6.5 η διαδικασία της παραγωγής ξεκινάει με την συλλογή πληροφοριών από τα αρχεία εισόδου όπου είναι το XML σχήμα και η οντολογία που έχει παραχθεί από το σχήμα αυτό. Απαιτείται η ολοκλήρωση και των δυο διαδικασιών για την εκκίνηση της επόμενη δραστηριότητα. Με την ολοκλήρωση τους και καθώς έχουν συλλεχθεί οι απαραίτητες πληροφορίες από τα δυο αρχεία περνάμε στην διαδικασία προσδιορισμού αντιστοιχήσεων μεταξύ στοιχείων της οντολογίας (Κλάσεων και Ιδιοτήτων) και μονοπατιών (Xpaths). Τέλος και εφόσον έχουν προσδιοριστεί οι απαραίτητες αντιστοιχήσεις ξεκινάει η διαδικασία εγγραφής και αποθήκευσης των αντιστοιχήσεων ακολουθώντας το σχήμα που έχει οριστεί για την δομή των εγγράφων αυτών .

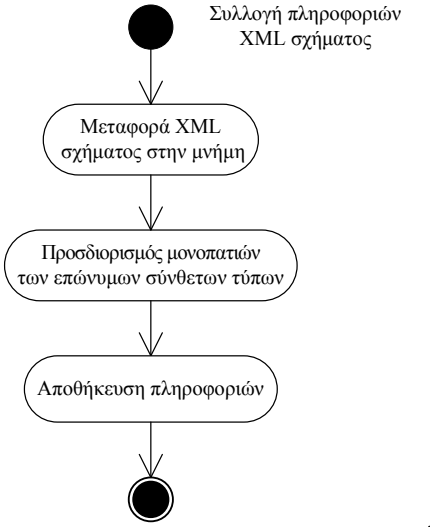


Εικόνα 6.5 Ανακάλυψη και Αυτόματη Παραγωγή

6.4.1 Συλλογή πληροφοριών XML σχήματος

Για την ανακάλυψη και παραγωγή των αντιστοιχήσεων αρχικά ανακτώνται οι απαραίτητες πληροφορίες από το XML σχήμα, πληροφορίες οι οποίες στην συνέχεια θα χρησιμοποιηθούν για τον προσδιορισμό των αντιστοιχήσεων.

Σε αυτό το σημείο γίνεται η ανάλυση της δραστηριότητας **Συλλογή πληροφοριών από το XML Schema**. Όπως απεικονίζεται στην Εικόνα 6.6 η δραστηριότητα αυτή έχει διασπαστεί στις επιμέρους υπό-δραστηριότητες :



Εικόνα 6.6 Συλλογής Πληροφοριών XML Σχήματος

Μεταφορά XML σχήματος στην μνήμη

Το XML σχήμα πρέπει να προσπελαστεί-διαβαστεί και να αποθηκευτεί στην μνήμη, ώστε στην συνέχεια να γίνει η συλλογή τον πληροφοριών από αυτό. Η μεταφορά XML σχήματος στην μνήμη γίνεται με την χρίση του XML beans API και αποθηκεύεται με την μορφή DOM εγγράφου.

Προσδιορισμός μονοπατιών των επώνυμων σύνθετων τύπων (named complex types)

Ο όρος επώνυμος σύνθετος τύπων (named complex type) αναφέρεται στους σύνθετους τύπους οι οποίοι έχουν το χαρακτηριστικό (attribute) name και αντίστοιχα όρος ανώνυμος σύνθετος τύπων αναφέρεται στους σύνθετους τύπους όπου το χαρακτηριστικό name απουσιάζει. Καθώς το XML σχήμα δίνει την δυνατότητα στους σύνθετους τύπους να μην περιέχουν το χαρακτηριστικό name όταν αυτοί δεν είναι ανωτέρου επιπέδου (top level) , δηλαδή να ορίζονται στο εσωτερικό κάποιου XML στοιχείου (element) είτε στο εσωτερικό κάποιου σύνθετου τύπου.

Όπως γνωρίζουμε από το σύστημα αντιστοιχήσεων XS2OWL, οι σύνθετοι XML Schema τύποι αναπαρίστανται στην κύρια οντολογία ως κλάσεις (που ορίζονται μέσω της OWL δομής owl:Class), καθώς τόσο οι σύνθετοι XML Schema τύποι όσο και οι OWL κλάσεις αναπαριστούν σύνολα οντοτήτων με κοινά χαρακτηριστικά. Η αντιστοίχηση αυτή είναι 1:1 που σημαίνει ότι για κάθε σύνθετο τύπο αντιστοιχεί μια κλάση και κάθε κλάση αντιστοιχεί σε έναν σύνθετο τύπο. Το χαρακτηριστικό όνομα της κλάσης (rdf:id) προκύπτει ανάλογα με την περίπτωση, αν ο σύνθετος τύπος που αναπαρίσταται είναι επώνυμος τότε είναι ίδιο με το

όνομα (name) του σύνθετου τύπου , αντίθετα στην περίπτωση ανώνυμου σύνθετου τύπου το όνομα της κλάσης προκύπτει ανάλογα με την θέση στην οποία ορίζεται ο ανώνυμος σύνθετος τύπος. (για λεπτομέρειες βλέπε [17]).

Εφόσον το XML σχήμα είναι αποθηκευμένο στην μνήμη σε μορφή DOM με την βοήθεια του Xpath Over Schema API [51] βρίσκεται για κάθε επώνυμο σύνθετο τύπο τα πιθανά μονοπάτια των στοιχείων (elements) που ανήκουν σε αυτόν τον τύπο. Αυτές οι αντιστοιχίες θα αποτελέσουν την βάση για παραγωγή στην συνέχεια όλων των περαιτέρω αντιστοιχήσεων.

Παράδειγμα 6.2

(Διαδικασία : Προσδιορισμός μονοπατιών των επώνυμων σύνθετων τύπων.)

Για τους σύνθετους τύπους δεδομένων προσδιορίζονται τα πιθανά μονοπάτια στο XML έγγραφο στα οποία εμφανίζονται στοιχεία με τύπο τον συγκεκριμένο σύνθετο τύπο.

- Για τον σύνθετο τύπο Persons_Type: το μονοπάτι /Persons
- Για τον σύνθετο τύπο Person_Type: το μονοπάτι /Persons/Person
- Για τον σύνθετο τύπο Employee_Type : το μονοπάτι /Persons/Employee.
- Για τον σύνθετο τύπο Address_Type : τα μονοπάτια /Persons/Person/Address και /Persons/Employee/Address.

Αποθήκευση πληροφοριών

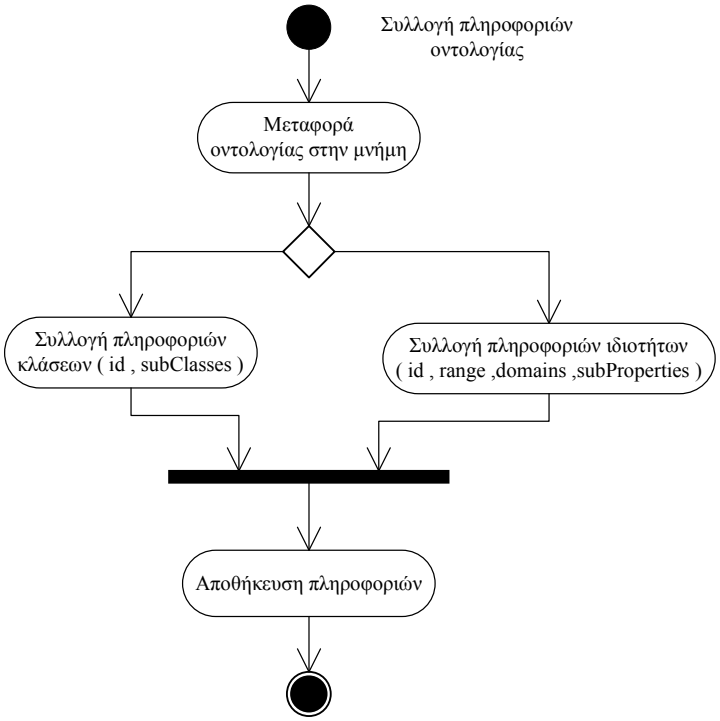
Τέλος αποθηκεύονται στην μνήμη οι επώνυμοι σύνθετοι τύποι με τα αντίστοιχα μονοπάτια όπως αυτά υπολογίστηκαν παραπάνω.

6.4.2 Συλλογή πληροφοριών οντολογίας

Για την ανακάλυψη και παραγωγή των αντιστοιχήσεων αρχικά ανακτώνται οι απαραίτητες πληροφορίες από την οντολογία, πληροφορίες οι οποίες στην συνέχεια θα χρησιμοποιηθούν για τον προσδιορισμό των αντιστοιχήσεων

Οι πληροφορίες που χρειάζεται να αντληθούν από την οντολογία είναι οι πληροφορίες όπου σχετίζονται με τις βασικές έννοιες της οντολογίας δηλαδή πληροφορίες σχετικά με τις κλάσεις και τις ιδιότητες.

Σε αυτό το σημείο γίνεται η ανάλυση της δραστηριότητα **Συλλογή πληροφοριών από την κύρια οντολογία** .Όπως απεικονίζεται στην Εικόνα 6.7 η δραστηριότητα αυτή έχει διασπαστεί στις επιμέρους υπό-δραστηριότητες :



Εικόνα 6.7 Συλλογή Πληροφοριών Οντολογίας

Μεταφορά οντολογίας στην μνήμη

Οντολογία όπου έχει παραχθεί από το σύστημα XS2OWL και αναπαρίσταται με την γλώσσα OWL-DL πρέπει να προσπελαστεί-διαβαστεί και να αποθηκευτεί στην μνήμη, ώστε στην συνέχεια να γίνει η συλλογή των πληροφοριών από αυτήν. Η μεταφορά της οντολογία στην μνήμη γίνεται με την χρήση του Jena API και μεταφέρεται στην μνήμη με μορφή Jena Graph.

Συλλογή πληροφοριών κλάσεων

Όπως είναι γνωστό από το σύστημα αντιστοιχήσεων XS2OWL, οι σύνθετοι XML Schema τύποι αναπαρίστανται στην κύρια οντολογία ως κλάσεις (που ορίζονται μέσω της OWL δομής owl:Class), καθώς τόσο οι σύνθετοι XML Schema τύποι όσο και οι OWL κλάσεις αναπαριστούν σύνολα οντοτήτων με κοινά χαρακτηριστικά.

Οι πληροφορίες που απαιτούνται για τις κλάσεις είναι :

- Το **χαρακτηριστικό όνομα** της κλάσης που αναπαρίσταται μέσω της OWL δομής rdf:id. Οι οντολογίες που δίνονται σαν είσοδο έχουν παραχθεί με το σύστημα XS2OWL και ακολουθούν έναν ενιαίο κανόνα για την ονομασία της κάθε κλάσης (βλέπε ενότητα 6.4.1).

- Τις **υποκλάσεις της κλάσης** που αναπαρίσταται μέσω της OWL δομής owl:subClassOf. Υποκλάσεις δημιουργούνται στην περίπτωση όπου ένας σύνθετος τύπος επεκτείνεται (`extension`) από κάποιον άλλο σύνθετο τύπο , τότε οι κλάσεις που αναπαριστούν αυτούς τους τύπους είναι μεταξύ τους υποκλάση και υπέρ-κλάση αντίστοιχα.

Παράδειγμα 6.3

(Διαδικασία : Συλλογή πληροφοριών κλάσεων.)

Από την οντολογία του παραδείγματος στην υπό-ενότητα 5.2.2 :

- **Πληροφορίες Κλάσεων :**
  - Persons_Type
  - Person_Type
    - Η κλάση "Employee_Type" είναι υπό-κλάση της κλάσης "Person_Type"
  - Employee_Type
  - Address_Type

Συλλογή πληροφοριών ιδιοτήτων

Όπως έχει ήδη αναφερθεί στο συστημα αντιστοιχήσεων XS2OWL, τα XML στοιχεία αναπαρίστανται στην κύρια οντολογία ως OWL ιδιότητες: Τα στοιχεία που έχουν ως τύπο κάποιο απλό τύπο δεδομένων αναπαρίστανται ως ιδιότητες τύπου δεδομένων, (μέσω της δομής owl:DatatypeProperty) και τα στοιχεία σύνθετου τύπου αναπαρίστανται ως ιδιότητες αντικειμένων (μέσω της δομής owl:ObjectProperty).

Οι πληροφορίες που απαιτούνται για τις ιδιότητες είναι :

- Το **χαρακτηριστικό όνομα** της ιδιότητας που αναπαρίσταται μέσω της OWL δομής rdf:id. Θα έχει την τιμή concatenate(name, '__', type) όπου name είναι η τιμή του χαρακτηριστικού name του στοιχείου και όπου type είναι η τιμή του χαρακτηριστικού type του στοιχείου.
- Τα **πεδία ορισμού** (domains) της ιδιότητας, που αναπαρίσταται μέσω της OWL δομής rdfs:domain και έχει την τιμή της κλάσης που αναπαριστά τον σύνθετο τύπου δεδομένου μέσα στο οποίον είναι δηλωμένο το στοιχείο που αναπαριστά η ιδιότητα.

  Στην περίπτωση που το στοιχείο είναι ανώτερου επίπεδου το πεδίο ορισμού παραλείπεται. Επίσης υπάρχει περίπτωση για μια ιδιότητα να ορίζονται μέσω της OWL δομής rdfs:domain παραπάνω από ένα πεδία ορισμού. Αυτό συμβαίνει στην περίπτωση όπου ένα στοιχείο έχει ίδιο όνομα και ίδιο τύπο με κάποιο άλλο που είναι ορισμένο σε διαφορετικό σημείο στο σχήμα.

- Το **πεδίο τιμών** (range) της ιδιότητας, που αναπαρίσταται μέσω της OWL δομής rdfs:range και έχει την τιμή της κλάσης που αναπαριστά τον τύπο του στοιχείο που αναπαριστά η ιδιότητα (Η πληροφορία αυτή είναι απαραίτητη μόνο για ιδιότητες αντικειμένων).
- Οι **υπό-ιδιότητες** της ιδιότητας όπου αναπαρίσταται μέσω της OWL δομής rdfs:subPropertyOf. Υπο-ιδιότητες έχουμε στην περίπτωση όπου ένα στοιχείο επεκτείνει-υποκαθιστά (substitutionGroup) κάποιον άλλο.

Παράδειγμα 6.4

(Διαδικασία : Συλλογή πληροφοριών ιδιοτήτων.)

Από την οντολογία του παραδείγματος στην υπό-ενότητα 5.2.2 :

- **Πληροφορίες Ιδιοτήτων :**
  - Persons
    - Πεδίο Ορισμού : OWL:Thing
    - Πεδίο Τιμών : Persons_Type
  - Person
    - Πεδίο Ορισμού : Persons_Type
    - Πεδίο Τιμών : Person_Type
  - Employee
    - Πεδίο Ορισμού : Persons_Type
    - Πεδίο Τιμών : Employee_Type
  - Address
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : Address_Type
  - Telephone
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : xs:String
  - Age
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : validAgeType
  - FirstName
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : xs:String
  - NickName
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : xs:String

- LastName
    - Πεδία Ορισμών : Person_Type , Employee_Type
    - Πεδίο Τιμών : xs:String
- Salary
    - Πεδίο Ορισμού : Employee_Type
    - Πεδίο Τιμών : xs:integer
- City
    - Πεδίο Ορισμού : Address_Type
    - Πεδίο Τιμών : xs:String
- Street
    - Πεδίο Ορισμού : Address_Type
    - Πεδίο Τιμών : xs:String
    - Υπό-ιδιότητα : Road
- Road
    - Πεδίο Ορισμού : Address_Type
    - Πεδίο Τιμών : xs:String
- Number
    - Πεδίο Ορισμού : Address_Type
    - Πεδίο Τιμών : xs:integer

Αποθήκευση πληροφοριών

Τέλος αποθηκεύονται στην μνήμη οι πληροφορίες που συλλέχθηκαν από την οντολογία, για την χρήση τους στον υπολογισμό των αντιστοιχιών.

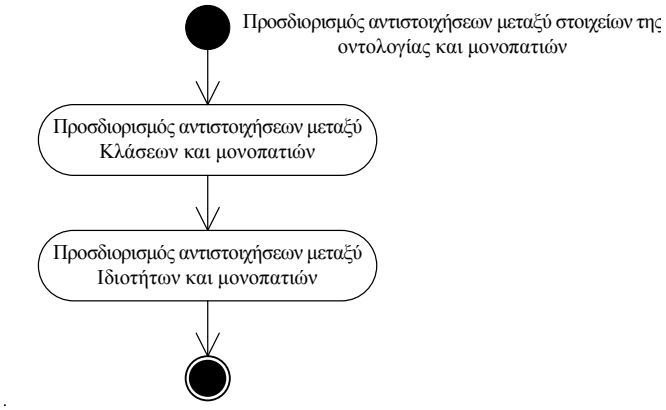
6.4.3 Προσδιορισμός αντιστοιχήσεων μεταξύ στοιχείων της οντολογίας και μονοπατιών

Με την ολοκλήρωση των διαδικασιών που αφορούν στην συλλογή πληροφοριών από τα αρχεία εισόδου και καθώς έχουν συλλεχθεί οι απαραίτητες πληροφορίες από αυτά, περνάμε στην διαδικασία προσδιορισμού αντιστοιχήσεων μεταξύ εννοιών της οντολογίας (Κλάσεων και Ιδιοτήτων) και μονοπατιών (Xpaths) όπως φαίνεται και στην Εικόνα 6.5.

Η δραστηριότητα Προσδιορισμού αντιστοιχήσεων μεταξύ στοιχείων της οντολογίας και μονοπατιών είναι η βασικότερη δραστηριότητα που λαμβάνει χώρα κατά την διαδικασία ανακάλυψης και αυτόματης παραγωγής των αντιστοιχήσεων.

Λόγω της σπουδαιότητας και της πολυπλοκότητας της παρούσας δραστηριότητας θα πραγματοποιηθεί η λεπτομερής ανάλυσης της, για τον λόγο αυτό γίνεται και η διάσπαση της σε δυο υπό-δραστηριότητες όπως φαίνεται και στην Εικόνα 6.8. Σε πρώτο στάδιο, γίνεται ο **Προσδιορισμός των αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών**, αντιστοιχήσεις οι οποίες θα αποτελέσουν τη βάση για τον προσδιορισμό των αντιστοιχήσεων μεταξύ Ιδιοτήτων οντολογίας και μονοπατιών. Με το πέρας της δραστηριότητας και καθώς έχουν προσδιορισθεί οι αντιστοιχήσεις μεταξύ Κλάσεων και μονοπατιών, ξεκινάει η δραστηριότητα προσδιορισμού των αντιστοιχήσεων Ιδιοτήτων και μονοπατιών. Σε αυτή την δραστηριότητά είναι απαραίτητες οι αντιστοιχήσεις μεταξύ κλάσεων και μονοπατιών όπως αυτές έχουν προσδιοριστεί από την προηγούμενη δραστηριότητα.

Σε αυτό το σημείο γίνεται η ανάλυση της δραστηριότητας **Προσδιορισμός αντιστοιχήσεων μεταξύ στοιχείων της οντολογίας και μονοπατιών**. Όπως απεικονίζεται στην Εικόνα 6.8 η δραστηριότητα αυτή έχει διασπαστεί στις επιμέρους υπό-δραστηριότητες :

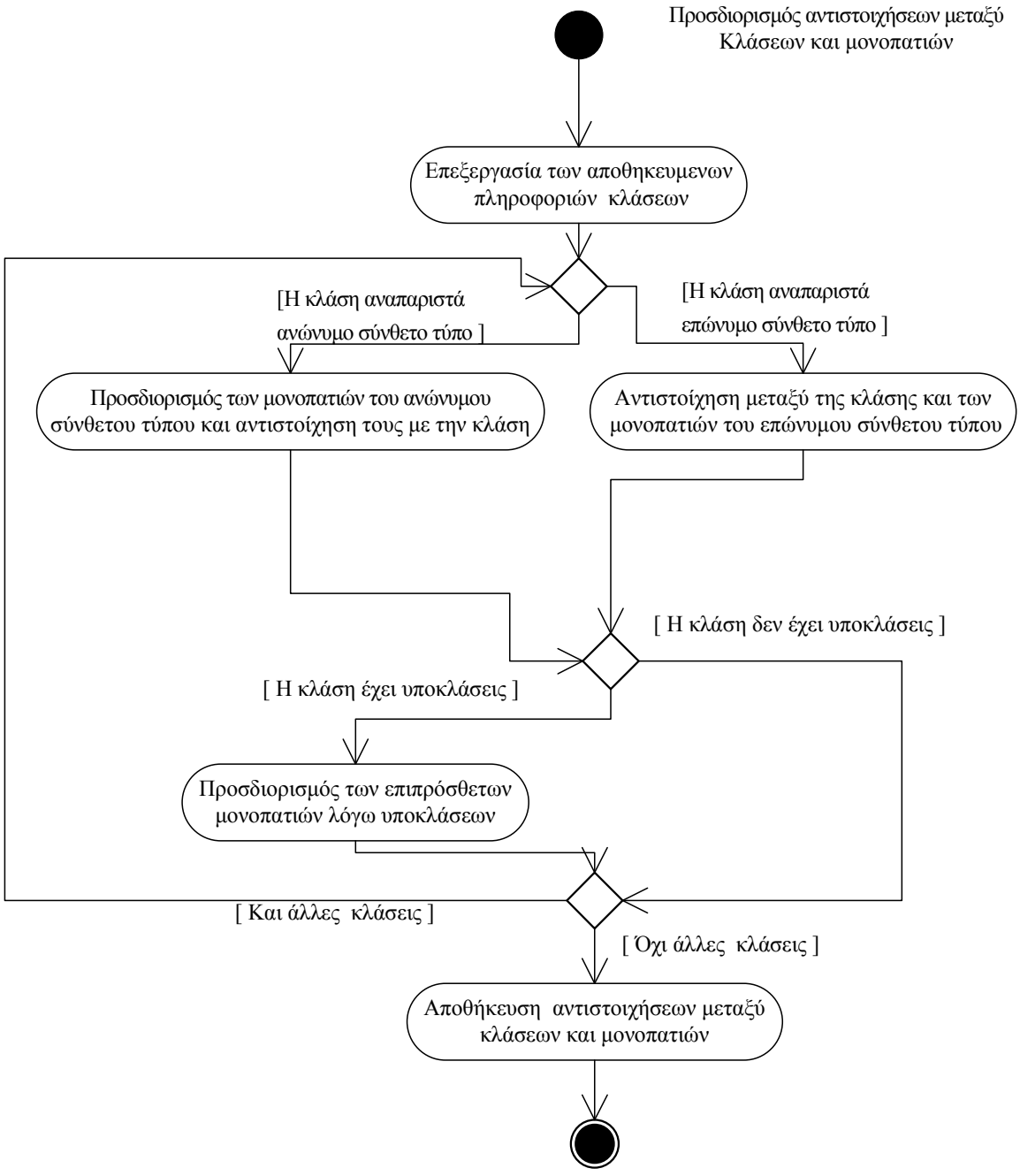


Εικόνα 6.8 Προσδιορισμός αντιστοιχήσεων μεταξύ στοιχείων της οντολογίας και μονοπατιών

6.4.3.1 Προσδιορισμός αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών

Για τον προσδιορισμό των αντιστοιχήσεων σε πρώτο στάδιο, προσδιορίζονται οι αντιστοιχήσεις μεταξύ Κλάσεων της οντολογίας και μονοπατιών, αντιστοιχήσεις οι οποίες θα αποτελέσουν τη βάση για τον προσδιορισμό των αντιστοιχήσεων μεταξύ Ιδιοτήτων της οντολογίας και μονοπατιών. Το διάγραμμα δραστηρι-

οτήτων για τον Προσδιορισμό αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών φαίνεται στην Εικόνα 6.9, επίσης για την καλύτερη κατανόηση και λεπτομερή ανάλυση της δραστηριότητας έχουν αναπτυχθεί οι αντίστοιχοι αλγόριθμοι σε ψευδογλώσσα (Αλγόριθμος 6.1) οι οποίοι προσδιορίζουν την λειτουργικότητα της δραστηριότητας.



Εικόνα 6.9 Προσδιορισμός αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών

Σε αυτό το σημείο γίνεται η ανάλυση της δραστηριότητας **Προσδιορισμός αντιστοιχήσεων μεταξύ κλάσεων και μονοπατιών**. Όπως απεικονίζεται στην Εικόνα 6.9 η δραστηριότητα αυτή έχει διασπαστεί στις επιμέρους υπό-δραστηριότητες :

Επεξεργασία των αποθηκευμένων πληροφοριών κλάσεων

Όπως έχει αναφερθεί σε προηγούμενη ενότητα (βλέπε υπό-ενότητα 6.4.2) έχουν αποθηκευτεί οι απαραίτητες για τις κλάσεις πληροφορίες, οι πληροφορίες αυτές περιλαμβάνουν τα χαρακτηριστικά ονόματα των κλάσεων καθώς και τις υποκλάσεις τους. Επίσης από προηγούμενη δραστηριότητα έχουν αποθηκευτεί τα ονόματα των επωνύμων σύνθετων τύπων. Με χρήση αυτών των πληροφοριών και συγκρίνοντας το χαρακτηριστικό όνομα της κλάσης με τα ονόματα των επωνύμων σύνθετων τύπων γνωρίζουμε αν η κλάση αναπαριστά επώνυμο ή ανώνυμο σύνθετο τύπο. Καθώς όπως έχει αναφερθεί στην περίπτωση επωνύμου σύνθετου τύπου το χαρακτηριστικό όνομα της κλάσης ταυτίζεται με το όνομα του σύνθετου τύπου. Με βάση αυτή την διάκριση συνεχίζουμε στις επόμενες δραστηριότητες.

Αντιστοίχηση μεταξύ της κλάσης και των μονοπατιών του επώνυμου σύνθετου τύπου

Στην περίπτωση που η κλάση αναπαριστά επώνυμο σύνθετο τύπο τα μονοπάτια της ταυτίζονται με τα μονοπάτια του επώνυμου σύνθετου τύπου και έχουν προσδιοριστεί από προηγούμενη δραστηριότητα. Σαν μονοπάτια σύνθετου τύπου έχουμε υπολογίσει τα μονοπάτια των στοιχείων αυτού του τύπου.

Παράδειγμα 6.5

(Διαδικασίες : Επεξεργασία των αποθηκευμένων πληροφοριών κλάσεων και Αντιστοίχηση μεταξύ της κλάσης και των μονοπατιών του επώνυμου σύνθετου τύπου.)

Όπως αναφέρθηκε και στο Παράδειγμα 6.1, ο σύνθετος τύπος Person_Type του XML Σχήματος αντιστοιχεί στην οντολογία στην OWL κλάση Person_Type, το ίδιο ισχύει και για τους σύνθετους τύπους Employee_Type και Address_Type στις OWL κλάσεις Employee_Type και Address_Type αντίστοιχα. Από το παράδειγμα 6.2 έχουν προσδιοριστεί τα μονοπάτια των σύνθετων τύπων, τα μονοπάτια αυτά αντιστοιχούν στις OWL κλάσεις. Επομένως (σε αυτό το σημείο δεν συμπεριλαμβάνονται οι σχέσεις υπόκλάσεων) :

- Για την κλάση Persons_Type: το μονοπάτι /Persons .
- Για την κλάση Person_Type: το μονοπάτι /Persons/Person .
- Για την κλάση Employee_Type : το μονοπάτι /Persons/Employee.

- Για την κλάση Address_Type : τα μονοπάτια /Persons/Person/Address και /Persons/Employee/Address.

Προσδιορισμός των μονοπατιών του ανώνυμου σύνθετου τύπου και αντιστοίχηση τους με την κλάση

Στην περίπτωση που η κλάση αναπαριστά ανώνυμο σύνθετο τύπο, πρέπει να προσδιοριστούν τα μονοπάτια του ανώνυμου τύπου καθώς δεν ήταν δυνατός ο προσδιορισμός τους σε προηγούμενη δραστηριότητα. Καθώς δεν εμφανίζονται στο XML σχήμα με κάποιο όνομα ώστε να μπορεί να γίνει αναφορά σε αυτούς , όπως επίσης είναι απαραίτητη η πληροφορία των αντιστοιχήσεων των επώνυμων σύνθετων τύπων. Ο τρόπος προσδιορισμού των μονοπατιών των ανώνυμων σύνθετων τύπων αναπτύσσεται από το αλγόριθμο **findUnnamedComplexTypeXpaths** (Αλγόριθμος 6.1).

Ο αλγόριθμος **findUnnamedComplexTypeXpaths** δέχεται ως είσοδο την συμβολοσειρά του χαρακτηριστικού ονόματος της κλάσης**(className)** που αναπαριστά κάποιοι ανώνυμο σύνθετο τύπο, ως έξοδο επιστρέφει τα μονοπάτια της κλάσης. Ο αλγόριθμος εκμεταλλεύεται τον κανόνα του XS2OWL συστήματος για την ονομασία των κλάσεων που αναπαριστούν ανώνυμους σύνθετους τύπους για τον προσδιορισμό των μονοπατιών. Σύμφωνα με τον οποίο το χαρακτηριστικό όνομα της κλάσης έχει την τιμή :

- `concatenate(ct_name, '_', element_name, '_UNType')` αν ο σύνθετος τύπος `ct` είναι ανώνυμος τύπος εμφωλευμένος σε κάποιο στοιχείο e, όπου:
  - `ct_name` είναι η τιμή του "`name`" γνωρίσματος του τύπου μέσα στον οποίο ορίζεται το στοιχείο e. Αν το e είναι στοιχείο κορυφαίου επιπέδου, η τιμή του `ct_name` είναι "`NS`".
  - `element_name` είναι η τιμή του "`name`" γνωρίσματος του στοιχείου e.

Προσδιορισμός των επιπρόσθετων μονοπατιών λόγω υποκλάσεων

Η δραστηριότητα αυτή λαμβάνει χώρα στην περίπτωση που οι κλάσεις έχουν υποκλάσεις, και εφόσον πρώτα έχει γίνει ο προσδιορισμός των αντιστοιχήσεων μεταξύ κλάσεων και μονοπατιών χωρίς να λαμβάνονται υπόψη η πιθανή ύπαρξη υποκλάσεων. Οι υποκλάσεις της κλάσης αναπαρίστανται μέσω της OWL δομής owl:subClassOf. Υποκλάσεις έχουμε στην περίπτωση όπου ένας σύνθετος τύπος επεκτείνεται (extension) από κάποιον άλλο σύνθετο τύπο , τότε οι κλάσεις που αναπαριστούν αυτούς τους τύπους είναι μεταξύ τους υποκλάση και υπέρ-κλάση αντίστοιχα.

Εφόσον ο σύνθετος τύπος που αναπαρίσταται από την υπερκλάση επεκτείνεται από τον σύνθετο τύπο που αναπαρίσταται με την υποκλάση, σημαίνει ότι η πληροφορία που περιγράφεται με τον πρώτο τύπο

είναι "πιο γενική" από αυτή που περιγράφει ο δεύτερος. Με βάση αυτή την λογική πρέπει να προσθέσουμε στα μονοπάτια της υπερκλάσης τα μονοπάτια των υποκλάσεων της. Ο τρόπος προσδιορισμού των επιπρόσθετων μονοπατιών που προκύπτουν λόγω της ύπαρξης των υποκλάσεων αναπτύσσεται από το αλγόριθμο **determSubClassesXpaths** (Αλγόριθμος 6.1) .

Ο αλγόριθμος **determSubClassesXpaths** δέχεται ως είσοδο τις αντιστοιχήσεις μεταξύ σύνθετων τύπων και μονοπατιών **(classesXpathsHashTable)** καθώς και τις αντιστοιχήσεις μεταξύ κλάσεων και μονοπατιών **(classesHashTable)**. Ο αλγόριθμος για κάθε κλάση που έχει υποκλάσεις, προσθέτει στα μονοπάτια της κλάσης τα μονοπάτια των υποκλάσεων. Εμπλουτίζοντας με αυτό τον τρόπο τις αντιστοιχήσεις μεταξύ κλάσεων και μονοπατιών λαμβάνοντας υπόψη τις subClassOf σχέσεις μεταξύ τον κλάσεων.

Παράδειγμα 6.6

(Διαδικασία : Προσδιορισμός των επιπρόσθετων μονοπατιών λόγω υποκλάσεων.)

Όπως φαίνεται και στο Παράδειγμα 6.1, η μόνη κλάση που έχει υπό-κλάσεις είναι η κλάση Person_Type. Από το παράδειγμα 6.5 έχουν προσδιοριστεί οι αντιστοιχήσεις για τις κλάσεις Person_Type και Employee_Type (χωρίς να συμπεριληφθούν οι σχέσεις υπό-κλάσεων) :

- Για την κλάση Person_Type: το μονοπάτι /Persons/Person .
- Για την κλάση Employee_Type : το μονοπάτι/Persons/Employee.

Λαμβάνοντας υπόψη τις σχέσεις υπό-κλάσεων, τα μονοπάτια που αντιστοιχούν στην κλάση Employee_Type στην θα προστεθούν στα μονοπάτια της κλάσης Person_Type η οποία αποτελεί υπέρκλάσης της (Βλέπε Αλγόριθμο 6.1 determSubClassesXpaths). Επομένως τελικά προκύπτει:

- Για την κλάση Person_Type: τα μονοπάτια /Persons/Person και /Persons/Employee.
- Για την κλάση Employee_Type : το μονοπάτι /Persons/Employee .

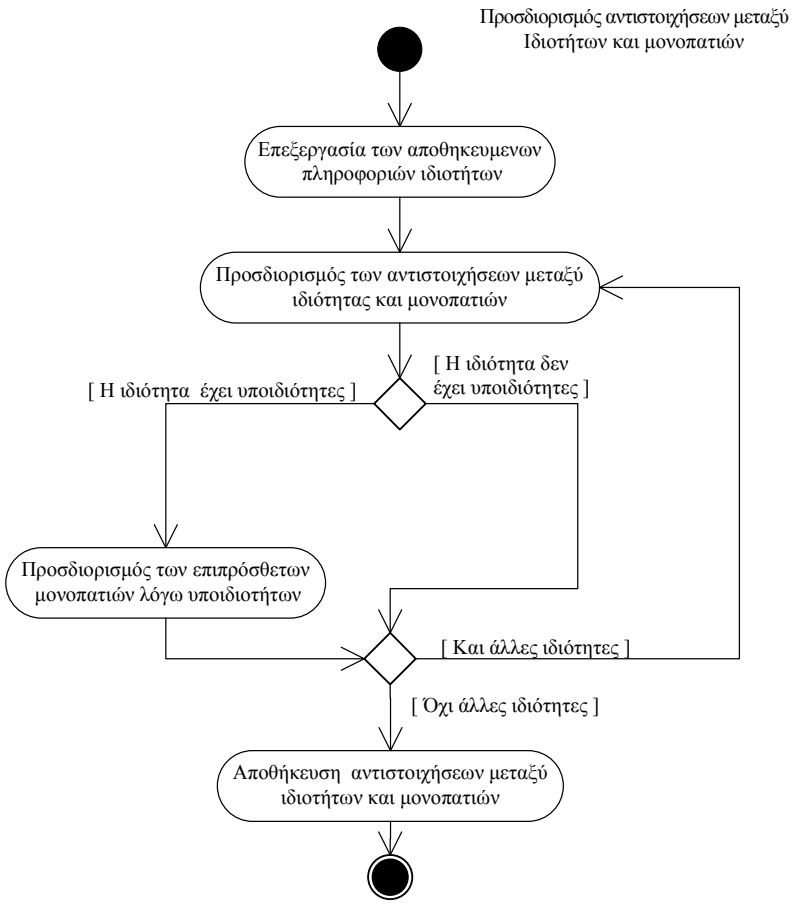
```
algorithm findUnnamedComplexTypeXpaths ( className )
   //Analyze className String starting form the end until
   //we find a Named Complex type or the string beginning

    for each element we found          //για κάθε όνομα element που περιέχει το className
     tmpXpath.addAtbegin("/element") //πρόσθεση στην αρχή του tmpXpath το όνομα του element με πρόθεμα '//'
     if Named coplex type found        //αν βρεθεί επώνυμος σύνθετος τύπος
        for ctXpath in complexType.Xpaths       //για κάθε xpath του σύνθετου τύπου
          xpahts.add( ctXpath.append(tmpXpath)) //στα xpath του σύνθετου τύπου επισυνάπτεται το tmpXpath
        end for
        break
     end if
    end for
   return xpahts  //επιστρέφει τα xpath του ανωνύμου σύνθετου τύπου
end algorithm


//προσδιορίζει τα επιπρόσθετα μονοπάτια λόγω υπό-κλάσεων
algorithm determSubClassesXpaths ( classesHashTable , classesXpathsHashTable )

   for each class in classesHashTable   //επανάληψη για όλες τις κλάσεις
    if class has subclasses             //αν η κλάση έχει υπό- κλάσεις
     for each subclass                  //επανάληψη για κάθε υπό- κλάση
        subClassXpaths = classesXpathsHashTable.get(subclass)   //ανάκτησης των Xpaths της υπό-κλάσης
        add subClassXpaths to class.xpahts             //πρόσθεση των xpaths της υπό-κλάσης στην κλάση
     end for
    end if
   end for
end algorithm
```

Αλγόριθμος 6.1 Αλγόριθμοι Προσδιορισμού αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών

6.4.3.2 Προσδιορισμός αντιστοιχήσεων μεταξύ Ιδιοτήτων και μονοπατιών

Όπως έχει ήδη αναφερθεί στο σύστημα αντιστοιχήσεων XS2OWL(Πίνακας 6.1), τα XML Schema στοιχεία αναπαρίστανται στην κύρια οντολογία ως OWL ιδιότητες: Τα στοιχεία που έχουν ως τύπο κάποιο απλό τύπο δεδομένων, αναπαρίστανται ως ιδιότητες τύπου δεδομένων(μέσω της δομής owl:DatatypeProperty) , τα στοιχεία σύνθετου τύπου αναπαρίστανται ως ιδιότητες αντικειμένων (μέσω της δομής owl:ObjectProperty). Η χαρακτηριστική ονομασία των ιδιοτήτων με βάση το σύστημα αντιστοιχήσεων XS2OWL προκύπτει ως concatenate(name, '__', type) όπου name είναι η τιμή του χαρακτηριστικού name του στοιχείου και όπου type είναι η τιμή του χαρακτηριστικού type του στοιχείου.

Σε αυτό το σημείο γίνεται η ανάλυση της δραστηριότητας **Προσδιορισμός αντιστοιχήσεων μεταξύ Ιδιοτήτων και μονοπατιών**. Όπως απεικονίζεται στην Εικόνα 6.10 η δραστηριότητα αυτή έχει διασπαστεί στις επιμέρους υπό-δραστηριότητες :



Εικόνα 6.10 : Προσδιορισμός αντιστοιχήσεων μεταξύ Ιδιοτήτων και μονοπατιών

Επεξεργασία των αποθηκεμένων πληροφοριών ιδιοτήτων

Όπως έχει αναφερθεί σε προηγούμενη δραστηριότητα (βλέπε υπό-ενότητα 6.4.2) οι απαραίτητες πληροφορίες για τις ιδιότητες έχουν αποθηκευτεί, οι πληροφορίες αυτές περιλαμβάνουν τα χαρακτηριστικά ονόματα των ιδιοτήτων, το πεδίο τιμών τους (Range) , τα πεδία ορισμού (Domains) καθώς και τις υπό-ιδιότητες της εκάστοτε ιδιότητας. Οι παραπάνω πληροφορίες είναι απαραίτητες για τον προσδιορισμό των αντιστοιχήσεων που λαμβάνει χώρα στο επόμενο στάδιο από την δραστηριότητα 'Προσδιορισμός των αντιστοιχήσεων μεταξύ ιδιότητας και μονοπατιών' .

Προσδιορισμός των αντιστοιχήσεων μεταξύ ιδιότητας και μονοπατιών

Ο τρόπος με τον οποίο προσδιορίζονται οι αντιστοιχήσεις μεταξύ ιδιοτήτων και μονοπατιών αναπαριστάται από τον αλγόριθμο **determPropertiesXpaths** Αλγόριθμος 6.2. Ο αλγόριθμος δέχεται ως είσοδο τις αποθηκευμένες για τις ιδιότητες πληροφορίες (**propertiesHashTable**) όπως επίσης και τις αντιστοιχή-

σεις μεταξύ κλάσεων και μονοπατιών (**classesXpathsHashTable**) όπως αυτές έχουν προσδιοριστεί από την δραστηριότητα "Προσδιορισμού αντιστοιχήσεων μεταξύ Κλάσεων και μονοπατιών" Ενότητα 6.4.3.1. Κάνοντας χρήση αυτών των δυο εισόδων ο αλγόριθμος προσδιορίζει αρχικά τις αντιστοιχήσεις μεταξύ ιδιοτήτων και μονοπατιών, αγνοώντας την ύπαρξη υπό-ιδιοτήτων. Με το τέλος των προσδιορισμών ο αλγόριθμος καλεί τον αλγόριθμο **determSubPropertiesXpaths** (Αλγόριθμος 6.2) για τον προσδιορισμό των επιπρόσθετων μονοπατιών στις ιδιότητες τις οποίες έχουν υπό-ιδιότητες.

Παράδειγμα 6.7

(Διαδικασίες : Επεξεργασία των αποθηκεμένων πληροφοριών ιδιοτήτων και Προσδιορισμός των αντιστοιχήσεων μεταξύ ιδιότητας και μονοπατιών)

Από το Παράδειγμα 6.4 έχουν προσδιοριστεί οι πληροφορίες σχετικά με τις παρακάτω ιδιότητες :

- Person
  - Πεδία Ορισμών : Persons_Type
  - Πεδίο Τιμών : Person_Type
- FirstName
  - Πεδία Ορισμών : Person_Type , Employee_Type
  - Πεδίο Τιμών : xs:String
- Street
  - Πεδία Ορισμών : Address_Type
  - Πεδίο Τιμών : xs:String
  - Υπό-ιδιότητα : Road
- Road
  - Πεδία Ορισμών : Address_Type
  - Πεδίο Τιμών : xs:String

Από τα Παραδείγματα 6.5 και 6.6 έχουν προσδιοριστεί οι παρακάτω αντιστοιχήσεις μεταξύ κλάσεων και μονοπατιών:

- Για την κλάση Persons_Type: το μονοπάτι /Persons .
- Για την κλάση Person_Type: τα μονοπάτια /Persons/Person και /Persons/Employee .
- Για την κλάση Employee_Type : το μονοπάτι /Persons/Employee .
- Για την κλάση Address_Type : τα μονοπάτια /Persons/Person/Address και /Persons/Employee/Address .

Επομένως τα μονοπάτια των ιδιοτήτων ανάλογα με τα πεδία ορισμού τους είναι (σε αυτό το σημείο δεν συμπεριλαμβάνονται οι σχέσεις υπό-ιδιοτήτων, βλέπε Αλγόριθμο 6.2 determPropertiesXpaths):

- Για την ιδιότητα Person : το μονοπάτι /Persons/Person
- Για την ιδιότητα FirstName : τα μονοπάτια /Persons/Person/FirstName και /Persons/Employee/FirstName .
- Για την ιδιότητα Street : τα μονοπάτια /Persons/Person/Address/Street και /Persons/Employee/Address/Street.
- Για την ιδιότητα Road : τα μονοπάτια /Persons/Person/Address/Road και /Persons/Employee/Address/Road.

Προσδιορισμός των επιπρόσθετων μονοπατιών λόγω υπό-ιδιοτήτων

Ο προσδιορισμός των επιπρόσθετων μονοπατιών λόγω ύπαρξης υπό-ιδιοτήτων πραγματοποιείται από τον αλγόριθμο **determSubPropertiesXpaths** Αλγόριθμος 6.2, ο οποίος εκτελείται εφόσον πραγματοποιηθεί ο προσδιορισμός των αντιστοιχήσεων μεταξύ ιδιοτήτων και μονοπατιών από τον αλγόριθμο **determPropertiesXpaths**. Ο αλγόριθμος δέχεται σαν είσοδο τις αποθηκευμένες για τις ιδιότητες πληροφορίες (**propertiesHashTable**) και υπολογίζει τα επιπρόσθετα μονοπάτια των ιδιοτήτων οι οποίες έχουν υπό-ιδιότητες.

Παράδειγμα 6.8

(Διαδικασία : Προσδιορισμός των επιπρόσθετων μονοπατιών λόγω υπό-ιδιοτήτων.)

Όπως φαίνεται και στο Παράδειγμα 6.4, η μόνη ιδιότητα που έχει υπό-ιδιότητες είναι η ιδιότητα Street. Από το παράδειγμα 6.7 έχουν προσδιοριστεί οι αντιστοιχήσεις για τις ιδιότητες Road και Street (χωρίς να συμπεριληφθούν οι σχέσεις υπό-ιδιοτήτων) .

Λαμβάνοντας υπόψη τις σχέσεις υπό-ιδιοτήτων, τα μονοπάτια που αντιστοιχούν στην ιδιότητα Road θα προστεθούν στα μονοπάτια της ιδιότητα Street η οποία αποτελεί υπέρ-ιδιότητα της. Άρα τελικά προκύπτει (βλέπε Αλγόριθμο 6.2 determSubPropertiesXpaths) :

- Για την ιδιότητα Street : τα μονοπάτια /Persons/Person/Address/Street , /Persons/Employee/Address/Street, /Persons/Person/Address/Road /Persons/Employee/Address/Road.

```
//αλγόριθμος προσδιορισμού των αντιστοιχήσεων μεταξύ ιδιοτήτων και μονοπατιών
algorithm determPropertiesXpaths( propertiesHashTable , classesXpathsHashTable )
  for each property in propertiesHashTable //επανάληψη για κάθε ιδιότητα
   domains = property.getDomains()            //τα πεδία ορισμού της ιδιότητας
   for each domain in domains                 //για κάθε πεδίο ορισμού της ιδιότητας
    domainXapths = classesXpathsHashTable.get( domain )
    propertyDomainXpahts = domainXpahts
    //τα Xpaths τις κλάσης πεδίου ορισμού ανατίθενται στα xpahts του πεδίου ορισμού της ιδιότητα
    for each dXpath in domainXpaths
       propertyRangeXpahts.add( dxpath.append( "/property.getlabel()" ) )
       //τα Xpaths του πεδίων τιμών, δημιουργείται αν επισυνάψουμε το όνομα του στοιχειού που αντι-
       //στοιχεί αυτή η ιδιότητα και περιέχεται στο label της ιδιότητας.
    end for
   end for
   propertiesXpathsHashTable.put(property.Name , propertyRangeXpahts , propertyDomainXpahts)
  end for

  determSubPropertiesXpaths ( propertiesHashTable )
  //καλείται η συνάρτηση για τον υπολογισμό των επιπροσθέτων λόγω υπό-ιδιοτήτων Xpaths
end algorithm



algorithm determSubPropertiesXpaths ( propertiesHashTable )
  for each property in propertiesHashTable     //για κάθε ιδιότητα
    if property has subproperties              //αν έχει υπό-ιδιότητες
       for each subproperty                    //για κάθε υπό-ιδιότητα
            add subproperty range Xpaths to property range Xpaths
       end for
       if property is Object Property  //αν είναι ιδιότητα αντικειμένων
            for each subproperties and property //για κάθε ιδιότητα και υπό-ιδιότητα
              add to range classes the new xpaths produced by subproperty relation
              //προσθέτει στις κλάσεις των πεδίων τιμών των ιδιοτήτων τα νέα Xpaths
            end for
       end if
  end if
end for
```

Αλγόριθμος 6.2 Αλγόριθμοι Προσδιορισμού αντιστοιχήσεων μεταξύ Ιδιοτήτων και μονοπατιών

6.5 Περίληψη

Στο παρόν κεφάλαιο, έγινε η περιγραφή της μορφής με την οποία αποθηκεύονται οι αντιστοιχήσεις μεταξύ της οντολογίας και του XML σχήματος. ι αντιστοιχήσεις ορίζονται μεταξύ στοιχείων της οντολογίας (κλάσεις και ιδιότητες) και μονοπατιών στα XML δεδομένα. Με αυτό τον τρόπο επιτυγχάνεται η "άμεση" σύνδεση των στοιχείων της οντολογίας με τα XML δεδομένα. Οι αντιστοιχήσεις αυτές θα χρησιμοποιηθούν κατά τις διαδικασίες της μεταφράσεις (Κεφάλαια 9-12) για την ανάπτυξη των XQuery εκφράσεων, καθώς τα στοιχεία της οντολογίας που περιέχονται στις SPARQL ερωτήσεις θα αντικατασταθούν με μονοπάτια στις XQuery εκφράσεις.

Επιπρόσθετα αναλύθηκε ο τρόπος με τον οποίο γίνεται η ανακάλυψη(discovery) και η αυτόματη παραγωγή των αντιστοιχήσεων, στην περίπτωση την οποία η οντολογία έχει παραχθεί με χρήση της μεθόδου XS2OWL . Όπως έχει αναφερθεί η διαδικασία της μετάφρασης που περιγράφεται στα επόμενα κεφάλαια είναι ανεξάρτητη από τη μορφή ,την σύνταξη και τον τρόπο δημιουργίας και αποθήκευσης των αντιστοιχήσεων όπως επίσης και από το σύστημα XS2OWL.

Με την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων επιτυγχάνεται μια πλήρως αυτοματοποιημένη διαδικασία, χωρίς να είναι απαραίτητη η παρέμβαση ανθρώπινου παράγοντα. Επίσης επιτυγχάνεται η πλήρης αντιστοίχηση όλων των στοιχείων τόσο της οντολογίας όσο και του XML σχήματος, με αντιστοιχήσεις απολύτως σημασιολογία ορθές, χωρίς την πιθανότητα σφάλματος ή παρερμηνείας κατά την δημιουργία τους, κάτι που μπορεί να συμβεί στην περίπτωση "χειροκίνητου"(manual) ορισμού των αντιστοιχήσεων από εξειδικευμένο χρήστη.

Στο επόμενο κεφάλαιο (Κεφάλαιο 7) ξεκινάει η ανάλυση της διαδικασίας μετάφρασης των σημασιολογικών ερωτήσεων, αναλύοντας την διαδικασία "Κανονικοποίηση Σχηματομορφών Γράφων" η οποία πραγματοποιεί την κανονικοποίηση της σχηματομορφής γράφου των SPARQL ερωτήσεων. Η κανονικοποίηση έχει ως σκοπό την βελτιστοποίηση και απλοποίηση της διαδικασίας της μετάφρασης.

7 Κανονικοποίηση Σχηματο - μορφών Γράφων

7.1 Εισαγωγή

Η διαδικασία "**Κανονικοποίηση Σχηματομορφών Γράφων**" **(Graph Pattern Normalization)** πραγματοποιεί την κανονικοποίηση της σχηματομορφής γράφου των SPARQL ερωτήσεων. Η κανονικοποίηση έχει ως σκοπό την βελτιστοποίηση και απλοποίηση της διαδικασίας της μετάφρασης. Αναδρομική εφαρμογή κανόνων ισοδυναμίας , ιδιοτήτων των τελεστών που εμφανίζονται μεταξύ των σχηματομορφών γράφων και re-writing τεχνικών , έχουν ως αποτέλεσμα τη δημιουργία μιας κανονικοποιημένης γραμματικής για τη σχηματομορφή γράφου της ερώτησης.

Η διαδικασία "Κανονικοποίηση Σχηματομορφών Γράφων" είναι η πρώτη διαδικασία που εκτελείται κατά την μετάφραση των ερωτήσεων, η διαδικασία δέχεται ως είσοδο την SPARQL ερώτηση και πραγματοποιεί

την κανονικοποίηση της σχηματομορφή γράφου που περιέχει. Η κανονικοποιημένη αυτή σχηματομορφή γράφων πρόκειται να επεξεργαστεί στην συνέχεια από τις υπόλοιπες διαδικασίες της μετάφρασης.

Η κανονικοποιημένη σχηματομορφή γράφου έχει την μορφή P1 union P2 union…union Pn, όπου P1,P2…Pn είναι union free σχηματομορφές γράφων (βλέπε Συμπέρασμα 7.1). Αυτό έχει ως αποτέλεσμα την απλοποίηση της διαδικασίας μετάφρασης, καθώς η αρχική σχηματομορφή διασπάται σε n σχηματομορφές από τις οποίες η κάθε μια μεταφράζεται ανεξάρτητα από τις άλλες.

Η εφαρμογή των κανόνων ισοδυναμίας, ιδιοτήτων των τελεστών που εμφανίζονται μεταξύ των σχηματομορφών γράφων και re-writing τεχνικών έχει ως αποτέλεσμα την μετακίνηση των σχηματομορφών γράφων και την δημιουργία μεγαλύτερων(συνεχόμενων) βασικών σχηματομορφών γράφων. Ο στόχος είναι να επιτευχθεί όσο το δυνατόν μεγαλύτερος αριθμός συνεχόμενων τριπλετών σχηματομορφών με τον τελεστή AND(Ορισμός 2.15) ανάμεσα τους. Ο λόγος αυτής της προσπάθειας είναι η ελαχιστοποίηση του αριθμού εκτελέσεων των διαδικασιών "Μετάφραση Βασικών Σχηματομορφών Γράφων" (Κεφάλαιο 11) και "Σύνδεση Μεταβλητών"(Κεφάλαιο 10) ώστε να απλοποιηθεί και να βελτιστοποιηθεί (Optimization) η διαδικασία της μετάφρασης.

Η εμφάνιση του τελεστή AND μεταξύ τριπλετών σχηματομορφών και η μετάφραση του με τον αλγόριθμο BGP2XQuery(ο οποίος μεταφράζει Βασικές Σχηματομορφές Γράφων (BGPs) σε XQuery εκφράσεις) έχει το πλεονέκτημα, ότι ο τελεστής προσομοιώνεται από την σύνταξη των XQuery εκφράσεων και δεν γίνεται εφαρμογή του τελεστή σε επίπεδο αποτελεσμάτων, με αποτέλεσμα βελτιστοποίηση της XQuery ερώτησης(βλέπε και Κεφάλαιο 11).

Στην προηγούμενη υπό-ενότητα για λόγους απλούστευσης και για την αποφυγή πιθανής ασάφειας, στον ορισμό των σχηματομορφών γράφων δεν γίνετε χρήση εννοιών και σύνταξης από την γραμματική της γλώσσας, αλλά δίνεται ένας αλγεβρικός ορισμός (Ορισμός 2.8) των σχηματομορφών γράφων. Στην Εικόνα 7.1 φαίνεται η αντιστοιχία ανάμεσα στην σύνταξη με χρήση κανόνων της γραμματικής της γλώσσας και στην αλγεβρική σύνταξη, καθώς στα παραδείγματα που ακολουθούν για λόγους κατανόησης γίνεται χρήση και των δυο τρόπων σύνταξης. Έστω P1 και P2 σχηματομορφές γράφων και R SPARQL έκφρασης όπως ορίζεται στον Ορισμό 2.8.

SPARQL Σύνταξη	**Αλγεβρική Σύνταξη**
{ P1 P2 }	(P1 AND P2)
{ P1 OPTIONAL { P2 }}	(P1 OPT P2)
{ P1 } UNION { P2 }	(P1 UNION P2)
{ P1 FILTER (R) }	(P1 FILTER R)

Εικόνα 7.1 : Αντιστοιχήσεις Τρόπου Σύνταξης

Σημείωση: *Όπως παρατηρείται από την Εικόνα 7.1 ο τελεστής AND παραλείπεται κατά την σύνταξη της γλώσσας SPARQL, το οποίο φαίνεται και από τον Ορισμό 2.15.*

Στην συνέχεια της ενότητας ορίζονται οι κανόνες ισοδυναμίας (υπό-ενότητα 7.2) οι οποίοι έχουν αποδεικτή στο [11] και χρησιμοποιούνται για την κανονικοποίηση της σχηματομορφής γράφου των SPARQL ερωτήσεων. Με χρήση των κανόνων αυτών και re-writing τεχνικών προκύπτει η κανονικοποιημένη γραμματική όπως αναλύεται στην υπό-ενότητα 7.3. Οι κανόνες ισοδυναμίας και ως επέκταση η κανονικοποιημένη γραμματική διαφοροποιείται για τις "καλά σχεδιασμένες"*(Ορισμός 2.*16) και όχι "καλά σχεδιασμένες" σχηματομορφές γράφων. Η κανονικοποίηση έχει ως σκοπό την βελτιστοποίηση και απλοποίηση της διαδικασίας της μετάφρασης.

7.2 Κανόνες Ισοδυναμίας

Για την συνέχεια θεωρείται ότι έστω P_i μια σχηματομορφή γράφων, $P_\emptyset$ η κενή σχηματομορφή, tp_i τριπλέτα σχηματομορφής και R μια SPARQL έκφρασης.

Κανόνας 7.1 [11] Από τον ορισμό των AND , OPT και UNION ισχύουν οι εξής κανόνες.

Για το AND και UNION ισχύουν:

- Προσεταιριστική Ιδιότητα (Associative):

  P_1 AND (P_2 AND P_3) ≡ (P_1 AND P_2) AND P_3 ,

  P_1 UNION (P_2 UNION P_3) ≡ (P_1 UNION P_2) UNION P_3

- Αντιμεταθετική Ιδιότητα (Commutative):

  P_1 AND P_2 ≡ P_2 AND P_1 ,

  P_1 UNION P_2 ≡ P_2 UNION P_1

Για το OPT ισχύει :

- Αριστερή Προσεταιριστική Ιδιότητα (Left Associative):

  P_1 OPT P_2 OPT P_3 ≡ (P_1 OPT P_2)OPT P_3

Κανόνας 7.2 Απαλείφονται όλες οι κενές σχηματομορφές γράφων $P_{\emptyset}$ (Empty Graph Patterns)

Απόδειξη

Από τις σημειώσεις των ορισμών 2.9 και 2.11 προκύπτει ότι η κενή αντιστοίχηση $\mu_{\emptyset}$ που προκύπτει από την κενή σχηματομορφή γράφων $P_{\emptyset}$,ταυτίζεται (match) με οποιοδήποτε γράφο (συμπεριλαμβανομένου του κενού γράφου).Οπότε, η αφαίρεση του δεν επηρεάζει τις αντιστοιχίες των αποτελεσμάτων (solutions mappings).

Παραδειγμα 7.1

Με εφαρμογή του Κανόνα 7.2 προκύπτει :

- $\{P_1 \text{ AND } P_{\emptyset}\} \Rightarrow \{P_1\}$
- $\{P_1 \text{ UNION } P_{\emptyset}\} \Rightarrow \{P_1\}$
- $\{P_1 \text{ OPT } P_{\emptyset}\} \Rightarrow \{P_1\}$
- $\{ P_{\emptyset} \text{ OPT } P_1\} \Rightarrow \{P_1\}$

Στην γραμματική της γλώσσας ο κενός γράφος σχηματομορφής συμβολίζετε ως '{ }'

- $\{\{P_1\}\} \Rightarrow \{P_1\}$
- $\{\{ \}P_1\}\} \Rightarrow \{P_1\}$
- $\{P_1 \ \{ \} \ P_2\} \Rightarrow \{P_1 \ P_2\}$
- { P_1 OPTIONAL{ }}=> $\{P_1\}$

Κανόνας 7.3 $(P_1 \text{ AND } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ AND } P_2) \text{ UNION } (P_1 \text{ AND } P_3))$

Απόδειξη Στο [11]

Κανόνας 7.4 $(P_1 \text{ OPT } (P_2 \text{ UNION } P_3)) \equiv ((P_1 \text{ OPT } P_2) \text{ UNION } (P_1 \text{ OPT } P_3))$

Απόδειξη Στο [11]

Κανόνας 7.5 $((P_1 \text{ UNION } P_2) \text{OPT } P_3) \equiv ((P_1 \text{ OPT } P_3) \text{ UNION } (P_2 \text{ OPT } P_3))$

Απόδειξη Στο [11]

Κανόνας 7.6 (P_1 UNION P_2)FILTER R) ≡ ((P_1 FILTER R) UNION (P_2 FILTER R))

Απόδειξη Στο [11]

Συμπέρασμα 7.1 [11] Με χρήση των Κανόνων **7.1 , 7.3 , 7.4 , 7.5** και **7.6** είναι δυνατή η μετατροπή κάθε σχηματομορφής γράφου σε μια ισοδύναμη της μορφής :

P_1 UNION P_2 UNION P_3 UNION … UNION P_n

όπου κάθε σχηματομορφή γράφου P_i ($1 \leq i \leq n$) δεν περιέχει τον τελεστή UNION (UNION-free).

Κανόνας 7.7 Αφαίρεση των περιττών αγκύλων '{ }'

Παραδειγμα 7.2

Με εφαρμογή του Κανόνα 7.7 προκύπτει:

{{tp_1}{tp_2}{tp_3}}=> { tp_1 tp_2 tp_3 }

Σημείωση : *{ tp_1 tp_2 tp_3 } = (tp_1 AND tp_2 AND tp_3)*

Κανόνας 7.8 (P_1 AND (P_2 OPT P_3)) ≡ ((P_1 AND P_2) OPT P_3)

Απόδειξη Στο [11]

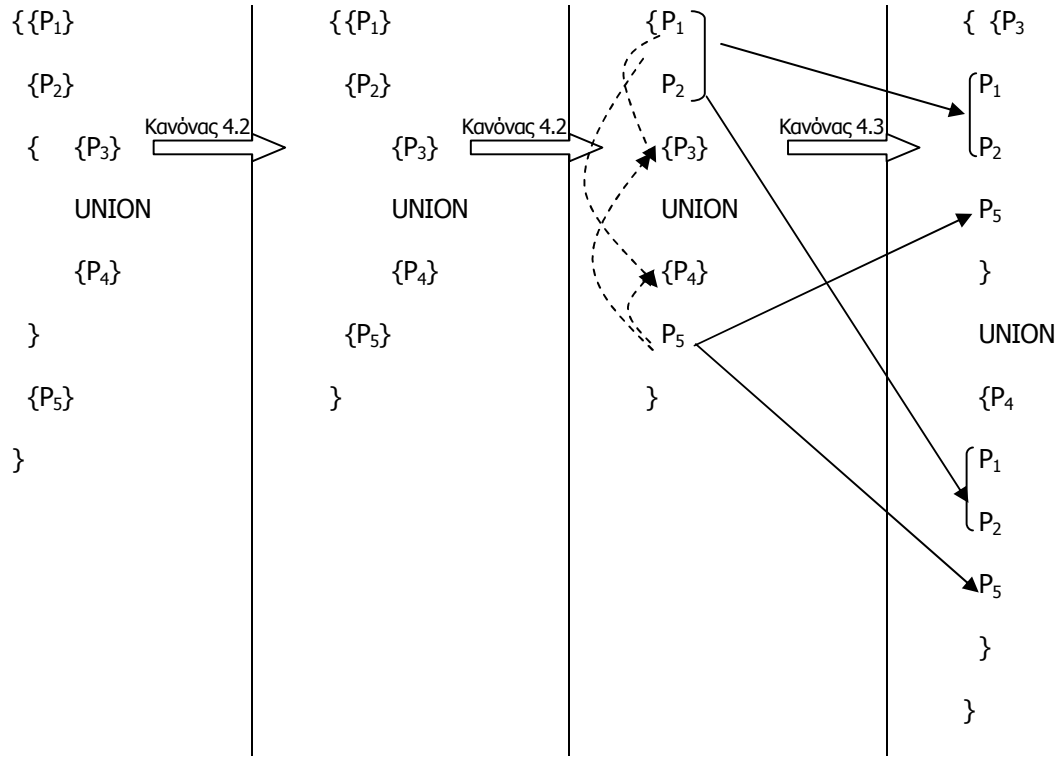
Σημείωση : Ι*σχύει <u>μόνο</u> για τους "καλά σχεδιασμένους"(Ορισμός 2.16) γράφους .*

Κανόνας 7.9 ((P_1 OPT P_2) OPT P_3) ≡ ((P_1 OPT P_3) OPT P_2).

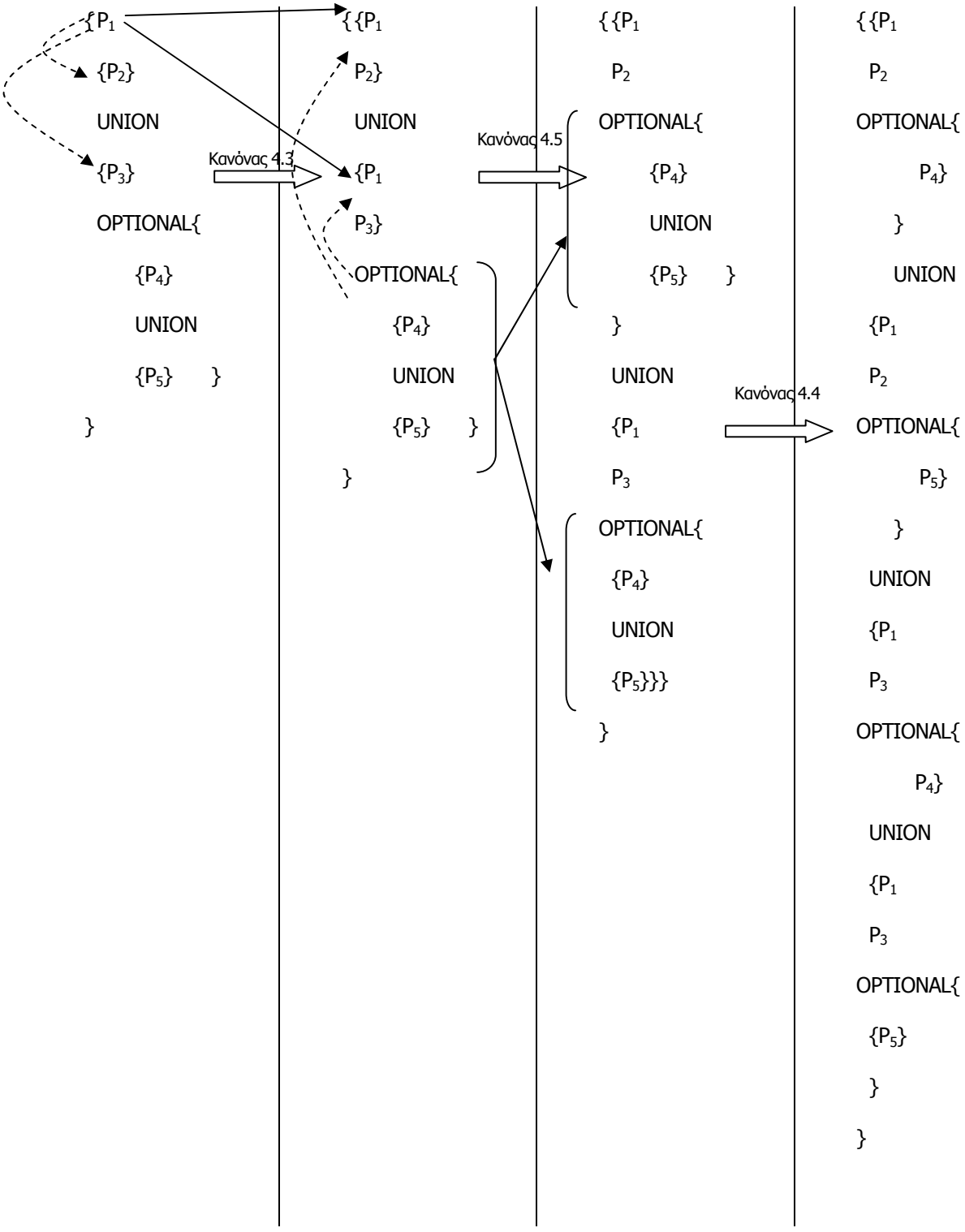
Απόδειξη Στο [11]

Σημείωση : Ι*σχύει <u>μόνο</u> για τους "καλά σχεδιασμένους"(Ορισμός 2.16) γράφους .*

Παραδειγμα 7.3



Παραδειγμα 7.4



7.3 Κανονικοποιημένη Γραμματική (Normalized Grammar)

Όπως αναφέρθηκε στην προηγούμενη ενότητα 7.2 και αποδεικνύεται στο [11], οι κανόνες ισοδυναμίας 7.8 και 7.9 ισχύουν μόνο για τις "καλά σχεδιασμένες"*(Ορισμός 2.16)* σχηματομορφές γράφων, αυτό έχει ως αποτέλεσμα οι γραμματικές που προκύπτουν από την εφαρμογή των κανόνων ισοδυναμίας (της ενότητας 7.2) να διαφέρουν μεταξύ των "καλά σχεδιασμένων" και των " όχι καλά σχεδιασμένων" σχηματομορφών γράφων.

7.3.1 Κανονικοποιημένη Γραμματική για τις Καλά Σχεδιασμένες Σχηματομορφές Γράφων (Well Designed Graph Patterns)

Με αναδρομική εφαρμογή των κανόνων ισοδυναμίας που καταγράφονται στην προηγούμενη υπό-ενότητα για τις "καλά σχεδιασμένες"*(Ορισμός 2.16)* σχηματομορφές γράφων προκύπτει η εξής κανονικοποιημένη γραμματική για τις "καλά σχεδιασμένες" σχηματομορφές γράφων (Εικόνα 7.2).

a) GP := BGP | Optional_Pattern | Union_Pattern
b) BGP := "("((tp "AND" tp)* Filter* (tp "AND" tp)*)* ")"
c) Optional_Pattern := BGP ("OPT" "("Optional_Pattern Filter* ")")+
d) Union_free_Pattern := BGP | Optional_Pattern
e) Union_Pattern := "("Union_free_Pattern ")" ("UNION" "("Union_free_Pattern) ")" *
f) tp := Triple Pattern
g) Filter := FILTER(R)
h) R := SPARQL expression

Εικόνα 7.2 : Κανονικοποιημένη Γραμματική (Well Designed Pattern)

Όπως παρατηρείται και στην Εικόνα 7.2 έχει επιτευχθεί ο αρχικός σκοπός ,δηλαδή η δημιουργία όσο το δυνατόν μεγαλύτερων βασικών σχηματομορφών γράφων. Επίσης, έχει επιτευχθεί το βέλτιστο αποτέλεσμα καθώς οι τελεστές AND εμφανίζονται μόνο μεταξύ τριπλετών σχηματομορφών, δηλαδή μόνο στο εσωτερικό των βασικών σχηματομορφών γράφων. Έτσι ο τελεστής AND για τους "καλά σχεδιασμένους" γράφους υλοποιείται με τον αλγόριθμο

BGP2XQuery καθώς αυτός εμφανίζεται μόνο στο εσωτερικό των βασικών γράφων σχηματομορφών.

Στην συνέχεια αναλύονται οι κανόνες της Κανονικοποιημένη Γραμματική της Εικόνας 7.2 και παρουσιάζονται οι πιθανές μορφές της κανονικοποιημένη σχηματομορφής γράφου.

- **Κανόνας h)** Οι εκφράσεις των φίλτρων δεν διαφοροποιούνται και παραμένουν όπως της αρχικής γραμματικής(από την προδιαγραφή της γλώσσας SPARQL[5]) και αναλύονται στον Ορισμό 2.8.

  Παραδείγματα: ?χ=20 , ?χ= "John" , ?x>?y , ?salary>2000 , ?x=?y&&?y>20 || ?z<10 , isBlank(?x), isBlank(?x)=true , isBlank(?x)=false , bound(?x) && ?x>100 κτλ.

- **Κανόνας f)** Οι Τριπλέτες σχηματομορφών (triples patterns) δεν διαφοροποιούνται και παραμένουν όπως της αρχικής γραμματικής (από την προδιαγραφή της γλώσσας SPARQL[5]).

  Παραδείγματα:

  - <http://example.org/book/book1> dc:title ?title
  - ?x ns:Firstname "john"
  - ?x ?p "John"
  - ?x ?p ? o

- **Κανόνας b)** Οι Βασικές Σχηματομορφές Γράφων (Basic Graph Pattern) δεν διαφοροποιούνται και παραμένουν όπως της αρχικής γραμματικής (από την προδιαγραφή της γλώσσας SPARQL[5]).

  Παραδείγματα:

  - ?X ns:FirstName "John" .
    ?X ns:LastName ?lastName.
    ?X ns:Address ?addr.
  - ?X ns:FirstName "John" .
  - ?X ns:FirstName ?fn .
    FILTER(?fn= "John")
  - ?X ns:FirstName ?fn .
    FILTER(?fn= "John")

?X ns:LastName ?lastName.

?X ns:Address ?addr.

FILTER(?lastName= "Jonshon")

- **Κανόνας c)**

  Παραδείγματα :

  - {?X ns:FirstName "John" .
    OPTIONAL{?X ns:Address ?addr.}}
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr.} }
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr. FILTER(bound(?X))} }
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr.}
    OPTIONAL{?X ns:LastName ?lastName.} }
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr.
    OPTIONAL{?X ns:LastName ?lastName.} }}
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr.
    OPTIONAL{?X ns:LastName ?lastName.} }}
  - {?X ns:FirstName "John" .
    FILTER(bound(?X))
    OPTIONAL{?X ns:Address ?addr.
    OPTIONAL{?X ns:LastName ?lastName.} }
    OPTIONAL{?X ns:NickName ?nickName.} }

- **Κανόνας e)**

  Παραδείγματα:

  - {?X ns:FirstName "John" .
    ?X ns:LastName ?lastName.
    ?X ns:Address ?addr. }
    UNION
    {?X ns:NickName ?nickName.}
  - {?X ns:FirstName "John" .
    ?X ns:LastName ?lastName.
    ?X ns:Address ?addr. }
    UNION
    {?X ns:NickName ?nickName.
    FILTER(?nickName="nick")}
  - {?X ns:FirstName "John" .
    OPTIONAL{?X ns:Address ?addr.}
    UNION
    {?x ?p ? o }
    UNION
    {?X ns:LastName ?lastName.
    FILTER(?lastName= "Jonshon") }

7.3.2 Κανονικοποιημένη Γραμματική για τις Όχι Καλά Σχεδιασμένες Σχηματομορφές Γράφων (non-Well Designed Graph Patterns)

Όπως έχει αναφερθεί και προηγουμένως για τους γράφους που δεν είναι "καλά σχεδιασμένοι"*(Ορισμός 2.16)* δεν ισχύουν οι κανόνες 7.8 και 7.9. Με εφαρμογή των υπόλοιπων κανόνων ισοδυναμίας από την υπό-ενότητα 7.2 έχουμε την παρακάτω κανονικοποιημένη γραμματική για τις όχι "καλά σχεδιασμένες" σχηματομορφές γράφων (Εικόνα 7.3) :

a) **GP := BGP| And_Pattern | Optional_Pattern | Union_Pattern**
b) **BGP := "(" ((tp "AND" tp)* Filter* (tp "AND" tp)*)* ")"**
c) **And_Pattern := "(" (Union_free_Pattern "AND" Union_free_Pattern)* ")"**
d) **Optional_Pattern := Union_free_Pattern ("OPT" "("Union_free_Pattern ")")+**
e) **Union_free_Pattern := BGP| And_Pattern | Optional_Pattern**
f) **Union_Pattern := "("Union_free_Pattern")" ("UNION" "("Union_free_Pattern ")")***
g) **tp := Triple Pattern**
h) **Filter := FILTER(R)**
i) **R := SPARQL expression**

Εικόνα 7.3 : Κανονικοποιημένη Γραμματική (non-Well Designed Pattern)

Όπως παρατηρείται και στην Εικόνα 7.3 δεν έχει επιτευχθεί το βέλτιστο αποτέλεσμα όπως για τους "καλά σχεδιασμένους" γράφους αφού δεν μεγιστοποιείται το μέγεθος των βασικών σχηματομορφών γράφων, υπάρχουν όμως περιπτώσεις που αυξάνεται σημαντικά. Μπορεί όμως κανείς να παρατηρείται ότι ο τελεστής AND δεν εμφανίζεται μόνο ανάμεσα σε τριπλέτες σχηματομορφών, άρα υπάρχει η ανάγκη της εφαρμογής του τελεστή AND στο επίπεδο των αποτελεσμάτων ερωτήσεων XQuery .

Στην συνέχεια αναλύεται ο κανόνας c) (που είναι ο μόνος που διαφοροποιείται από την γραμματική της εικόνας 7.2) της Κανονικοποιημένη Γραμματική της Εικόνας 7.3 και παρουσιάζονται οι πιθανές μορφές του.

- **Κανόνας e)**

  Παραδείγματα:

  - {?x a ns:Person_Type
    OPTIONAL { ?x ns:FirstName ?n} }
    {?y a ns:Person_Type
    OPTIONAL { ?y ns:NickName ?n} }}
  - { {?x FirstName "Paul"
    OPTIONAL { ?y NickName ?n} }
    ?y FirstName "George" }

- { {?x FirstName "Paul"
  OPTIONAL { ?y NickName ?n} }
  ?y FirstName "George" }
  UNION
  {?X ns:LastName ?lastName.
  FILTER(?lastName= "Jonshon") }

7.4 Περίληψη

Στο κεφάλαιο αυτό, έγινε περιγραφή της διαδικασία "Κανονικοποίηση Σχηματομορφών Γράφων" η οποία πραγματοποιεί την κανονικοποίηση της σχηματομορφής γράφου των SPARQL ερωτήσεων. Η κανονικοποίηση έχει ως σκοπό την βελτιστοποίηση και απλοποίηση της διαδικασίας της μετάφρασης.

Επίσης ορίστηκαν οι κανόνες ισοδυναμίας οι οποίοι χρησιμοποιούνται για την κανονικοποίηση, ορίστηκε η κανονικοποιημένη γραμματική η οποία προκύπτει με χρήση των κανόνων αυτών και re-writing τεχνικών. Η κανονικοποιημένη αυτή σχηματομορφή γράφων πρόκειται να επεξεργαστεί στην συνέχεια από τις υπόλοιπες διαδικασίες της μετάφρασης.

Στο επόμενο κεφάλαιο (Κεφάλαιο 7) γίνεται η περιγραφή της διαδικασίας "προσδιορισμός των τύπων των μεταβλητών" (Variables Types Determination), η οποία προσδιορίζει τον τύπο των μεταβλητών που περιέχονται στην SPARQL ερώτηση.

8 Προσδιορισμός Τύπων Μεταβλητών

8.1 Εισαγωγή

Η διαδικασία **"προσδιορισμός των τύπων των μεταβλητών" (Variables Types Determination)**, προσδιορίζει τον τύπο των μεταβλητών που περιέχονται στην SPARQL ερώτηση. Η διαδικασία πραγματοποιείται κυρίως ώστε οι τύποι των μεταβλητών να χρησιμοποιηθούν στην συνέχεια για την ανάπτυξη του return clause των XQuery ερωτήσεων, καθώς ανάλογα με τον τύπο της μεταβλητής διαφοροποιείται και η μορφή των αποτελεσμάτων που παράγονται από τα XQuery ερωτήματα(βλέπε υπό-ενότητα 8.3). Επίσης οι τύποι των μεταβλητών χρησιμοποιούνται κατά την διαδικασία "Σύνδεση Μεταβλητών" για τον προσδιορισμό των συνδέσεων. Τέλος με αυτόν τον τρόπο ελέγχεται και η συνέπεια χρήσης των μεταβλητών στην ερώτηση(βλέπε Παράδειγμα 8.2). Στην περίπτωση παραβίασης της συνέπειας, ακυρώνεται η μετάφραση της Union-Free σχηματομορφή γράφου που εμφανίζεται η παραβίαση ή και όλης της ερώτησης (στην περίπτωση που αποτελείται από μια μόνο Union-Free σχηματομορφή), άλλωστε αυτός είναι και ο λόγος που ο προσδιορισμός των

τύπων των μεταβλητών πραγματοποιείται στα πρώτα στάδια της διαδικασίας της μετάφρασης. Με αυτό τον τρόπο επιτυγχάνεται βελτιστοποίηση(optimization) της διαδικασίας μετάφρασης.

Η διαδικασία εκτελείται αμέσως μετά την διαδικασία κανονικοποίησης της σχηματομορφής γράφου για κάθε Union-Free σχηματομορφή γράφου που περιέχεται στην κανονικοποιημένη σχηματομορφή γράφου. Δέχεται σαν είσοδο μια Union-Free σχηματομορφή γράφου και προσδιορίζει τους τύπους των μεταβλητών που περιέχονται σε αυτή. Οι τύποι των μεταβλητών πρόκειται στην συνέχεια να χρησιμοποιηθούν από την διαδικασία Σύνδεσης Μεταβλητών (Κεφάλαιο 10) και από την διαδικασία Μετάφραση Βασικών Σχηματομορφών Γράφων(Κεφάλαιο 11).

Στην συνέχεια του κεφαλαίου ορίζονται οι πιθανοί τύποι των μεταβλητών (Ορισμός 8.3) , οι κανόνες προσδιορισμού των τύπων των μεταβλητών (υπό-ενότητα 8.2) και τέλος η συσχέτιση της μορφής των αποτελεσμάτων από τον τύπο των μεταβλητών (υπό-ενότητα 8.3).

Ορισμός 8.1 Συνέπεια χρήσης μεταβλητών Ορίζεται η αντιστοίχηση λύσεων, που προκύπτει από μια τριπλέτα T και για μια μεταβλητή V σε RDF-όρους ως η συνάρτηση μ_{VT} (Παραλλαγή Ορισμού 2.11). Η συνέπεια χρήσης των μεταβλητών σε ένα σύνολο από τριπλέτες σχηματομορφών, τηρείται αν για κάθε μεταβλητή x και για κάθε τριπλέτα t του συνόλου σχηματομορφών το πεδίο τιμών της συνάρτησης μ_{xt} ανήκει σε ένα μόνο από τα σύνολα RDF-L , RDF-B , I_C , I_{DTP} I_{OP} (για τα RDF-L και RDF-B, βλέπε Ορισμός 2.1) . Θεωρούνται τα σύνολα I_C , I_{DTP} I_{OP} να είναι υποσύνολα του I , με το I_C να περιέχει τα στοιχεία του I που είναι στιγμιότυπα κλάσεων, I_{DTP} να περιέχει τα στοιχεία του I που είναι μεταβλητές τύπων δεδομένων και I_{OP} να περιέχει τα στοιχεία του I που είναι ιδιότητες αντικειμένων.

Ορισμός 8.2 Κενοί Κομβόι (Blank Nodes) στην SPARQL Σύμφωνα με την προδιαγραφή της γλώσσας SPARQL [5] η χρήση κενών κόμβων (Blank Nodes) στις σχηματομορφές γράφων είναι ένας εναλλακτικός ,"μη-κομψός" τρόπος χρήσης μεταβλητών, με την διαφοροποίηση ότι κάθε κενός κόμβος έχει εμβέλεια(scope) μόνο στην βασική σχηματομορφή γράφων που χρησιμοποιείται, και δεν μπορεί να χρησιμοποιηθεί σε άλλη. Για την συνέχεια ο όρος μεταβλητή θα αφορά τόσο στις μεταβλητές όσο και στους κενούς κόμβους.

Ορισμός 8.3 Τύποι Μεταβλητών Ορίζονται οι εξής τύποι μεταβλητών:

- **Μεταβλητή Τύπου Στιγμιότυπου Κλάσης – CIVT (**Class Instance Variable Type)
- **Μεταβλητή Τύπου Σταθεράς - LVT** (Literal Variable Type)
- **Άγνωστου Τύπου Μεταβλητή – UVT** (Unknown Variable Type)
- **Μεταβλητή Ιδιότητας Τύπου Δεδομένων - DTPVT**(Data Type Predicate Variable Type)
- **Μεταβλητή Ιδιότητας Αντικειμένων - OPVT** (Object Predicate Variable Type)
- **Μεταβλητή Ιδιότητας Αγνώστου Τύπου– UPVT** (Unknown Predicate Variable Type)

8.2 Κανόνες Προσδιορισμού Τύπου Μεταβλητών

Οι κανόνες προσδιορισμού εφαρμόζονται επαναληπτικά σε όλο το σύνολο των τριπλετών που εμφανίζονται στην σχηματομορφή γράφου της ερώτησης, με σκοπό να προσδιορίσουν τους τύπους των μεταβλητών που περιέχονται σε αυτή. Από αυτές τις τριπλέτες εξαιρούνται όλες οι Onto τριπλέτες(Ορισμός 9.1), εκτός της ειδικής περίπτωσης της τριπλέτας rdf:type. Ο λόγος αυτής της εξαίρεσης, όπως αναλύεται στο Κεφάλαιο 9 είναι ότι οι μεταβλητές αυτών των τριπλετών δεν συμπεριλαμβάνονται στο return clause των XQueries. Στην περίπτωση ύπαρξης τελεστή UNION, όπως είναι λογικό οι κανόνες εφαρμόζονται ανεξάρτητα στις σχηματομορφές που χωρίζονται από UNION. Τέλος, στην περίπτωση την οποία για μια μεταβλητή γίνει προσδιορισμός διαφορετικών τύπων στο σύνολο των τριπλετών, η μετάφραση ακυρώνεται καθώς δεν μπορεί να πραγματοποιηθεί λόγω λανθασμένης αρχικής ερώτησης.

Σημείωση : *Όπως είναι λογικό οι Άγνωστοι τύποι(UTV - UTPV) έχουν μικρότερη "βαρύτητα" έναντι των άλλων τύπων. Για τον λόγο αυτό ο προσδιορισμός μια μεταβλητής ως Αγνώστου Τύπου ενώ έχει ήδη προσδιοριστεί κάποιος άλλος μη- Άγνωστος τύπος δεν προκαλεί ακύρωση της μετάφρασης αλλά ούτε και αλλαγή τύπου από μη-Ανώνυμο σε Ανώνυμο. Ομοίως, αν μια μεταβλητή προσδιοριστεί ως Αγνώστου τύπου και στην συνέχεια προσδιορίζεται ως μη-Αγνώστου τύπου δεν ακυρώνεται η μετάφραση αλλά ούτε γίνεται αλλαγή τύπου.*

Ορίζουμε τα σύνολα **DTPS** (Data Type Properties Set) που περιέχουν τις ιδιότητες τιμών δεδομένων, **OPS** (Object Properties Set) που περιέχουν τις ιδιότητες αντικειμένων, **V** (Variable) που περιέχουν τις μεταβλητές που εμφανίζονται στις τριπλέτες που εφαρμόζονται οι κανόνες και **L**(Literal) που περιέχουν τις σταθερές. Ως T_X ορίζεται ο τύπος της μεταβλητής χ.

Έστω η τριπλέτα σχηματομορφών **S P O**

a) **Αν S ϵ V => T_S = CIVT** . Αν το υποκείμενο είναι μεταβλητή, τότε ο τύπος της θα είναι Μεταβλητή Τύπου Στιγμιότυπου Κλάσης

b) **Αν P ϵ DTPS => Αν O ϵ V , T_O = LVT**. Αν το κατηγόρημα είναι ιδιότητα τύπων δεδομένων, τότε αν το αντικείμενο είναι μεταβλητή ο τύπος της θα είναι Μεταβλητή Τύπου Σταθεράς.

c) **Αν P ϵ OPS => Αν O ϵ V , T_O = CIVT** . Αν το κατηγόρημα είναι ιδιότητα αντικειμένων, τότε αν το αντικείμενο είναι μεταβλητή, ο τύπος της θα είναι Μεταβλητή Τύπου Στιγμιότυπου Κλάσης.

d) **T_P = DTPVT <=> T_O = LVT** $\forall$ **P,O ϵ V** . Αν το κατηγόρημα είναι μεταβλητή και είναι τύπου Μεταβλητή Ιδιότητας Τύπου Δεδομένων, τότε αν και το αντικείμενο είναι μεταβλητή ο τύπος της θα είναι Μεταβλητή Τύπου Σταθεράς. Επίσης ισχύει και το αντίστροφο.

e) **T_P = OPVT <=> T_O = CIVT** $\forall$ **P,O ϵ V.** Αν το κατηγόρημα είναι μεταβλητή και είναι τύπου Μεταβλητή Ιδιότητας Τύπου Αντικειμένων, τότε αν και το αντικείμενο είναι μεταβλητή θα είναι τύπου Μεταβλητή Τύπου Στιγμιότυπου Κλάσης. Επίσης ισχύει και το αντίστροφο.

f) **Αν O ϵ L => Αν P ϵ V , T_P = DTPVT .** Αν το αντικείμενο είναι σταθερά, τότε αν το κατηγόρημα είναι μεταβλητή, θα είναι τύπου Μεταβλητή Τύπου Στιγμιότυπου Σταθεράς.

g) **Άγνωστοι τύποι(UTV - UTPV)** εμφανίζονται στην περίπτωση που μια τριπλέτα έχει μεταβλητές στο κατηγόρημα και στο αντικείμενο και κανένας από τους τύπους των μεταβλητών δεν προσδιορίζεται από άλλη τριπλέτα.

Παράδειγμα 8.1

Έστω μια βασική σχηματομορφή γράφων αποτελούμενη από τις εξής τριπλέτες σχηματομορφών :

?x ?p ?n .

?y FirstName ?n .

?k ?p "john" .

Εφαρμόζοντας τους κανόνες στις παραπάνω τριπλέτες

- Για την πρώτη τριπλέτα (?x ?p ?n) ισχύει :
  - Tx = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανονας α)
  - Tp = Μεταβλητή Ιδιότητας Αγνώστου Τύπου– UPVT (Κανονας g)
  - Tn = Άγνωστου Τύπου Μεταβλητή – UVT (Κανονας g)
- Για την δεύτερη τριπλέτα (?y FirstName ?n) ισχύει :
  - Ty = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανονας α)
  - Tn = Μεταβλητή Τύπου Σταθεράς - LVT (Κανονας b)
- Για την τρίτη τριπλέτα (?k ?p "John") ισχύει :
  - Tk = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανόνας α)
  - Tp = Μεταβλητή Ιδιότητας Τύπου Δεδομένων – DTPVT (Κανόνας d)

Επομένως συνολικά ισχύει :

- Tx = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανόνας α)
- Ty = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανόνας α)
- Tk = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανόνας α)

- Tp = Μεταβλητή Ιδιότητας Τύπου Δεδομένων – DTPVT (Κανόνας d)
- Tn = Μεταβλητή Τύπου Σταθεράς - LVT (Κανόνας b)

Παράδειγμα 8.2

Έστω μια βασική σχηματομορφή γράφων αποτελούμενη από τις εξής τριπλέτες σχηματομορφών :

?n ?p ?k .

?y FirstName ?n .

Εφαρμόζοντας τους κανόνες στις παραπάνω τριπλέτες

- Για την πρώτη τριπλέτα (?n ?p ?k) ισχύει :
  - Tn = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανονας α)
  - Tp = Μεταβλητή Ιδιότητας Αγνώστου Τύπου– UPVT (Κανονας g)
  - Tk = Άγνωστου Τύπου Μεταβλητή – UVT (Κανονας g)
- Για την δεύτερη τριπλέτα (?y FirstName ?n) ισχύει :
  - Ty = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανονας α)
  - Tn = Μεταβλητή Τύπου Σταθεράς - LVT (Κανονας b)

Επομένως συνολικά ισχύει :

- Ty = Μεταβλητή Τύπου Στιγμιότυπου Κλάσης - CIVT (Κανόνας α)
- Tk = Άγνωστου Τύπου Μεταβλητή – UVT (Κανονας g)
- Tp = Μεταβλητή Ιδιότητας Αγνώστου Τύπου– UPVT (Κανονας g)
- Tn = **? (Παραβίαση στη Συνέπειας Χρήσης)**

Για την μεταβλητή n εμφανίζεται παραβίαση στη συνέπειας χρήσης, καθώς από την πρώτη τριπλέτα προσδιορίζεται ως Μεταβλητή Τύπου Στιγμιότυπου Κλάσης – CIVT ενώ από την δεύτερη τριπλέτα προσδιορίζεται ως Μεταβλητή Τύπου Σταθεράς - LVT . Επόμενος μια τέτοια σχηματομορφή δεν είναι δυνατόν να επαληθευτεί (γίνει match) σε RDF δεδομένα.

8.3 Συσχέτιση τύπων μεταβλητών με μορφή αποτελεσμάτων

Στην παρούσα εργασία για την μετάφραση SPARQL σε XQuery δεν έχουν αξιοποιηθεί οι μηχανισμοί περιορισμού ταυτότητας (Identity Constraints), οι οποίοι προσφέρονται από το XML Σχήμα. Στην πράξη, οι μηχανισμοί αυτοί χρησιμοποιούνται σπανίως και ειδικότερα στα ευρέως αποδεκτά XML πρότυπα όπως MPEG-7 MDS, MPEG-21 DIA Architecture, IEEE LOM, SCORM, METS κτλ δεν χρησιμοποιούνται, για τον λόγο αυτό θα αποτελέσουν μια από τις μελλοντικές επεκτάσεις της παρούσας εργασίας. Οι μηχανισμοί αυτοί είναι ID/IDREF/IDREFS, που έχουν "κληρονομηθεί" από τα DTD's (Document Type Definition) και οι Key/Keyref/Unique οι οποίοι προσφέρουν την δυνατότητα δήλωσης μοναδικότητας (uniqueness) και αναφοράς (reference) σε στοιχεία και χαρακτηριστικά μέσα στο XML έγγραφο.

Παρατήρηση 8.1 Εφόσον δεν έχουν αξιοποιηθεί οι μηχανισμοί που προσφέρει το XML σχήμα για περιορισμούς ταυτότητας δεν γίνεται να έχουμε αναφορές μέσα στο XML έγγραφο. Αυτό, έχει ως συνέπεια κάθε στοιχείο ή γνώρισμα να βρίσκεται σε μια μόνο θέση μέσα στο XML έγγραφο.

Ορισμός 8.3 Ταυτότητα στοιχείων και χαρακτηριστικών στο XML έγγραφο Από την Παρατήρηση 8.1 προκύπτει ότι κάθε στοιχείο ή γνώρισμα μπορεί να βρίσκεται σε μια θέση μέσα στο XML έγγραφο. Αυτή η θέση μπορεί να προσδιοριστεί επακριβώς με την χρήση μονοπατιών (Xpath) και αρίθμηση κόμβων. Ο συνδυασμός αυτός θα αποτελεί και την ταυτότητα των στοιχείων και χαρακτηριστικών σε ένα XML έγγραφο.

Παρατήρηση 8.2 Ο έλεγχος ισότητας μεταξύ XML κόμβων, που αντιστοιχούν σε απλά στοιχεία η χαρακτηριστικά, μπορεί να πραγματοποιηθεί με την χρήση του τελεστή "=" στα δεδομένα των κόμβων (Atomic Values) με αποτέλεσμα να είναι εφικτή η σύγκριση κόμβων που αποτελούν απλά στοιχεία ή χαρακτηριστικά. Από την άλλη, η ισότητα μεταξύ σύνθετων στοιχείων μπορεί να οριστεί με την χρήση αναφορών που υποστηρίζονται από τους μηχανισμούς περιορισμού ταυτότητας, ενώ ο έλεγχος ισότητας σύνθετων στοιχείων βασισμένος στην ισότητα των δεδομένων απλών στοιχείων και χαρακτηριστικών που περιέχουν σε αυτά, δεν θα ήταν ορθολογικός. Για τον λόγο αυτόν και εφόσον δεν γίνεται αξιοποίηση των μηχανισμών περιορισμού ταυτότητάς, θεωρείται ότι δεν υπάρχουν ισότητες μεταξύ σύνθετων στοιχείων. Για παράδειγμα να μην μπορεί να θεωρηθεί πιθανή η ισότητα μεταξύ των σύνθετων στοιχείων που βρίσκονται στα μονοπάτια /Persons/Person και /Persons/Employee, κάτι που θα μπορούσε να ισχύει με την χρήση μηχανισμών περιορισμού ταυτότητας. Τα παραινώ λαμβάνονται υπόψη κατά την διαδικασία της μετάφρασης.

a. **Για τις μεταβλητές που έχουν προσδιοριστεί ως : Μεταβλητή Τύπου Στιγμιότυπου Κλάσης – CIVT**. Η εκτέλεση της SPARQL ερώτησης σε RDF δεδομένα, θα επέστρεφε σαν αποτέλεσμα τα IRI's των πόρων (Resources) που αντιστοιχηθήκαν σε αυτές, τα οποία θα αποτελούσαν στιγμιότυπα (Instances) κάποιας κλάσης. Το IRI αυτό αποτελεί και την ταυτότητα (Identity) του πόρου. Σε αντιστοιχία με το IRI, όπως αναφέρεται και στον Ορισμό 8.3, ο συνδυασμός μονοπατιών (Xpath) και η αρίθμηση κόμβων αποτελεί την ταυτότητα των στοιχείων και χαρακτηριστικών σε ένα XML έγγραφο. Σε σύνδεση με το URI του XML εγγράφου στο οποίο βρίσκεται ο κόμβος, δημιουργείται και η πλήρης ταυτότητα των XML δεδομένων. Αυτή την μορφή θα έχουν και τα αποτελέσματα για τη Μεταβλητή Τύπου Στιγμιότυπου Κλάσης – CIVT.

b. **Για τις μεταβλητές που έχουν προσδιοριστεί ως : Μεταβλητή Ιδιότητας Τύπου Δεδομένων** ,**Μεταβλητή Ιδιότητας Αντικειμένων** και **Μεταβλητή Ιδιότητας Αγνώστου Τύπου**. Η εκτέλεση της SPARQL ερώτησης σε RDF δεδομένα θα επέστρεφε σαν αποτέλεσμα τα IRI's των ιδιοτήτων που αντιστοιχηθήκαν σε αυτές. Σε αντιστοιχία, επιστρέφεται το μονοπάτι του κόμβου που έχει ανατεθεί στην μεταβλητή, σε σύνδεση με το με το URI του XML εγγράφου στο οποίο βρίσκεται ο κόμβος .

c. **Για τις μεταβλητές που έχουν προσδιοριστεί ως : Μεταβλητή Τύπου Σταθεράς**. Η εκτέλεση της SPARQL ερώτησης σε RDF δεδομένα θα επέστρεφε σαν αποτέλεσμα τις σταθερές που αντιστοιχηθήκαν σε αυτές. Σε αντιστοιχία επιστρέφονται τα περιεχόμενα των XML κόμβων.

d. **Για τις μεταβλητές που έχουν προσδιοριστεί ως : Αγνώστου Τύπου Μεταβλητή**. Η μορφή των αποτελεσμάτων εξαρτάται από το είδος του XML κόμβου που έχει αντιστοιχηθεί η μεταβλητή . Για τα σύνθετα στοιχεία η μορφή των αποτελεσμάτων είναι ιδία με τις μεταβλητές τύπου στιγμιότυπου κλάσης (a), ενώ για τα απλά στοιχεία ή γνωρίσματα τα αποτελέσματα έχουν ιδία μορφή με τις μεταβλητές τύπου σταθεράς (c).

Σημείωση: *Για λεπτομέρειες υλοποιήσεις βλέπε υπό-ενότητα 11.5.*

8.4 Επίλογος

Στο κεφάλαιο αυτό, οριστήκαν οι τύποι των μεταβλητών που περιέχονται στην SPARQL ερώτηση, επίσης ορίστηκαν οι κανόνες με του οποίους προσδιορίζονται οι τύποι των μεταβλητών. Οι τύποι των μεταβλητών θα χρησιμοποιηθούν στην συνέχεια για την ανάπτυξη του return clause των XQuery ερωτήσεων, καθώς ανάλογα με τον τύπο της μεταβλητής διαφοροποιείται και η μορφή των αποτελεσμάτων που παράγονται από τα XQuery ερωτήματα. Πιο συγκεκριμένα ο τύπος των

μεταβλητών και η συσχέτιση τους με την μορφή των αποτελεσμάτων θα χρησιμοποιηθεί στην διαδικασία Μετάφραση Βασικών Σχηματομορφών Γράφων (Κεφάλαιο 10) και κατά την διαδικασία "Σύνδεση Μεταβλητών"(Κεφάλαιο 9) για τον προσδιορισμό των συνδέσεων.

Τέλος με τον προσδιορισμό των τύπων των μεταβλητών ελέγχεται και η συνέπεια χρήσης των μεταβλητών στην ερώτηση. Στην περίπτωση παραβίασης της συνέπειας, ακυρώνεται η μετάφραση της Union-Free σχηματομορφή γράφου που εμφανίζεται η παραβίαση ή και όλης της ερώτησης (στην περίπτωση που αποτελείται από μια μόνο Union-Free σχηματομορφή), άλλωστε αυτός είναι και ο λόγος που ο προσδιορισμός των τύπων των μεταβλητών πραγματοποιείται στα πρώτα στάδια της διαδικασίας της μετάφρασης. Με αυτό τον τρόπο επιτυγχάνεται βελτιστοποίηση(optimization) της διαδικασίας μετάφρασης.

Στο επόμενο κεφάλαιο(Κεφάλαιο 9) γίνεται η περιγραφή της διαδικασίας "Επεξεργασία Οντο τριπλέτων", αναλύεται ο τρόπος χειρισμού των ερωτήσεων που περιέχουν Οντο τριπλέτες και ο τρόπος σύνδεσης(bind) των μεταβλητών με μονοπάτια του XML εγγράφου, που προκύπτουν από τις Οντο τριπλέτες.

9 Επεξεργασία Όντο Τριπλέτων

9.1 Εισαγωγή

Στο παρόν κεφάλαιο περιγράφεται η διαδικασία **"Επεξεργασίας Οντο Τριπλέτων"(Onto Triples Processing)**, δηλαδή επεξεργασία των τριπλέτων οι οποίες περιέχουν έννοιες από τα λεξιλόγια των RDF Schema [3] και OWL [1] (Ορισμός 9.1). Η διαδικασία εκτελείται αμέσως μετά την διαδικασία Προσδιορισμού τύπων μεταβλητών για κάθε Union-Free σχηματομορφή γράφου που περιέχεται στην κανονικοποιημένη σχηματομορφή γράφου. Η διαδικασία δέχεται σαν είσοδο μια Union-Free σχηματομορφή γράφου και σκοπός της είναι η σύνδεση(bind) των μεταβλητών με μονοπάτια του XML εγγράφου, που προκύπτουν από τις Οντο τριπλέτες. Αυτές οι συνδέσεις πρόκειται να χρησιμοποιηθούν ως αρχικές συνδέσεις (Initial bindings) στην διαδικασία Σύνδεσης Μεταβλητών (Κεφάλαιο 10).

Αρχικά αναλύεται ο τρόπος χειρισμού των ερωτήσεων που περιέχουν Οντο τριπλέτες, γίνεται ο διαχωρισμός των ερωτήσεων σε δυο κατηγορίες τις "Ερωτήσεις ως προς το Σχήμα και την Σημασιολογία της Οντολογίας" (υπό-ενότητα 9.2) και τις "Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας" (υπό-ενότητα 9.3), ανάλογα με την ύπαρξη των Οντο τριπλέτων.

Ορισμός 9.1 'Οντο τριπλέτες (Onto Triples) Ως 'Οντο τριπλέτες (Onto Triples) ονομάζονται οι τριπλέτες σχηματομορφών, οι οποίες αναφέρονται στο σχήμα ή στην σημασιολογία της οντολογίας(και όχι στα δεδομένα). Πιο συγκεκριμένα ονομάζονται οι τριπλέτες οι οποίες περιέχουν έννοιες από τα λεξιλόγια των RDF Schema [3] και OWL [1] .

Έστω η SPARQL ερώτηση με συνθήκη (Where Clause), την σχηματομορφή γράφου αποτελούμενη από την τριπλέτα σχηματομορφών ?s ?p ?o . Η εκτέλεση της ερώτησης σε μια οντολογία από ένα SPARQL Engine, θα έχει ως αποτελέσματα τις τριπλέτες από τα πιθανά RDF δεδομένα που περιέχει η οντολογία, καθώς και τις τριπλέτες που περιγράφουν το σχήμα και την σημασιολογία της οντολογία με χρήση του RDF και Owl λεξιλόγιο (vocabulary).

Στην παρούσα προσέγγιση, καθώς τα δεδομένα προέρχονται από XML έγραφα, ο συνδυασμός αυτών με τα δεδομένα από το σχήμα και την σημασιολογία της οντολογίας είναι στις περισσότερες περιπτώσεις αδύνατος. Επίσης οι ερωτήσεις για τις οποίες γίνεται η επεξεργασία αναφέρονται στα δεδομένα που περιγράφει η οντολογία και όχι στο σχήμα ή την σημασιολογία της οντολογίας, καθώς η οντολογία αποτελεί ένα ενδιάμεσο σχήμα (mediated schema).

Παρατήρηση 9.1 Για τους παραπάνω λόγους, τα αποτελέσματα των ερωτήσεων θα περιέχουν είτε δεδομένα από τις XML πηγές , είτε δεδομένα που αναφέρονται στο σχήμα και στην σημασιολογία της οντολογίας .

Όπως γίνεται αντιληπτό, οι Οντο τριπλέτες μπορούν να χρησιμοποιηθούν είτε για την ανάκτηση πληροφορίας για την δομή και την σημασιολογία της οντολογίας("Ερωτήσεις ως προς το Σχήμα και την Σημασιολογία της Οντολογίας")(βλέπε υπό-ενότητα 9.2), είτε για την αρχικοποίηση μεταβλητών που περιέχονται και σε άλλες τριπλέτες ("Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας")(βλέπε υπό-ενότητα 9.3). Στην πρώτη περίπτωση η σχηματομορφή γράφου της συνθήκη (where clause) θα πρέπει να περιέχει μόνο Οντο τριπλέτες.

9.2 Ερωτήσεις ως προς το Σχήμα και την Σημασιολογία της Οντολογίας

Σε αυτή την περίπτωση, η σχηματομορφή γράφου της ερώτησης αποτελείται μόνο από Όντο τριπλέτες. Η ερώτηση δεν μεταφράζεται, αλλά εκτελείται από ένα SPARQL engine και επιστρέφονται τα αποτελέσματα της εκτέλεσης.

Παρατήρηση 9.2 Ειδική περίπτωση Οντο τριπλέτας, είναι η τριπλέτα η οποία έχει ως κατηγόρημα την ιδιότητα rdf:type [1] (στην γλώσσα SPARQL μπορεί να οριστεί εναλλακτικά και ως a). Η ιδιότητα αυτή έχει ως πεδίο ορισμού rdf:Resource και ως πεδίο τιμών rdf:class. Η σημασιολογια της ιδιοτητας είναι ότι κάθε rdf:Resource της ιδιότητας, θα αποτελεί στιγμιότυπο της rdf:class .

Εάν στο σύνολο των Οντο τριπλέτων περιέχεται τριπλέτα με κατηγόρημα την ιδιότητα rdf:type και σαν αντικείμενο επώνυμη κλάση ή μεταβλητή που έχει αντιστοιχηθεί σε επώνυμη κλάση, τότε η ερώτηση είναι τύπου "Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας" καθώς το υποκείμενο τις τριπλέτας αντιστοιχεί σε XML δεδομένα. Ο τρόπος εκτελέσεις της αναλύεται παρακάτω.

Τα παρακάτω παραδείγματα αποτελούν ερωτήσεις προς το σχήμα και την σημασιολογία της οντολογίας.

Παράδειγμα 9.1

Έστω η ερώτηση "Επέστρεψε τις υπό-κλάσεις της κλάσης Person_Type" :

```
SELECT ?x
WHERE{ ?x  rdfs:subClassOf        Person_Type . }
```

Η παραπάνω ερώτηση αναφέρεται εξολοκλήρου στο σχήμα της οντολογίας και δεν σχετίζεται με τα XML δεδομένα, επομένως δεν θα μεταφραστεί σε XQuery αλλά εκτελείται σαν SPARQL.

[1] Θεωρούνται τα εξής Names Spaces:
rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs=http://www.w3.org/2000/01/rdf-schema#
owl=http://www.w3.org/2002/07/owl#

Παράδειγμα 9.2

Έστω η ερώτηση "Επέστρεψε τις υπό-κλάσεις της κλάσης Person_Type και τις ιδιότητες που έχουν ως πεδίο ορισμού τις υπό-κλάσεις αυτές " :

```
SELECT   ?x  ?p
WHERE { ?x rdfs:subClassOf        Person_Type .
        ?p    rdfs:domain              ?x .}
```

Η παραπάνω ερώτηση αναφέρεται εξολοκλήρου στο σχήμα της οντολογίας και δεν σχετίζεται με τα XML δεδομένα, επομένως δεν θα μεταφραστεί σε XQuery αλλά εκτελείται σαν SPARQL.

Παράδειγμα 9.3

Έστω η ερώτηση "Επέστρεψε τις υπό-κλάσεις της κλάσης Person_Type και τις ιδιότητες που έχουν ως πεδίο ορισμού τις υπό-κλάσεις αυτές και έχουν οριστεί ως FunctionalProperty " :

```
SELECT   ?x  ?p
WHERE{ ?x  rdfs:subClassOf        Person_Type .
       ?p  rdfs:domain            ?x .
       ?p   rdf:type        owl:FunctionalProperty .  }
```

Η παραπάνω ερώτηση αναφέρεται εξολοκλήρου στο σχήμα και την σημασιολογία της οντολογίας και δεν σχετίζεται με τα XML δεδομένα, επομένως δεν θα μεταφραστεί σε XQuery αλλά εκτελείται σαν SPARQL.

9.3 Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας

Σε αυτή την κατηγορία ερωτήσεων, οι Όντο τριπλέτες αφαιρούνται και εκτελούνται ανεξάρτητα από ένα SPARQL engine. Στην συνέχεια, τα αποτελέσματα της ερώτησης, τα οποία αποτελούντα από RDF όρους αντιστοιχούνται σε μονοπάτια με χρήση των αρχικών αντιστοιχήσεων. Αυτά τα μονοπάτια, θα αποτελέσουν την αρχικοποίηση των μεταβλητών που πιθανά εμφανίζονται και εκτός των Οντο τριπλέτων στην διαδικασία "Αντιστοίχηση Μεταβλητών".

Παρατήρηση 9.3 Η Οντο τριπλέτας με κατηγόρημα την ιδιότητα rdf:type δεν εκτελείται από το SPARQL engine, στην περίπτωση που το αντικείμενο είναι : μεταβλητή , επώνυμη κλάση (IRI) , rdf:Property , owl:ObjectProperty, owl:DatatypeProperty, καθώς σε αυτήν την περίπτωση η το υποκείμενο της τριπλέτα μπορεί να αντιστοιχεί άμεσα σε XML δεδομένα. Η διαχείριση της τριπλέτας αναλύεται παρακάτω.

Χειρισμός Οντο τριπλέτας με κατηγόρημα την ιδιότητα rdf:type

Όπως αναφέρεται και στην Παρατήρηση 9.3, υπάρχουν περιπτώσεις στις οποίες οι Οντο τριπλέτες με κατηγόρημα την ιδιότητα rdf:type δεν απαιτούν την εκτέλεση τους ώστε να προσδιοριστούν οι αντιστοιχήσεις των μεταβλητών. Οι περιπτώσεις είναι οι ακόλουθες :

a. ?x rdf:type I Το I είναι IRI το οποίο αναφέρεται σε μια επώνυμη κλάση.

Σε αυτήν την περίπτωση, η μεταβλητή x περιέχει τα στιγμιότυπα της κλάσης που αναφέρεται το I . Συνεπώς η μεταβλητή x αντιστοιχίζεται με τα μονοπάτια που έχουν αντιστοιχηθεί με την κλάση που αναφέρεται το I (μέσω τον αρχικών αντιστοιχήσεων).

Παράδειγμα 9.4

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα της κλάσης Person_Type " :

```
SELECT   ?x
WHERE { ?x    rdf:type        Person_Type . }
```

Από την υπό-ενότητα 5.5.1 έχει οριστεί η εξής αντιστοίχηση :

$\mathbf{X}_{\mathbf{Person\_Type}}$ = { /Persons/Person , /Persons/Employee }

Επομένως η μεταβλητή x μπορεί να αντιστοιχηθεί άμεσα με τα μονοπάτια που έχουν οριστεί για την κλάση Person_Type στις αντιστοιχήσεις. Άρα :

$\mathbf{X}_{\mathbf{x}}$ = { /Persons/Person , /Persons/Employee }

b. ?x rdf:type rdf:Property

Σε αυτήν την περίπτωση, η μεταβλητή x περιέχει τα στιγμιότυπα της κλάσης rdf:property κοινώς τα IRI's που αντιστοιχούν στις ιδιότητες (rdf:property). Οι κλάσεις owl:ObjectProperty και owl:DatatypeProperty είναι υπό-κλάσεις της κλάσης rdf:property. Επομένως η μεταβλητή x περιέχει και τα στιγμιότυπα των κλάσεων owl:ObjectProperty και

owl:DatatypeProperty. Η μεταβλητή x αντιστοιχίζεται με τα μονοπάτια όλων των ιδιοτήτων για τις οποίες έχουν οριστεί οι αντιστοιχήσεις.

Παράδειγμα 9.5

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα όλων rdf:Property (δηλαδή όλων των ιδιοτήτων)" :

```
SELECT   ?p
WHERE { ?p    rdf:type        rdf:Property. }
```

Η μεταβλητή p θα αντιστοιχηθεί με τα μονοπάτια όλων των ιδιοτήτων για τις οποίες έχουν οριστεί στις αντιστοιχήσεις υπό-ενότητα 5.5.1, επομένως :

$\mathbf{X_x}$ = { /Persons , /Persons/Person , /Persons/Employee , /Persons/Person/Address ,
/Persons/Employee/Address, /Persons/Person/Telephone ,
/Persons/Employee/Telephone, Persons/Person/Age , /Persons/Employee/Age,
/Persons/Person/FirstName, /Persons/Employee/FirstName,
/Persons/Person/LastName, /Persons/Employee/LastName,
/Persons/Person/NickName, /Persons/Employee/NickName,
/Persons/Employee/Salary, /Persons/Person/Address/@city,
/Persons/Employee/Address/@city, Persons/Person/Address/Street,
/Persons/Employee/Address/Street, /Persons/Person/Address/Road,
/Persons/Employee/Address/Road, /Persons/Person/Address/Road,
/Persons/Employee/Address/Road , /Persons/Person/Address/Number,
/Persons/Employee/Address/Number}

c. ?x rdf:type owl:ObjectProperty

Σε αυτήν την περίπτωση, η μεταβλητή x περιέχει τα στιγμιότυπα της κλάσης owl:ObjectProperty, κοινώς τα IRI's που αντιστοιχούν στις ιδιότητες αντικειμένων (owl:ObjectProperty). Επομένως η μεταβλητή x αντιστοιχίζεται με τα μονοπάτια όλων των ιδιοτήτων αντικειμένων για τις οποίες έχουν οριστεί οι αντιστοιχήσεις.

Παράδειγμα 9.6

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα όλων owl:ObjectProperty (δηλαδή όλων των ιδιοτήτων αντικειμένων)" :

```
SELECT   ?p
WHERE{ ?p    rdf:type        owl:ObjectProperty. }
```

Η μεταβλητή p θα αντιστοιχηθεί με τα μονοπάτια όλων των ιδιοτήτων αντικειμένων για τις οποίες έχουν οριστεί στις αντιστοιχήσεις υπό-ενότητα 5.5.1, επομένως :

$\mathbf{X_x}$ = { /Persons , /Persons/Person , /Persons/Employee , /Persons/Person/Address , /Persons/Employee/Address}

d. ?x rdf:type owl:DatatypeProperty

Σε αυτήν την περίπτωση, η μεταβλητή x περιέχει τα στιγμιότυπα της κλάσης owl:DatatypeProperty, κοινώς τα IRI's που αντιστοιχούν στις ιδιότητες τιμών δεδομένων (owl:DatatypeProperty). Επομένως η μεταβλητή x αντιστοιχίζετε με τα μονοπάτια όλων των ιδιοτήτων τιμών δεδομένων, για τις οποίες έχουν οριστεί οι αντιστοιχήσεις.

Παράδειγμα 9.7

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα όλων owl: DatatypeProperty (δηλαδή όλων των ιδιοτήτων τιμών δεδομένων)" :

SELECT ?p

WHERE { ?p rdf:type owl: DatatypeProperty. }

Η μεταβλητή p θα αντιστοιχηθεί με τα μονοπάτια όλων των ιδιοτήτων τιμών δεδομένων για τις οποίες έχουν οριστεί στις αντιστοιχήσεις υπό-ενότητα 5.5.1, επομένως :

$\mathbf{X_x}$ = { /Persons/Person/Telephone , /Persons/Employee/Telephone, Persons/Person/Age , /Persons/Employee/Age, /Persons/Person/FirstName, /Persons/Employee/FirstName, /Persons/Person/LastName, /Persons/Employee/LastName, /Persons/Person/NickName, /Persons/Employee/NickName, /Persons/Employee/Salary, /Persons/Person/Address/@city, /Persons/Employee/Address/@city, Persons/Person/Address/Street, /Persons/Employee/Address/Street, /Persons/Person/Address/Road, /Persons/Employee/Address/Road, /Persons/Person/Address/Road, /Persons/Employee/Address/Road , /Persons/Person/Address/Number, /Persons/Employee/Address/Number}

e. ?x rdf:type ?o

- Αν η μεταβλητή ο έχει μια από τις τιμές των αντικείμενων, των παραπάνω τριπλέτων, τότε η τριπλέτα αναγάγεται στην αντίστοιχη περίπτωση .
- Αν η μεταβλητή ο δεν περιέχεται σε κάποια άλλη Οντο τριπλέτα (κοινώς δεν προσδιορίζεται), τότε η μεταβλητή x, αντιστοιχίζεται με τα μονοπάτια όλων των ιδιοτήτων και όλων των κλάσεων για τις οποίες έχουν οριστεί αντιστοιχήσεις. Στην συνέχεια, ανάλογα

με την θέση της μεταβλητής x στις υπόλοιπες τριπλέτες (μη-Οντο τριπλέτες), αντιστοιχίζεται είτε με τα μονοπάτια όλων των κλάσεων, στην περίπτωση που βρίσκεται σε θέση υποκειμένου ή αντικειμένου ,είτε με τα μονοπάτια όλων των ιδιοτήτων αν βρίσκετε σε θέση κατηγορήματος .

Παράδειγμα 9.8

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα όλων owl:ObjectProperty (δηλαδή όλων των ιδιοτήτων αντικειμένων)" :

SELECT ?p

WHERE { Persons rdf:type ?o

?p rdf:type ?o. }

Εκτελείται το SPARQL ερώτημα αποτελούμενο από την Οντο τριπλέτα :

Persons rdf:type ?o

Τα αποτελέσματα της εκτέλεσης της SPARQL ερώτησης είναι :

?o -> [owl:ObjectProperty]

Επομένως η τριπλέτα ?p rdf:type ?o αναγάγεται στην τριπλέτα ?p rdf:type owl:ObjectProperty δηλαδή στην περίπτωση c. Άρα :

Η μεταβλητή p θα αντιστοιχηθεί με τα μονοπάτια όλων των ιδιοτήτων αντικειμένων για τις οποίες έχουν οριστεί στις αντιστοιχήσεις υπό-ενότητα 5.5.1, επομένως :

$\mathbf{X_x}$ = { /Persons , /Persons/Person , /Persons/Employee , /Persons/Person/Address , /Persons/Employee/Address}

Ο λόγος για τον οποίο γίνεται ο υπολογισμός των αντιστοιχήσεων των μεταβλητών στις Οντο τριπλέτες, είναι για να χρησιμοποιηθούν ως αρχικοποιήσεις στις μεταβλητές που εμφανίζονται και εκτός των Οντο τριπλέτων.

Οι μεταβλητές των Οντο τριπλέτων που μπορούν να διαμοιράζονται και σε άλλες (μη-Οντο) τριπλέτες είναι :

a. Το υποκείμενο των τριπλέτων που έχουν ως κατηγόρημα την ιδιότητα rdf:type .

b. Το υποκείμενο και το αντικείμενο των τριπλέτων που έχουν ως κατηγόρημα την ιδιότητα rdfs:subClassOf .

c. Το υποκείμενο των τριπλέτων που έχουν ως κατηγόρημα την ιδιότητα rdfs:domain

d. Το υποκείμενο των τριπλέτων που έχουν ως κατηγόρημα την ιδιότητα rdfs:range

Παράδειγμα 9.9

Έστω η ερώτηση "Επέστρεψε τα στιγμιότυπα όλων υπό-κλάσεων της κλάσης Person_Type" :

```
SELECT  ?x
WHERE  { ?e       rdfs:subClassOf      Person_Type .
         ?x       rdf:type             ?e .
         ?p       rdfs:domain          ?e .
         ?x       ?p                   ?o . }
```

Εκτελείται το SPARQL ερώτημα αποτελούμενο απο τις Οντο τριπλέτες :

```
?e    rdfs:subClassOf      Person_Type .
?p    rdfs:domain          ?e .
```

Τα αποτελέσματα της εκτέλεσης της SPARQL ερώτησης είναι :

?e -> [Employee_Type]

?p -> [Telephone , Age , FirstName , NickName , LastName , Salary ,City ,Street ,Road ,Number , Address]

Στην συνέχεια συμφώνα με τα παραπάνω από την τριπλέτα ?x rdf:type ?e η μεταβλητή x αντιστοιχίζεται με τα μονοπάτια της κλάσης Employee_Type.

Επίσης η μεταβλητή p, αντιστοιχίζετε με τα μονοπάτια των : Telephone , Age , FirstName , NickName , LastName , Salary ,City ,Street ,Road ,Number , Address.

Άρα, προκύπτουν οι εξής αντιστοιχήσεις :

$\mathbf{X_x}$ = { / Persons/Employee }

$\mathbf{X_p}$={/Persons/Person/Telephone,/Persons/Employee/Telephone,/Persons/Person/Age, /Persons/Employee/Age,/Persons/Person/FirstName,/Persons/Employee/FirstName, /Persons/Person/LastName,/Persons/Employee/LastName,/Persons/Person/NickName, /Persons/Employee/NickName, /Persons/Employee/Salary }

Οι αντιστοιχίσεις αυτές ,θα αποτελέσουν τις αρχικοποιήσεις των μεταβλητών x και p στην διαδικασία της ανάθεσης μεταβλητών .

9.4 Όντο Τριπλέτες και Αντιστοιχήσεις

Όπως γίνεται αντιληπτό και από τα παραπάνω, στην παρούσα προσέγγιση μόνο για την Οντο τριπλέτα με κατηγόρημα την ιδιότητα rdf:type είναι δυνατό να οριστούν "άμεσα" οι αντιστοιχίες (μέσω των αντιστοιχήσεων), χωρίς να είναι αναγκαία η εκτέλεση SPARQL ερώτησης. Για τις υπόλοιπες Οντο τριπλέτες απαιτείται η εκτέλεση SPARQL ερώτησης και στην συνέχεια μέσω των αντιστοιχήσεων τα αποτέλεσμα της ερώτησης να αντιστοιχηθούν σε μονοπάτια. Αυτό συμβαίνει, καθώς έχει επιλεχθεί (για τους λόγους που αναλύονται παρακάτω) να "αποθηκεύονται" μόνο αντιστοιχήσεις μεταξύ στοιχείων της οντολογίας και μονοπατιών και όχι άλλες πληροφορίες όπως ιεραρχίες κλάσεων και ιδιοτήτων ή άλλες σημασιολογικές έννοιες της οντολογίας.

Μια άλλη προσέγγιση θα ήταν στις αντιστοιχήσεις να συμπεριληφθούν οι "βασικές" πληροφορίες για το σχήμα της οντολογίας, όπως οι ιεραρχίες και συσχετίσεις κλάσεων και ιδιοτήτων της οντολογίας. Όμως, και σε αυτήν την περίπτωση υπάρχουν Οντο τριπλέτες για τις οποίες δεν είναι δυνατό να οριστούν "άμεσα" οι αντιστοιχίες. Ένα παράδειγμα είναι οι τριπλέτες :

?x rdfs:subClassOf ?y .

?p rdfs:range ?x .

?p rdf:type owl:FunctionalProperty .

Όπως παρατηρείται για την τρίτη τριπλέτα, η πληροφορία των αντιστοιχήσεων δεν θα επαρκεί καθώς δεν "αποθηκεύονται" σημασιολογικές πληροφορίες όπως owl:FunctionalProperty και απαιτείται η εκτέλεση ερώτησης προς την οντολογία.

Όμως, ακόμα και στην περίπτωση που επαρκούν οι πληροφορίες από της "υποθηκευμένες" αντιστοιχήσεις, ο χρόνος που απαιτείται για την ανάκτηση ,επεξεργασία και εξαγωγή συμπερασμάτων δεν θα διαφέρει σημαντικά από τον χρόνο εκτέλεσης μια SPARQL ερώτησης από ένα SPARQL engine.

Για τους παραπάνω λόγους, επιλέχτηκε στις αντιστοιχήσεις που αποθηκεύονται να περιλαμβάνει μόνο αντιστοιχήσεις και όχι παραπάνω πληροφορίες για το σχήμα και την σημασιολογία της οντολογίας. Η μελέτη αποδοτικότερης διαχείρισης των Όντο τριπλέτων αποτελεί μια από τις μελλοντικές επεκτάσεις της παρούσας προσέγγισης.

9.5 Επίλογος

Στο κεφάλαιο αυτό έγινε η περιγραφή της διαδικασία "Επεξεργασία Οντο τριπλέτων", δηλαδή επεξεργασία των τριπλέτων οι οποίες περιέχουν έννοιες από τα λεξιλόγια των RDF Schema και OWL. Έγινε ο διαχωρισμός των ερωτήσεων σε δυο κατηγορίες τις "Ερωτήσεις ως προς το Σχήμα και την Σημασιολογία της Οντολογίας" και τις "Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας", ανάλογα με την ύπαρξη των Οντο τριπλέτων.

Στην πρώτη κατηγορία ερωτήσεων("Ερωτήσεις ως προς το Σχήμα και την Σημασιολογία της Οντολογίας"), η σχηματομορφή γράφου της ερώτησης αποτελείται μόνο από Όντο τριπλέτες. Η ερώτηση δεν μεταφράζεται, αλλά εκτελείται από ένα SPARQL engine και επιστρέφονται τα αποτελέσματα της εκτέλεσης.

Στην δεύτερη κατηγορία ερωτήσεων ("Ερωτήσεις που συνδυάζουν πληροφορίες από το Σχήμα και την Σημασιολογία της Οντολογίας"), οι Όντο τριπλέτες αφαιρούνται και εκτελούνται ανεξάρτητα από ένα SPARQL engine. Στην συνέχεια, τα αποτελέσματα της ερώτησης, τα οποία αποτελούντα από RDF όρους αντιστοιχούνται σε μονοπάτια με χρήση των αρχικών αντιστοιχήσεων. Αυτές οι συνδέσεις πρόκειται να χρησιμοποιηθούν ως αρχικές συνδέσεις (Initial bindings) στην διαδικασία Σύνδεσης Μεταβλητών (Κεφάλαιο 10).

Στο επόμενο κεφάλαιο(Κεφάλαιο 10) γίνεται η περιγραφή της διαδικασίας " Σύνδεσης Μεταβλητών" (Variables Binding), η οποία εφαρμόζεται σε βασικές σχηματομορφές γράφων ώστε να επιτευχθεί η σύνδεση των μεταβλητών που περιέχονται σε αυτές με μονοπάτια του XML εγγράφου.

10 Σύνδεση Μεταβλητών

10.1 Εισαγωγή

Στο παρόν κεφάλαιο περιγράφεται η διαδικασία **"Σύνδεση Μεταβλητών" (Variables Binding)**, η οποία εφαρμόζεται σε βασικές σχηματομορφές γράφων (Basic graph pattern-BGP) , δηλαδή σε ακολουθίες από τριπλέτες σχηματομορφών, ώστε να επιτευχθεί η σύνδεση των μεταβλητών που περιέχονται στο BGP με μονοπάτια του XML εγγράφου. Η σύνδεση των μεταβλητών της SPARQL ερώτησης με μονοπάτια του XML εγγράφου είναι απαραίτητο για να μπορέσει να πραγματοποιηθεί η μετάφραση Βασικών Σχηματομορφών Γράφων σε XQuery (Κεφάλαιο 11). Καθώς με αυτόν τον τρόπο συνδέονται οι SPARQL μεταβλητές με τα XML δεδομένα μέσω των μονοπατιών, στην συνέχεια με την χρήση των μονοπατιών και XQuery εκφράσεων δίνεται η δυνατότητα διάσχισης και ανάκτησης της αντίστοιχη(ανάλογα με την μεταβλητή) XML πληροφορίας. Όπως γίνεται αντιληπτό

και στην συνέχεια του κεφαλαίου η σύνδεση των μεταβλητών με τα "επιθυμητά" μονοπάτια δεν είναι μια απλή διαδικασία.

Η διαδικασία σύνδεσης μεταβλητών δέχεται ως είσοδο και χρησιμοποιεί σαν αρχικές συνδέσεις, τις συνδέσεις που έχουν προκύψει από την διαδικασία **"Επεξεργασία Οντο τριπλέτων"** (Κεφάλαιο 9). Τέλος οι συνδέσεις που προσδιορίζονται από την διαδικασία θα χρησιμοποιηθούν από τον αλγόριθμο μετάφρασης Βασικών Σχηματομορφών Γράφων σε XQuery (Κεφάλαιο 11).

Στην συνέχεια της παρούσας ενότητας αναλύεται η αντιστοιχία μονοπατιών με (μια) τριπλέτα σχηματομορφών (υπό-ενότητα 10.2) , στην οποία περιγράφονται οι σχέσεις που ισχύουν μεταξύ των μονοπατιών που αντιστοιχούν στα μέρη (υποκείμενο-κατηγόρημα-αντικείμενο) της τριπλέτας. Οι σχέσεις αυτές βοηθάνε στην κατανόηση της ανάπτυξης του αλγόριθμου Σύνδεσης Μεταβλητών (υπό-ενότητα 10.4). Επίσης αναλύεται η αντιστοιχία μονοπατιών με (πολλές) τριπλέτες σχηματομορφών (υπό-ενότητα 10.3) , στην οποία περιγράφονται οι σχέσεις που ισχύουν μεταξύ των μονοπατιών που αντιστοιχούν στα μέρη (υποκείμενο-κατηγόρημα-αντικείμενο) των τριπλετών και βοηθάνε στην κατανόηση της ανάπτυξης του αλγόριθμου Σύνδεσης Μεταβλητών. Τέλος περιγράφεται ο αλγόριθμος για τον προσδιορισμό των συνδέσεων(υπό-ενότητα 10.4) καθώς και τις Αλληλεξαρτήσεις Μεταξύ των Συνόλων Μονοπατιών στις Τριπλέτες Σχηματομορφών (υπό-ενότητα 10.5)

Το RDF μοντέλο δεδομένων (RDF Data Model) όπως αναφέρθηκε είναι ένα Graph–Based μοντέλο. Πιο αναλυτικά ένας RDF γράφος από τον Ορισμό 2.3 είναι μια ακολουθία από τριπλέτες σχηματομορφών, που μπορεί να θεωρηθεί ως ένα γράφημα αποτελούμενο από κόμβους και κατευθυνόμενες ακμές στο οποίο μια τριπλέτα σχηματομορφής αναπαριστά ένα κόμβο-ακμή-κόμβο σύνδεσμο. Η ακμή έχει κατεύθυνση πάντα από τον κόμβο που αναπαριστά το υποκείμενο προς τον κόμβο που αναπαριστά το αντικείμενο. Επίσης τόσο οι κόμβοι όσο και οι ακμές έχουν τίτλους (Labels). Αντιθέτως το XML μοντέλο δεδομένων (XML Data Model) είναι ένα δενδρικό μοντέλο που περιέχει τίτλους μόνο στους κόμβους.

Η διαδικασία της Σύνδεσης Μεταβλητών (Variables Binding) εφαρμόζετε σε ένα BGP(Basic Graph Pattern) με σκοπό να προσδιοριστούν οι αντιστοιχίσεις μεταξύ μεταβλητών και μονοπατιών στα XML δεδομένα, ώστε να είναι εφικτή η αποτίμηση του BGP σε XML δεδομένα και όχι σε RDF γράφους όπως ορίζεται στον Ορισμό 2.8.

Όπως είναι γνωστό(Ορισμός 2.7), οι BGP αποτελούνται από τριπλέτες σχηματομορφών και φίλτρα. Κατά τη διαδικασία της σύνδεσης μεταβλητών ωστόσο, δεν συμμετέχουν τα φίλτρα καθώς δεν περιέχουν πληροφορία που να μπορεί να χρησιμοποιηθεί, όπως επίσης και τα Onto Triples(βλέπε κεφάλαιο 10) καθώς η επεξεργασία τους έχει γίνει σε προηγούμενο στάδιο. Το αποτέλεσμα της επεξεργασίας των Onto Triples χρησιμοποιείται σαν αρχικοποίηση στην Σύνδεση Μεταβλητών (βλέπε Παράδειγμα 10.5). Οι τριπλέτες σχηματομορφών όπως προκύπτει και από την Παρατήρηση 10.1, αποτελούνται από

μεταβλητές, σταθερές και IRI ιδιότητες (properties). Για τα IRI των ιδιοτήτων έχουν οριστεί αρχικά οι αντιστοιχήσεις. Αυτές οι αντιστοιχήσεις όπως και η αρχικοποίηση από την επεξεργασία των Onto τριπλετών θα χρησιμοποιηθούν για τον προσδιορισμό των αντιστοιχήσεων μεταξύ μεταβλητών και μονοπατιών των XML δεδομένων. Με τον προσδιορισμό αυτών των αντιστοιχήσεων, δίνεται στην συνέχεια η δυνατότητα αποτίμησης των BGP στα XML δεδομένα με την χρήση XQuery.

Παρατήρηση 10.1 Από Ορισμός 2.2 και Ορισμός 2.6 προκύπτει ότι **Τριπλέτα Σχηματομορφής** ονομάζετε η τριπλέτα για την οποία ισχύει **(s,p,o) ϵ (I** U **B** U **V)** x **(I** U **V)** x **(I** U **B** U **L** U **V) .** Οι οντολογίες που αναφέρονται οι ερωτήσεις δεν περιέχουν στιγμιότυπα των κλάσεων ,ενώ και στην περίπτωση που περιέχουν αγνοούνται καθώς οι απαντήσεις θα περιέχουν XML δεδομένα .Για τον λόγο αυτό από τον ορισμό της Τριπλέτας Σχηματομορφής μπορούν να αφαιρεθούν τα IRIs από τις θέσεις του υποκειμένου και του αντικειμένου. Ο νέος ορισμός προκύπτει ως **(s,p,o) ϵ (B** U **V)** x **(I** U **V)** x **(B** U **L** U **V)** . Στην συνέχεια, θα γίνεται αναφορά στην Τριπλέτα Σχηματομορφής με βάση αυτόν τον ορισμό (Ο παραπάνω ορισμός δεν ισχύει για την περίπτωση των Onto τριπλετών, βλέπε Κεφάλαιο 9).

10.2 Αντιστοιχία Μονοπατιών με Τριπλέτα Σχηματομορφών

Στην παρούσας υπό-ενότητας αναλύεται η αντιστοιχία μονοπατιών με (μια) τριπλέτα σχηματομορφών και περιγράφονται οι σχέσεις που ισχύουν μεταξύ των μονοπατιών που αντιστοιχούν στα μέρη (υποκείμενο-κατηγόρημα-αντικείμενο) της τριπλέτας. Οι σχέσεις αυτές βοηθάνε στην κατανόηση της ανάπτυξης του αλγόριθμου Σύνδεσης Μεταβλητών (υπό-ενότητα 10.4).

Η τριπλέτα σχηματομορφών αναπαριστά ένα τμήμα (κόμβος-ακμή-κόμβος) του RDF γράφου. Επομένως θα αντιστοιχίζεται με ένα τμήμα της δενδρικής αναπαράστασης των XML εγγράφων. Κοινώς, μια τριπλέτα αντιστοιχεί σε ένα μονοπάτι, κατ' επέκταση και κάθε μέρος της τριπλέτας (Υποκείμενο-Κατηγόρημα-Αντικείμενο) αντιστοιχεί σε ένα τμήμα του.

- **Σχέσεις Συνόλων Μονοπατιών για Τριπλέτα Σχηματομορφών**

  Έστω η τριπλέτα σχηματομορφών **S P O** , για την οποία ισχύουν οι εξής σχέσεις :

  Έστω $\mathbf{X_s}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Υποκείμενο (**S**).

  Έστω $\mathbf{X_p}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Κατηγόρημα (**P**).

  Έστω $\mathbf{X_o}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Αντικείμενο (**O**).

a. $X_{pD} = X_p^P$

Το Συμπέρασμα 5.4 αναφέρει : $\forall \text{Property pr} => X_{prR} = X_{pr}$ και $X_{prD} = X_{pr}^P = X_{prR}^P$, εφόσον και το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα , θα ισχύει η ίδια ισότητα με το συμπέρασμα 5.4 . Επομένως το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος **X_{pD}** είναι ίσο με το σύνολο μονοπατιών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής **^{P} X_p^P**.

b. $X_{pD} = X_{pR}^P$

Το Συμπέρασμα 5.4 αναφέρει :

$\forall \text{Property pr} => X_{prR} = X_{pr}$ και $X_{prD} = X_{pr}^P = X_{prR}^P$, εφόσον και το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα , θα ισχύει η ίδια ισότητα με το συμπέρασμα 5.4 . Επομένως το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος **X_{pD}** είναι ίσο με το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο τιμών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής **^{P} X_{pR}^P**

c. $X_S = X_{pD}$

Το Συμπέρασμα 5.2 αναφέρει :

$$\forall class => X_C \subseteq X_{pr}^P \text{ ή } X_C \supseteq X_{pr}^P \text{ , όπου pr ειναι ιδιοτητα με domain την class ,}$$

εφόσον το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα και το υποκείμενο S σε κάποια κλάση, η οποία αποτελεί πεδίο ορισμού της ιδιότητας P. Τότε το σύνολο μονοπατιών **X_S** που αντιστοιχεί στο υποκείμενο , βασισμένο στο Συμπέρασμα 5.2 θα είναι είτε υπό-σύνολο είτε υπέρ-σύνολο του συνόλου μονοπατιών **X_{pD}** που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος.

Όμως καθώς μελετάται η περίπτωση μόνο μιας τριπλέτας θα ισχύει η ισότητα, αυτό συμβαίνει καθώς το μόνο σταθερό (όχι μεταβλητή) μέρος της τριπλέτας μπορεί να υπάρξει το κατηγόρημα (και το αντικείμενο , στην περίπτωση που είναι σταθερά, όμως η σταθερά δεν προσδιορίζει κάποιες αντιστοιχήσεις) από το οποίο προσδιορίζονται οι τιμές στα άλλα μέρη της τριπλέτας.

d. $X_{pR} = X_O$

Το Συμπέρασμα 5.3 αναφέρει :

$\forall class => X_C \subseteq X_{pr}$ ή $X_C \supseteq X_{pr}$, όπου pr ειναι ιδιοτητα με range την class ,

εφόσον το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα και το υποκείμενο S σε κάποια κλάση, η οποία αποτελεί πεδίο τιμών της ιδιότητας P. Τότε το σύνολο μονοπατιών $\mathbf{X_o}$ που αντιστοιχεί στο αντικείμενο, βασισμένο στο Συμπέρασμα 5.3 θα είναι είτε υπό-σύνολο είτε υπέρ-σύνολο του συνόλου μονοπατιών $\mathbf{X_{pR}}$ που αντιστοιχεί στο πεδίο τιμών του κατηγορήματος.

Όμως καθώς μελετάται η περίπτωση μόνο μιας τριπλέτας θα ισχύει η ισότητα, αυτό συμβαίνει καθώς το μόνο σταθερό (όχι μεταβλητή) μέρος της τριπλέτας μπορεί να υπάρξει το κατηγόρημα (και το αντικείμενο , στην περίπτωση που είναι σταθερά, όμως η σταθερά δεν προσδιορίζει κάποιες αντιστοιχήσεις) από το οποίο προσδιορίζονται οι τιμές στα άλλα μέρη της τριπλέτας.

Εξισώνοντας τα κοινά μέρη των παραπάνω ισοτήτων, προκύπτει η ισότητα :

$$\mathbf{X_S = X_{pD} = X_{pR}^P = X_p^P = X_o^P} \qquad \textbf{Σχέση 10.1}$$

Από την Σχέση 10.1 ισχύει: Το σύνολο μονοπατιών $\mathbf{X_S}$ που αντιστοιχεί στο υποκείμενο είναι ίσο με το σύνολο μονοπατιών $\mathbf{X_{pD}}$ που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος, είναι ίσο με το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο τιμών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής $^{\mathbf{P}}\ \mathbf{X_{pR}}^{\mathbf{P}}$, είναι ίσο με το σύνολο μονοπατιών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής $^{\mathbf{P}}\ \mathbf{X_p}^{\mathbf{P}}$ και τέλος είναι ίσο με το σύνολο μονοπατιών που αντιστοιχεί στο αντικείμενο εφόσον εφαρμοστεί ο τελεστής $^{\mathbf{P}}\ \mathbf{X_o}^{\mathbf{P}}$.

Στην συνέχεια παρατίθεται μια σειρά από παραδείγματα για την κατανόηση και την απόδειξη της ισχύος της Σχέσης 10.1

Σημείωση : *Στην συνέχεια του κειμένου, για λόγους απλοποίησης παραλείπετε η πρόθεση (prefix) του Namespace της οντολογία που απευθύνονται οι ερωτήσεις.*

Παράδειγμα 10. 1

Έστω η ερώτηση : "Επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/τα μικρό/α τους όνομα/τα " (η οποία αποτελείται από μια τριπλέτα σχηματομορφής)

SELECT ?x ,?n
WHERE { ?x FirstName ?n }

Η τριπλέτα :

?x FirstName ?n

Από την υπό-ενότητα 5.5.1 έχει οριστεί η εξής αντιστοίχηση :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

Άρα, με βάση τις παραπάνω ισότητες c και d προκύπτει για τα **?x** και **?n** :

?x = { /Persons/Person, /Persons/Employee }

?n = { /Persons/Person/FirstName , /Persons/Employee/FirstName }

Όπως παρατηρείται το ?χ περιέχει τα επιθυμητά μονοπάτια για τα άτομα (και εργαζόμενους), το ίδιο και το ?n για τα μικρά ονόματα.

10.3 Αντιστοιχία Μονοπατιών με Τριπλέτες Σχηματομορφών

Στην παρούσας υπό-ενότητας αναλύεται η αντιστοιχία μονοπατιών με (πολλές) τριπλέτες σχηματομορφών και περιγράφονται οι σχέσεις που ισχύουν μεταξύ των μονοπατιών που αντιστοιχούν στα μέρη (υποκείμενο-κατηγόρημα-αντικείμενο) της τριπλέτας. Οι σχέσεις αυτές βοηθάνε στην κατανόηση της ανάπτυξης του αλγόριθμου Σύνδεσης Μεταβλητών (υπό-ενότητα 10.4).

Στην περίπτωση ύπαρξης περισσοτέρων από μιας τριπλέτας σχηματομορφών οι παραπάνω συνθήκες αλλάζουν λόγω της ύπαρξης διαμοιραζομένων μεταβλητών (Shared Variables) .

Ορισμός 10.1 : **Διαμοιραζόμενες Μεταβλητές (Shared Variables)** Μια μεταβλητή ονομάζεται Διαμοιραζόμενη Μεταβλητή (Shared Variable) όταν εμφανίζεται σε παραπάνω από μια τριπλέτα σχηματομορφών, χωρίς να είναι απαραίτητη η εμφάνιση της στην ιδία θέση σε κάθε τριπλέτα.

Λόγω των Διαμοιραζόμενων μεταβλητών οι υπολογισμοί των αντιστοιχήσεων γίνονται πιο πολύπλοκοι, μια μεταβλητή μπορεί να εμφανίζεται σε παραπάνω από μια τριπλέτες με αποτέλεσμα για την συγκεκριμένη μεταβλητή να υπολογίζονται διαφορετικές αντιστοιχήσεις από την κάθε τριπλέτα που εμφανίζεται.

- **Σχέσεις Συνόλων Μονοπατιών για Τριπλέτες Σχηματομορφών**

Έστω ένα σύνολο από ν τριπλέτες σχηματομορφών $\mathbf{S_i\ P_i\ O_i}$ για $1 \le i \le v$, για κάθε τριπλέτα σχηματομορφών $\mathbf{S_i\ P_i\ O_i}$ ισχύουν οι εξής σχέσεις :

Έστω $\mathbf{X_s}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Υποκείμενο ($\mathbf{S_i}$).

Έστω $\mathbf{X_p}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Κατηγόρημα ($\mathbf{P_i}$).

Έστω $\mathbf{X_o}$ το σύνολο των μονοπατιών που αντιστοιχεί στο Αντικείμενο ($\mathbf{O_i}$).

a. $\mathbf{X_{pD} = X_p^P}$

Το Συμπέρασμα 5.4 αναφέρει

: $\forall \text{Property pr} \Rightarrow X_{prR} = X_{pr}$ και $X_{prD} = X_{pr}^P = X_{prR}^P$, εφόσον και το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα, θα ισχύει η ίδια ισότητα με το συμπέρασμα 5.4 . Επομένως το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος $\mathbf{X_{pD}}$ είναι ίσο με το σύνολο μονοπατιών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής $\mathbf{^P\ X_p^P}$.

b. $\mathbf{X_{pD} = X_{pR}^P}$

Το Συμπέρασμα 5.4 αναφέρει

: $\forall \text{Property pr} \Rightarrow X_{prR} = X_{pr}$ και $X_{prD} = X_{pr}^P = X_{prR}^P$, εφόσον και το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα, θα ισχύει η ίδια ισότητα με το συμπέρασμα 5.4 . Επομένως το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος $\mathbf{X_{pD}}$ είναι ίσο με το σύνολο μονοπατιών που αντιστοιχεί στο πεδίο τιμών του κατηγορήματος εφόσον εφαρμοστεί ο τελεστής $\mathbf{^P\ X_{pR}^P}$

c. $\mathbf{X_s \subseteq X_{pD}}$ ή $\mathbf{X_s \supseteq X_{pD}}$

Το Συμπέρασμα 5.2 αναφέρει

: $\forall class \Rightarrow X_C \subseteq X_{pr}^P$ ή $X_C \supseteq X_{pr}^P$, όπου pr ειναι ιδιοτητα με domain την class, εφόσον το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα και το υποκείμενο S σε κάποια κλάση, η οποία αποτελεί πεδίο ορισμού της ιδιότητας P. Τότε το σύνολο μονοπατιών $\mathbf{X_s}$ που αντιστοιχεί στο υποκείμενο , βασισμένο στο Συμπέρασμα 5.2 θα είναι είτε υπό-

σύνολο είτε υπέρ-σύνολο του συνόλου μονοπατιών $\mathbf{X_{pD}}$ που αντιστοιχεί στο πεδίο ορισμού του κατηγορήματος.

d. $\mathbf{X_{pR} \subseteq X_o}$ ή $\mathbf{X_{pR} \supseteq X_o}$

Το Συμπέρασμα 5.3 αναφέρει

: $\forall class => X_c \subseteq X_{pr}$ ή $X_c \supseteq X_{pr}$, όπου pr ειναι ιδιοτητα με range την class , εφόσον το κατηγόρημα P αντιστοιχεί σε κάποια ιδιότητα και το υποκείμενο S σε κάποια κλάση, η οποία αποτελεί πεδίο τιμών της ιδιότητας P. Τότε το σύνολο μονοπατιών $\mathbf{X_o}$ που αντιστοιχεί στο αντικείμενο, βασισμένο στο Συμπέρασμα 5.3 θα είναι είτε υπό-σύνολο είτε υπέρ-σύνολο του συνόλου μονοπατιών $\mathbf{X_{pR}}$ που αντιστοιχεί στο πεδίο τιμών του κατηγορήματος.

10.4 Αλγόριθμος Σύνδεσης Μεταβλητών (Variables Binding Algorithm)

Όπως φαίνεται από τις σχέσεις c και d της προηγούμενη υπό-ενότητας, μεταξύ των συνόλων μονοπατιών του υποκειμένου του αντικειμένου και του κατηγορήματος ισχύουν σχέσεις υπό-συνόλων ή υπέρ-συνόλων, κάτι το οποίο δεν είναι το επιθυμητό. Ο αλγόριθμος σύνδεσης μεταβλητών προσπαθεί να προσδιορίσει τις συνδέσεις των μεταβλητών για τις οποίες ισχύει η Σχέση 10.1 σε όλο το σύνολο των τριπλετών.

Για την επίτευξη της ισότητας ανάμεσα σε σύνολα μονοπατιών τόσο των διαμοιραζομένων μεταβλητών όσο και ανάμεσα στα μέρη των τριπλετών, γίνεται χρήση του τελεστή της τομής ($\cap$) και του τελεστή αριστερού παιδιού($\oplus$) στα σύνολα που πρέπει να επιτευχθεί η Σχέση 10.1 . Ως αποτέλεσμα λαμβάνεται το μεγαλύτερο δυνατό υποσύνολο για το οποίο ισχύει η ισότητα, αυτό επιτυγχάνεται με τον Αλγόριθμος Σύνδεσης Μεταβλητών(Αλγόριθμος 10.1) και συγκεκριμένα με την χρήση των κανόνων της υπό-ενότητας 10.4.1 (βλέπε Παραδείγματα 10.1 - 10.6).

Η χρήση του τελεστή της τομής και του τελεστή $\oplus$ για την επίτευξη ισότητας για τις διαμοιραζόμενες μεταβλητές ισχύει τόσο για τις μεταβλητές Τύπου Στιγμιότυπου Κλάσης – CIVT (Ορισμός 8.3) όσο και για τις μεταβλητές ιδιοτήτων (DTPVT , OPVT, UPVT). Δεν ισχύει το ίδιο για τις μεταβλητές Τύπου Σταθεράς - LVT, καθώς ο υπολογισμός του ελάχιστου συνόλου μονοπατιών με τους τελεστές, δεν είναι ο επιθυμητός όπως φαίνεται και στο παράδειγμα 10.2. Αυτό συμβαίνει καθώς η ισότητα μεταξύ σταθερών, όπως είναι

λογικό μπορεί να επιτευχθεί ανεξάρτητα από τα μονοπάτια από τα οποία εμφανίζονται οι σταθερές. (όπως πχ ισότητα μεταξύ σταθερών μπορεί να ισχύει μεταξύ κόμβων με μονοπάτια /Persons/Person/First_Name και /Persons/Person/Nick_Name) Επομένως τα μονοπάτια που αντιστοιχούν σε μεταβλητές τύπου LVT δεν υπολογίζονται από τον αλγόριθμο και θα προσδιοριστούν στην συνέχεια.

Ο παρακάτω αλγόριθμος αναπτύχθηκε με βάση τις παραπάνω παρατηρήσεις και συνθήκες για τον υπολογισμό των αντιστοιχήσεων των μεταβλητών ενός BGP.

```
VariablesBinding ( BGP , initialBindings , variablesTypes ){
    bindings = initialBindings
    Repeat
        oldBindings = bindings
        For each triple in BGP
            bindings = DetermineBinding(triple, oldBindings, variablesTypes)
        End For
    Until(oldBindings = Bindings)
}
```

Αλγόριθμος 10.1 Αλγόριθμος Σύνδεσης Μεταβλητών

Ο αλγόριθμος δέχεται σαν παράμετρο μια Βασική Σχηματομορφή Γράφων (*BGP*) (δηλαδή ένα σύνολο από τριπλέτες), ένα σύνολο από συνδέσεις μεταβλητών *initialBindings* και τους τύπους των μεταβλητών *variablesTypes* που προσδιοριστήκαν από την διαδικασία "Προσδιορισμός Τύπων Μεταβλητών"(Κεφάλαιο 8). Οι συνδέσεις *initialBindings* προέρχονται από την "Επεξεργασία των Onto Τριπλετών" (Κεφάλαιο 9) και αρχικοποιούν τις συνδέσεις του αλγόριθμου (*bindings = initialBindings*). Ο αλγόριθμος εκτελεί μια επανάληψη ,που στην αρχή της αποθηκεύονται οι παρούσες αντιστοιχήσεις των μεταβλητών (o*ldBindings = bindings*) . Στο εσωτερικό της επανάληψης για κάθε τριπλέτα του BGP καλείται η συνάρτηση *DetermineBinding* . Η συνάρτηση αυτή υλοποιεί τους κανόνες υπολογισμού που αναφέρονται στην επόμενη υπό-ενότητα (10.4.1) για τον υπολογισμό των συνδέσεων. Τέλος η επανάληψη εκτελείται εώς οι αντιστοιχίσεις να παραμένουν ίδιες μεταξύ δυο επαναλήψεων(*Until(oldBindings = bsindings))* .

10.4.1 Κανόνες Υπολογισμού Συνδέσεων Μεταβλητών

Οι παρακάτω κανόνες υλοποιούνται από την συνάρτηση *DetermineBinding* του αλγόριθμου σύνδεσης μεταβλητών (Αλγόριθμος 10.1). Οι κανόνες αυτοί εφαρμόζονται σε κάθε τριπλέτα, λαμβάνοντας υπόψη

τις αντιστοιχήσεις που έχουν υπολογιστεί μέχρι το σημείο εκτελέσεις της συνάρτησης (πιθανά από άλλες τριπλέτες), υπολογίζουν τις νέες αντιστοιχήσεις για τις οποίες ισχύει η Σχέση 10.1 για τη συγκεκριμένη τριπλέτα. Το ίδιο εφαρμόζεται σε όλες τις τριπλέτες που περιέχει η βασική σχηματομορφή γράφου, με αποτέλεσμα να υπολογίζονται οι συνδέσεις για τις οποίες ισχύει η Σχέση 10.1 για όλες τις τριπλέτες. Ανάλογα με τον τύπο-μορφή της τριπλέτας, οι κανόνες διαφοροποιούνται καθώς όπως είναι λογικό η θέση και ο αριθμός των μεταβλητών είναι δυνατόν να μεταβάλλονται σε διαφορετικές τριπλέτες. Στη συνέχεια ορίζονται οι πιθανοί τύποι τριπλετών και σε αναλογία οι κανόνες που εφαρμόζονται ώστε να προσδιοριστούν οι συνδέσεις.

Με βάση την Παρατήρηση 10.1, από τους πιθανούς συνδυασμούς, προκύπτουν τέσσερις διαφορετικοί τύποι τριπλετών σχηματομορφών.

- **Τύπος 1 :** S ϵ V , P ϵ I, O ϵ L . Το υποκείμενο(S) είναι μεταβλητή, το κατηγόρημα(P) είναι IRI και το αντικείμενο(O) είναι σταθερά (L).
- **Τύπος 2 :** S, O ϵ V, P ϵ I. Το υποκείμενο(S) και το αντικείμενο(O) είναι μεταβλητή και το κατηγόρημα(P) είναι IRI (I).
- **Τύπος 3 :** S, P ϵ V, O ϵ L .Το υποκείμενο(S) και το κατηγόρημα(P) είναι μεταβλητή και το αντικείμενο(O) είναι σταθερά (I).
- **Τύπος 4 :** S, P, O ϵ V . Το υποκείμενο(S) το κατηγόρημα(P) και το αντικείμενο(O) είναι μεταβλητές.

Ανάλογα με τον τύπο, ορίζονται και οι κανόνες υπολογισμού των συνδέσεων για τις μεταβλητές. Με την χρήση του χαρακτήρα ’ σε σύνολα μονοπατιών , συμβολίζεται η νέα τιμή του συνόλου που προκύπτει από τους κανόνες. Επίσης το σύμβολο ← αναπαριστά την ανάθεση τιμής σε σύνολα μονοπατιών(Οι κανόνες αναλύονται και στα παραδείγματα στην συνέχεια) .

- Αν η τριπλέτα **S P O** είναι **Τύπος 1 :**
  - $\mathbf{X_s' \leftarrow X_{pD} \cap X_s}$
- Αν η τριπλέτα **S P O** είναι **Τύπος 2 :**
  - $\mathbf{X_s' \leftarrow X_{pD} \cap X_s \cap X_o^P}$
  - Αν P είναι ιδιότητα Αντικειμένων
    - $\mathbf{X_o' \leftarrow X_s' \oplus X_o}$
  - Αν P είναι ιδιότητα Τιμών Δεδομένων
    - Δεν υπολογίζεται σε αυτό το σημείο το O

- Αν η τριπλέτα **S P O** είναι **Τύπος 3 :**
    - $\mathbf{X_s' \leftarrow X_{pD} \cap X_s}$
    - $\mathbf{X_p' \leftarrow X_s' \oplus X_p}$
    - Ειδική περίπτωση είναι όταν καμιά από τις S P μεταβλητές δεν είναι διαμοιραζόμενη , οπότε δεν μπορεί να προσδιοριστεί κάποιο από τα σύνολα. Σε αυτή την περίπτωση
        - $\mathbf{X_s \leftarrow Xc_1\ U\ Xc_2\ U\ \ldots\ U\ Xc_n}$,όπου $\mathbf{Xc_i}$ $(1 \le i \le n)$ είναι τα σύνολα μονοπατιών των κλάσεων για τις οποίες έχουν ορίσει αντιστοιχήσεις.
        - $\mathbf{X_p = X_s/*\ U\ X_s/@*}$
    - Ειδική περίπτωση είναι όταν η P μεταβλητές δεν είναι διαμοιραζόμενη (Παράδειγμα 10.5), οπότε δεν μπορεί να προσδιοριστεί το σύνολο της . Σε αυτή την περίπτωση
        - $\mathbf{X_p = X_s/*\ U\ X_s/@*}$
- Αν η τριπλέτα **S P O** είναι **Τύπος 4 :**
    - $\mathbf{X_s' \leftarrow X_{pD} \cap X_s \cap X_o^P}$
    - $\mathbf{X_p' \leftarrow X_s' \oplus X_p}$
    - Αν P είναι ιδιότητα Αντικειμένων
        - $\mathbf{X_o' \leftarrow X_p' \cap X_o}$
    - Αν P είναι Μεταβλητή Ιδιότητας Τύπου Δεδομένων ή Μεταβλητή Ιδιότητας Αγνώστου Τύπου
        - Δεν υπολογίζεται σε αυτό το σημείο το O
    - Ειδική περίπτωση είναι όταν καμιά από τις S P O μεταβλητές δεν είναι διαμοιραζομένη (Παράδειγμα 4.15), οπότε δεν μπορεί να προσδιοριστεί κάποιο από τα σύνολα. Σε αυτή την περίπτωση
        - $\mathbf{X_s \leftarrow Xc_1\ U\ Xc_2\ U\ \ldots\ U\ Xc_n}$,όπου $\mathbf{Xc_i}$ $(1 \le i \le n)$ είναι τα σύνολα μονοπατιών των κλάσεων για τις οποίες έχουν ορίσει αντιστοιχήσεις.
        - $\mathbf{X_p = X_o = X_s/*\ U\ X_s/@*}$
    - Ειδική περίπτωση είναι όταν η P μεταβλητές δεν είναι διαμοιραζόμενη (Παράδειγμα 10.6), οπότε δεν μπορεί να προσδιοριστεί το σύνολο της . Σε αυτή την περίπτωση
        - $\mathbf{X_p = X_s/*\ U\ X_s/@*}$

Σημείωση : *Θεωρείται ότι τα σύνολα X_i αρχικά έχουν την τιμή null. Αυτό έχει ως αποτέλεσμα, ο τελεστής της τομής να έχει ειδική συμπεριφορά στην περίπτωση συνόλων που δεν έχουν αρχικοποιηθεί (δηλαδή έχουν την τιμή null). Έστω X={ null } και Y= { /a/b , d/e } τότε X ∩ Y ={ /a/b , d/e } .*

Παράδειγμα 10. 2

Σε αυτό το παράδειγμα γίνεται φανερός ο λόγος για τον οποίο δεν είναι δυνατόν να προσδιορίσουν οι συνδέσεις για μεταβλητές τύπου σταθεράς.

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) τα οποία έχουν το ίδιο ψευδώνυμο (Nickname) και μικρό όνομα"

```
SELECT ?x
WHERE { ?x  FirsName ?n .
        ?x NickName ?n . }
```

Οι τριπλέτες :

?x FirstName ?n .

?x NickName ?n .

Από την υπό-ενότητα 4.5.1 έχουν οριστεί οι εξής αντιστοιχήσεις :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

$\mathbf{X_{NickName}}$ **=** { /Persons/Person/NickName, /Persons/Employee/NickName }

Αρχικά δεν εφαρμόζεται ο αλγόριθμος αλλά με βάση την Σχέση 10.1 υπολογίζονται οι συνδέσεις για κάθε μια τριπλέτα χωριστά (ώστε να γίνει πιο κατανοητό).

Για την πρώτη τριπλέτα **?x FirstName ?n** ισχύει :

$\mathbf{X_{x1}}$ = { /Persons/Person, /Persons/Employee }

$\mathbf{X_{n1}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

Για την πρώτη τριπλέτα **?x NickName ?n** ισχύει :

$\mathbf{X_{x2}}$ = { /Persons/Person, /Persons/Employee}

$\mathbf{X_{n2}}$ = { /Persons/Person/NickName, /Persons/Employee/NickName }

Επόμενος και για τις δυο τριπλέτες πρέπει να βρεθούν οι συνδέσεις για τις οποίες τα σύνολα των ίδιων μεταβλητών να είναι ίσα, Αυτό επιτυγχάνεται από τον αλγόριθμο με την χρήση του τελεστή της τομής . Άρα για την μεταβλητή x η σύνδεση θα ήταν η

$\mathbf{X_x}$ = { /Persons/Person, /Persons/Employee}

Ενώ όπως παρατηρείται για την μεταβλητή n δεν υπάρχει κάποια κοινή σύνδεση, επόμενος η χρήση του τελεστή της τομής θα είχε σαν αποτέλεσμα το κενό σύνολο. Για τον λόγο αυτόν δεν υπολογίζεται από τον αλγόριθμο οι συνδέσεις για μεταβλητές τύπου σταθεράς. Ο τρόπος προσδιορισμού των συνδέσεων αυτού του είδους μεταβλητών περιγράφεται στην υπό-ενότητα 10.5.

Με βάση των αλγόριθμο σύνδεσης μεταβλητών (Αλγόριθμος 10.1) θα ισχύει:

Αρχικοποιούνται τα σύνολα των μεταβλητών :

- X_x = null
- X_n = null

<u>Πρώτη Επανάληψη :</u>

- Για την πρώτη τριπλέτα **?x FirstName ?n** (**Τύπος 2**) ισχύει :

  $X_x' \leftarrow X_{FirstNameD} \cap X_x \cap X_n^P$ = null $\cap$ {/Persons/Person, /Persons/Employee }$\cap$ null = {/Persons/Person, /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

- Για την πρώτη τριπλέτα **?x NickName?n** (**Τύπος 2**) ισχύει :

  $X_x' \leftarrow X_{NickNameD} \cap X_x \cap X_s^P$ = {/Persons/Person, /Persons/Employee } $\cap$ { /Persons/Employee }$\cap$ null = { /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

<u>Δεύτερη Επανάληψη :</u>

Δεν αλλάζουν οι συνδέσεις, οπότε ο αλγόριθμος τερματίζει.

Τελικές Συνδέσεις:

X_x ={ /Persons/Person, /Persons/Employee}

Παράδειγμα 10. 3

Έστω η ερώτηση : "Επέστρεψε όλα τους εργαζόμενους, το/τα μικρό/α τους όνομα/τα και τον μισθό τους" (η οποία αποτελείται από δυο τριπλέτες σχηματομορφής)

SELECT ?x ?n ?s
WHERE{ ?x FirstName ?n .
?x Salary ?s . }

Οι τριπλέτες :

?x FirstName ?n .

?x Salary ?s .

Από την υπό-ενότητα 4.5.1 έχουν οριστεί οι εξής αντιστοιχήσεις :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

$\mathbf{X_{Salary}}$ **=** { /Persons/Employee/Salary }

Με βάση των αλγόριθμο σύνδεσης μεταβλητών (Αλγόριθμος 10.1) θα ισχύει:

Αρχικοποιούνται τα σύνολα των μεταβλητών :

- X_x = null
- X_n = null
- X_s = null

Πρώτη Επανάληψη :

- Για την πρώτη τριπλέτα **?x FirstName ?n** (**Τύπος 2**) ισχύει :

  $X_x' \leftarrow X_{FirstNameD} \cap X_x \cap X_n^P$ = null ∩ {/Persons/Person, /Persons/Employee }∩ null = {/Persons/Person, /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

- Για την πρώτη τριπλέτα **?x Salary ?s** (Τύπος 2) ισχύει :

  $X_x' \leftarrow X_{SalaryD} \cap X_x \cap X_s^P$ = {/Persons/Person, /Persons/Employee } ∩ { /Persons/Employee }∩ null = { /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

Δεύτερη Επανάληψη :

- Για την πρώτη τριπλέτα **?x FirstName ?n** (**Τύπος 2**) ισχύει :

  $X_x' \leftarrow X_{FirstNameD} \cap X_x \cap X_n^P$ = { /Persons/Employee }∩ {/Persons/Person, /Persons/Employee }∩ null = { /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

- Για την πρώτη τριπλέτα **?x Salary ?s** (**Τύπος 2**) ισχύει :

  $X_x' \leftarrow X_{SalaryD} \cap X_x \cap X_s^P$ = { /Persons/Employee } ∩ { /Persons/Employee }∩ null = { /Persons/Employee }

  $X_n' \leftarrow$ Δεν υπολογίζεται σε αυτό το σημείο το X_n

Τρίτη Επανάληψη :

Δεν αλλάζουν οι συνδέσεις , οπότε ο αλγόριθμος τερματίζει.

Τελικές Συνδέσεις :

X_x = { /Persons/Employee }

Παράδειγμα 10. 4

Έστω η ερώτηση : "Επέστρεψε όλα τα άτομα(και εργαζόμενους) ,τον οποίων το μικρό τους όνομα είναι John.

SELECT ?x
WHERE{ ?x FirstName "John" . }

Η τριπλέτα :

?x FirstName "John" .

Από την υπό-ενότητα 4.5.1 έχουν οριστεί οι εξής αντιστοιχήσεις :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

Με βάση των αλγόριθμο σύνδεσης μεταβλητών (Αλγόριθμος 10.1) θα ισχύει:

Αρχικοποιούνται τα σύνολα των μεταβλητών :

- X_x = null

<u>Πρώτη Επανάληψη :</u>

- Για την πρώτη τριπλέτα **?x FirstName "John"** (**Τύπος 1**) ισχύει :

  $\mathbf{X_x'} \leftarrow \mathbf{X_{FirstNameD}} \cap \mathbf{X_x}$ ={ /Persons/Person, /Persons/Employee}∩null = { /Persons/Person, /Persons/Employee}

<u>Δεύτερη Επανάληψη :</u>

Δεν αλλάζουν οι συνδέσεις , οπότε ο αλγόριθμος τερματίζει.

Τελικές Συνδέσεις :

X_x ={ /Persons/Person, /Persons/Employee}

Παράδειγμα 10. 5

Έστω η ερώτηση : "Επέστρεψε τους εργαζόμενους ,τον οποίων σε κάποια ιδιότητα έχουν τιμή την συμβολοσειρά John.

SELECT ?x
WHERE{ ?x rdf:type Employee.
?x ?p "John" . }

Η τριπλέτα :

?x rdf:type Employee.

Αποτελεί Οντο- τριπλέτα , από την επεξεργασία της έχουμε :

X_x = {/Persons/Employee}

Με βάση των αλγόριθμο σύνδεσης μεταβλητών (Αλγόριθμος 10.1) θα ισχύει:

Αρχικοποιούνται τα σύνολα των μεταβλητών :

- X_x = {/Persons/Employee} (αρχικοποίηση από επεξεργασία Οντο – τριπλετών)
- X_p = null

<u>Πρώτη Επανάληψη :</u>

- Για την πρώτη τριπλέτα **?x ?p "John"** (**Τύπος 3**) ισχύει :

  $\mathbf{X_x' \leftarrow X_{pD} \cap X_x}$ =null ∩ { /Persons/Employee} = { /Persons/Employee}

  $\mathbf{X_p' \leftarrow X_x' /@* \cup X_x' /*}$ = { /Persons/Employee/@* , /Persons/Employee/*}

<u>Δεύτερη Επανάληψη :</u>

Δεν αλλάζουν οι συνδέσεις , οπότε ο αλγόριθμος τερματίζει.

Τελικές Συνδέσεις :

X_x ={ /Persons/Employee}

X_p ={ /Persons/Employee/@*,/Persons/Employee/*}

Παράδειγμα 10. 6

Έστω η ερώτηση : "Επέστρεψε ότι πληροφορία υπάρχει" (η οποία αποτελείται από μια τριπλέτα σχηματομορφής)

SELECT ?S ?P ?O
WHERE { ?S ?P ?O }

Η τριπλέτα :

?S ?P ?O

Όπως παρατηρείται από την παραπάνω τριπλέτα, δεν υπάρχει κάποιο μέρος της για το οποίο να είναι γνωστές κάποιες αντιστοιχήσεις ή να μπορούν να προσδιοριστούν, άρα πρέπει να αντιστοιχηθούν όλα τα πιθανά μονοπάτια.

Με βάση των αλγόριθμο σύνδεσης μεταβλητών (Αλγόριθμος 10.1) θα ισχύει:

Αρχικοποιούνται τα σύνολα των μεταβλητών :

- X_s = null
- X_p = null
- X_o = null

<u>Πρώτη Επανάληψη :</u>

- Για την πρώτη τριπλέτα **?S ?P ?O** (**Τύπος 4**) ισχύει :

  $\mathbf{X_s \leftarrow Xc_1\ U\ Xc_2\ U\ \ldots\ U\ Xc_n = X_{Persons\_Type}\ U\ X_{Person\_Type}\ U\ X_{Employee\_Type}\ U\ X_{Address\_Type}}$ = { /Persons } U { /Persons/Person , /Persons/Employee } U { /Persons/Employee } U {/Persons/Person/Address , /Persons/Employee/Address } = { /Persons , /Persons/Person , /Persons/Employee , /Persons/Person/Address , /Persons/Employee/Address }

  $\mathbf{X_p' \leftarrow X_x' / @* \ U\ X_x' / *}$ = { /Persons/* , /Persons/Person/* , /Persons/Employee/*,/Persons/Person/Address/*, /Persons/Employee/Address/* , /Persons/@* ,/Persons/Person/@* , /Persons/Employee/@* , /Persons/Person/Address/@*, /Persons/Employee/Address/@* }

<u>Δεύτερη Επανάληψη :</u>

Δεν αλλάζουν οι συνδέσεις , οπότε ο αλγόριθμος τερματίζει.

10.5 Αλληλεξαρτήσεις Μεταξύ Συνόλων Μονοπατιών στις Τριπλέτες Σχηματομορφών

Σε αυτή την ενότητα θα εξεταστούν οι αλληλεξαρτήσεις μεταξύ των συνόλων μονοπατιών στις τριπλέτες σχηματομορφών, εφόσον αυτά έχουν υπολογίζονται από τον αλγόριθμο σύνδεσης μεταβλητών(Αλγόριθμο 10.1). Δείχνοντας ότι τα σύνολα μονοπατιών στις τριπλέτες μπορούν να οριστούν με "βάση" το σύνολο μονοπατιών που έχει υπολογιστεί από τον αλγόριθμο σύνδεσης μεταβλητών(Αλγόριθμο 10.1) για το υποκείμενο της κάθε τριπλέτας (βλέπε Παραδείγματα 10.7-10.9). Αυτό έχει ως αποτέλεσμα κατά την ανάπτυξη των XQuery ερωτήσεων να είναι εφικτή η "σύνδεση" μεταξύ των δεδομένων τα οποία "αλληλοεξαρτώνται" (όπως των δεδομένων που αντιστοιχούν στα υποκείμενο-κατηγόρημα-αντικείμενο) με την κατάλληλη σύνταξη των for και let δομών(clauses)(για λεπτομέρειες βλέπε Κεφάλαιο 11). Λόγω της δενδρικής δομής των XML εγγράφων οι αλληλεξαρτήσεις μεταξύ των δεδομένων μπορούν να ελεγχθούν με βάση τα μονοπάτια τους. Για παράδειγμα τα δεδομένα που βρίσκονται στα μονοπάτια /Persons/Person και /Persons/Person/FirstName αλληλοεξαρτώνται.

> **Ορισμός 10.1 Επέκταση συνόλων μονοπατιών** Ένα σύνολο μονοπατιών A θα ονομάζεται **επέκταση (extension)** ενός συνόλου μονοπατιών B, αν τα όλα τα μονοπάτια του A είναι απόγονοι (descendants) μονοπατιών του B. Σύνολα επέκτασης ενός αρχικού συνόλου μπορούν να προκύψουν με χρήση του **τελεστής προσάρτησης (append) /** (Ορισμός 5.6).
>
> **Σημείωση :** *Λόγω της δενδρικής δομής των XML εγγράφων, γίνεται κατανοητό ότι τα δεδομένα που αντιστοιχούν στα σύνολα μονοπατιών A και B , αλληλοεξαρτώνται.*

Έστω η τριπλέτα σχηματομορφών **S P O** . Από την Σχέση 10.1 της υπό-παραγράφου 10.2, γίνεται φανερό ότι τα σύνολα μονοπατιών που αντιστοιχούν στα S,P και O αλληλοεξαρτώνται . Πιο συγκεκριμένα τα σύνολα P και O είναι **επεκτάσεις** του S.

Πιο συγκεκριμένα από την Σχέση 10.1 προκύπτει $\mathbf{X_P = X_O = X_S/X_P^{LN}}$, που σημαίνει ότι τα σύνολα μονοπατιών του αντικειμένου ($\mathbf{X_O}$) και του κατηγορήματος ($\mathbf{X_P}$) αποτελούν επέκταση του συνόλου μονοπατιών του υποκειμένου ($\mathbf{X_S}$) και ισούνται με το σύνολο μονοπατιών του υποκειμένου εάν σε αυτό προσαρτήσουν οι κομβοί που προκύπτουν από την χρήση του τελεστή $^{\mathbf{LN}}$ στο σύνολο του κατηγορήματος($\mathbf{X_P^{LN}}$).

Όπως παρατηρείται σε αυτό το σημείο δεν είναι απαραίτητο να έχει πραγματοποιηθεί ο υπολογισμός του συνόλου του αντικειμένου, καθώς αυτό μπορεί να προκύψει από τα σύνολα του υποκειμένου και του κατηγορήματος. Με τον τρόπο αυτόν αντιμετωπίζεται και το πρόβλημα στην μη δυνατότητα υπολογισμού των μεταβλητών τύπου σταθεράς από τον αλγόριθμο σύνδεσης μεταβλητών.

Η σχέση $\mathbf{X_P = X_O = X_S/X_P^{LN}}$ που προκύπτει από την Σχέση 10.1, διαχωρίζεται σε δυο περιπτώσεις όπως φαίνεται στην Σχέση 10.2. Στην περίπτωση την οποία το κατηγόρημα (P) είναι μεταβλητή και στην περίπτωση που το κατηγόρημα P είναι IRI.

Στην πρώτη περίπτωση στην οποία το κατηγόρημα (P) είναι μεταβλητή ισχύει η σχέση $\mathbf{X_P = X_O = X_S/X_P^{LN}}$ που σημαίνει ότι τα σύνολα μονοπατιών του αντικειμένου ($\mathbf{X_O}$) και του κατηγορήματος ($\mathbf{X_P}$) αποτελούν επέκταση του συνόλου μονοπατιών του υποκειμένου ($\mathbf{X_S}$) και ισούνται με το σύνολο μονοπατιών του υποκειμένου εάν σε αυτό προσαρτήσουν οι κόμβοι που προκύπτουν από την χρήση του τελεστή $^{\mathbf{LN}}$ στο σύνολο του κατηγορήματος.(Σχέση 10.2)

Στην δεύτερη περίπτωση στην οποία το κατηγόρημα (P) είναι IRI και δεν είναι μεταβλητή ισχύει η σχέση $\mathbf{X_P = X_O = X_S/y_p = X_S/y_p}$. Εφόσον για το P δεν έχει πραγματοποιηθεί ο υπολογισμός του από τον αλγόριθμο σύνδεσης μεταβλητών, καθώς είναι κάτι σταθερό(IRI όχι μεταβλητή) που προκύπτει από τις αντιστοιχήσεις και θα ήταν λάθος να προσδιοριστεί, εφόσον όπως είναι λογικό μπορεί να εμφανίζεται σε παραπάνω από μια τριετές και με διαφορετική(εάν υπολογιστεί) σε συνδυασμό με το υποκείμενο της τριπλέτας τιμή. Για τον λόγο αυτόν το σύνολο του αντικείμενου υπολογίζεται ως επέκταση του $\mathbf{y_p}$, το οποίο προκύπτει από την εφαρμογή του τελεστή $\oplus$ μεταξύ του συνόλου του υποκειμένου και του κατηγορήματος και την εφαρμογή στο αποτέλεσμα του τελεστή ^{LN} ,$\mathbf{(X_S \oplus X_P)^{LN}}$ (Σχέση 10.2).

Αν P είναι Μεταβλητή :

$$X_P = X_0 = X_S/X_P^{LN}$$

Αν P είναι IRI:

$$X_P = X_0 = X_S/y_p = X_S/y_p$$

$$\text{Όπου } y_p = (X_S \oplus x_P)^{LN}$$

Σχέση 10.2

Στην συνέχεια παρατίθεται μια σειρά από παραδείγματα για την κατανόηση και την απόδειξη της ισχύος της Σχέσης 10.2

Παράδειγμα 10. 7

Από το Παράδειγμα 10.1 έχουμε για την τριπλέτα :

?x FirstName ?n

Από την υπό-ενότητα 4.5.1 έχουν οριστεί οι εξής αντιστοιχήσεις :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

Από την εκτέλεση του αλγόριθμου σύνδεσης μεταβλητών στο Παράδειγμα 10.1 υπολογίστηκε :

$\mathbf{X_x}$ = { /Persons/Person, /Persons/Employee }

Οπως παρατηρείται ισχύει η Σχέση 10.2 :

$\mathbf{y_p} = \mathbf{(X_x} \oplus$ **FirstName)**$^{\mathbf{LN}}$ ={ FirstName}

$\mathbf{X_n}$ **=** $\mathbf{X_x / y_p}$ **=**{ /Persons/Person, /Persons/Employee } / { FirstName} =
{/Persons/Person/FirstName , /Persons/Employee/FirstName }

Όπως γίνεται αντιληπτό το σύνολο $\mathbf{X_n}$ περιέχει τα μονοπάτια των μικρών ονομάτων των ατόμων και των εργαζομένων. Που ταυτίζεται με αυτό που ρωτάει η ερώτηση στο Παράδειγμα 10.1

Παράδειγμα 10. 8

Από το Παράδειγμα 10.3 έχουμε για τις τριπλέτες :

?x FirstName ?n .
?x Salary ?s .

Από την υπό-ενότητα 4.5.1 έχουν οριστεί οι εξής αντιστοιχήσεις :

$\mathbf{X_{FirstName}}$ **=** { /Persons/Person/FirstName , /Persons/Employee/FirstName }

$\mathbf{X_{Salary}}$ **=** { /Persons/Employee/Salary }

Από την εκτέλεση του αλγόριθμου σύνδεσης μεταβλητών στο Παράδειγμα 10.3 υπολογίστηκε :

$\mathbf{X_x}$ = { /Persons/Employee }

Οπως παρατηρείται ισχύει η Σχέση 10.2 :

Για την πρώτη τριπλέτα (?x FirstName ?n .) :

$\mathbf{y_p} = \mathbf{(X_x \oplus FirstName)^{LN}} =\{$ FirstName$\}$

$\mathbf{X_n = X_x / y_p =}\{$ /Persons/Employee $\}/ \{$ FirstName$\} =\{$ /Persons/Employee/FirstName $\}$

Για την δεύτερη τριπλέτα (?x Salary ?s .) :

$\mathbf{y_p} = \mathbf{(X_x \oplus FirstName)^{LN}} =\{$ Salary$\}$

$\mathbf{X_s = X_x / y_p =}\{$ /Persons/Employee $\} / \{$ Salary$\} = \{$ /Persons/Employee/Salary $\}$

Όπως γίνεται αντιληπτό το σύνολο $\mathbf{X_n}$ περιέχει τα μονοπάτια των μικρών ονομάτων των εργαζομένων και το σύνολο $\mathbf{X_s}$ περιέχει τα μονοπάτια των μισθών των εργαζομένων. Που ταυτίζονται με αυτά που ρωτάει η ερώτηση στο Παράδειγμα 10.3.

Παράδειγμα 10. 9

Από το Παράδειγμα 10.5 έχουμε για την τριπλέτα :

?S ?P "John"

Από την εκτέλεση του αλγόριθμου σύνδεσης μεταβλητών στο Παράδειγμα 10.9 υπολογίστηκε :

$\mathbf{X_x} =\{$ /Persons/Employee$\}$

$\mathbf{X_p} =\{$ /Persons/Employee/@*,/Persons/Employee/*$\}$

Οπως παρατηρείται ισχύει η Σχέση 10.2 :

$\mathbf{X_p^{LN}} = \{ * , @* \}$

$\mathbf{X_p = X_x / X_p^{LN}} =\{$ /Persons/Employee$\} / \{ * , @* \} =$
$\{$ /Persons/Employee/@*,/Persons/Employee/*$\}$

Όπως γίνεται αντιληπτό το σύνολο $\mathbf{X_p}$ περιέχει τα μονοπάτια των στοιχείων ή χαρακτηριστικών που περιέχονται μέσα στα στοιχεία των εργαζομένων. Που ταυτίζονται με αυτά που ρωτάει η ερώτηση στο Παράδειγμα 10.5.

10.6 Επίλογος

Στο κεφάλαιο αυτό, έγινε η περιγραφή της διαδικασίας "Σύνδεση Μεταβλητών" (Variables Binding), η οποία εφαρμόζεται σε βασικές σχηματομορφές γράφων ώστε να επιτευχθεί η σύνδεση των μεταβλητών που περιέχονται σε αυτές με μονοπάτια του XML εγγράφου. Πιο συγκεκριμένα, αναλυθήκαν οι σχέσεις που ισχύουν μεταξύ των μονοπατιών που αντιστοιχούν στα μέρη (υποκείμενο-κατηγόρημα-αντικείμενο) της τριπλέτας, ορίστηκε ο αλγόριθμος για τον προσδιορισμό των συνδέσεων και τέλος εξεταστήκαν οι αλληλεξαρτήσεις μεταξύ των συνόλων μονοπατιών στις τριπλέτες σχηματομορφών.

Οι συνδέσεις μεταβλητών με μονοπάτια που προσδιορίζονται από τον αλγόριθμο υπολογισμού σύνδεσης μεταβλητών και οι αλληλεξαρτήσεις των συνόλων μονοπατιών, χρησιμοποιούνται από τον αλγόριθμο μετάφρασης των βασικών σχηματομορφών γράφων, ο οποίος αναλύεται στο επόμενο κεφάλαιο(Κεφάλαιο 11) και κάνει χρήση των μονοπατιών για την ανάπτυξη των XQuery εκφράσεων επιτυγχάνοντας με αυτό τον τρόπο την διάσχισης και ανάκτησης της αντίστοιχη(ανάλογα με την μεταβλητή) XML πληροφορίας.

Στο επόμενο κεφάλαιο (Κεφάλαιο 11) περιγράφεται η διαδικασία "Μετάφρασης Βασικών Σχηματομορφών Γράφων" (Basic Graph Pattern Translation). Η διαδικασία δέχεται ως εισόδους μια βασική σχηματομορφή γράφου, τις αντιστοιχήσεις, τις συνδέσεις των μεταβλητών που έχουν προσδιοριστεί από την διαδικασία σύνδεσης μεταβλητών καθώς και τους τύπους των μεταβλητών όπως έχουν προσδιοριστεί από την διαδικασία προσδιορισμού τύπων μεταβλητών(Κεφάλαιο 8) και παράγει ως έξοδο XQuery εκφράσεις σημασιολογικά ισοδύναμες με την βασική σχηματομορφή γράφου της εισόδου.

11 Μετάφραση Βασικών Σχηματομορφών Γράφων

11.1 Εισαγωγή

Στο παρόν κεφάλαιο περιγράφεται η διαδικασία **"Μετάφρασης Βασικών Σχηματομορφών Γράφων" (Basic Graph Pattern Translation).** Η διαδικασία δέχεται ως εισόδους μια βασική σχηματομορφή γράφου, τις αντιστοιχήσεις (Κεφάλαιο 5), τις συνδέσεις των μεταβλητών που έχουν προσδιοριστεί από την διαδικασία σύνδεσης μεταβλητών(Κεφάλαιο 10) καθώς και τους τύπους των μεταβλητών όπως έχουν προσδιοριστεί από την διαδικασία προσδιορισμού τύπων μεταβλητών(Κεφάλαιο 8) και παράγει ως έξοδο XQuery εκφράσεις σημασιολογικά ισοδύναμες με την βασική σχηματομορφή γράφου της εισόδου.

Ένα από τα βασικότερα και πιο πολύπλοκα κομμάτια της μετάφρασης, είναι η μετάφραση της συνθήκης που βρίσκεται στο Where κομμάτι (Where Clause) της ερώτησης. Αυτή η συνθήκη πρέπει να μεταφραστεί με τέτοιο τρόπο ώστε να είναι δυνατή η αποτίμηση της στα XML δεδομένα με χρήση της γλώσσας XQuery. Είναι φανερή η ανάγκη ανάπτυξης αλγορίθμου ο οποίος θα πραγματοποιεί την μετάφραση της συνθήκης σε XQuery εκφράσεις.

Στην γλώσσα SPARQL η συνθήκη είναι μια Σχηματομορφή Γράφου (Ορισμός 2.8). Από τον ορισμό γίνεται αντιληπτό ότι η πιο απλή μορφή σχηματομορφής γράφου είναι η τριπλέτα σχηματομορφής και κατά επέκταση η Βασική Σχηματομορφή Γράφου καθώς αυτή αποτελείται από τριπλέτες σχηματομορφών και φίλτρα (Ορισμός 2.7). Άρα το πρώτο βήμα στην μετάφραση της συνθήκης είναι η ανάπτυξη αλγόριθμου για την μετάφραση του απλούστερου συστατικού των σχηματομορφών γράφων.

Όπως γίνεται φανερό και από τους Ορισμούς 2.7 και 2.15 , μια Βασική Σχηματομορφή Γράφων είναι μια ακολουθία από τριπλέτες σχηματομορφών με τον τελεστή AND (υποθετικά, καθώς παραλείπεται κατά την σύνταξη) ανάμεσα τους, που έχει ως συνέπεια την εφαρμογή της σύζευξης (Join) στις αντιστοιχήσεις λύσεων που προκύπτουν από την αποτίμηση τους (Ορισμός 2.13). Άρα, ο αλγόριθμος θα πρέπει να συντάξει ερωτήσεις XQuery που θα υλοποιούν την σύζευξη των αντιστοιχήσεων λύσεων. Δηλαδή να υλοποιεί τον τελεστή Join που εφαρμόζεται μεταξύ τριπλετών σχηματομορφών στο επίπεδο των αντιστοιχήσεων λύσεων.

Η υλοποίηση του τελεστή Join από τον αλγόριθμο επιτυγχάνεται από τον τρόπο σύνταξης και ανάπτυξης των εκφράσεων του XQuery ερωτήματος. Αυτό έχει ως αποτέλεσμα τη βελτιστοποίηση των XQuery ερωτήσεων καθώς ο τελεστής δεν εφαρμόζεται σε επίπεδο δεδομένων.

Σύμφωνα με την σημασιολογία της γλώσσας SPARQL η αποτίμηση μια βασικής σχηματομορφής γράφου BGP (Basic Graph Pattern) σε ένα γράφο G και σε ένα RDF Σύνολο Δεδομένων D, συμβολίζεται ως $\mathbf{[[BGP]]_G^D}$**.** Ο αλγόριθμος **BGP2XQuery**(Basic Graph Pattern to XQuery) προσομοιώνει την αποτίμηση της BGP, σε ένα γράφο G και σε ένα σύνολο XD, XML δεδομένων $\mathbf{[[BGP]]_G^{XD}}$.

Οι στόχοι τεθήκαν κατά την σχεδίαση του αλγορίθμου είναι : α) Η ανάπτυξη μια γενικής και κατανοητής διαδικασίας για την μετάφραση Βασικών Σχηματομορφών Γράφων σε ερώτηση XQuery. β) Αυστηρή τήρηση της σημασιολογίας κατά την διαδικασία της μετάφρασης γ) Όσο το δυνατόν μικρότερα και λιγότερο πολύπλοκα XQuery. δ) Ανάπτυξη και σύνταξη των ερωτήσεων με τρόπο ώστε οι "αντιστοιχίες" μεταξύ των δυο ερωτήσεων(SPARQL-XQuery) και ο τρόπος μετάφρασης να γίνονται εύκολα αντιληπτά. ε) Σε συνδυασμό με τους άλλους στόχους όσο το δυνατόν βελτιστοποίηση(Optimization) των XQuery .(πχ στον τρόπο υλοποίησης της σύζευξης). Τέλος αυτό που γίνεται προσπάθεια να επιτευχθεί δεν είναι η απάντηση των SPARQL ερωτήσεων σε XML δεδομένα, αλλά η μετάφραση των SPARQL ερωτήσεων σε σημασιολογικές ισοδύναμες XQuery ερωτήσεις.

Όπως αναλύεται και στην συνέχεια του κεφαλαίου η υλοποίηση του τελεστή Join ακολουθεί αυστηρά την σημασιολογία που έχει οριστεί από την παρούσα προδιαγραφή (Specification)[5] της γλώσσας SPARQL για τον χειρισμό των κενών (unbound) μεταβλητών όπως αυτή έχει σχολιαστεί και [13][14].

Στην συνέχεια του κεφαλαίου παρουσιάζεται και αναλύεται ο αλγόριθμος για την μετάφραση βασικών σχηματομορφών γράφων σε σημασιολογικά ισοδύναμες XQuery εκφράσεις. Ο αλγόριθμος και η ανάλυση του χωρίζεται σε δυο μέρη, στο πρώτο μέρος παρουσιάζεται ο αλγόριθμος στην περίπτωση απουσίας διαμοιραζόμενων μεταβλητών (ενότητα 11.3), ενώ στο δεύτερο μέρος αναλύεται ο αλγόριθμος στην περίπτωση ύπαρξης διαμοιραζόμενων μεταβλητών (ενότητα 11.4). Τέλος παρουσιάζεται η δομή και η μορφή των αποτελεσμάτων που παραγάγουν τα XQuery ερωτήματα, βάση των οποίων γίνεται και η παραγωγή των XQuery return δομών από τον αλγόριθμο.(ενότητα 11.5)

11.2 Εισαγωγή στον Αλγόριθμο BGP2XQuery (Basic Graph Pattern to XQuery)

Για την καλύτερη κατανόηση των αλγορίθμων σε αυτό το σημείο παρατίθεται μια σειρά από ορισμούς των οποίον θα γίνει χρήση κατά την παρουσίαση και ανάλυση των αλγορίθμων.

Ορισμός 11.1 Ορίζεται η **αφηρημένη(Abstract) σύνταξη των For και Let δομών (Clause)** στην γλώσσα XQuery ως :

for $Var in expr
let $Var := expr

Ορισμός 11.1 Η μεταβλητή $doc Ορίζεται η μεταβλητή $doc, η οποία περιέχει έναν κόμβο κειμένου (Document node) ή μια ακολουθία από κόμβους κειμένου. Αρχικοποιείται με τις δομές : Let $doc := fn:doc (URI) ή Let $doc := fn:collection (URI) . Όπου URI είναι η διεύθυνση για το XML έγγραφο ή την συλλογή εγγράφων.

Σημείωση : *Για τον ορισμό της μεταβλητής $doc, όπως και άλλων μεταβλητών ($BGP_i ,$UNION_i ,$AND_i , $Results κτλ) που ορίζονται στην συνέχεια του κειμένου για τις ανάγκες της μεταφράσεις, γίνεται έλεγχος να μην συμπίπτουν με ονόματα μεταβλητών οι οποίες χρησιμοποιούνται στην SPARQL ερώτηση και πιθανός να δημιουργήσουν σύγχυση κατά την ανάγνωση των μεταφρασμένων ερωτήσεων. Στην περίπτωση εμφάνισης μεταβλητών με αυτά τα ονόματα στην SPARQL ερώτηση, οι βοηθητικές μεταβλητές διαφοροποιούν το όνομα τους επισυνάπτοντας ένα αύξοντα αριθμό στο όνομα τους.*

Ορισμός 11.2 Μεταβλητές Επιστροφής (Return Variables) ονομάζονται οι μεταβλητές για τις οποίες, επιστρέφει κάποια πληροφορία μια SPARQL ερώτηση και προκύπτουν ανάλογα με τον τύπο της ερώτησης .

- Για τις ερωτήσεις τύπου **SELECT** : είναι το σύνολο των μεταβλητών, που αναφέρονται αμέσως μετά το SELECT ενώ στην περίπτωση που υπάρχει * είναι το σύνολο όλων των μεταβλητών που αναφέρονται στην σχηματομορφή γράφου της ερώτησης.
- Για τις ερωτήσεις τύπου **DESCRIBE** : όμοια με τύπου SELECT.
- Για τις ερωτήσεις τύπου **CONSTRUCT** : είναι το σύνολο των μεταβλητών, που αναφέρονται στον πρότυπο γράφο (graph template). (Στις μεταβλητές δεν συμπεριλαμβάνονται οι κενοί κόμβοι του πρότυπου γράφου)
- Για τις ερωτήσεις τύπου **ASK** : δεν έχουμε μεταβλητές επιστροφής εφόσον οι ερωτήσεις αυτού του τύπου επιστρέφουν yes ή no.

Ορισμός 11.3 Όνομα μεταβλητής Ως N_V συμβολίζεται το όνομα της μεταβλητής V.

Σημείωση : *Τα σύνολα V και I είναι τα γνωστά σύνολα μεταβλητών και IRI.*

11.3 Ο Αλγόριθμος BGP2XQuery Με Απουσία Διαμοιραζόμενων Μεταβλητών

Σαν αρχική προσέγγιση, θεωρείται ότι στην βασική σχηματομορφή γράφου δεν εμφανίζονται διαμοιραζόμενες μεταβλητές (οι περιπτώσεις ύπαρξης διαμοιραζομένων μεταβλητών αναλύονται στην ενότητα 11.4).

Ο αλγόριθμος BGP2XQuery(Basic Graph Pattern to XQuery) με Απουσία Διαμοιραζόμενων Μεταβλητών (Αλγόριθμος 11.1) δέχεται ως εισόδους μια βασική σχηματομορφή γράφου **(BGP)**, τις αντιστοιχήσεις**(mappings)**, τις συνδέσεις των μεταβλητών**(variablesBindings)** που έχουν προσδιοριστεί από την διαδικασία σύνδεσης μεταβλητών(Κεφάλαιο 10), τους τύπους των μεταβλητών **(varsTypes)** όπως έχουν προσδιοριστεί από την διαδικασία προσδιορισμού τύπων μεταβλητών (Κεφάλαιο 8), τον τύπο της SPARQL ερώτησης **(queryType)** και τις μεταβλητές επιστροφής **(RV)** και

παράγει ως έξοδο XQuery εκφράσεις σημασιολογικά ισοδύναμες με την βασική σχηματομορφή γράφου της εισόδου.

Στο εσωτερικό του ο αλγόριθμος καλεί με την σειρά τους αλγορίθμους : s*ubjectsTranslation* ο οποίος πραγματοποιεί την μετάφραση των υποκειμένων των τριπλετών (υπό-ενότητα 11.3.1), τον p*redicatesTranslation* ο οποίος πραγματοποιεί την μετάφραση των κατηγορημάτων των τριπλέτων (υπό-ενότητα 11.3.2), o*bjectsTranslation* ο οποίος πραγματοποιεί την μετάφραση των αντικειμένων των τριπλέτων (υπό-ενότητα 11.3.3), *filtersTranslation* ο οποίος πραγματοποιεί την μετάφραση των φίλτρων που περιέχονται στην βασική σχηματομορφή γράφων (υπό-ενότητα 11.3.4) και τέλος τον r*eturnClauseBuilt* ο οποίος παράγει την return δομή των XQuery εκφράσεων(υπό-ενότητα 11.3.5).

```
BGP2XQueryShareVarsAbsence( BGP , mappings , variablesBindings , varsTypes , RV , queryType) {
    subjectsTranslation( BGP , mappings , variablesBindings )
    predicatesTranslation( BGP , mappings , variablesBindings )
    objectsTranslation( BGP , mappings , variablesbindings )
    filtersTranslation( BGP )
    returnClauseBuilt( RV,  queryType)
    return XQuery_Expressions
}
```

Αλγόριθμος 11.1 Αλγόριθμος BGP2XQuery με Απουσία Διαμοιραζόμενων Μεταβλητών

11.3.1 Μετάφραση Υποκειμένων

Η μετάφραση των υποκειμένων πραγματοποιείται από τον αλγόριθμο **subjectsTranslation** (Αλγόριθμος 11.2), ο οποίος δέχεται ως εισόδους μια βασική σχηματομορφή γράφου **(BGP)**, τις αντιστοιχήσεις**(mappings)** και τις συνδέσεις των μεταβλητών**(variablesBindings).**

Ανάλυση Αλγορίθμου:

- **Για κάθε υποκείμενο των τριπλετών που είναι μεταβλητή**.

  Ορίζεται μια δομή For ή Let (For/Let) .Τα κριτήρια επιλογής μεταξύ For και Let δομής αναλύονται στην υπό-ενότητα 11.3.6 (Όπως προκύπτει και από την παρατήρηση 10.1, το υποκείμενο των τριπλέτων είναι πάντα μεταβλητή)

- Η μεταβλητή (Var) της For/Let δομής έχει το όνομα της μεταβλητής του υποκειμένου της τριπλέτας. Με τον τρόπο αυτό, επιτυγχάνεται απόλυτη αντιστοιχία στα ονόματα των μεταβλητών στις δυο ερωτήσεις .
- Η έκφραση (expr) της For/Let δομής προκύπτει από την ένωση (union) των επεκτάσεων της $doc μεταβλητής, οι οποίες προκύπτουν από την προσάρτηση των στοιχείων του συνόλου μονοπατιών του υποκειμένου στην μεταβλητή $doc.

subjectsTranslation(BGP , mappings , variablesBindings){

$\forall$ **S** ϵ **V** $\Rightarrow$**Δημιούργησε ένα For/Let Clause**

Var = N_S

**expr=$doc/**$x_1$ **union $doc/**$x_2$ **union … union $doc/**$x_n$ $\forall$ $x_i \epsilon X_S$

}

Αλγόριθμος 11.2 Αλγόριθμος μετάφρασης Υποκειμένων

11.3.2 Μετάφραση Κατηγορημάτων

Η μετάφραση των κατηγορημάτων πραγματοποιείται από τον αλγόριθμο **predicateTranslation** (Αλγόριθμος 11.3), ο οποίος δέχεται ως εισόδους μια βασική σχηματομορφή γράφου **(BGP)**, τις αντιστοιχήσεις**(mappings)** και τις συνδέσεις των μεταβλητών**(variablesBindings).**

Ανάλυση Αλγορίθμου:

- **Για κάθε κατηγόρημα των τριπλέτων που είναι μεταβλητή**.

  Ορίζεται μια δομή For ή Let (For/Let)

  - Η μεταβλητή (Var) της For/Let δομής έχει το όνομα της μεταβλητής του κατηγορήματος της τριπλέτας. Με τον τρόπο αυτό, επιτυγχάνεται απόλυτη αντιστοιχία στα ονόματα των μεταβλητών στις δυο ερωτήσεις .
  - Η έκφραση (expr) της For/Let δομής αποτελείται από την ένωση των μονοπατιών που προκύπτουν, από την προσάρτηση στην μεταβλητή που έχει οριστεί (στο XQuery) για το υποκείμενο της τριπλέτας, του ονόματος των τελευταίων κόμβων του συνόλου μονοπατιών του κατηγορήματος.

- Στην ειδική περίπτωση όπου η μεταβλητή του κατηγορήματος δεν μπορεί να προσδιοριστεί και προκύψει από προέκταση της μεταβλητής του υποκειμένου με χρήση των wildcard * και @*. Γίνετε έλεγχος αν το XPath του κόμβου που έχει ανατεθεί στην μεταβλητή περιέχεται στα Xpaths που έχουν οριστεί στις αντιστοιχήσεις των ιδιοτήτων. Ώστε να εξασφαλιστεί ότι οι τιμές που επιστρέφονται προέχονται από XML στοιχεία ή χαρακτηρίστηκα τα οποία έχουν αντιστοιχηθεί.

predicatesTranslation(BGP , mappings , variablesBindings){

$\forall$ **P** ϵ **V** $\Rightarrow$ **Δημιούργησε ένα For/ Let Clause**

Var = N_p

Expr = $\$N_S/x_1$ **union** $\$N_S/x_2$ **union ... union** $\$N_S/x_n$ $\forall$ $x_i \epsilon X_p^{LN}$

IF δημιουργήθηκε For και X_p^{LN} **={ * , @* }$\Rightarrow$ Δημιούργησε ένα Let Clause**

Var = properties_xpaths

expr= $\text{“xpath}_1\text{”,“xpath}_2\text{”,...,“xpath}_n\text{”}$ $\forall$ $xpath_i$ ϵ **mappings and**

$xpath_i^P \epsilon X_s$

Συνθήκη στη where δομή (func:xpath($**$N_P$**)=$ properties_xpaths)

}

Αλγόριθμος 11.3 Αλγόριθμος μετάφρασης Κατηγορημάτων

11.3.3 Μετάφραση Αντικειμένων

Η μετάφραση των αντικειμένων πραγματοποιείται από τον αλγόριθμο **objectTranslation** (Αλγόριθμος 11.4), ο οποίος δέχεται ως εισόδους μια βασική σχηματομορφή γράφου **(BGP)**, τις αντιστοιχήσεις**(mappings)** και τις συνδέσεις των μεταβλητών**(variablesBindings).**

Ανάλυση Αλγορίθμου:

- **Για κάθε αντικείμενο των τριπλέτων που είναι μεταβλητή .**
  - **Αν το κατηγόρημα της τριπλέτας είναι μεταβλητή.**

    Σε αυτή την περίπτωση, η δήλωση της μεταβλητής αντικειμένου στην XQuery θα μπορούσε να παραληφθεί καθώς συμπίπτει με την μεταβλητή του κατηγορήματος. Όμως, επιλέχτηκε η δήλωση της για λόγους συνέπειας μεταξύ των μεταβλητών των δυο ερωτήσεων. Επομένως, ορίζετε μια δομή Let για την απλή ανάθεση τιμής από την μεταβλητή του κατηγορήματος στην μεταβλητή του αντικειμένου. Η επιλογή της δομής

Let έγινε, καθώς τα κριτήρια επιλογής μεταξύ δομών For ή Let έχουν εφαρμοστεί στην μεταβλητή του κατηγορήματος.

- Η μεταβλητή (Var) της δομής Let έχει το όνομα της μεταβλητής του αντικειμένου της τριπλέτας. Με τον τρόπο αυτό, επιτυγχάνεται απόλυτη αντιστοιχία στα ονόματα των μεταβλητών στις δυο ερωτήσεις.
- Η έκφραση (expr) της δομής είναι η μεταβλητή που αντιστοιχεί(στο XQuery) στο κατηγόρημα της τριπλέτας.
- Στην περίπτωση την οποία για την μεταβλητή του κατηγορήματος, έχει επιλεχθεί η δομή Let ,τότε προστίθεται μια συνθήκης ελέγχου ανάθεσης τιμής για την μεταβλητή του αντικειμένου, στην where δομή ,με χρήση της built-in συνάρτησης exists. Με τον τρόπο αυτό, ελέγχετε η επαλήθευση της τριπλέτας στα XML δεδομένα. (για λεπτομέρειες ,βλέπε υπό-ενότητα 11.3.7)

- **Αν το κατηγόρημα της τριπλέτας είναι IRI .**

  Ορίζεται μια δομή For ή Let (For/Let).

  - Η μεταβλητή (Var) της δομής Let έχει το όνομα της μεταβλητής του αντικειμένου της τριπλέτας. Με τον τρόπο αυτό, επιτυγχάνεται απόλυτη αντιστοιχία στα ονόματα των μεταβλητών, στις δυο ερωτήσεις.
  - Η έκφραση (expr) της δομής αποτελείται από την ένωση των μονοπατιών που προκύπτουν, από την προσάρτηση στην μεταβλητή που έχει οριστεί (στο XQuery) για το υποκείμενο της τριπλέτας, των ονομάτων των τελευταίων κόμβων του συνόλου μονοπατιών της ιδιότητας του κατηγορήματος.
  - Στην περίπτωση την οποία για την μεταβλητή του αντικειμένου έχει επιλεχθεί η δομή Let ,τότε προστίθεται μια συνθήκης ελέγχου ανάθεσης τιμής για την μεταβλητή, στην where δομή ,με χρήση της built-in συνάρτησης exists. Με τον τρόπο αυτό, ελέγχεται η επαλήθευση της τριπλέτας στα XML δεδομένα.(για λεπτομέρειες ,βλέπε υπό-ενότητα 11.3.7)

- **Για κάθε αντικείμενο των τριπλέτων που είναι σταθερά.**

  Ορίζεται μια συνθήκη ισότητας στην where δομή (clause) της ερώτησης.

  - **Αν το κατηγόρημα της τριπλέτας είναι μεταβλητή**

  Σε αυτή την περίπτωση, η συνθήκη ισότητας ορίζεται μεταξύ της μεταβλητής του κατηγορήματος και της σταθεράς του αντικειμένου.

- **Αν το κατηγόρημα της τριπλέτας είναι IRI**

  Σε αυτή την περίπτωση, η συνθήκη ισότητας ορίζεται μεταξύ των κόμβων που προκύπτουν από την προσάρτηση στην μεταβλητής που έχει οριστεί (στο XQuery) για το υποκείμενο της τριπλέτας, των ονομάτων των τελευταίων κόμβων του συνόλου μονοπατιών της ιδιότητας του κατηγορήματος και της σταθεράς.

objectsTranslation(BGP , mappings , variablesBindings) {

$\forall$ **O** ϵ **V** ⇨ **IF P** ϵ **V** ⇨ **Δημιούργησε ένα Let Clause**

Var = N_O

expr=$ N_P

IF για το P δημιουργήθηκε Let Clause ⇨

Exist Condition

ELSE ⇨ **Δημιούργησε ένα For/Let Clause**

Var = N_O

Expr = $\$N_S/x_1$ **union** $\$N_S/x_2$ **union ... union** $\$N_S/x_n$ $\forall x_i \epsilon (X_s \oplus X_P)^{LN}$

IF δημιουργήθηκε Let Clause ⇨

Exist Condition

$\forall$ **O** ϵ **L** ⇨ **IF P** ϵ **V**

Δημιούργησε Συνθήκη ισότητας στο expr της For/Let δομής του P([.= "O"])

IF για το P δημιουργήθηκε Let Clause ⇨

Exist Condition for P

ELSE

Δημιούργησε Συνθήκη ισότητας $\forall x_i \epsilon X_S$ **στο expr της For/Let δομής του S**

($[./y_1 =$ "O" **or** $./y_2 =$ "O" **or ... or** $./y_n =$ "O"$]$ $\forall y_i \epsilon (x_i \oplus X_P)^{LN}$ **)**

IF για το S δημιουργήθηκε Let Clause ⇨

Exist Condition for S

}

Αλγόριθμος 11.4 Αλγόριθμος μετάφρασης Αντικειμένων

11.3.4 Μετάφραση Φίλτρων

Η μετάφραση των φίλτρων πραγματοποιείται από τον αλγόριθμο **filterTranslation** (Αλγόριθμος 11.5), ο οποίος δέχεται ως είσοδο μια βασική σχηματομορφή γράφου **(BGP).**

Ανάλυση Αλγορίθμου:

- **Για κάθε δομή φίλτρου (FILTER)**

  Ορίζεται μια συνθήκη ανάλογη με τον περιορισμό του φίλτρου στην where δομή (clause) της ερώτησης. Λόγω των περιορισμών που μπορεί να περιέχει το φίλτρο, δεν είναι δυνατόν πάντα να εφαρμοστούν στις Let/For δομές, ώστε να επιτευχθεί βελτιστοποίηση .Για τον λόγο αυτό και για λογούς συνέπειας ο περιορισμός εφαρμόζεται πάντα στην where δομή.

```
filtersTranslation ( BGP ){

    ∀ FILTER ⇨ Condition at Where Clause
}
```

Αλγόριθμος 11.5 Αλγόριθμος μετάφρασης Φίλτρων

11.3.5 Παραγωγής Return δομής

Η παραγωγή της return δομή των XQuery εκφράσεων γίνεται από τον αλγόριθμο **returnClauseBuilt** (Αλγόριθμος 11.6), ο οποίος δέχεται ως εισόδους τον τύπο της SPARQL ερώτησης **(queryType)** και τις μεταβλητές επιστροφής **(RV)**. Για τον ακριβή τρόπος σύνταξης της δομής return βλέπε υπό-ενότητα 11.5.

Ανάλυση Αλγορίθμου:

- **Για κάθε μεταβλητή που ανήκει στις μεταβλητές επιστροφής**

  Κάθε μεταβλητή της βασικής σχηματομορφής γράφου (BGP) που ανήκει στις μεταβλητές επιστροφής (Ορισμός 11.2) τοποθετείται στην return δομή του XQuery ερωτήματος. Για τις ερωτήσεις τύπου ASK στην δομή return τοποθετείται το "yes". (Για λεπτομέρειες σύνταξης της Return δομής ,βλέπε υπό-ενότητα 11.5)

```
returnClauseBuilt ( RV,  queryType) {
    IF Query Type is ASK
        Insert "yes" at Return Clause
    ELSE
        ∀ Variable ϵ BGP and ϵ RV (Return Variables) ⇨ Insert it at Return Clause
}
```

Αλγόριθμος 11.6 Αλγόριθμος παραγωγής Return δομής

Παρατήρηση 11.1 Όπως γίνεται αντιληπτό από την ανάλυση του αλγορίθμου, η σειρά με την οποία ορίζονται οι δομές For/Let στο ερώτημα XQuery δεν μπορεί να είναι τυχαία, καθώς στις δομές των μεταβλητών που βρίσκονται σε θέση αντικειμένου ή κατηγορήματος γίνεται χρήση των μεταβλητών των υποκειμένων. Για τον λόγο αυτό, ο αλγόριθμος ορίζει πρώτα τις For/Let δομές των υποκειμένων , στη συνέχεια τις δομές των μεταβλητών των κατηγορημάτων και τέλος τις δομές των μεταβλητών των αντικειμένων . Όπως θα αναλυθεί και στη συνέχεια αυτή η σειρά δήλωσης δεν είναι δυνατόν να ακολουθείται πάντα στην περίπτωση διαμοιραζόμενων μεταβλητών.

Σημείωση : *Για τα παρακάτω παραδείγματα στις περιπτώσεις επιλογής For ή Let δομής ,επιλέγεται αφηρημένα η ανάπτυξη For δομών (τα κριτήρια επιλογής For ή Let δομής αναλύονται στην υπό-ενότητα 11.3.6).*

Παράδειγμα 11.1

Από τις αντιστοιχήσεις που έχουν προκύψει, για την ερώτηση στο Παράδειγμα 10.1 ισχύει:

SELECT ?x ,?n
WHERE { ?x FirstName ?n }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation* για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation* για την μεταβλητή ?n :

for $n in $x/FirstName

Από τον Αλγόριθμο *returnClauseBuilt* η return δομή θα περιέχει τις μεταβλητές:

x, n

Παράδειγμα 11.2

Από τις αντιστοιχήσεις που έχουν προκύψει, για την ερώτηση στο Παράδειγμα 10.6 ισχύει:

```
SELECT  *
WHERE { ?S ?P ?O }
```

Βάση τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation* για την μεταβλητή ?S :

```
for $S in $doc/Persons union $doc/Persons/Person union
    $doc/Persons/Employee  union  $doc/Persons/Person/Address
        union  $doc/Persons/Employee/Address
```

Από τον Αλγόριθμο *predicateTranslation* για την μεταβλητή ?P :

```
for $P in  $S/* union  $S/@*
let $properties_xpaths:=( "/Person/Person/FirstName",
"/Person/Person/LastName",....)
where ( func:xpath($P)= $properties_xpaths)
```

Από τον Αλγόριθμο *objectTranslation* για την μεταβλητή ?O :

```
let $O := $P
```

Από τον Αλγόριθμο *returnClauseBuilt* η return δομή θα περιέχει τις μεταβλητές: **S, P, O**

Σημείωση : *Ως func:xpaths θεωρείται η συνάρτηση func:xpath(βλέπε Εικόνα 11.1) δέχεται ως όρισμα ένα σύνολο από κόμβους και επιστρέφει τα μονοπάτια τους. Η συνάρτηση έχει προκύψει από τροποποίηση συνάρτησης από το [23].*

```
declare function func:xpath ( $nodes as node()* )  as xs:string * {
        $nodes/string-join(ancestor-or-self::*/name(.) , '/' )
};
```

Εικόνα 11.1 :Συνάρτηση func:xpath για υπολογισμό των Xpaths κόμβων

Παράδειγμα 11.3

Από τις αντιστοιχήσεις που έχουν προκύψει, για την ερώτηση στο Παράδειγμα 10.4 ισχύει:

ASK

WHERE { ?x FirstName "John" }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation* για την μεταβλητή ?χ :

let $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation* για την σταθερά "John":

let $x in $doc/Persons/Person[./FirstName="John"] union

$doc/Persons/Employee[./FirstName="John"]

Από τον Αλγόριθμο *returnClauseBuilt* η return δομή θα περιέχει : **"yes"**

Παράδειγμα 11.4

Από τις αντιστοιχήσεις που έχουν προκύψει, για την ερώτηση στο Παράδειγμα 10.5 ισχύει:

PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?S vcard:FN "John"}
WHERE { ?x rdf:type Employee.

?S ?P "John" }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation* για την μεταβλητή ?S :

for $S in $doc/Persons/Employee

Από τον Αλγόριθμο *predicateTranslation* για την μεταβλητή ?P :

for $P in $S/* union $S/@*

Από τον Αλγόριθμο *objectTranslation* για την σταθερά "John":

for $P in $S/*[.="John"] union $S/@*[.="John"]

Από τον Αλγόριθμο *returnClauseBuilt* η return δομή θα περιέχει

την μεταβλητή: **S**

11.3.6 Κριτήρια Επιλογής For ή Let δομής (For/Let Clause)

Ένα από τα βασικότερα προβλήματα που καλείται να αντιμετώπιση ο αλγόριθμος, είναι τα XQuery ερωτήματα να παράγουν ακολουθίες λύσεων σύμφωνα με την σημασιολογία της SPARQL και χωρίς να είναι αναγκαία η περεταίρω επεξεργασία τους ώστε να επιτευχθούν οι επιθυμητές ακολουθίες λύσεων.

Η For δομή στην γλώσσα XQuery, πραγματοποιεί μια επανάληψη αναθέτοντας κάθε φορά στην μεταβλητή της δομής μια από τις τιμές που έχουν υπολογιστεί από την έκφραση(expr) της δομής. Από την άλλη, η δομή Let πραγματοποιεί την ανάθεση των τιμών που έχουν υπολογιστεί από την έκφραση(expr) στην μεταβλητή, χωρίς όμως να εκτελεί επανάληψη . Η βασική τους διαφοροποίηση είναι ότι με την δομή For οι τιμές που έχουν προκύψει από την έκφραση αναθέτονται μια-μια εκτελώντας επανάληψη, ενώ με την δομή Let γίνεται ανάθεση όλου του συνόλου των τιμών στην μεταβλητή.

Όπως έχει ήδη αναφερθεί, υπάρχουν εξαρτήσεις μεταξύ των μεταβλητών καθώς πολλές αποτελούν επεκτάσεις άλλων μεταβλητών. Αυτό, όπως είναι λογικό δημιουργεί και αλληλεξαρτήσεις στο επίπεδο των δεδομένων. Ένα πρόβλημα που προκύπτει είναι πως εξασφαλίζεται η ακολουθία λύσεων(solution sequence) να ταυτίζεται με την ακολουθία λύσεων που θα παρήγαγε η SPARQL ερώτηση, ακολουθώντας την σημασιολογία της SPARQL. Δηλαδή πως εξασφαλίζεται ο σωστός αριθμός αποτελεσμάτων και η σωστή αντιστοιχία στα αποτελέσματα, μεταξύ αλληλεξαρτωμένων μεταβλητών. Αυτό γίνεται φανερό και στα παρακάτω παραδείγματα.

Παράδειγμα 11.4

Έστω η ερώτηση : "Επέστρεψε τον τηλεφωνικό αριθμό, όλων των ατόμων (και εργαζομένων) "

SELECT ?t

WHERE { ?x Telephone ?t }

Από την υπό-ενότητα 5.5.1 έχει οριστεί η εξής αντιστοίχηση :

$\mathbf{X_{Telephone}}$ **=** { /Persons/Person/Telephone , /Persons/Employee/Telephone }

Άρα προκύπτει :

$\mathbf{X_x}$**= { /Persons/Person, /Persons/Employee }**

$\mathbf{X_t}$ **= ?x/ Telephone**

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει(αν επιλεγόταν από τον η δομή Let για τις μεταβλητές) :

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?x :

let $x := $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?t :

let $t := $x/Telephone

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.2) **-** Exist Condition

where (exists($t))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) η return δομή :

return ($t)

Το παραπάνω ερώτημα θα μεταβεί στο return μια φορά και θα επιστρέψει όλους τους τηλεφωνικούς αριθμούς όλων των ατόμων, ως μια απάντηση. Αυτό, δεν είναι σωστό καθώς το επιθυμητό είναι να επιστρέψει τόσες απαντήσεις, όσες και οι τηλεφωνικοί αριθμοί. Από την άλλη, δεν υπάρχει κάποιο πρόβλημα στις αντιστοιχίσεις μεταξύ των αποτελεσμάτων καθώς επιστρέφεται μόνο η μεταβλητή t.

Έστω ότι επιλεγόταν οι παρακάτω For/Let δομές από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* :

let $x := $doc/Persons/Person union $doc/Persons/Employee
for $t in $x/Telephone
return ($t)

Το παραπάνω ερώτημα θα μεταβεί στο return τόσες φορές, όσοι είναι και ο αριθμοί των τηλεφωνικών αριθμών επιστρέφοντας κάθε φορά έναν τηλεφωνικό αριθμό .Αυτό, είναι και το επιθυμητό. Επομένως σε αυτό το παράδειγμα, με την επιλογή της δομής For για την μεταβλητή επιστροφής, επιτυγχάνεται το σωστό αποτέλεσμα.

Σημείωση : *Η σύνταξη της δομής Return είναι απλοποιημένη και περιέχει μόνο μεταβλητές χωρίς να καθορίζεται η μορφή των αποτελεσμάτων ανάλογα με τον τύπο της μεταβλητής ,όπως αναλύθηκε στην υπό-ενότητα 8.3 .Για τις λεπτομέρειες υλοποίησης και σύνταξης της return δομής, βλέπε υπό-ενότητα 11.5 .*

Παράδειγμα 11.5

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) και τον/τους τηλεφωνικό/ους αριθμό/ους τους"

SELECT ?x , ?t
WHERE { ?x Telephone ?t }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει(αν επιλεγόταν από τον η δομή Let για τις μεταβλητές) :

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?x :

let $x := $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?t :

let $t := $x/Telephone

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) **-** Exist Condition

where (exists($t))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) η return δομή :

return ($x , $t)

Το παραπάνω ερώτημα θα μεταβεί στο return μια φορά και θα επιστρέψει όλα τα άτομα και στην συνέχεια όλους τους τηλεφωνικούς αριθμούς, χωρίς να υπάρχει καμιά αντιστοιχία μεταξύ τους. Καθώς η δομή let πραγματοποιεί απλή ανάθεση τιμής στην μεταβλητή χωρίς να εκτελεί επανάληψη. Επίσης, δεν είναι δυνατόν να βρεθούν οι αντιστοιχίες(με περεταίρω επεξεργασία) καθώς δεν είναι αναγκαία 1:1, εφόσον ένα άτομο μπορεί να μην έχει τηλεφωνικό αριθμό ή να έχει πάνω από έναν.

Έστω ότι επιλεγόταν οι παρακάτω For/Let δομές από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* :

for $x in $doc/Persons/Person union $doc/Persons/Employee
let $t := $x/Telephone
where (exists($t))
return ($x , $t)

Το παραπάνω ερώτημα θα μεταβεί στο return για κάθε άτομο, θα επιστρέφει το άτομο και όλους τους τηλεφωνικούς αριθμούς που αντιστοιχούν σε αυτό το άτομο (καθώς η δομή let πραγματοποιεί απλή ανάθεση τιμής στην μεταβλητή χωρίς να εκτελεί επανάληψη) πράγμα που δεν είναι σωστό, καθώς θα έπρεπε να επιστρέφει διαφορετικό αποτέλεσμα για κάθε συνδυασμό ατόμου και τηλεφωνικού αριθμού.

Έστω ότι επιλεγόταν οι παρακάτω For/Let δομές από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* :

```
let $x := $doc/Persons/Person union $doc/Persons/Employee
for $t in $x/Telephone
return ( $x , $t )
```

Το παραπάνω ερώτημα θα μεταβεί στο return τόσες φορές, όσα είναι τα τηλεφωνικά νούμερα όλων των ατόμων, και θα επέστρεφε όλα τα άτομα και έναν τηλεφωνικό αριθμό που όπως είναι φανερό ότι δεν είναι το επιθυμητό.

Έστω ότι επιλεγόταν οι παρακάτω For/Let δομές από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* :

```
for $x in $doc/Persons/Person union $doc/Persons/Employee
for $t in $x/Telephone
return ( $x , $t )
```

Το παραπάνω ερώτημα θα μεταβεί στο return τόσες φορές όσοι είναι και οι συνδυασμοί ατόμων και τηλεφωνικών αριθμών, επιστρέφοντας κάθε φορά ένα άτομο και έναν τηλεφωνικό αριθμό που αντιστοιχεί σε αυτό το άτομο.

Επομένως σε αυτό το παράδειγμα με την επιλογή της δομής For για τις μεταβλητές επιστροφής επιτυγχάνεται το σωστό αποτέλεσμα.

Παράδειγμα 11.6

Έστω η ερώτηση : " Για όλα τα άτομα (και εργαζόμενοι) επέστρεψε το/α μικρό/ά όνομα/τα του και τον/τους τηλεφωνικό/ους αριθμό/ους του"

```
SELECT ?n , ?t
WHERE { ?x FirstName ?n .
        ?x  Telephone  ?t . }
```

Έστω ότι επιλέγετε από τον αλγόριθμο *BGP2XQueryShareVarsAbsence* η δομή For για τις μεταβλητές επιστροφής :

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?x :

```
let  $x := $doc/Persons/Person union $doc/Persons/Employee
```

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?n :

```
for  $n in  $x/FirstName
```

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?t :

```
for  $t in   $x/Telephone
return ( $n , $t)
```

Το παραπάνω ερώτημα θα μεταβεί στο return τόσες φορές όσοι είναι και οι συνδυασμοί των μικρών ονομάτων με τους τηλεφωνικούς αριθμούς, επιστρέφοντας όλους τους συνδυασμούς μικρό όνομα και τηλεφωνικών αριθμών ασχέτως με το αν ανήκουν στο ίδιο άτομο.

Έστω ότι επιλεγόταν το εξής ερώτημα :

```
for  $x in $doc/Persons/Person union $doc/Persons/Employee
for  $n in $x/FirstName
for  $t in $x/Telephone
return ( $n , $t)
```

Το παραπάνω ερώτημα θα μεταβεί στο return τόσες φορές όσοι είναι και οι συνδυασμοί των μικρών ονομάτων με τους τηλεφωνικούς αριθμούς. Όμως, με την προϋπόθεση ότι το όνομα και ο τηλεφωνικός αριθμός αντιστοιχούν στο ίδιο άτομο, που είναι και το επιθυμητό .

Ορισμός 11.4 Επέκταση μεταβλητών Σε αναλογία με τον Ορισμό 10.1 για την επέκταση συνόλων μονοπατιών, μια XQuery μεταβλητή A ονομάζεται **επέκταση** της XQuery μεταβλητής B, εάν στην μεταβλητή A ανατίθενται τιμές των οποίων τα μονοπάτια είναι επέκτασης των μονοπατιών των τιμών της μεταβλητής B. Επεκτάσεις μεταβλητών μπορούν να δημιουργηθούν στην γλώσσα XQuery με την χρήση του συμβόλου **/** .

Σημείωση : *Λόγω της δενδρικής δομής των XML εγγράφων, γίνεται κατανοητό ότι τα δεδομένα που αντιστοιχούν στις μεταβλητές A και B , αλληλοεξαρτώνται.*

Παράδειγμα 11.7

Έστω η τριπλέτα:

?x FirstName ?n

Η μεταβλητή **n είναι επέκταση της x** , καθώς όπως έχει δειχθεί και στο Παράδειγμα 10.1 :

?n = FirstName = ?x / FirstName

Τα δεδομένα των n και x αλληλοεξαρτώνται

Έστω η τριπλέτα:

?S ?P ?O

Οι μεταβλητές **P και O είναι επεκτάσεις της S** , καθώς όπως έχει δειχθεί και στο Παράδειγμα 10.6 :

?O = ?P = ?S/* U ?S/@*

Τα δεδομένα των S, P και O αλληλοεξαρτώνται

Παράδειγμα 11.8

Έστω οι τριπλέτες:

?x Person ?y

?y ?p ?z

Η μεταβλητή **y είναι επέκταση της x .**

Οι μεταβλητές **p και z είναι επεκτάσεις της y.**

Επομένως οι μεταβλητές **y, p, z είναι επεκτάσεις της x .**

Τα δεδομένα των x, y, p και z αλληλοεξαρτώνται

Με βάση όλα τα παραπάνω προκύπτουν οι εξής κανόνες για την επιλογή της κατάλληλης δομής For ή Let .

- Ορίζεται **For** δομή για μια μεταβλητή V :
  - Αν η μεταβλητή **V** περιέχεται στις **μεταβλητές επιστροφής** .
  - Αν η κάποια μεταβλητή από τις **μεταβλητές επιστροφής**, αποτελεί **επέκταση** της μεταβλητή **V**.
- Ορίζεται **Let** δομή για μια μεταβλητή V :
  - Σε όλες τις άλλες περιπτώσεις
- Για τις SPARQL ερωτήσεις **ASK** μορφής, για όλες τις μεταβλητές : Ορίζονται **Let** δομές

Αλγόριθμος 11.7 Αλγόριθμος επιλογής For/Let

11.3.7 Έλεγχος ανάθεσης τιμής μεταβλητής (Exist Condition)

Σύμφωνα με την παρούσα προδιαγραφή της γλώσσας SPARQL, για κάθε βασική σχηματομορφή γράφων απαιτείται σε κάθε μεταβλητή να ανατίθεται τιμή(να μην είναι unbound) για όλες τις λύσεις , της ακολουθίας λύσεων(solution sequence). Ο Έλεγχος ανάθεσης τιμής μεταβλητής (Exist Condition) που εφαρμόζεται στον αλγόριθμο objectTranslation(Αλγόριθμος 11.4) διασφαλίζει την ανάθεση τιμών σε όλες τις μεταβλητές της βασικής σχηματομορφής γράφου για όλες τις λύσεις της ακολουθίας λύσεων.

Παρατήρηση 11.2 Όπως είναι γνωστό στην γλώσσα SPARQL, υπάρχουν περιπτώσεις στις οποίες οι μεταβλητές δεν έχουν αντιστοιχηθεί με κάποια τιμή, αυτές οι μεταβλητές ονομάζονται unbound. Σύμφωνα με την παρούσα προδιαγραφή της γλώσσας SPARQL, στην σύζευξη (Join) μεταβλητών έχουμε απόρριψη αποτελέσματος μόνο όταν υπάρχει σύγκρουση (conflict) στις τιμές των μεταβλητών (δηλαδή στις δυο μεταβλητές να έχουν αντιστοιχηθεί τιμές και να είναι διαφορετικές)και όχι στην περίπτωση όπου μια μεταβλητή είναι unbound(βλέπε Παράδειγμα 11.9). Εάν μια από τις δυο μεταβλητές της σύζευξης είναι unbound, το αποτέλεσμα θα έχει την τιμή της bound μεταβλητής (βλέπε Παράδειγμα 11.9). Στην περίπτωση που και οι δυο μεταβλητές της σύζευξης είναι unbound,

το αποτέλεσμα και πάλι δεν απορρίπτεται αλλά προκύπτει unbound. Τα παραπάνω έρχονται σε αντίθεση με την σύζευξη της σχεσιακής άλγεβρας στην περίπτωση null τιμής, όπου το αποτέλεσμα απορρίπτεται. Τα παραπάνω έχουν αναλυθεί και στα [13][14] (βλέπε Παράδειγμα 11.9).

Παρατήρηση 11.3 Σύμφωνα με την παραπάνω παρατήρηση, η αυστηρή διατύπωση του Ορισμού 2.11 είναι: **Συμβατές Αντιστοιχήσεις (Compatible Mappings)** Δυο αντιστοιχήσεις **μ_1 : V-> T** και **μ_2 : V-> T** είναι συμβατές αν για κάθε ?X ϵ dom(μ_1) ∩ dom(μ_2) ισχύει μ_1(?X) = μ_2(?X) ή μ_1(?X)=unbound ή μ_2(?X)=unbound , όπου unbound είναι η άγνωστη τιμή (σε αντιστοιχία με την τιμή null της σχεσιακής άλγεβρας), στην περίπτωση που στην μεταβλητή δεν έχει ανατεθεί κάποια τιμή από τα σύνολα B ,I και L. (βλέπε Παράδειγμα 11.9)

Παράδειγμα 11.9

Στο παράδειγμα εμφανίζονται οι διαφορές μεταξύ της πράξης της σύζευξης (Join $\triangleright\triangleleft$) στην σχεσιακή άλγεβρα και στην άλγεβρα της SPARQL, στις περιπτώσεις null και unbound τιμών αντίστοιχα.

Σύμφωνα με την σχεσιακή άλγεβρα για τον τελεστή σύζευξη ισχύει ($X_1 \neq X_2$):

- $X_1 \triangleright\triangleleft X_2$ = null
- $X_1 \triangleright\triangleleft$ null = null
- null $\triangleright\triangleleft X_2$ = null
- null $\triangleright\triangleleft$ null = null
- $X_1 \triangleright\triangleleft X_1 = X_1$

Σύμφωνα με την SPARQL άλγεβρα για τον τελεστή σύζευξη ισχύει ($X_1 \neq X_2$):

- $X_1 \triangleright\triangleleft X_2$ = unbound *(not Compatible mappings)*
- $X_1 \triangleright\triangleleft$ unbound = X_1 *(Compatible mappings)*
- unbound $\triangleright\triangleleft X_2 = X_2$ *(Compatible mappings)*
- unbound $\triangleright\triangleleft$ unbound = unbound *(Compatible mappings)*
- $X_1 \triangleright\triangleleft X_1 = X_1$ *(Compatible mappings)*

Στην περίπτωση των βασικών σχηματομορφών γράφων (BGP), σύμφωνα με την προδιαγραφή της γλώσσας SPARQL, δεν επιτρέπεται η ύπαρξη unbound μεταβλητών. Με αποτέλεσμα ο αλγόριθμος BGP2XQuery να μην χρειάζεται να ελέγχει τις περιπτώσεις unbound μεταβλητών. Από την άλλη η μη

δυνατότητα ύπαρξης unbound μεταβλητών, συνεπάγει ότι όλες οι τριπλέτες (και τα φίλτρα) της BGP πρέπει να επαληθεύονται (match) στα δεδομένα.

Με βάση τα παραπάνω, τα ερωτήματα XQuery που προκύπτουν από τον αλγόριθμο BGP2XQuery, θα πρέπει να επιστρέφουν αποτελέσματα μόνο στην περίπτωση την οποία, σε όλες τις μεταβλητές έχουν ανατεθεί τιμές από τα XML δεδομένα.

Όπως έχει ήδη αναφερθεί η δομή For στην γλώσσα XQuery πραγματοποιεί μια επανάληψη αναθέτοντας κάθε φορά στην μεταβλητή της δομής μια από τις τιμές που έχουν υπολογιστεί από την έκφραση(expr). Η δομή For πραγματοποιεί με έμμεσο τρόπο, τον έλεγχο ανάθεσης τιμής σε μια μεταβλητή. Καθώς στην περίπτωση την οποία δεν υπολογίζεται κάποια τιμή από την expr της δομής, δηλαδή δεν επιστρέφονται κόμβοι από τα μονοπάτια της expr, δεν εκτελείτε επανάληψη και επομένως δεν είναι δυνατή η μετάβαση στην Return δομή. Συνεπώς δεν επιστρέφονται δεδομένα από την ερώτηση.(από πιθανές άλλες μεταβλητές που περιέχονται στην Return δομή)

Από την άλλη, η δομή Let πραγματοποιεί την ανάθεση των τιμών που έχουν υπολογιστεί από την έκφραση(expr) στην μεταβλητή χωρίς όμως να εκτελεί επανάληψη. Αυτό έχει ως αποτέλεσμα, στην περίπτωση που δεν επιστρέφονται κόμβοι από τα μονοπάτια της expr, να αντιστοιχίζεται μια κενή ακολουθία στην μεταβλητή Var, ενώ σύγχρονος είναι δυνατή η μετάβαση στην Return δομή. Επομένως για τις δομές Let, απαιτείται ο έλεγχος αν στην μεταβλητή της δομής έχει ανατεθεί κάποια τιμή.

Σε κάθε τριπλέτα όπως έχει αναφερθεί, οι μεταβλητές του αντικειμένου και του κατηγορήματος προκύπτουν ως επεκτάσεις της μεταβλητής του υποκειμένου. Σε αυτή την περίπτωση για τις μεταβλητές της τριπλέτας, αρκεί ο έλεγχος ανάθεσης τιμής της μεταβλητή να εφαρμοστεί στην μεταβλητή του κατηγορήματος ή του αντικειμένου, στις περιπτώσεις ύπαρξης μεταβλητών στις θέσεις αυτές.

> **Παρατήρηση 11.1** Στην γενική περίπτωση (καλύπτει και τις περιπτώσεις ύπαρξης διαμοιραζόμενων μεταβλητών) ο έλεγχος ανάθεσης τιμής, σε μια ακολουθία μεταβλητών στην οποία η κάθε μεταβλητή προκύπτει από επέκταση της προηγούμενης, μπορεί να πραγματοποιηθεί εφαρμόζοντας τον έλεγχο στην τελευταία μεταβλητή της ακολουθίας (η οποία είναι επέκταση των άλλων και δεν αποτελεί επέκταση καμιάς) .

Ο έλεγχος ανάθεσης τιμής σε μια μεταβλητή εφαρμόζεται είτε από την δομή For, στην περίπτωση που έχει επιλεχθεί αυτή η δομή για την μεταβλητή, είτε από την προσθήκη μια συνθήκης της μορφής "exists ($var) " στην where δομή του ερωτήματος.

Η συνάρτηση exists είναι ενσωματωμένη (built-in) συνάρτηση της γλώσσας XQuery.

fn:exists($arg as item()*) as xs:Boolean

Ελέγχεται η τιμή του $arg , αν δεν είναι κενή ακολουθία, επιστρέφει true αλλιώς false.

Παράδειγμα 11.10

Από το Παράδειγμα 11.8

Έστω οι τριπλέτες:

?x Person ?y

?y ?p ?z

Οι μεταβλητές **y είναι επέκταση της x .**

Οι μεταβλητές **p και z είναι επεκτάσεις της y.**

Επομένως οι μεταβλητές **y, p, z είναι επεκτάσεις της x**

Έστω ότι απαιτείται έλεγχος ανάθεση τιμής στις μεταβλητές ?y και ?z , ο έλεγχος μπορεί να αντικαταστεί εφαρμόζοντας τον μόνο στην μεταβλητή ?z.

Παράδειγμα 11.11

Παραλλαγή του Παραδείγματος 11.1 με εφαρμογή του αλγορίθμου επιλογής For/Let (Αλγόριθμος 11.7) και του ελέγχου ανάθεσης τιμής μεταβλητής (Exist Condition) που αποτελεί μέρος του αλγορίθμου objectTranslation (Αλγόριθμος 11.4)

SELECT ?x
WHERE { ?x FirstName ?n }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?n :

let $n in $x/FirstName

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4)**-** Exist Condition

where (exists($n))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) η return δομή θα περιέχει τις μεταβλητές: **x, n**

Παράδειγμα 11.12

Παραλλαγή του Παραδείγματος 11.2 με εφαρμογή του αλγορίθμου επιλογής For/Let (Αλγόριθμος 11.7) και του ελέγχου ανάθεσης τιμής μεταβλητής (Exist Condition) που αποτελεί μέρος του αλγορίθμου objectTranslation (Αλγόριθμος 11.4)

SELECT ?S
WHERE { ?S ?P ?O }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?S :

for $S in $doc/Persons union $doc/Persons/Person union
$doc/Persons/Employee union $doc/Persons/Person/Address
union $doc/Persons/Employee/Address

Από τον Αλγόριθμο *predicateTranslation*(Αλγόριθμος 11.3) για την μεταβλητή ?P :

let $P in $S/* union $S/@*
let $properties_xpaths:=("/Person/Person/FirstName",
"/Person/Person/LastName",….)
where (func:xpath($P)= $properties_xpaths)

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?O :

let $O := $P

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) **-** Exist Condition

where (exists($O))

Από τον Αλγόριθμο *returnClauseBuilt* (Αλγόριθμος 11.6) η return δομή θα περιέχει τις μεταβλητές: **S, P, O**

Παράδειγμα 11.13

Παραλλαγή του Παραδείγματος 11.3 με εφαρμογή του αλγορίθμου επιλογής For/Let (Αλγόριθμος 11.7) και του ελέγχου ανάθεσης τιμής μεταβλητής (Exist Condition) που αποτελεί μέρος του αλγορίθμου objectTranslation (Αλγόριθμος 11.4)

SELECT ?x
WHERE { ?x FirstName "John" }

Από τον Αλγόριθμο *BGP2XQueryShareVarsAbsence* ισχύει:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

let $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την σταθερά "John":

let $x in $doc/Persons/Person[./FirstName="John"] union $doc/Persons/Employee[./FirstName="John"]

Από τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) **-** Exist Condition

where (exists($x))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) η return δομή θα περιέχει:

"yes"

11.4 Ο Αλγόριθμος BGP2XQuery Με Ύπαρξη Διαμοιραζόμενων Μεταβλητών

Στην παρούσα ενότητα θα γίνει η παρουσίαση και η ανάλυση του αλγορίθμου BGP2XQuery για τις περιπτώσεις ύπαρξης διαμοιραζόμενων μεταβλητών στην βασική σχηματομορφή γράφου. Ο αλγόριθμος διαφοροποιείται ανάλογα με την θέση(υποκείμενο–κατηγόρημα–αντικείμενο) των διαμοιραζόμενων μεταβλητών στις τριπλέτες σχηματομορφών.

Έστω μια βασική σχηματομορφή γράφων, αποτελείται από v τριπλέτες σχηματομορφών.

Οι πιθανές περιπτώσεις ύπαρξης διαμοιραζόμενων μεταβλητών είναι :

Έστω οι τριπλέτες : $S_1\ P_1\ O_1\ ,\ S_2\ P_2\ O_2\ ,\ \dots\ ,\ S_v\ P_v\ O_v$

- **Case 1 :** $S_1 \equiv S_2 \equiv \dots \equiv S_\mu$ **, με** $\mu \leq v$ **και** $S_1, S_2, \dots, S_\mu \in V$ Περίπτωση διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του υποκειμένου.
- **Case 2 :** $P_1 \equiv P_2 \equiv \dots \equiv P_\mu$ **, με** $\mu \leq v$ **και** $P_1, P_2, \dots, P_\mu \in V$ Περίπτωση διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του κατηγορουμένου.
- **Case 3 :** $O_1 \equiv O_2 \equiv \dots \equiv O_\mu$ **, με** $\mu \leq v$ **και** $O_1, O_2, \dots, O_\mu \in V$ Περίπτωση διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του αντικειμένου.
- **Case 4 :** $S_1 \equiv S_2 \equiv \dots \equiv S_\mu \equiv O_1 \equiv O_2 \equiv \dots \equiv O_\kappa$ **, με** $\mu,\kappa \leq v$ **και** $S_1, S_2, \dots, S_\mu, O_1, O_2 \dots, O_\kappa \in V$ Περίπτωση διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του υποκειμένου και σε κ σε θέση αντικειμένου.

Ανάλυση των περιπτώσεων ύπαρξης διαμοιραζόμενων μεταβλητών

Case 1 : $S_1 \equiv S_2 \equiv \dots \equiv S_\mu$ **, με** $\mu \leq v$ **και** $S_1, S_2, \dots, S_\mu \in V$ **.** Διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του υποκειμένου.

Στον Ορισμό 8.3 αναφέρεται ότι ο συνδυασμός μονοπατιών (Xpaths) και αρίθμηση κόμβων αποτελεί την ταυτότητα των στοιχείων και χαρακτηριστικών σε ένα XML έγγραφο. Σε συνδυασμό με το γεγονός ότι τα υποκείμενα τον τριπλετών αντιστοιχούν σε σύνθετα στοιχεία στο XML έγγραφο, για τα οποία δεν ορίζεται ισότητα (Παρατήρηση 8.2), η σύζευξη των υποκειμένων ανάγεται στην τομή των μονοπατιών. Όπως έχει αναφερθεί και πραγματοποιηθεί στην διαδικασία

ανάθεσης μεταβλητών. Συνεπώς σε αυτή την περίπτωση διαμοιραζόμενων μεταβλητών ο αλγόριθμος BGP2XQuery δεν διαφοροποιείται.

Case 1 :

Καμία διαφοροποίηση με τον αλγόριθμο *subjectsTranslation* *(Αλγόριθμος 11.2)*

Αλγόριθμος 11.8 Αλγόριθμος Μετάφρασης Διαμοιραζόμενων Υποκειμένων

Παράδειγμα 11.14

Έστω η ερώτηση : " Για όλα τα άτομα (και εργαζόμενοι) επέστρεψε το/α μικρό/ά όνομα/τα τους το/α επίθετο/α τους και τον/τους τηλεφωνικό/ους αριθμό/ούς τους"

```
SELECT ?Fn ,?Ln ?tel
WHERE { ?x FirstName ?Fn .
        ?x  LastName  ?Ln .
        ?x  Telephone  ?tel . }
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation* (Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *objectsTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?Fn :

for $Fn in $x/FirstName

Από τον Αλγόριθμο *objectsTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?Ln :

for $Ln in $x/LastName

Από τον Αλγόριθμο *objectsTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?tel :

for $tel in $x/Telephone

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) :

return ($Fn , $Ln , $tel)

Case 2 : $P_1 \equiv P_2 \equiv \ldots \equiv P_\mu$, με $\mu \leq v$ και $P_1, P_2, \ldots, P_\mu \in V$. Διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του κατηγορουμένου.

Οι μεταβλητές των κατηγορημάτων και αντικειμένων κάθε τριπλέτας, προκύπτουν ως επέκτασεις της μεταβλητής του υποκειμένου. Είναι λοιπόν αναγκαίο για κάθε τριπλέτα με διαφορετικό υποκείμενο στην οποία εμφανίζεται κοινή μεταβλητή κατηγορήματος, να ορίζεται μια δομή For/Let .

- Η μεταβλητή (Var) της δομής For/Let έχει το όνομα της μεταβλητής του κατηγορήματος της τριπλέτας, με την προσάρτηση σε αυτό ενός αριθμού n. Όπου $1<n<k$, με κ ο αριθμός των τριπλέτων με ιδία μεταβλητή κατηγορήματος και διαφορετική μεταβλητή υποκειμένου.
- Η έκφραση (expr) της For/Let δομής προκύπτει από την προσάρτηση στην μεταβλητή που έχει οριστεί (στο XQuery) για το υποκείμενο της τριπλέτας, των ονομάτων των τελευταίων κόμβων του συνόλου μονοπατιών του κατηγορήματος.
- Όταν υπάρχει ισότητα μεταβλητών που βρίσκονται στην θέση του κατηγορήματος και στις τριπλέτες που περιέχεται η μεταβλητή έχουν διαφορετικές μεταβλητές υποκειμένων, όπως ορίζεται παραπάνω ορίζονται οι ανάλογες for/let δομές με τις ανάλογες μεταβλητές που δημιουργούνται για την αναπαράσταση της μεταβλητής του κατηγορήματος. Στην **ειδική περίπτωση** στην οποία το σύνολο των μονοπατιών του κατηγορήματος ισούται με την ένωση περισσοτέρων από ενός συνόλου μονοπατιών ιδιοτήτων(δηλαδή τα μονοπάτια του κατηγορήματος αντιστοιχούν σε περισσότερες από μια ιδιότητες) .Όπως στην περίπτωση στην οποία τα μονοπάτια της μεταβλητής του κατηγορήματος δεν έχουν προσδιοριστεί επακριβώς, δηλαδή το X_p^{LN} της μεταβλητής είναι το * και @* (βλέπε Παράδειγμα 11.15). Ακόμα στην περίπτωση στην οποία με χρήση Οντο τριπλετών, η μεταβλητή του κατηγορήματος έχει αντιστοιχηθεί με μονοπάτια παραπάνω από μιας ιδιότητας. Τότε απαιτείται ο έλεγχος ότι όλες οι μεταβλητές(του XQuery) που αναφέρονται στην ιδία μεταβλητή κατηγορήματος, έχουν αντιστοιχηθεί σε τιμές από στοιχεία ή χαρακτηριστικά τα οποία ανήκουν στην ίδια ιδιότητα(της οντολογίας) σύμφωνα με τις αντιστοιχήσεις. Αυτό επιτυγχάνεται με την προσθήκη συνθηκών στην δομή where, οι οποίες ελέγχουν αν τα μονοπάτια όλων των μεταβλητών που αναφέρονται στην ιδία μεταβλητή κατηγορήματος, ανήκουν σε μονοπάτια που έχουν αντιστοιχηθεί στην ιδία ιδιότητας.

  Ο παραπάνω έλεγχος πραγματοποιείται ακολουθώντας την εξής διαδικασία:

  Έστω τα $P_1 \equiv P_2 \equiv \ldots \equiv P_\mu$, με $\mu \leq v$ ανήκουν σε τριπλέτες με $S_1 \neq S_2 \neq \ldots \neq S_\mu$. Αν η μεταβλητή P περιέχει μονοπάτια που περιέχονται σε διαφορετικές ιδιότητες ,δηλαδή $X_P = x_{Prop\_1} U x_{Prop\_2} U \ldots U x_{Prop\_k}$ όπου $x_{Prop\_i} \subseteq X_{Prop\_i}$,οπου X_{Prop_i} οι αντιστοιχήσεις που έχουν οριστεί για την ιδιότητα **prop_i** . Για κάθε ένα από τα x_{Prop_i} ορίζεται μια δομή let η οποία σαν μεταβλητή έχει το όνομα prop_i και σαν έκφραση (expr) έχει τα μονοπάτια που περιέχονται

στο x_{Prop_i} . Στην where δομή εφαρμόζεται ο έλεγχος αν τα μονοπάτια όλων των μεταβλητών που αναφέρονται στην ιδία μεταβλητή κατηγορήματος, ανήκουν σε μονοπάτια μιας από τις παραπάνω ιδιοτήτες (βλέπε Αλγόριθμος 11.10) .

Case 2 :

SharePredicateTranslation (BGP , mappings , variablesBindings) {

∀ **Διαφορετική μεταβλητή Υποκειμένου SK (1≤K≤μ)** ⇨

Δημιούργησε ένα For/Let Clause

Var = N_P_K

Expr = $\$N_{SK} / x_1$ **union** $\$N_{SK} / x_2$ **... union** $\$N_{SK} / x_n$ $\forall x_i \in X_P^{LN}$

- **IF** $X_P = y_{Prop\_1} \cup y_{Prop\_2} \cup \dots \cup y_{Prop\_k}$ όπου $y_{Prop\_i} \subseteq X_{Prop\_i}$,όπου X_{Prop_i} οι αντιστοιχήσεις που έχουν οριστεί για την ιδιότητα **prop_i.**

Equality Condition at where clause (Αλγόριθμος 11.10)

}

Αλγόριθμος 11.9 Αλγόριθμος Μετάφρασης Διαμοιραζόμενων Κατηγορημάτων

PredicateEqualityCondition {

- Έστω ν τριπλέτες με $P_1 \equiv P_2 \equiv \dots \equiv P_\mu$, με $\mu \le v$ και $S_1 \neq S_2 \neq \dots \neq S_\mu$ και $X_P = y_{Prop\_1} \cup y_{Prop\_2} \cup \dots \cup y_{Prop\_k}$ όπου $y_{Prop\_j} \subseteq X_{Prop\_i}$ $1 \le j \le k$,όπου X_{Prop_i} οι αντιστοιχήσεις που έχουν οριστεί για την ιδιότητα **prop_i.** ($\neg\exists$ $z_{Prop\_j} \supseteq y_{Prop\_j}$ με $z_{Prop\_j} \subseteq X_{prop\_i}$)
- $\forall$ y_{prop_j} ορίζεται μια let δομή .
- Έστω ότι η παραπάνω συνθήκη ισχύει για **κ** ιδιότητες

  let $prop_1 := y_{prop_1} **let $prop_2 :=** y_{prop_2} **... let $prop_k :=** y_{prop_k}
- Η συνθήκη ελέγχου ισότητας στην **where** δομή :

((xpaths($P_1) = $prop_1 and xpaths($P_2) = $prop_1 and … and xpaths($P_μ) = $prop_1) or

(xpaths($P_1) = $prop_2 and xpaths($P_2) = $prop_2 and … and xpaths($P_μ) = $prop_2) or … or

(xpaths($P_1) = $prop_k and xpaths($P_2) = $prop_k and … and xpaths($P_μ) = $prop_k))

}

Αλγόριθμος 11.10 Αλγόριθμος Ισότητας Μεταβλητών Κατηγορήματος

Σημείωση : *Ως xpaths θεωρείται η συνάρτηση func:xpath(βλέπε Εικόνα 11.1) δέχεται ως όρισμα ένα σύνολο από κόμβους και επιστρέφει τα μονοπάτια τους. Η συνάρτηση έχει προκύψει από τροποποίηση συνάρτησης από το [23].*

Παράδειγμα 11.15

Έστω η ερώτηση : " Επέστρεψε δυο "ομάδες" ατόμων, όπου η μια να έχει την σταθερά John σαν τιμή στην ιδία ιδιότητα που το η άλλη (ομάδα) έχει την τιμή George"

```
SELECT ?x , ?y
WHERE { ?x rdf:type  Person_Type
        ?y rdf:type Person_Type
        ?x   ?p     "John" .
        ?y   ?p "George" . }
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

```
for  $x in  $doc/Persons/Person
```

Από τον Αλγόριθμο *SharePredicateTranslation*(Αλγόριθμος 11.9) για την μεταβλητή ?p και τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.6) για την σταθερά "John": :

```
for $p_1 in  $x/*[.="John"]  union $x/@*[.="John"]
```

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?y :

```
for $y in  $doc/Persons/Person
```

Από τον Αλγόριθμο *SharePredicateTranslation*(Αλγόριθμος 11.9) για την μεταβλητή ?p και τον Αλγόριθμο *objectTranslation*(Αλγόριθμος 11.4) για την σταθερά "George" :

```
for $p_2 in  $y/*[.="George"]  union $y/@*[.="George"]
```

Από τον Αλγόριθμο *PredicateEqualityCondition*(Αλγόριθμος 11.10) :

```
let $prop_1 := ( "/Persons/Person/Telephone" )
let $prop_2 := ( "/Persons/Person/Age" )
let $prop_3 := ( "/Persons/Person/FirstName" )
let $prop_4 := ( "/Persons/Person/LastName" )
let $prop_5 := ( "/Persons/Person/NickName" )
```

where((func:xpath($p_1)=$prop_1 and func:xpath ($p_2)=$prop_1) or
(func:xpath ($p_1)=$prop_2 and func:xpath ($p_2)=$prop_2) or
(func:xpath ($p_1)=$prop_3 and func:xpath ($p_2)=$prop_3) or
(func:xpath ($p_1)=$prop_4 and func:xpath ($p_2)=$prop_4) or
(func:xpath ($p_1)=$prop_5 and func:xpath ($p_2)=$prop_5))

Βάση τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) :

return ($x , $y)

Case 3 : $O_1 \equiv O_2 \equiv \ldots \equiv O_\mu$, με $\mu \leq v$ και $O_1, O_2, \ldots, O_\mu \in V$. Διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του αντικειμένου

Σε αυτή την περίπτωση ο αλγόριθμος διαφοροποιείται ανάλογα με τον τύπο της διαμοιραζόμενης μεταβλητής.

- **Μεταβλητή Τύπου Στιγμιότυπου Κλάσης**

  Στην περίπτωση που στις μ τριπλέτες που εμφανίζουν κοινή μεταβλητή αντικειμένου, υπάρχουν τριπλέτες με μη μεταβλητό κατηγορούμενο (δηλαδή IRI), ορίζεται η μεταβλητή του αντικειμένου ως επέκταση του υποκειμένου της τριπλέτας με το σταθερό κατηγορήματα (με χρήση του BGP2XQuery), οι υπόλοιπες τριπλέτες με τα σταθερά κατηγορήματα αγνοούνται καθώς έχουν "ληφθεί υπόψη" κατά την διαδικασία ανάθεση μεταβλητών. Στην περίπτωση μη ύπαρξης τριπλέτας με σταθερό κατηγόρημα, η μεταβλητή του αντικειμένου ορίζεται με βάση των BGP2XQuery από οποιαδήποτε τριπλέτα. Στην συνέχεια εφαρμόζεται ένας έλεγχος ισότητας στην where δομή της XQuery ερώτησης. Από τις τριπλέτες με μεταβλητή στην θέση του κατηγορήματος (εάν υπάρχουν), ελέγχεται η ισότητα της πλήρης διαδρομής (xpath με αρίθμηση κόμβων) της μεταβλητής του αντικειμένου(που ορίστηκε με βάση τα παραπάνω) καθώς η πλήρης διαδρομή είναι η ταυτότητα (Identity) των σύνθετων στοιχείων, με τις πλήρες διαδρομές των μεταβλητών των κατηγορημάτων των μ τριπλέτων. Από τις τριπλέτες με IRI στην θέση του κατηγορήματος (εάν υπάρχουν), ελέγχεται η ισότητα της διαδρομής της μεταβλητής του αντικειμένου(που ορίστηκε με βάση τα παραπάνω) με τις διαδρομές που προκύπτουν από την προσάρτηση των κόμβων (^{LN}) του κατηγορήματος στο υποκείμενο τις κάθε τριπλέτας.

- **Μεταβλητή Τύπου Σταθεράς**

  Όπως έχει αναφερθεί και στην υπό-ενότητα 10.4 η ισότητα μεταξύ μεταβλητών τύπου σταθεράς είναι ανεξάρτητη, από τα μονοπάτια των μεταβλητών. Επίσης η περίπτωση διαμοιραζόμενων μεταβλητών τύπου σταθεράς στην θέση του αντικειμένου δεν γίνεται ο υπολογισμός των αντιστοιχήσεων μονοπατιών αυτών των μεταβλητών κατά την διαδικασία Σύνδεσης Μεταβλητών. Ο προσδιορισμός των μονοπατιών θα προέλθει από επέκταση των μονοπατιών των υποκειμένων.

  Επίσης, ένα θέμα που προέκυψε στην Παράγραφο 10.4, ήταν αν είναι το επιθυμητό σε αυτή την περίπτωση για τον προσδιορισμό των αντιστοιχήσεων, η πράξη της ένωσης αντί της τομής. Όπως πλέον γίνεται αντιληπτό η παραγωγή ενός συνόλου μονοπατιών με χρήση της ένωσης, δεν θα μπορούσε να αξιοποιηθεί καθώς οι μεταβλητές των αντικειμένων προκύπτουν από επέκταση των μεταβλητών των υποκειμένων.

  Αυτό που πραγματοποιεί ο αλγόριθμος είναι:

  Ορίζετε μια δομή For ή Let (For/Let) για την διαμοιραζόμενη μεταβλητή.

  - Η μεταβλητή (Var) της δομής For/Let έχει το όνομα της μεταβλητής του αντικειμένου της τριπλέτας. Με τον τρόπο αυτό, επιτυγχάνεται απόλυτη αντιστοιχία στα ονόματα των μεταβλητών, στις δυο ερωτήσεις.
  - Η έκφραση (expr) της For/Let δομής προκύπτει, από την συνθήκη ισότητας μεταξύ των μεταβλητών των κατηγορημάτων για τις τριπλέτες που έχουν μεταβλητή ως κατηγόρημα και μεταξύ των επεκτάσεων των υποκειμένων των τριπλετών για τις τριπλέτες που δεν έχουν μεταβλητή ως κατηγόρημα. Στην περίπτωση των κατηγορημάτων που δεν είναι μεταβλητές και το X^{LN} περιέχει παραπάνω από ένα κόμβο, δημιουργείτε η ένωση όλων των ισοτήτων όλων των συνδυασμών (βλέπε Παράδειγμα 11.19).

Θεωρούνται τα σύνολα:

μ_triples : που περιέχει μ τριπλέτες όπου εμφανίζονται η διαμοιραζόμενη μεταβλητή.

triples_μI = **μ_triples| P ϵ I**

triples_μV= **μ_triples| P ϵ V**

Case 3 :

ShareObjectTranslation(BGP , mappings , variablesBindings) {

IF O είναι μεταβλητή Τύπου Σταθεράς

- **Δημιούργησε ένα For/Let Clause**

  Var = N_O

  Expr = $X_1[.=X_2[\ldots\ [.=X_\mu]]\ldots]$

  IF κατηγόρημα τις i τριπλέτας είναι μεταβλητή

  $X_i = \$P_i$

  ELSE

  $X_i = (\$S_i/X_{K\_1}$ **union** $\$S_i/X_{K\_2}$ **union ...union** $\$S_i/X_{K\_n}) \ \forall\ X_{K\_i} \epsilon\ (X_{si} \oplus X_{Pi})^{LN}$

  IF Let Clause => Exist Condition

ELSE

IF triples_μI≠Ø

Από μια τριπλέτα **t ϵ triples_μI** δημιούργησε ένα For/Let Clause **Var = N_O**

Expr = (Εφαρμόζοντας τον BGP2XQuery για περίπτωση απουσίας διαμοιραζομένων μεταβλητών)

ELSE

Από μια τριπλέτα **t ϵ triples_μV** δημιούργησε ένα Let Clause

Var = N_O

expr=$ N_P

Συνθήκη στην where δομή :

- Έλεγχος ισότητας του πλήρης μονοπατιού μεταξύ της μεταβλητής αντικειμένου της τριπλέτας t και των μεταβλητών των κατηγορημάτων :

  func:nodeURI($No)=func:nodeURI($Np1)and…and func:nodeURI($No)= func:nodeURI ($Npi) $\forall$ pi ϵ triples_μV-t.

- Έλεγχος :

  func:xpath($No)=func:xpath($S_1/X_{K_1}$ union $\$S_{i1}X_{K\_2}$ union...union $\$S_1/X_{K\_n} \forall\ X_{K\_i1} \epsilon\ (X_{s1} \oplus X_{P1})^{LN}$) and …and
  func:xpath($No)=func:xpath($\$S_i/X_{K_1}$ union $\$S_i/X_{K\_2}$ union...union $\$S_i/X_{K\_n} \forall\ X_{K\_i} \epsilon\ (X_{si} \oplus X_{Pi})^{LN}$) $\forall\ S_i \epsilon$ triples_μI-t

}

Αλγόριθμος 11.11 Αλγόριθμος Μετάφρασης Διαμοιραζόμενων Αντικειμένων

Παράδειγμα 11.16

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενους) τα οποία έχουν το ίδιο ψευδώνυμο (Nickname) και μικρό όνομα"

```
SELECT ?x
WHERE { ?x  FirsName ?n .
        ?x NickName ?n . }
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

```
for  $x in  $doc/Persons/Person union $doc/Persons/Employee
```

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) για την μεταβλητή ?n

```
let $n :=  $x/FirstName[.=$x/NickName]
```

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) για την μεταβλητή ?n **-** Exist Condition

```
where ( exists($n) )
```

Από τον Αλγόριθμο *returnClauseBuilt* (Αλγόριθμος 11.6):

```
return ( $x )
```

Παράδειγμα 11.17

Έστω η ερώτηση : " Επέστρεψε δυο "ομάδες" ατόμων (και εργαζομένων) οι οποίες η μια έχει το ίδιο ψευδώνυμο (Nickname) με το μικρό όνομα της άλλης"

```
SELECT ?x ?y
WHERE {?x  FirsName ?n .
       ?y NickName ?n . }
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $y in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) για την μεταβλητή ?n

let $n := $x/FirstName[.=$y/NickName]

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) **-** Exist Condition :

where (exists($n))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) :

return ($x)

Παράδειγμα 11.18

Έστω παραλλαγή της παραπάνω ερώτησης : " Επέστρεψε δυο "ομάδες" ατόμων (και εργαζομένων) οι οποίες η δεύτερη έχει σαν τιμή κάποια ιδιότητας την το μικρό όνομα της άλλης"

SELECT ?x ?y
WHERE {?x FirsName ?n .
?y ?p ?n . }

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?y :

for $y in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *predicateTranslation*(Αλγόριθμος 11.3) για την μεταβλητή ?p :

let $p := $y/* union $y/@*

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) για την μεταβλητή ?n

let $n := $x/FirstName[.=$p]

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) **-** Exist Condition :

where (exists($n))

Από τον Αλγόριθμο *returnClauseBuilt* (Αλγόριθμος 11.6):

return ($x)

Παράδειγμα 11.19

Έστω η ερώτηση : " Επέστρεψε το επίθετο ατόμων(και εργαζομένων) το οποίο εμφανίζεται και σαν όνομα διεύθυνσης (street) (ή δρόμου (road)) "

```
SELECT ?n
WHERE {?x  LastName ?n .
       ?a      Street    ?n . }
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?χ :

for $x in $doc/Persons/Person union $doc/Persons/Employee

Από τον Αλγόριθμο *subjectsTranslation*(Αλγόριθμος 11.2) για την μεταβλητή ?a :

for $a in $doc/Persons/Person/Address union

$doc/Persons/Employee/Address

Από τον Αλγόριθμο *ShareObjectTranslation*(Αλγόριθμος 11.11) για την μεταβλητή ?n

for $n in $x/LastName[.=$a/Street union $a/Road]

Από τον Αλγόριθμο *returnClauseBuilt* (Αλγόριθμος 11.6):

return ($n)

Case 4 : $S_1 \equiv S_2 \equiv \ldots \equiv S_\mu \equiv O_1 \equiv O_2 \equiv \ldots \equiv O_\kappa$, με $\mu,\kappa \leq v$ και $S_1, S_2, \ldots, S_\mu, O_1, O_2 \ldots, O_\kappa \in V$
Διαμοιραζόμενη μεταβλητή σε μ τριπλέτες στην θέση του υποκειμένου και σε κ σε θέση αντικειμένου.

Σε αυτή την περίπτωση ο αλγόριθμος ορίζει μια For/Let δομή για τις διαμοιραζόμενες μεταβλητές στην θέση του αντικειμένου με βάση αυτά που ορίζονται στο c. παραλείποντας τις μεταβλητές που βρίσκονται στην θέση υποκειμένου.

Όπως γίνεται αντιληπτό σε αυτή την περίπτωση η σειρά δήλωσης των δομών For/Let δεν μπορεί να ακολουθεί την σειρά που αναφέρεται στην Παρατήρηση 11.1 . Ο λόγος είναι ότι επιβάλλεται να γίνουν πρώτα οι δηλώσεις των δομών για τις μεταβλητές τις τριπλέτας που εμφανίζει την διαμοιραζόμενη μεταβλητή ως αντικείμενο και στην συνέχεια οι δηλώσεις των δομών για τις μεταβλητές των κατηγορημάτων και αντικειμένων των τριπλέτων που εμφανίζουν την διαμοιραζόμενη μεταβλητή στην θέση του υποκειμένου.(βλέπε Παράδειγμα 11.21)

Case 4 :

ShareSubjectObjectTranslation(BGP , mappings , variablesBindings) {

Δημιούργησε ένα For/Let Clause για ένα O_i $1 \leq i \leq k$ (εφαρμόζοντας των BGP2XQuery)

- **Ignore S_1 , S_2 , S_μ**
- **Ορθής σειράς δήλωσης των for/let clauses, εφαρμόζοντας πρώτα των αλγόριθμο BGP2XQuery στην τριπλέτα που περιέχει το O_i**

}

Αλγόριθμος 11.12 Αλγόριθμος Μετάφρασης Διαμοιραζόμενων Υποκειμένων- Αντικειμένων

Παράδειγμα 11.20

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (οχι και εργαζόμενους) και το/α μικρό/α όνομα/τα τους "

```
SELECT ?x ?n
WHERE { ?x    FirsName ?n .
        ?y    Person   ?x  .}
```

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *ShareSubjectObjectTranslation*(Αλγόριθμος 11.12) για την μεταβλητή ?y :

for $y in $doc/Persons

Από τον Αλγόριθμο *ShareSubjectObjectTranslation*(Αλγόριθμος 11.12) για την μεταβλητή ?x :

for $x in $y/Person

Από τον Αλγόριθμο *objectsTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?n :

for $n in $x/FirstName

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) :

return ($x , $n)

Παράδειγμα 11.21

Έστω η παραλλαγή της παραπάνω ερώτησης : " Επέστρεψε όλα τα Persons"

SELECT ?y
WHERE { ?x ?p ?n .
?y Person ?x . }

Με χρήση του αλγόριθμου *BGP2XQuery* παράγεται XQuery ερώτημα:

Από τον Αλγόριθμο *ShareSubjectObjectTranslation*(Αλγόριθμος 11.12) για την μεταβλητή ?y :

for $y in $doc/Persons

Από τον Αλγόριθμο *ShareSubjectObjectTranslation*(Αλγόριθμος 11.12) για την μεταβλητή ?x :

let $x := $y/Person

Από τον Αλγόριθμο *predicatesTranslation*(Αλγόριθμος 11.3) για την μεταβλητή ?p :

let $p := $x/* union $x/@*

Από τον Αλγόριθμο *objectsTranslation*(Αλγόριθμος 11.4) για την μεταβλητή ?n :

let $n := $p

Από τον Αλγόριθμο *objectTranslation*για **-** Exist Condition :

where (exists($n))

Από τον Αλγόριθμο *returnClauseBuilt*(Αλγόριθμος 11.6) :

return ($x)

11.5 Μορφή και Δομή των Αποτελεσμάτων των XQuery Ερωτήσεων

Όπως έχει αναφερθεί στην return δομή των XQuery ερωτήσεων που προκύπτουν από την εφαρμογή του αλγόριθμο BGP2XQuery περιέχονται οι μεταβλητές οι οποίες ανήκουν στις μεταβλητές επιστροφής και περιλαμβάνονται στην βασική σχηματομορφή γράφων(BGP) που εφαρμόζεται ο αλγόριθμος.

Η εκτέλεση μια ερώτησης XQuery σε XML δεδομένα, επιστρέφει ένα σύνολο αποτελεσμάτων (Result Set). Η μορφή και η δομή των αποτελεσμάτων εξαρτάται από την σύνταξη της return δομής.

11.5.1 Δομή των Αποτελεσμάτων

Στην παρούσα προσέγγιση είναι απαραίτητο για το σύνολο αποτελεσμάτων, να είναι εφικτή η ανάγνωση και επεξεργασία τους ανεξάρτητα από το XQuery engine που εκτέλεσε την ερώτηση. Όπως γίνεται αντιληπτό η δομή των αποτελεσμάτων πρέπει να παρέχει την δυνατότητα διαχωρισμού ενός αποτελέσματος από ένα σύνολο αποτελεσμάτων, όπως επίσης να είναι δυνατός ο προσδιορισμός των αντιστοιχήσεων μεταξύ αποτελέσματος και μεταβλητών.

Για την επίτευξη των παραπάνω είναι αναγκαίο τα δεδομένα τα οποία επιστρέφονται από τις XQuery ερωτήσεις να έχουν κάποια καθορισμένη δομή. Για τον λόγο αυτό επιλέχτηκε οι ερωτήσεις να επιστρέφουν τα αποτελέσματα με την ημί-δομημένη XML μορφή. Με τον τρόπο αυτό τα ημί-δομημένα δεδομένα που επιστρέφουν οι ερωτήσεις, μπορούν εύκολα να επεξεργαστούν. Ένα αποτέλεσμα από το σύνολο αποτελεσμάτων επιστρέφονται σε μορφή XML κάθε φορά που εκτελείται η δομή return.

Η XML δομή του κάθε αποτελέσματος είναι καθορισμένη, ως ριζικό στοιχείο(root element) ορίζεται το Result. Με τον τρόπο αυτό επιτυγχάνεται ο διαχωρισμός του αποτελέσματος από ένα σύνολο αποτελεσμάτων. Ως παιδιά του στοιχείου Result, ορίζονται στοιχεία με τα ονόματα των μεταβλητών που περιέχονται στο return, το περιεχόμενο (content) των στοιχείων προκύπτει από το περιεχόμενο των αντίστοιχων μεταβλητών. Με τον τρόπο αυτό γίνεται δυνατή η αντιστοιχία αποτελέσματος με μεταβλητές. Τέλος όλα τα παραπάνω σε απόλυτη ανεξαρτησία από το περιβάλλον εκτέλεσης και το XQuery engine.

Η σύνταξη της return δομής :

return (<Result>{<x>{…}</x> , <y>{…}</y> , <z>{…}</z> , … }</Result>)

Η δομή του συνόλου αποτελεσμάτων μετά την εκτέλεση της ερώτησης :

<Result> <x>x_1</x> <y>y_1</y> <z>z_1</z> ... </Result>
<Result> <x>x_2</x> <y>y_2</y> <z>z_2</z> ... </Result>
<Result> <x>x_3</x> <y>y_3</y> <z>z_3</z> ... </Result>
....
....
<Result> <x>x_n</x> <y>y_n</y> <z>z_n</z> ... </Result>

11.5.2 Μορφή των Αποτελεσμάτων

Όπως έχει αναλυθεί και στην υπό-ενότητα 8.3 η μορφή των αποτελεσμάτων της εκάστοτε μεταβλητής εξαρτάται από τον τύπο της. Πιο αναλυτικά :

a. Για τις μεταβλητές που έχουν προσδιοριστεί ως **Μεταβλητή Τύπου Στιγμιότυπου Κλάσης – CIVT** , η τιμή των αποτελεσμάτων αυτού του τύπου μεταβλητών, προκύπτει από την συνένωση της διεύθυνση URI, του XML εγγράφου που βρίσκεται ο κόμβος που έχει ανατεθεί στην μεταβλητή, με την πλήρη διαδρομή του κόμβου μέσα στο έγγραφο(Ως πλήρης ονομάζεται η χρήση Xpaths με αρίθμηση κόμβων). Με αυτό τον τρόπο δημιουργείται ένας XPointer[57] για τον κόμβο. Η μορφή των αποτελεσμάτων για αυτού του τύπου μεταβλητές θα είναι της μορφής:

XMLDocumentURI # xpointer(ExplictPath)

Έστω πχ

www.music.tuc.gr/xmlDoc_1.xml#xpointer(PERSONS/Employee[3]/FirstName[1])

Για την παραγωγή αποτελεσμάτων της παραπάνω μορφής έχει αναπτυχθεί η XQuery συνάρτηση:

func:nodeURI ($arg as node()?) as xs:string (Εικόνα 11.2)

Η συνάρτηση έχει προκύψει από τροποποίηση συνάρτησης από το [23] . Η συνάρτηση δέχεται ως όρισμα μια μεταβλητή και επιστρέφει ως συμβολοσειρά την "πλήρη" διεύθυνση του κόμβου που έχει ανατεθεί στην μεταβλητή.

Για να μπορεί στην συνέχεια να επιτευχθεί ο μετασχηματισμός των αποτελεσμάτων, σύμφωνα με την XML μορφοποίησης (SPARQL Query Results XML Format) [56] , που προτείνεται από τον W3C για την αναπαράσταση και αποθήκευση των αποτελεσμάτων των SPARQL ερωτήσεων. (για

περισσότερες λεπτομέρειες βλέπε Κεφάλαιο 16).Είναι απαραίτητο να μπορεί γίνει η διάκριση, μεταξύ των αποτελεσμάτων που αποτελούν IRI τιμές και των αποτελεσμάτων που αποτελούν σταθερές (literal) τιμές, Για τον λόγο αυτό η συνάρτηση func:nodeURI επιστρέφει τα αποτελέσματα περιβαλλόμενα από τα XML tags <IRI> </IRI> . Επομένως το παραπάνω URI θα επιστρεφόταν από την συνάρτηση με την μορφή :

<IRI> www.music.tuc.gr/xmlDoc_1.xml#xpointer(PERSONS/Employee[3]/FirstName[1]) </IRI>

```
declare function func:index-of-node
        ( $nodes as node()* , $nodeToFind as node() )  as xs:integer* {
        for $seq in (1 to count($nodes))
        return $seq[$nodes[$seq] is $nodeToFind]
} ;
declare function func:nodeURI  ( $node as node()? )  as xs:string{
        concat("&lt;IRI&gt;",base-uri($node),"#",string-join(
                for $ancestor in $node/ancestor-or-self::*
                let $sibsOfSameName := $ancestor/../*[name() = name($ancestor)]
                return  concat(  name($ancestor),
                        if (count($sibsOfSameName) <= 1)   then ''
                        else concat(  '[',func:index-of-node($sibsOfSameName,$ancestor),']'))
                                        , '/'), "&lt;/IRI&gt;")
 } ;
```

Εικόνα 11.2 : Η συνάρτηση func:nodeURI για τον προσδιορισμό URI XML κόμβου

b. Για τις μεταβλητές που έχουν προσδιοριστεί ως : **Μεταβλητή Ιδιότητας Τύπου Δεδομένων** ,**Μεταβλητή Ιδιότητας Αντικειμένων** και **Μεταβλητή Ιδιότητας Αγνώστου Τύπου**. Σε αντίθεση με τις μεταβλητές Τύπου Στιγμιότυπου Κλάσης όπου επιστρέφεται η ακριβής διεύθυνση του συγκεκριμένου κόμβου. Στις μεταβλητές τύπων ιδιοτήτων το επιθυμητό είναι να επιστραφεί το μονοπάτι του οποίου οι κόμβοι αντιστοιχούν σε αυτή την μεταβλητή, επομένως δεν είναι απαραίτητη η αρίθμηση κόμβων. Η τιμή των αποτελεσμάτων αυτών των τύπων μεταβλητών, προκύπτει από την συνένωση της διεύθυνση URI, του XML εγγράφου που βρίσκεται ο κόμβος που έχει ανατεθεί στην μεταβλητή, με το μονοπάτι του κόμβου(χωρίς αρίθμηση κόμβων), που έχει ανατεθεί στην μεταβλητή.

Η μορφή το αποτελεσμάτων για αυτού του τύπου μεταβλητές θα είναι της μορφής :

XMLDocumentURI # xpointer(XPath)

Έστω πχ

www.music.tuc.gr/xmlDoc_1.xml#xpointer(PERSONS/Employee/FirstName)

Για την παραγωγή αποτελεσμάτων της παραπάνω μορφής έχει αναπτυχθεί η XQuery συνάρτηση:

func:predURI ($arg as node()?) as xs:string (Εικόνα 11.3)

Η συνάρτηση έχει προκύψει από τροποποίηση συνάρτησης από το [23] . Η συνάρτηση δέχεται ως όρισμα μια μεταβλητή και επιστρέφει ως συμβολοσειρά την διεύθυνση του κόμβου που έχει ανατεθεί στην μεταβλητή.

Για να μπορεί στην συνέχεια να επιτευχθεί ο μετασχηματισμός των αποτελεσμάτων, σύμφωνα με την XML μορφοποίησης (SPARQL Query Results XML Format) [56] , που προτείνεται από τον W3C για την αναπαράσταση και αποθήκευση των αποτελεσμάτων των SPARQL ερωτήσεων. (για περισσότερες λεπτομέρειες βλέπε κεφάλαιο 16).Είναι απαραίτητο να μπορεί γίνει η διάκριση, μεταξύ των αποτελεσμάτων που αποτελούν IRI τιμές και των αποτελεσμάτων που αποτελούν σταθερές (literal) τιμές, Για τον λόγο αυτό η συνάρτηση func:predURI επιστρέφει τα αποτελέσματα περιβαλλόμενα από τα XML tags <IRI> </IRI> . Επομένως το παραπάνω URI θα επιστρεφόταν από την συνάρτηση με την μορφή :

<IRI> www.music.tuc.gr/xmlDoc_1.xml#xpointer(PERSONS/Employee/FirstName) </IRI>

```
declare function func: predURI ( $nodes as node()* )  as xs:string * {

concat("&lt;IRI&gt;",base-uri($nodes),"#",$nodes/string-join(ancestor-or-self::*/name(.) , '/')    ,"&lt;/IRI&gt;")

};
```

Εικόνα 11.3 : Η συνάρτηση func:predURI για τον προσδιορισμό URI XML κόμβου κατηγορήματος

c. Για τις μεταβλητές που έχουν προσδιοριστεί ως **Μεταβλητή Τύπου Σταθεράς**, η τιμή των αποτελεσμάτων αυτού του τύπου μεταβλητών, προκύπτει από την τιμή του περιεχομένου του κόμβου. Η τιμή αυτή υπολογίζεται από την built-in XQuery συνάρτηση :

fn:string(**$arg as item()?**) **as xs:string**

Η συνάρτηση δέχεται ως όρισμα μια μεταβλητή και επιστρέφει την συμβολοσειρά του περιεχομένου του κόμβου, που έχει ανατεθεί στην μεταβλητή.

d. Για τις μεταβλητές που έχουν προσδιοριστεί ως **Αγνώστου Τύπου Μεταβλητή**, η μορφή των αποτελεσμάτων εξαρτάται από το είδος του XML κόμβου που έχει αντιστοιχηθεί η μεταβλητή . Για τα σύνθετα στοιχεία η μορφή των αποτελεσμάτων είναι ιδία με τις μεταβλητές τύπου στιγμιότυπου κλάσης (a), ενώ για τα απλά στοιχεία ή γνωρίσματα τα αποτελέσματα έχουν ιδία μορφή με τις μεταβλητές τύπου σταθεράς (c).

Για τον προσδιορισμό του τύπου του κόμβου που έχει ανατεθεί στην μεταβλητή, έχει αναπτυχθεί μια XQuery συνάρτηση (Εικόνα 11.4):

func:UnknownVarType ($arg as node()?) as xs:string (Εικόνα 11.4)

Η συνάρτηση δέχεται ως όρισμα μια μεταβλητή και ελέγχει αν η μεταβλητή αυτή περιέχει κόμβο σύνθετου στοιχείου. Ο έλεγχος αυτός πραγματοποιείται ελέγχοντας αν το στοιχείο του κόμβου περιέχει άλλα στοιχεία ή/και χαρακτηριστικά . Στην περίπτωση που αντιστοιχεί σε σύνθετο στοιχείο, καλείται η συνάρτηση **func:nodeURI** , ενώ σε αντίθετη περίπτωση καλείται η συνάρτηση **fn:string.**

```
declare function func:unKnownTypeVar( $node as node() )  as xs:string  {
        return (
                if( exists($node/*) or  exists($node/@*))then
                        func:nodeURI($node)
                else
                fn:string ($node)
                )
} ;
```

Εικόνα 11.4 : Η συνάρτηση func:UnknownVarType για την επιστροφή αποτελεσμάτων για μεταβλητές άγνωστου τύπου

Σημείωση : *Όλα τα παραπάνω ισχύουν για τις περιπτώσεις οπού ο τύπος της SPARQL ερώτησης είναι SELECT,DESCRIBE,CONSTRUCT. Στην περίπτωση των ASK ερωτήσεων η return δομή περιλαμβάνει* μόνο *το "yes".*

Παράδειγμα 11.22

Έστω ότι προκύπτει μια XQuery ερώτηση από τον αλγόριθμο BGP2XQuery, και περιλαμβάνει στην return δομή τις μεταβλητές : **$x**, **$y**, **$z**, **$k**. Για τις μεταβλητές έχουν προσδιοριστεί οι τύποι **: $x** Μεταβλητή Τύπου Στιγμιότυπου Κλάσης, **$y** Μεταβλητή Ιδιότητας Τύπου Δεδομένων ,**$z** Μεταβλητή Τύπου Σταθεράς και **$k** Αγνώστου Τύπου Μεταβλητή . Η return δομή θα έχει την εξής σύνταξη :

return (<Result>{<x>{ func:nodeURI ($x)}</x> , <y>{func:predURI($y)}</y> , <z>{string($k)}</z> , <k>{func:UnknownVarType ($k)}</k> }</Result>)

11.6 Επίλογος

Στο κεφάλαιο αυτό, παρουσιάστηκε και αναλύθηκε ο αλγόριθμος για την μετάφραση βασικών σχηματομορφών γράφων σε σημασιολογικά ισοδύναμες XQuery εκφράσεις. Ο αλγόριθμος και η ανάλυση του χωρίστηκαν σε δυο μέρη, στο πρώτο μέρος παρουσιάζεται ο αλγόριθμος στην περίπτωση απουσίας διαμοιραζόμενων μεταβλητών, ενώ στο δεύτερο μέρος αναλύεται ο αλγόριθμος στην περίπτωση ύπαρξης διαμοιραζόμενων μεταβλητών. Τέλος παρουσιάστηκε η δομή και η μορφή των αποτελεσμάτων που παραγάγουν τα XQuery ερωτήματα. Η δομή των αποτελεσάντων παίζει καθοριστικό ρόλο στην δυνατότητα εφαρμογής των τελεστών AND και OPT (όπως αναλύονται στο επόμενο κεφάλαιο) για την μετάφραση των σχηματομορφών γράφων.

Στην εισαγωγή του κεφαλαίου τέθηκαν κάποιοι στόχοι για τους οποίους έγινε προσπάθεια να επιτευχθούν κατά την σχεδίαση των αλγορίθμων όπως: ανάπτυξη μια γενικής και κατανοητής διαδικασίας , αυστηρή τήρηση της σημασιολογίας, όσο το δυνατόν μικρότερα και λιγότερο πολύπλοκα XQuery, ανάπτυξη και σύνταξη των ερωτήσεων με τρόπο ώστε οι "αντιστοιχίες" μεταξύ των δυο ερωτήσεων(SPARQL-XQuery) και ο τρόπος μετάφρασης να γίνονται εύκολα αντιληπτά κτλ. Σε αυτό το σημείο μπορεί να γίνει κατανοητό ότι οι στόχοι αυτοί επιτεύχθηκαν από τους αλγορίθμους που περιγράφονται στο κεφάλαιο που μόλις προηγήθηκε.

Το βασικό επίτευγμα του κεφαλαίου είναι η δυνατότητα μετάφρασης βασικών σχηματομορφών γράφων σε σημασιολογικά ισοδύναμες XQuery εκφράσεις. Από τον ορισμό της σχηματομορφής γράφου(Ορισμός 2.8) γίνεται αντιληπτό ότι η πιο απλή μορφή (σχηματομορφής γράφου) είναι η τριπλέτα σχηματομορφής και κατά επέκταση η Βασική Σχηματομορφή Γράφου καθώς αυτή αποτελείται από τριπλέτες σχηματομορφών και φίλτρα (Ορισμός 2.7). Επομένως έχει επιτευχθεί η μετάφραση του "βασικού συστατικού" των σχηματομορφών γράφων.

Τις XQuery εκφράσεις που προκύπτουν από την μετάφραση των βασικών σχηματομορφών γράφων επεξεργάζεται το επόμενο κεφάλαιο (Κεφάλαιο 12) για την εφαρμογή των τελεστών(AND, OPT, UNION) που εμφανίζονται μεταξύ των βασικών σχηματομορφών γράφων, ώστε να επιτευχθεί η μετάφραση όλων των πιθανών σχηματομορφών γράφων των SPARQL ερωτήσεων.

12 Μετάφραση Σχηματομορφών Γράφων

12.1 Εισαγωγή

Στο παρόν κεφάλαιο θα γίνει η ανάλυση της διαδικασίας **"Μετάφρασης Σχηματομορφών Γράφων"(Graph Pattern Translation)** και του αλγορίθμου **GP2XQuery (Graph Pattern to XQuery)** ο οποίος την πραγματοποιεί. Όπως ορίζεται και στον Ορισμό 2.8, οι σχηματομορφές γράφων ορίζονται αναδρομικά, άρα αρκεί να οριστεί ο τρόπος μετάφρασης των "συστατικών" του αναδρομικού ορισμού.

Στο προηγούμενο κεφάλαιο(Κεφάλαιο 11) επιτεύχθηκε η μετάφραση των βασικών σχηματομορφών γράφων σε σημασιολογικά ισοδύναμες XQuery εκφράσεις. Από τον ορισμό της σχηματομορφής γράφου(Ορισμός 2.8) γίνεται αντιληπτό ότι βασικές σχηματομορφές γράφων αποτελούν το απλούστερο και πιο βασικό "συστατικό" των σχηματομορφών γράφων.

Στο παρόν κεφάλαιο θα γίνει η ανάλυση της διαδικασίας μετάφρασης των υπολοίπων "συστατικών" του αναδρομικού ορισμού των σχηματομορφών γράφων, δηλαδή : (P_1 AND P_2) , (P_1 UNION P_2) , (P_1 OPT P_2) και (P_1 FILTER R) όπου P_1 , P_2 σχηματομορφές γράφων και R SPARQL έκφραση. Με την διαδικασία μετάφρασης των σχηματομορφών γραφων επιτυγχάνεται η μετάφραση του βασικότερου και πιο πολύπλοκου μέρους μιας SPARQL ερώτησης, καθώς αυτή αντιστοιχεί στην μετάφραση της where δομής της ερώτησης.

Στην συνέχεια του κεφαλαίου παρουσιάζεται ο αλγόριθμος GP2XQuery ενότητα 12.2, στην συνέχεια ο αλγόριθμος διασπάται και αναλύεται ανάλογα με τον SPARQL τελεστή που εμφανίζεται μεταξύ των σχηματομορφών γράφων. Στην υπό-ενότητα 12.3 αναλύεται η μετάφραση του τελεστή AND, επίσης στην υπό-ενότητα 12.4 αναλύεται ο τελεστής OPT και ο τελεστής UNION στην υπό-ενότητα 12.5. Τέλος αναλύονται οι εναλλακτικοί τρόποι αποτίμησης της σχηματομορφής γράφου ώστε να βελτιστοποιηθεί η διαδικασία της αποτίμησης υπό-ενότητα 12.6.

12.2 Ο Αλγόριθμος GP2XQuery (Graph Pattern to XQuery)

Σε αυτό το σημείο θα γίνει η περιγραφή του αλγορίθμου μετάφρασης σχηματομορφών γράφων (GP2XQuery). Λόγω του αναδρομικού ορισμού των σχηματομορφών γράφων είναι απαραίτητη και η δημιουργία αναδρομικού αλγορίθμου για την μετάφραση τους, σε αυτό το σημείο δεν θα δοθεί έμφαση στην αναδρομική συμπεριφορά και ορισμό της συνάρτησης για λόγους πολυπλοκότητας, αλλά θα παρουσιαστεί η διαδικασία που ακολουθεί ο αλγόριθμος για την μετάφραση των τελεστών που εμφανίζονται μεταξύ των σχηματομορφών γράφων.

Ο αλγόριθμος GP2XQuery (Αλγόριθμος 12.1) δέχεται ως είσοδο μια σχηματομορφή γράφου P, ανάλογα με τον τελεστή που εμφανίζεται ανάμεσα στις σχηματομορφές γράφου καλεί και τον αντίστοιχο αλγόριθμο. Για τον τελεστή AND καλεί τον αλγόριθμο ANDPattAlgo (Αλγόριθμος 12.3), ο οποίος αναλύεται στην υπό-ενότητα 12.3, για τον τελεστή OPT καλεί τον αλγόριθμο OptPattAlgo(Αλγόριθμος 12.5 ο οποίος αναλύεται στην υπό-ενότητα 12.4 , για τον τελεστή union καλεί τον αλγόριθμο UnionPattAlgo(Αλγόριθμος 12.6) , ο οποίος αναλύεται στην υπό-ενότητα 12.5.

```
GP2XQuery ( P ) {
  IF  P =  P1 AND  P2  , όπου P1 , P2 είναι union-free graph patterns
        ANDPattAlgo ( P )
  ELSE IF P = P1 OPT P2  , όπου P1 , P2 είναι union-free graph patterns
        OptPattAlgo ( P )
  ELSE
        UnionPattAlgo ( P )
}
```

Αλγόριθμος 12.1 : GP2XQuery - Αλγόριθμος Μετάφρασης Σχηματομορφών Γράφων

12.3 Ο Τελεστής AND

Ο τελεστής AND παραλείπεται κατά την σύνταξη των SPARQL ερωτήσεων και κατά συνεπεία δεν αναφέρεται από την γραμματική της γλώσσας. Ο τελεστής AND αντιστοιχεί στην σύζευξη(Join) στο επίπεδο των αντιστοιχήσεων λύσεων (Ορισμός 2.13). Η σύζευξη σύμφωνα με την σημασιολογία της γλώσσας SPARQL διαφοροποιείται από την σύζευξη της σχεσιακής άλγεβρας, στις περιπτώσεις unbound μεταβλητών, όπως έχει αναλυθεί και στην Παρατήρηση 11.2 και 11.3.

12.3.1 Καλά Σχεδιασμένοι Γράφοι (Well designed graph)

Όπως παρατηρείται από την κανονικοποιημένη γραμματική των "καλά σχεδιασμένων"(Ορισμός 2.16) σχηματομορφών γράφων (Εικόνα 7.2 – σχέση b) ο τελεστής AND εμφανίζεται μόνο στον κανόνα :

BGP := "(" ((tp "AND" tp)* Filter* (tp "AND" tp)*)* ")"

Όπως φαίνεται από τον παραπάνω κανόνα, ο τελεστής AND στην κανονικοποιημένη γραμματική εμφανίζεται μόνο ανάμεσα σε τριπλέτες σχηματομορφών. Επομένως δεν χρειάζεται να οριστεί κάτι καινούργιο, καθώς ο τελεστής AND ανάμεσα σε τριπλέτες σχηματομορφών μεταφράζεται από τον αλγόριθμο BGP2XQuery.

Όπως παρατηρείται από την κανονικοποιημένη γραμματική των "όχι καλά σχεδιασμένων" (Ορισμός 2.16) σχηματομορφών γράφων (Εικόνα 7.3), η βασική διαφοροποίηση τους σε σχέση με τις "καλά

σχεδιασμένες" σχηματομορφές γράφων (Εικόνα 7.2) ως προς τους τελεστές, είναι η εμφανίζει του τελεστή AND όχι μόνο ανάμεσα σε τριπλέτες σχηματομορφών όπως γίνεται στους "καλά σχεδιασμένους", αλλά και ανάμεσα σε σχηματομορφές γράφων . Για τον λόγο αυτό, απαιτείται η προσομοίωση με την γλώσσα XQuery του τελεστή AND, ο οποίος εφαρμόζεται σε σχηματομορφές γράφων.

12.3.2 Όχι Καλά Σχεδιασμένοι Γράφοι (non-Well designed graph)

Όπως παρατηρείται από την κανονικοποιημένη γραμματική των "όχι καλά σχεδιασμένων" (Ορισμός 2.16) σχηματομορφών γράφων (Εικόνα 7.3 - σχέσεις b , c) ο τελεστής AND εμφανίζεται στους κανόνες :

BGP := "(" ((tp "AND" tp)* Filter* (tp "AND" tp)*)* ")"

And_Pattern := "(" (Union_free_Pattern "AND" Union_free_Pattern)* ")"

Ο πρώτος κανόνας είναι κοινός και για τις "καλά σχεδιασμένες" σχηματομορφές γράφων, ο τρόπος μετάφρασης του έχει αναλυθεί σε προηγούμενες παραγράφους. Η διαφοροποίηση των "όχι καλά σχεδιασμένων" έναντι των "καλά σχεδιασμένων" σχηματομορφών γράφων διατυπώνεται από τον δεύτερο κανόνα.

Από την γραμματική των "όχι καλά σχεδιασμένων" σχηματομορφών γράφων(Εικόνα 7.3 - σχέση f) φαίνεται ότι οι Union_free_Pattern μπορεί να είναι βασικές σχηματομορφές γράφων, γράφοι που περιέχουν τον τελεστή OPT (Optional_Pattern) και γράφοι που περιέχουν τον τελεστή AND (And_Pattern).

Union_free_Pattern := BGP| And_Pattern | Optional_Pattern

Για την μετάφραση των βασικών σχηματομορφών γράφων (BGP) χρησιμοποιείται ο αλγόριθμος BGP2XQuery, ενώ για τις σχηματομορφές που περιέχουν τον τελεστή OPT (Optional_Pattern) ακολουθείται η διαδικασία που περιγράφηκε στην προηγούμενη υπό-ενότητα (OptPattAlgo Αλγόριθμος 12.5). Τέλος η διαδικασία μετάφρασης των σχηματομορφών που περιέχουν τον τελεστή AND(And_Pattern), αναλύεται στην παρούσα παράγραφο (ANDPattAlgo - Αλγόριθμος 12.3).

Για την προσομοίωση του τελεστή AND με χρήση της γλώσσας XQuery, υλοποιήθηκε μια XQuery συνάρτηση(Αλγόριθμος 12.2), η οποία δέχεται σαν είσοδο δυο ακολουθίες, προερχόμενες από την αποτίμηση σχηματομορφών γράφων, που έχουν μεταφραστεί με την χρήση των αλγορίθμων που έχουν είδη αναφερθεί. Η συνάρτηση εφαρμόζει σε αυτές τον τελεστή της σύζευξη , εφαρμόζοντας την αυστηρή σημασιολογία του τελεστή για τις περιπτώσεις unbound μεταβλητών.

```
//Η συνάρτηση δέχεται σαν ορίσματα δυο αποτελέσματα από XQuery ερωτηματα και
// και υπολογίζεις αν σύζευξη τους παράγει κάποιο αποτέλεσμα, ελέγχοντας αν υπάρχουν
//μεταβλητές με το ίδιο όνομα και την ιδία τιμή
declare function func:JOIN  ( $R1 as node()* ,$R2 as node()*   )  as xs:boolean* {
        for $var in $R1/*
        where(  $R2/*[name(.)=name($var)]!=$var)  // ελέγχει  αν υπάρχουν μεταβλητές με το
                                                  //ίδιο όνομα και την ιδία τιμή
        return false( )
 } ;


//Η συνάρτηση δέχεται σαν ορίσματα δυο αποτελέσματα από XQuery ερωτήματα και
// και καλείται στην περίπτωση στην οποία η σύζευξη των αποτελεσμάτων παράγει
//αποτελέσματα. Επιστρέφει τις μεταβλητές από το αποτέλεσμα R2 οι οποίες δεν
//περιέχονται στο R1 , ώστε να υλοποιηθεί το join και το  left outer join.
 declare function func:VarsFromRes2  ( $R1 as node()* , $R2 as node()*   )  as node()* {
        for $var in $R2/*
        //Ελέγχει αν η μεταβλητή var από το R2 περιέχεται στο R1
        where(exists( $R1/*[name(.)=name($var)])=false())
        return  $var
};


//Η συνάρτηση δέχεται σαν ορίσματα δυο ακολουθίες αποτελεσμάτων από XQuery
//ερωτήματα και εφαρμόζει τον τελεστή  AND (join )με τα semantics της SPARQL ,
// επιστέφει το αποτέλεσμα εφαρμογής του τελεστή
declare function func:AND  ( $Res1 as node()* , $Res2 as node()*   )  as item()*{
        for $R1 in $Res1      // επαναλήψεις για όλα τα αποτελέσματα των R1 και R2
        for $R2 in $Res2
                let $join:=func:JOIN($R1,$R2 )      // Ελέγχει αν η σύζευξη των
                                                    //αποτελεσμάτων είναι εφικτή
        return
```

```
                    if (exists($join)=false()) then
                            (<Result> {$R1/* ,func:VarsFromRes2($R1 ,$R2 ) }</Result> )
                            //επιστέφει όλες τις μεταβλητές του R1 και τις μεταβλητές του R2
                            //που δεν περιέχονται στο R1

                    else ( )    //αλλιώς δεν επιστρέφει τίποτα
        } ;
```

Αλγόριθμος 12.2 : Η συνάρτηση func:AND για την προσομοίωση του τελεστή AND

- ***Έστω P = P1 AND P2 , όπου P1 , P2 είναι union-free graph patterns***

```
ANDPattAlgo ( P ) {
    Χρήση BGP2XQuery και GP2XQuery⇒ XQuery1, XQuery2
    let $P:= (
            let $P1:= (XQuery1)
            let $P2:= (XQuery2)
            return(func:AND($P1 , $P2))  )
}
```

Αλγόριθμος 12.3 : ANDPattAlgo - Αλγόριθμος Μετάφρασης σχηματομορφών που περιέχουν τον τελεστή AND(AND_Pattern)

Παράδειγμα 12.1

Έστω η ερώτηση : " Επέστρεψε "δυο ομάδες" ατόμων (και εργαζόμενων) οπού η "πρώτη ομάδα" έχει μικρό όνομα Paul ενώ η "δεύτερη ομάδα" έχει μικρό όνομα George, επίσης προαιρετικά επέστρεψε το ψευδώνυμο της "δεύτερης ομάδας".

```
SELECT ?x ?y ?n
    WHERE { {?x FirstName "Paul"
                    OPTIONAL { ?y NickName ?n} }
                ?y FirstName "George"
             }
```

Παράγεται το εξής XQuery ερώτημα:

```
let $AND_1 := (              // με χρήση του αλγορίθμου  ANDPattAlgo
let $BGP_1 := (              // με χρήση του αλγορίθμου  BGP2XQuery
            for  $x  in $doc/Persons/Person[./FirstName="Paul"] union
                        $doc/Persons/Employee [./FirstName="Paul"]
            return (<Result>{<x>{ func:nodeURI ($x)}</x>}</Result>)
    )
    let $BGP_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
            for  $y  in $doc/Persons/Person union $doc/Persons/Employee
            for  $n in  $y/NickName
            return (<Result>{<y>{ func:nodeURI ($y)}</y> ,
                                <n>{string($n)}</n>}</Result>)
    )

    return ( func:OPT($BGP_1 , $BGP_2 ) )// με χρήση του αλγορίθμου  OptPattAlgo
 )

let $AND_2 := (              // με χρήση του αλγορίθμου  ANDPattAlgo
    let $BGP_1 := (
            for  $y  in $doc/Persons/Person[./FirstName="George"] union
                        $doc/Persons/Employee [./FirstName=" George"]
            return (<Result>{<y>{ func:nodeURI ($y)}</y>}</Result>)
    )
)
return ( func:AND( $AND_1 , $AND_2 ) ) // με χρήση του αλγορίθμου  ANDPattAlgo
```

Παράδειγμα 12.2

Έστω η ερώτηση : " Επέστρεψε "δυο ομάδες" ατόμων (και εργαζόμενων) τα οποία έχουν ίδιο μικρό όνομα με ψευδώνυμο , όπου το μικρό όνομα και το ψευδώνυμο είναι προαιρετικά.

```
SELECT ?x ?y ?n
    WHERE { {?x a Person_Type
                    OPTIONAL { ?x FirstName ?n} }
              {?y a Person_Type
                    OPTIONAL { ?y NickName ?n} }
            }
```

Παράγεται το εξής XQuery ερώτημα:

```
let $AND_1 := (              // με χρήση του αλγορίθμου  ANDPattAlgo
    let $BGP_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
          for  $x  in $doc/Persons/Person union $doc/Persons/Employee
          return (<Result>{<x>{ func:nodeURI ($x)}</x>}</Result>)
    )
    let $BGP_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
          for  $x  in $doc/Persons/Person union $doc/Persons/Employee
          for  $n in  $x/FirstName
          return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                                <n>{string($n)}</n>}</Result>)
    )

    return ( func:OPT($BGP_1 , $BGP_2 ) ) )// με χρήση του αλγορίθμου  OptPattAlgo
 )

let $AND_2 := (              // με χρήση του αλγορίθμου  ANDPattAlgo

    let $BGP_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
          for  $y  in $doc/Persons/Person union $doc/Persons/Employee
          return (<Result>{<y>{ func:nodeURI ($y)}</y>}</Result>)
    )
let $BGP_2 := (              // με χρήση του αλγορίθμου  BGP2XQuery

          for  $y  in $doc/Persons/Person union $doc/Persons/Employee
          for  $n in  $y/NickName
          return (<Result>{<y>{ func:nodeURI ($y)}</y> ,
                                <n>{string($n)}</n>}</Result>)
    )

    return ( func:OPT($BGP_1 , $BGP_2 ) ) )// με χρήση του αλγορίθμου  OptPattAlgo
 )

 return ( func:AND( $AND_1 , $AND_2 ) ) // με χρήση του αλγορίθμου  ANDPattAlgo
```

12.4 Ο Τελεστής OPT

Ο τελεστής OPT ή OPTIONAL(σύμφωνα σύνταξη της γλώσσας SPARQL), αντιστοιχεί σε αριστερή εξωτερική σύζευξη(Left Outer Join) στο επίπεδο των αντιστοιχήσεων λύσεων(Ορισμός 2.13). Όπως είναι γνωστό και από την σχεσιακή άλγεβρα ο τελεστής της αριστερής εξωτερικής σύζευξης ορίζεται ως $\Omega_1 ⟕ \Omega_2 = (\Omega_1 ⟖ \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$ (Ορισμός 2.12) . Η διαφοροποίηση του από τον τελεστή της σχεσιακής άλγεβρας οφείλεται στο γεγονός της "ιδιαίτερης" συμπεριφοράς της σύζευξης (Join), έναντι των μεταβλητών στις οποίες δεν έχει ανατεθεί τιμή(unbound) (Βλέπε Παρατήρηση 11.2 και 11.3)

Όπως παρατηρείται από την κανονικοποιημένη γραμματική των "καλά σχεδιασμένων" σχηματομορφών γράφων (Εικόνα 7.2 - σχέση c) ο τελεστής OPT εμφανίζεται μόνο στον κανόνα :

Optional_Pattern := BGP ("OPT" "(" Optional_Pattern Filter* ")")+

Αναπτύσσοντας τον παραπάνω κανόνα γίνεται αντιληπτό ότι έχουμε τις εξής εκφράσεις :

- Εμφωλευμένα OPT : BGP_1 OPT(BGP_2 OPT(BGP_3 … OPT(BGP_N))…)
- Συνεχόμενα OPT: BGP_1OPT(BGP_2)OPT(BGP_3)… OPT(BGP_N)
- Συνδυασμός των παραπάνω : BGP_1OPT(BGP_2)OPT(BGP_3OPT(BGP_4 OPT(BGP_5)))OPT (BGP_6)

Σύμφωνα με τον Ορισμό 2.13, η αποτίμηση των σχηματομορφών που περιέχουν τον τελεστή OPT γίνεται με το εξής τρόπο :

Έστω **P = BGP_1 OPT(BGP_2 OPT(BGP_3))**

$[[P]]_D$ =$[[BGP_1]]_D$ ⟕ ($[[BGP_2]]_D$ ⟕ $[[BGP_3]]_D$)

Η συνολική αποτίμηση της σχηματομορφής P ([[P]]) ισούται με την αριστερή σύζευξη μεταξύ των αποτιμήσεων των BGP_2 και BGP_3 ($[[BGP_2]]_D$ ⟕ $[[BGP_3]]_D$) και στο αποτέλεσμα που προκύπτει εφαρμόζεται αριστερή σύζευξη με την αποτίμηση του BGP_1.

Έστω **P = BGP_1 OPT(BGP_2) OPT(BGP_3)**

$[[P]]_D$ = ($[[BGP_1]]_D$ ⟕ $[[BGP_2]]_D$) ⟕ $[[BGP_3]]_D$

Η συνολική αποτίμηση της σχηματομορφής P ([[P]]) ισούται με την αριστερή σύζευξη μεταξύ των αποτιμήσεων των BGP_1 και BGP_2 ($[[BGP_1]]_D$ ⟕ $[[BGP_2]]_D$) και στο αποτέλεσμα που προκύπτει εφαρμόζεται αριστερή σύζευξη με την αποτίμηση του BGP_3.

Λόγω της πολυπλοκότητας του τελεστή καθώς και των πολλών πιθανών τρόπων εφαρμογής του δεν ήταν δυνατή η προσομοίωση του με εκφράσεις της γλώσσας XQuery, όπως αυτό έγινε στον αλγόριθμο BGP2XQuery για τον τελεστή AND όταν εμφανίζεται ανάμεσα σε τριπλέτες. Επομένως είναι αναγκαία η εφαρμογή του τελεστή σε επίπεδο αποτελεσμάτων.

Για την προσομοίωση του τελεστή OPT (OPTIONAL) με χρήση της γλώσσας XQuery, υλοποιήθηκε μια XQuery συνάρτηση(Αλγόριθμος 12.4) η οποία δέχεται σαν είσοδο δυο ακολουθίες λύσεων και εφαρμόζει σε αυτές τον τελεστή της αριστερή εξωτερική σύζευξη , εφαρμόζοντας την αυστηρή σημασιολογία του τελεστή για τις περιπτώσεις unbound μεταβλητών.

```
//Η συνάρτηση δέχεται σαν ορίσματα δυο ακολουθίες αποτελεσμάτων από XQuery
//ερωτήματα και εφαρμόζει τον τελεστή  OPT (left outer join) με τα semantics της
// SPARQL ,  επιστέφει το αποτέλεσμα εφαρμογής του τελεστή
declare function func:OPT  ( $Res1 as node()* , $Res2 as node()*   )  as item()*{
      for $R1 in $Res1    // επαναλήψεις για όλα τα αποτελέσματα των R1 και R2
      for $R2 in $Res2
      let $join:=func:JOIN($R1,$R2 )      // Ελέγχει αν η σύζευξη των αποτελεσμάτων είναι
                                          //εφικτή
      return
            if (exists($join)=false()) then  //αν είναι εφικτη η συζευξη
                  (<Result> {$R1/* ,func:VarsFromRes2($R1 ,$R2 ) }</Result> )
                  //επιστέφει όλες τις μεταβλητές του R1 και τις μεταβλητές του R2
                  //που δεν περιέχονται στο R1
             else       //αλλιώς επιστέφει μόνο το R1
                   (<Result> {$R1/*  }</Result> )
} ;
```

Αλγόριθμος 12.4 : Η συνάρτηση func:OPT για την προσομοίωση του τελεστή OPT

- ***Έστω P = P1 OPT P2 , όπου P1 , P2 είναι union-free graph patterns***

```
OptPattAlgo ( P ) {
        Χρήση BGP2XQuery και GP2XQuery⇒ XQuery1, XQuery2
        let $P:= (
                        let $P1:= (XQuery1)
                        let $P2:= (XQuery2)
                        return(func:OPT($P1 , $P2)) )
}
```

Αλγόριθμος 12.5 : OptPattAlgo - Αλγόριθμος Μετάφρασης σχηματομορφών που περιέχουν τον τελεστή OPT(Optional_Pattern)

Παράδειγμα 12.3

Έστω η ερώτηση : " Επέστρεψε για όλα τα άτομα (και εργαζόμενοι) το/α μικρό/ά όνομα/τα και για όσους έχουν, τα ψευδώνυμα τους.

```
SELECT ?x ?fn ?nn
    WHERE { ?x    FirsName   ?fn .
              OPTIONAL {?x NickName ?nn  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)

let $BGP_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in /Persons/Person union /Persons/Employee
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)

)
return ( func:OPT($BGP_1 , $BGP_2 ) )  // με χρήση του αλγορίθμου  OptPattAlgo
```

Παράδειγμα 12.4

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) ,το/α μικρό/ά όνομα/τα και για όσους έχουν, τα ψευδώνυμα τους ,επίσης για όσους έχουν, επέστρεψε τον τηλεφωνικό τους αριθμό.

```
SELECT ?x ?fn ?nn ?tel
    WHERE { ?x    FirsName   ?fn .
                OPTIONAL {?x NickName ?nn  .}
                OPTIONAL {?x Telephone ?tel .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <fn>{string($fn)}</fn>}</Result>)
)
let $BGP_2 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <nn>{string($nn)}</nn>}</Result>)
)

let $BGP_3 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $tel in  $x/Telephone
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <tel>{string($tel)}</tel>}</Result>)
)

let $OPT_1 :=func:OPT($BGP_1 , $BGP_2 ) // με χρήση του αλγορίθμου  OptPattAlgo

return ( func:OPT( $OPT_1 , $BGP_3 ) ) // με χρήση του αλγορίθμου  OptPattAlgo
```

Παράδειγμα 12.5

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι), το/α μικρό/ά όνομα/τα επίσης στην περίπτωση που έχει ψευδώνυμο επέστρεψε το ,στην περίπτωση που το άτομο έχει ψευδώνυμο επέστρεψε και τον τηλεφωνικό του αριθμό εάν υπάρχει.

```
SELECT ?x ?fn ?nn ?tel

    WHERE { ?x    FirsName   ?fn .
                OPTIONAL {?x NickName ?nn  .
                        OPTIONAL {?x Telephone ?tel .}}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <fn>{string($fn)}</fn>}</Result>)
)
let $BGP_2 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)
)

let $BGP_3 := (              // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $tel in  $x/Telephone
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <tel>{string($tel)}</tel>}</Result>)
)

let $OPT_1 :=func:OPT( $BGP_2 , $BGP_3 ) // με χρήση του αλγορίθμου  OptPattAlgo

return ( func:OPT( $BGP_1 , $OPT_1) ) // με χρήση του αλγορίθμου  OptPattAlgo
```

Παράδειγμα 12.6

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι), το/α μικρό/ά όνομα/τα επίσης στην περίπτωση που έχει ψευδώνυμο επέστρεψε, επίσης επέστρεψε την ηλικία εάν έχει οριστεί. Στην περίπτωση που το άτομο έχει ψευδώνυμο επέστρεψε και τον τηλεφωνικό του αριθμό εάν υπάρχει.

```
SELECT ?x ?fn ?nn ?tel ?age
    WHERE { ?x    FirsName   ?fn .
              OPTIONAL {?x NickName ?nn  .
                    OPTIONAL {?x Telephone ?tel .}}
              OPTIONAL {?x Age ?age  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <fn>{string($fn)}</fn>}</Result>)
)
let $BGP_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)
)

let $BGP_3 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $tel in  $x/Telephone
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <tel>{string($tel)}</tel>}</Result>)
)

let $BGP_4 := (
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $age in  $x/Age
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <age>{string($age)}</age>}</Result>)
)
```

let $OPT_1 :=func:OPT($BGP_2 , $BGP_3) *// με χρήση του αλγορίθμου OptPattAlgo*

let $OPT_2 :=func:OPT($BGP_1 , $OPT_1) *// με χρήση του αλγορίθμου OptPattAlgo*

return (func:OPT($OPT_2 , $BGP_4)) *// με χρήση του αλγορίθμου OptPattAlgo*

12.5 Ο Τελεστής UNION

Ο τελεστής UNION ο οποίος εμφανίζεται με το ίδιο όνομα και στην γραμματική της γλώσσας SPARQL, αντιστοιχεί στην ένωση (Union) στο επίπεδο των αντιστοιχήσεων λύσεων(Ορισμός 2.13). Ο τελεστής της ένωσης συμφώνα με την σημασιολογία της γλώσσας SPARQL δεν διαφοροποιείται από την ένωσης της σχεσιακής άλγεβρας.

Όπως παρατηρείται από την κανονικοποιημένη γραμματική των καλά σχεδιασμένων" σχηματομορφών γράφων (Εικόνα 7.2 - σχέση e) ο τελεστής UNION εμφανίζεται μόνο στον κανόνα :

Union_Pattern := "(" Union_free_Pattern ")" ("UNION" "("Union_free_Pattern) ")" *

Όπως έχει αναφερθεί και σε προηγούμενες παραγράφους για τους Union_free_Pattern που εμφανίζονται δεξιά και αριστερά του τελεστή UNION, εφαρμόζονται ανεξάρτητα οι διαδικασίες προσδιορισμού τύπων μεταβλητών και σύνδεση μεταβλητών. Καθώς οι αντιστοιχήσεις λύσεων των δυο αυτών γράφων σχηματομορφών είναι ανεξάρτητες.

Από την γραμματική των "καλά σχεδιασμένων" σχηματομορφών γράφων(Εικόνα 7.2 - σχέση d) φαίνεται ότι οι Union_free_Pattern μπορεί να είναι είτε βασικές σχηματομορφές γράφων είτε γράφοι που περιέχουν τον τελεστή OPT (Optional_Pattern) .

Union_free_Pattern := BGP | Optional_Pattern

Για την μετάφραση των βασικών σχηματομορφών γράφων (BGP) χρησιμοποιείται ο αλγόριθμος BGP2XQuery, ενώ για τις σχηματομορφές που περιέχουν τον τελεστή OPT (Optional_Pattern) ακολουθείται η διαδικασία που περιγράφηκε στην προηγούμενη υπό-ενότητα (OptPattAlgo Αλγόριθμος 12.6). Με τον τρόπο αυτό μεταφράζονται οι Union_free_Pattern , το XQuery που προκύπτει από κάθε μετάφραση ανατίθεται σε μια μεταβλητή με χρήση της δομής let, στην συνέχεια γίνεται ένωση των αποτελεσμάτων όλων των XQueries, επιστρέφοντας όλες τις μεταβλητές που έχουν ανατεθεί τα ερωτήματα, κάνοντας χρήση της δομής return.

- ***Έστω P = P1 UNION P2 UNION… UNION Pn , όπου P1, P2 , …, Pn είναι Union_free_Pattern***

UnionPattAlgo (P) {

Χρήση BGP2XQuery και GP2XQuery⇒ XQuery1, XQuery2, XQuery…, XQueryn

∀ XQuery ⇨ let $P_i := (XQuery$_i$) **(XQuery$_i$ έχει προκύψει από μετάφραση του P$_i$)**

Let P:= (let $P1:= (XQuery1) let $P2:= (XQuery2))… let $Pn:= (XQueryn)

return ($P1, $P2, …, $Pn))

}

Αλγόριθμος 12.6 : UnionPattAlgo - Αλγόριθμος Μετάφρασης σχηματομορφών που περιέχουν τον τελεστή UNION(Union_Pattern)

Παράδειγμα 12.7

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/α μικρό/ά όνομα/τα, επίσης επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/τα ψευδώνυμο/α τους.

```
SELECT ?x  ?n
   WHERE { {?x FirsName   ?n .}
                  UNION
           {?x NickName   ?n  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Union_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
   for  $x  in $doc/Persons/Person union $doc/Persons/Employee
   for  $n in  $x/FirstName
   return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)
let $Union_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
   for  $x  in $doc/Persons/Person union $doc/Persons/Employee
   for  $n in  $x/NickName
   return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)

return ( $Union_1 , $Union_2 )   // με χρήση του αλγορίθμου  UnionPattAlgo
```

Παράδειγμα 12.8

Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/α μικρό/ά όνομα/τα, επίσης επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/τα ψευδώνυμο/α τους, επιπρόσθετα επέστρεψε όλα τα άτομα (και εργαζόμενοι) και το/α επίθετα τους.

```
SELECT ?x ?n
    WHERE { {?x FirsName   ?n .}
                    UNION
            {?x NickName   ?n  .}
                    UNION
            {?x LastName   ?n  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Union_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $n in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)
let $Union_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $n in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)
let $Union_3 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in /Persons/Person union /Persons/Employee
    for  $n in  $x/LastName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)

return ( $Union_1 , $Union_2 , $Union_3) // με χρήση του αλγορίθμου  UnionPattAlgo
```

Παράδειγμα 12.9

Έστω η ερώτηση που επιστρέφει την ένωση των αποτελεσμάτων των ερωτήσεων από τα Παραδείγματα 12.4 και 12.5

```
SELECT ?x ?fn ?nn ?tel
   WHERE { { ?x FirsName   ?fn .
                OPTIONAL {?x NickName ?nn  .}
                OPTIONAL {?x Telephone ?tel .}}
                   UNION
             { ?x FirsName   ?fn .
               OPTIONAL {?x NickName ?nn  .
                        OPTIONAL {?x Telephone ?tel .}}}
           }
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Union_1 := (          // με χρήση του αλγορίθμου  UnionPattAlgo
        let $BGP_1 := (              // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $fn in  $x/FirstName
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <fn>{string($fn)}</fn>}</Result>)
   )
   let $BGP_2 := (              // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $nn in  $x/NickName
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <nn>{string($nn)}</nn>}</Result>)
   )

   let $BGP_3 := (              // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $tel in  $x/Telephone
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <tel>{string($tel)}</tel>}</Result>)
   )

   let $OPT_1 :=func:OPT($BGP_1 , $BGP_2 ) // με χρήση του αλγορίθμου OptPattAlgo

   return ( func:OPT( $OPT_1 , $BGP_3 ) ) // με χρήση του αλγορίθμου  OptPattAlgo
 )`
let $Union_2 := (          // με χρήση του αλγορίθμου  UnionPattAlgo
```

```
    let $BGP_1 := (            // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $fn in  $x/FirstName
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                           <fn>{string($fn)}</fn>}</Result>)
    )
    let $BGP_2 := (            // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $nn in  $x/NickName
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                           <nn>{string($nn)}</nn>}</Result>)
    )
    let $BGP_3 := (            // με χρήση του αλγορίθμου  BGP2XQuery
        for  $x  in $doc/Persons/Person union $doc/Persons/Employee
        for  $tel in  $x/Telephone
        return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                           <tel>{string($tel)}</tel>}</Result>)
    )

    let $OPT_1 :=func:OPT( $BGP_2 , $BGP_3 ) // με χρήση του αλγορίθμου  OptPattAlgo

    return ( func:OPT( $BGP_1 , $OPT_1) ) // με χρήση του αλγορίθμου  OptPattAlgo
)

return ( $Union_1 , $Union_2 )        // με χρήση του αλγορίθμου  UnionPattAlgo
```

Παράδειγμα 12.10

Έστω η παραλλαγή του ερωτήματος από το Παράδειγμα 12.7 , στην οποία επιστέφονται τρεις μεταβλητές οι οποίες δεν εμφανίζονται σε όλα τα μέρη της ερώτησης.

```
SELECT ?x  ?n ?nn
    WHERE { {?x FirsName   ?n .}
                    UNION
            {?x NickName   ?nn  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Union_1 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $n in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <n>{string($n)}</n>}</Result>)
)
let $Union_2 := (          // με χρήση του αλγορίθμου  BGP2XQuery
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                          <nn>{string($nn)}</nn>}</Result>)
)

return ( $Union_1 , $Union_2 )    // με χρήση του αλγορίθμου  UnionPattAlgo
```

Το σύνολο των αποτελεσμάτων της παραπάνω ερώτησης θα έχει την μορφή:

```
<Result> <x><IRI>…</IRI></x>  <n>…</n> </Result>
<Result> <x><IRI>…</IRI></x>  <n>…</n> </Result>
        ….
        ….
<Result> <x><IRI>…</IRI></x>  <n>…</n> </Result>
<Result> <x><IRI>…</IRI></x>  <nn>…</nn> </Result>
<Result> <x><IRI>…</IRI></x>  <nn>…</nn> </Result>
        ….
        ….
<Result> <x><IRI>…</IRI></x>  <nn>…</nn> </Result>
```

Που όπως παρατηρείται δεν εμφανίζονται όλες οι μεταβλητές σε όλα τα αποτελέσματα , οι μεταβλητές που δεν εμφανίζονται θεωρούνται unbound.

12.6 Τρόποι αποτίμησης - Βελτιστοποίηση

Στην παρούσα ενότητα θα παρουσιαστεί ένας εναλλακτικός "greedy"τρόπος αποτίμησης (έχει αποδειχθεί στο [11]) ο οποίος μπορεί να εφαρμοστεί για τους καλά σχεδιασμένες σχηματομορφές γράφων (Ορισμός 2.16), ο greedy" αυτός τρόπος αποτίμησης ακολουθεί έναν διαφορετικό τρόπο διάσχισης του δένδρου εκτέλεσης (execution tree) της ερώτησης. Αποτέλεσμα αυτού είναι η μείωση (σε ορισμένες περιπτώσεις σε σημαντικό βαθμό) των αριθμών των επαναλήψεων και των υπολογισμών στα XQuery ερωτήματα.

Όπως αναφέρεται και στο [11], οι τρόποι αποτίμησης των σχηματομορφών γράφων στις ερωτήσεις SPARQL είναι δυο.

Ο πρώτος χρησιμοποιήθηκε μέχρι τώρα για την αποτίμηση των ερωτήσεων. Είναι βασισμένος στην σημασιολογία της συνάρτησης αποτίμησης [[.]] (Ορισμοί 2.9 , 2.13 , 2.14 και 2.15). Αυτή η προσέγγιση όπως μπορεί να παρατηρηθεί, κατά την αποτίμηση των γράφων ακολουθεί bottom up διάσχιση του δένδρου εκτέλεσης (execution tree) που δημιουργείται με βάση την ερώτηση.

Ενώ υπάρχει και ένας άλλος "greedy" τρόπος αποτίμησης των σχηματομορφών γράφων(βλέπε Ορισμός 12.1). Στη συγκεκριμένη προσέγγιση σε αντίθεση με την προηγούμενη, για την αποτίμηση των γράφων ακολουθεί depth-first διάσχιση του δένδρου εκτέλεσης(execution tree), χρησιμοποιώντας τις αντιστοιχήσεις που έχουν υπολογιστεί σε προηγούμενες σχηματομορφές ώστε να μειωθεί ο αριθμός των υπολογισμών. Αυτή είναι και η προσέγγιση που χρησιμοποιείται από το ARQ engine[53] (SPARQL engine του Jena). Αυτός ο τρόπος αποτίμησης περιλαμβάνει σε κάθε στάδιο τρεις παραμέτρους : το σύνολο δεδομένων (dataset) D, το υπό-δένδρο της ερώτησης που θα αποτιμηθεί, ένα σύνολο αντιστοιχήσεων που έχουν είδη υπολογιστεί(βλέπε Ορισμό 12.1).

Ορισμός 12.1 [11] Η αναδρομική συνάρτηση $Eval_D(P , Ω)$ Έστω το σύνολο RDF δεδομένων D , η αποτίμηση της σχηματομορφής γράφου P σε συνδυασμό με ένα σύνολο από αντιστοιχήσεις Ω, συμβολίζεται ως $Eval_D(P , Ω)$. Η Eval είναι αναδρομική συνάρτηση και ορίζεται ως :

$Eval_D$(P : σχηματομορφή γράφου , Ω: σύνολο από αντιστοιχήσεις)

Αν Ω=Ø τότε return(Ø)

Αν P είναι σχηματομορφή τριπλέτας t τότε return($Ω \triangleright\triangleleft [[t]]_D$)

Αν $P=(P_1$ AND $P_2)$ τότε return $EVAL_D(P_2, EVAL_D(P_1, Ω))$

Αν $P=(P_1$ OPT $P_2)$ τότε return $EVAL_D(P_1, Ω)$ $EVAL_D \mathbin{>\!\!<} (P_2, EVAL_D(P_1, Ω))$

Αν $P=(P_1$ FILTER R) τότε return{ $μ \in EVAL_D(P_1, Ω)$ | $μ \models R$ }

Ορισμός 12.2 [11] Ισοδυναμία των τρόπων αποτιμήσεις Για κάθε σύνολο RDF δεδομένων D και για κάθε "καλά σχεδιασμένη" σχηματομορφή γράφου P ισχύει : $EVAL_D(P) = [[P]]_D$. Αυτή η ισότητα δεν ισχύει για τις "όχι-καλά σχεδιασμένες" σχηματομορφές γράφων.

Έστω $\mathbf{t_1, t_2, t_3}$ σχηματομορφές τριπλέτων

Αν P = t_1 OPT(t_2 OPT(t_3))

- $[[P]]_D = [[t_1]]_D \bowtie ([[t_2]]_D \bowtie [[t_3]]_D)$
- $EVAL_D(P) = [[t_1]]_D \bowtie (([[t_1]]_D \triangleright\!\triangleleft [[t_2]]_D) \bowtie ([[t_1]]_D \triangleright\!\triangleleft [[t_2]]_D \triangleright\!\triangleleft [[t_3]]_D))$

Αν P = t_1 OPT(t_2) OPT(t_3)

- $[[P]]_D = ([[t_1]]_D \bowtie [[t_2]]_D) \bowtie [[t_3]]_D$
- $EVAL_D(P) = [[t_1]]_D \bowtie (([[t_1]]_D \triangleright\!\triangleleft [[t_2]]_D) \bowtie ((([[t_1]]_D \bowtie (([[t_1]]_D \triangleright\!\triangleleft [[t_2]]_D)) \triangleright\!\triangleleft [[t_3]]_D))$

Με βάση τα παραπάνω, κατά την διαδικασία μετάφρασης των ερωτήσεων, στην περίπτωση "καλά σχεδιασμένων" σχηματομορφών γράφων μπορεί να ακολουθηθεί η δεύτερη προσέγγιση αποτίμησης. Αυτό έχει ως αποτέλεσμα την βελτιστοποίηση των XQuery ερωτήσεων. Η διαδικασία μετάφρασης παραμένει ίδια, με την διαφοροποίηση ότι ακολουθείται depth-first διάσχιση του δένδρου, όπως επίσης για τις μεταβλητές που έχουν υπολογιστεί σε προηγούμενο στάδιο χρησιμοποιούνται οι τιμές που έχουν υπολογιστεί και δεν πραγματοποιείται από την αρχή υπολογισμός. Παρακάτω καταγράφονται τα παραδείγματα από τις υπό-ενότητες 12.4 και 12.5, εφαρμόζοντας κατά την μετάφραση τον "greedy" τρόπο αποτίμησης .Στα παρακάτω παραδείγματα έχουν επισημανθεί οι αλλαγές που προέκυψαν, από την χρήση του "greedy" τρόπου αποτίμησης.

Παράδειγμα 12.11

Από το Παράδειγμα 12.3. Έστω η ερώτηση : " Επέστρεψε για όλα τα άτομα (και εργαζόμενοι) το/α μικρό/ά όνομα/τα και για όσους έχουν, τα ψευδώνυμα τους.

```
SELECT ?x ?fn ?nn
    WHERE { ?x    FirsName   ?fn .
            OPTIONAL {?x NickName ?nn  .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <n>{string($n)}</n>}</Result>)
)

let $BGP_2 := (
    for  $x  in $BGP_1/x
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)

)
return ( func:OPT($BGP_1 , $BGP_2 ) )
```

Αντί για :
$doc/Persons/Person union $doc/Persons/Employee

Όπως παρατηρείται στο παραπάνω παράδειγμα, στην for δομή της μεταβλητές ?x που προέρχεται από την μετάφραση της τριπλέτας (?x NickName ?nn) δεν έγινε η επανάληψη στο σύνολο $doc/Persons/Person union $doc/Persons/Employee , αλλά στις τιμές του ?χ που είχαν υπολογιστεί με βάση τη τριπλέτα ?x FirsName ?fn (εφαρμόζεται με έμμεσο τρόπο η σύζευξη στην μεταβλητή ?χ). Αυτό έχει ως αποτέλεσμα να μειώνονται οι επαναλήψεις τόσο του ?χ όσο και του ?nn , αλλά και η επεξεργασία πολύ λιγότερων δεδομένων κατά την εφαρμογή της συνάρτηση func:OPT.

Παράδειγμα 12.12

Από το Παράδειγμα 12.4. Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι) ,το/α μικρό/ά όνομα/τα και για όσους έχουν, τα ψευδώνυμα τους ,επίσης για όσους έχουν, επέστρεψε τον τηλεφωνικό τους αριθμό.

```
SELECT ?x ?fn ?nn ?tel
    WHERE { ?x    FirsName   ?fn .
                OPTIONAL {?x NickName ?nn  .}
                OPTIONAL {?x Telephone ?tel .}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <fn>{string($fn)}</fn>}</Result>)
)
let $BGP_2 := (
    for  $x  in  $BGP_1/x
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)
)
let $BGP_3 := (
    for  $x  in $BGP_1/x
    for  $tel in  $x/Telephone
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <tel>{string($tel)}</tel>}</Result>)
)

let $OPT_1 :=func:OPT($BGP_1 , $BGP_ 3)

return ( func:OPT( $OPT_1 , $BGP_2) )
```

Αντί για : $doc/Persons/Person union $doc/Persons/Employee

Αντί για : $BGP2

Αντί για : $BGP3

Όπως παρατηρείται στο παραπάνω παράδειγμα, στις for δομές της μεταβλητής ?x που προέρχεται από την μετάφραση των τριπλέτων (?x NickName ?nn και ?x Telephone ?tel) δεν έγινε η επανάληψη στο σύνολο $doc/Persons/Person union $doc/Persons/Employee , αλλά στις τιμές του ?χ που είχαν υπολογιστεί με βάση τη τριπλέτα ?x FirsName ?fn (εφαρμόζεται με έμμεσο τρόπο η σύζευξη στην μεταβλητή ?χ). Αυτό έχει ως αποτέλεσμα να μειώνονται οι επανάληψης τόσο του ?χ όσο και του ?nn και ?tel , αλλά και η επεξεργασία πολύ λιγότερων δεδομένων κατά την εφαρμογή των συναρτήσεων func:OPT.

Παράδειγμα 12.13

Από το Παράδειγμα 12.5 .Έστω η ερώτηση : " Επέστρεψε όλα τα άτομα (και εργαζόμενοι), το/α μικρό/ά όνομα/τα επίσης στην περίπτωση που έχει ψευδώνυμο επέστρεψε το ,στην περίπτωση που το άτομο έχει ψευδώνυμο επέστρεψε και τον τηλεφωνικό του αριθμό εάν υπάρχει.

```
SELECT ?x ?fn ?nn ?tel

    WHERE { ?x    FirsName   ?fn .
                OPTIONAL {?x NickName ?nn  .
                        OPTIONAL {?x Telephone ?tel .}}}
```

Παράγεται το εξής XQuery ερώτημα:

```
let $BGP_1 := (
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <fn>{string($fn)}</fn>}</Result>)
)
let $BGP_2 := (
    for  $x  in $BGP_1/x
    for  $nn in  $x/NickName
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <nn>{string($nn)}</nn>}</Result>)
)
let $OPT_1 :=func:OPT( $BGP_1 , $BGP_2 )

let $BGP_3 := (
    for  $x  in $OPT_1/x
    for  $tel in  $x/Telephone
    return (<Result>{<x>{ func:nodeURI ($x)}</x> ,
                        <tel>{string($tel)}</tel>}</Result>)
)

return ( func:OPT( $OPT_1 , $BGP_3) )
```

Αντί για : $doc/Persons/Person union $doc/Persons/Employee

Αντί για : $BGP_2 , $BGP_3

Αντί για : $doc/Persons/Person union $doc/Persons/Employee

Αντί για : $BGP_1 , $OPT_1

Όπως παρατηρείται στο παραπάνω παράδειγμα, στην for δομή της μεταβλητές ?x που προέρχεται από την μετάφραση της τριπλέτας (?x NickName ?nn) δεν έγινε η επανάληψη στο σύνολο $doc/Persons/Person union $doc/Persons/Employee , αλλά στις τιμές του ?χ που είχαν υπολογιστεί

με βάση τη τριπλέτα ?x FirsName ?fn (εφαρμόζεται με έμμεσο τρόπο η σύζευξη στην μεταβλητή ?χ). Αυτό έχει ως αποτέλεσμα να μειώνονται οι επανάληψης τόσο του ?χ όσο και του ?nn , αλλά και η επεξεργασία πολύ λιγότερων δεδομένων κατά την εφαρμογή της συνάρτηση func:OPT.

Επίσης στην for δομή της μεταβλητές ?x που προέρχεται από την μετάφραση της τριπλέτας (?x Telephone ?tel) δεν έγινε η επανάληψη στο σύνολο $doc/Persons/Person union $doc/Persons/Employee , αλλά στις τιμές του ?χ που είχαν υπολογιστεί με βάση την εφαρμογή του τελεστή OPT στα αποτελέσματα των τριπλετών ?x FirsName ?fn και ?x NickName ?nn. Αυτό έχει ως αποτέλεσμα να μειώνονται οι επανάληψης τόσο του ?χ όσο και του ?tel αλλά και η επεξεργασία πολύ λιγότερων δεδομένων κατά την εφαρμογή της συνάρτηση func:OPT.

Στην παραπάνω ενότητα έγινε η παρουσίαση ενός "greedy" τρόπος αποτίμησης των σχηματομορφών γράφων, η οποία μπορεί να εφαρμοστεί από την παρούσα προσέγγιση κατά την μετάφραση των ερωτήσεων. Ο συγκεκριμένος τρόπος αποτίμησης των γράφων, ακολουθεί depth-first διάσχιση του δένδρου εκτέλεσης (execution tree) , χρησιμοποιώντας τις αντιστοιχήσεις που έχουν υπολογιστεί σε προηγούμενες σχηματομορφές ώστε να μειωθεί ο αριθμός των υπολογισμών. Αξίζει να σημειωθεί ότι ο συγκεκριμένος τρόπος αποτίμησης χρησιμοποιείται από το ARQ engine (SPARQL engine του Jena) για την βελτιστοποίηση των ερωτημάτων. Τέλος , όπως γίνεται κατανοητό και από τα παραδείγματα η μέθοδος αυτή μειώνει αισθητά τον αριθμό των υπολογισμών και επαλείψεων στα μεταφρασμένα XQurery ερωτήματα.

12.7 Επίλογος

Στο κεφάλαιο αυτό έγινε η ανάλυση της διαδικασίας μετάφρασης σχηματομορφών γράφων και του αλγορίθμου GP2XQuery(Graph Pattern to XQuery) ο οποίος την πραγματοποιεί. Κάνοντας χρήση του αλγορίθμου BGP2XQuery (Κεφάλαιο 11) για την μετάφραση των βασικών σχηματομορφών γράφων, έγινε η ανάλυση της διαδικασίας μετάφρασης των υπολοίπων "συνθετικών" των σχηματομορφών γράφων: (P_1 AND P_2) , (P_1 UNION P_2) , (P_1 OPT P_2) και (P_1 FILTER R).

Πιο συγκεκριμένα για την προσομοίωση του τελεστή OPT με χρήση της γλώσσας XQuery, υλοποιήθηκε μια XQuery συνάρτηση η οποία δέχεται σαν είσοδο δυο ακολουθίες λύσεων και εφαρμόζει σε αυτές τον τελεστή της αριστερή εξωτερική σύζευξη, εφαρμόζοντας την αυστηρή σημασιολογία του τελεστή για τις περιπτώσεις unbound μεταβλητών. Ενώ για την προσομοίωση του τελεστή AND με χρήση της γλώσσας XQuery, υλοποιήθηκε μια XQuery συνάρτηση, η οποία δέχεται σαν είσοδο δυο ακολουθίες, προερχόμενες

από την αποτίμηση σχηματομορφών γράφων και εφαρμόζει σε αυτές τον τελεστή της σύζευξη, εφαρμόζοντας την αυστηρή σημασιολογία του τελεστή για τις περιπτώσεις unbound μεταβλητών. Τέλος για την προσομοίωση του τελεστή UNION γίνεται χρήση της XQuery δομή return.

Με την διαδικασία μετάφραση των σχηματομορφών γράφων επιτυγχάνεται η μετάφραση του "βασικότερου" και πιο πολύπλοκου μέρους μιας SPARQL ερώτησης, καθώς αυτή αντιστοιχεί στην μετάφραση της where δομής της ερώτησης.

Τέλος έγινε η παρουσίαση ενός "greedy" τρόπου αποτίμησης των σχηματομορφών γράφων, η οποία μπορεί να εφαρμοστεί από την παρούσα προσέγγιση κατά την μετάφραση των ερωτήσεων ώστε να γίνει βελτιστοποίηση των XQuery ερωτήσεων που προκύπτουν μειώνοντας τον αριθμός των υπολογισμών και επαναλήψεων.

Στο επόμενο κεφάλαιο(Κεφάλαιο 13) περιγράφεται η διαδικασία "Μετάφρασης των τροποποιητών της ακολουθίας λύσεων". Παρουσιάζεται η διαδικασία μετάφραση των τροποποιητών: DISTINCT, REDUCE, LIMIT, OFFSET και ORDERBY.

13 Μετάφραση Τροποποιητών Ακολουθίας Λύσεων

13.1 Εισαγωγή

Οι σχηματομορφές (Patterns) των ερωτήσεων δημιουργούν μη ταξινομημένες ακολουθίες λύσεων, στην γλώσσα SPARQL υπάρχει η δυνατότητα εφαρμογής τροποποιητών ώστε να δημιουργηθεί μια νέα ακολουθία, η οποία είναι αυτή που θα επιστραφεί από την ερώτηση. Οι τροποποιητές εφαρμόζονται σε ερωτήσεις μορφής SELECT, CONSTRUCT ή DESCRIBE και όχι σε ερωτήσεις ASK, πιο συγκεκριμένα οι τροποποιητές DISTINCT και REDUCE εφαρμόζονται μόνο σε SELECT ερωτήσεις.

Στο παρόν κεφάλαιο αναλύεται η διαδικασία **"Μετάφρασης Τροποποιητών Ακολουθίας Λύσεων" (Solution Sequence Modifiers Translation)**. Για την μετάφραση των τροποποιητών δεν διαφοροποιούνται σε τίποτα οι διαδικασίες και αλγόριθμοι που έχουν οριστεί μέχρι τώρα. Η διαδικασία

δέχεται ως εισόδους : την πληροφορία για τους τροποποιητές που περιέχει η SPARQL ερώτηση καθώς και τις XQuery εκφράσεις που έχουν προκύψει από την μετάφραση της σχηματομορφής γράφου της SPARQL ερώτηση. Ως έξοδο η διαδικασία παράγει XQuery εκφράσεις οι οποίες παράγουν ακολουθίες λύσεων σύμφωνα με τους τροποποιητές λύσεων που εμφανίζονται στην SPARQL ερώτηση, αυτές οι εκφράσεις στην συνέχεια θα εμπλουτιστούν από την διαδικασία μετάφρασης με βάση την μορφή της ερώτησης(Κεφάλαιο 14) ώστε η μορφή των λύσεων να προσαρμοστεί ανάλογα με την μορφή της SPARQL ερώτησης.

Στην ενότητα 13.2 παρουσιάζεται η μετάφραση του τροποποιητή DISTINCT , στην ενότητα 13.3 του τροποποιητή REDUCE , στην ενότητα 13.4 του τροποποιητή LIMIT , στην ενότητα 13.4 του τροποποιητή OFFSET και τέλος στην ενότητα 13.5 του τροποποιητή ORDERBY. Επίσης αναλύεται η σειρά με την οποία πρέπει να μεταφραστούν και να εφαρμοστούν οι τροποποιητές λύσεων ώστε το αποτέλεσμα της εφαρμογής τους να είναι σύμφωνο με την σημασιολογία της γλώσσας SPARQL.

13.2 DISTINCT

Ο τροποποιητής DISTINCT χρησιμοποιείται για την εξάλειψη των διπλών λύσεων και εφαρμόζεται μόνο στις SELECT ερωτήσεις. Για την εφαρμογή του τροποποιητή DISTINCT ακολουθείται η παρακάτω μεθοδολογία. Εφόσον πραγματοποιηθεί η μετάφραση της ερώτησης με την μεθοδολογία που έχει αναλυθεί, η ακολουθία των λύσεων που δημιουργείται από την XQuery ερώτηση, ανατίθεται σε μια μεταβλητή με χρήση της δομής let, στην συνέχεια αυτή η μεταβλητή θα αποτελέσει την είσοδο για μια XQuery συνάρτηση η οποία υλοποιεί την σημασιολογία του τροποποιητή DISTINCT.

Η συνάρτηση **func:distinct** υλοποιεί τον τροποποιητή DISTINCT, δέχεται σαν είσοδο μια μεταβλητή η οποία περιλαμβάνει την ακολουθία λύσεων που έχει δημιουργηθεί από την XQuery ερώτηση που προέκυψε από την μετάφραση. Η συνάρτηση εφαρμόζει σε αυτή την ακολουθία απαλοιφή διπλών τιμών, για τις μεταβλητές τις οποίες αποτελείται η λύση. Η συνάρτηση **func:distinct** δημιουργείται κάθε φορά ανάλογα με τον αριθμό και την ονομασία των μεταβλητών, για τις οποίες θα εφαρμοστεί η απαλοιφή διπλών τιμών. Ο τρόπος δημιουργίας της συνάρτησης περιγράφεται με το παρακάτω παράδειγμα .

- **SPARQL SELECT query με DISTINCT ⇨ Μετάφραση ⇨ XQuery**
- **let $Result := (XQuery)**

  return (func:DISTINCT($Result))

Αλγόριθμος 13.1 : Μεθοδολογία Μετάφρασης του Τροποποιητή DISTINCT

Παράδειγμα 13.1

Έστω ότι προκύπτει μετά από μετάφραση το XQuery ερώτημα xquery1, το οποίο επιστρέφει τις μεταβλητές x και y στις οποίες πρέπει να εφαρμοστεί ο τροποποιητής DISTINCT. Θα προκύψει το παρακάτω XQuery ερώτημα :

```
let $result := ( xquery1 )
return ( func:DISTINCT ($result ) )
```

Η συνάρτηση **func:DISTINCT** που δημιουργείται για να εφαρμόσει τον τροποποιητή DISTINCT , για της μεταβλητές x και y θα είναι :

```
declare function func:DISTINCT ( $res as node()* )  as node()*{
        let $dist_x :=distinct-values($res/x)
        let $dist_res_x:=(
                for $dx in  $dist_x
                let $res_x:=$res[./x=$dx ]
                return $res_x[position()<=1]
        )
        let $dist_y :=distinct-values($dist_res_x/age)
        let $dist_res_xy:=(
                for $dy in  $dist_y
                let $res_y:=$dist_res_xy[./age=$dy ]
                return $res_y[position()<=1]
        )
   return $dist_res_xy
 };
```

Αλλάζοντας ονόματα μεταβλητών και με επανάληψη των παραπάνω δομών δίνεται η δυνατότητα δημιουργίας συνάρτησης για διαφορετικό αριθμό και ονόματα μεταβλητών.

13.3 REDUCE

Ο τροποποιητής REDUCED χρησιμοποιείται για τον χειρισμό των διπλών λύσεων, όμως σε αντίθεση με τον DISTINCT δεν εξαλείφει τελείως τις διπλές λύσεις, αλλά επιτρέπει να εμφανίζονται με πολλαπλότητα (cardinality) μεταξύ του ένα και της πολλαπλότητας της λύσης πριν την εφαρμογή των REDUCED και DISTINCT τροποποιητών. Η επιλογή της πολλαπλότητας δεν καθορίζεται από την παρούσα προδιαγραφή της γλώσσας SPARQL αλλά επιλέγεται από το SPARQL query engine που εκτελεί την ερώτηση. Για την προσομοίωση του τροποποιητή REDUCE χρησιμοποιείται μια παραλλαγή της συνάρτησης func:DISTINCT, η οποία δέχεται ένα επιπλέον όρισμα το οποίο καθορίζει την μέγιστη πολλαπλότητα (maximum cardinality) των τιμών των μεταβλητών. Στην περίπτωση που αυτή η τιμή οριστεί ίση με ένα, τότε η συνάρτηση υλοποιεί τον DISTINCT τροποποιητή

- **SPARQL SELECT query με REDUCE⇨ Μετάφραση ⇨ XQuery**
- **let $Result := (XQuery)**

  return (func:REDUCE($Result , maximum cardinality number))

Αλγόριθμος 13.2 : Μεθοδολογία Μετάφρασης του Τροποποιητή REDUCE

Παράδειγμα 13.2 (παραλλαγή του Παραδείγματος 13.1)

Έστω ότι προκύπτει μετά από μετάφραση το XQuery ερώτημα xquery1, το οποίο επιστρέφει τις μεταβλητές x και y στις οποίες πρέπει να εφαρμοστεί ο τροποποιητής REDUCE. Θα προκύψει το παρακάτω XQuery ερώτημα :

let $result := (xquery1)

return (func:REDUCE ($result , 2))

Στο παρόν παράδειγμα ορίζεται ως μέγιστη πολλαπλότητα η τιμή 2.

Η συνάρτηση **func: REDUCE** που δημιουργείται για να εφαρμόσει τον τροποποιητή DISTINCT , για τις μεταβλητές x και y θα είναι :

```
declare function func:REDUCE ( $res as node()* , $max_card as xs:integer )  as node()*{
        let $dist_x :=distinct-values($res/x)
        let $dist_res_x:=(
            for $dx in  $dist_x
            let $res_x:=$res[./x=$dx ]
            return $res_x[position()<=$max_card]
        )
        let $dist_y :=distinct-values($dist_res_x/age)
        let $dist_res_xy:=(
            for $dy in  $dist_y
            let $res_y:=$dist_res_xy[./age=$dy ]
            return $res_y[position()<=$max_card]
        )
    return $dist_res_xy
};
```

Αλλάζοντας ονόματα μεταβλητών και με επανάληψη των παραπάνω δομών δίνεται η δυνατότητα δημιουργίας συνάρτησης για διαφορετικό αριθμό και ονόματα μεταβλητών.

13.4 LIMIT

Ο τροποποιητής LIMIT εφαρμόζει ένα άνω όριο στον αριθμό των λύσεων που επιστρέφονται. Στην περίπτωση που ο αριθμός των λύσεων είναι μεγαλύτερος από αυτόν που προσδιορίζεται από τον LIMIT , επιστρέφονται τόσες λύσεις όσες ορίζει ο LIMIT . Εάν ο LIMIT έχει οριστεί μηδέν (0) δεν έχει καμιά επίπτωση . Για την εφαρμογή του τροποποιητή LIMIT ακολουθείται η παρακάτω μεθοδολογία. Εφόσον πραγματοποιηθεί η μετάφραση της ερώτησης με την μεθοδολογία που έχει αναλυθεί, η ακολουθία των λύσεων που δημιουργείται από την XQuery ερώτηση, ανατίθεται σε μια μεταβλητή με χρήση της δομής let, στην συνέχεια από αυτή την μεταβλητή θα επιστραφεί ο αριθμός λύσεων που ορίζεται από τον τροποποιητή LIMIT, αυτό πραγματοποιείται με την χρήση της built-in XQuery συνάρτηση position.

- **SPARQL query με LIMIT=n ⇨ Μετάφραση ⇨ XQuery**
- **let $Result := (XQuery)**

  return ($Result[position() <= n])

Αλγόριθμος 13.3 : Μεθοδολογία Μετάφρασης του Τροποποιητή LIMIT

Παράδειγμα 13.3

Έστω η ερώτηση : " Επέστρεψε τον μισθό δέκα εργαζομένων"

```
SELECT ?sal
        WHERE { ?x Salary ?sal }
LIMIT 10
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Result := (
      for  $x  in $doc/Persons/Employee
         for  $sal in  $x/Salary
         return (<Result>{ <sal>{string($sal)}</sal>}</Result>)
)
return ($Result[position() <=10] )
```

13.5 OFFSET

Ο τροποποιητής OFFSET έχει ως αποτέλεσμα, η ακολουθία λύσεων να δημιουργείται στην περίπτωση που περιέχει περισσότερες λύσεις από αυτές που προσδιορίζει ο OFFSET. Για την εφαρμογή του τροποποιητή OFFSET ακολουθείται η παρακάτω μεθοδολογία. Εφόσον πραγματοποιηθεί η μετάφραση της ερώτησης με την μεθοδολογία που έχει αναλυθεί, η ακολουθία των λύσεων που δημιουργείται από την XQuery ερώτηση, ανατίθεται σε μια μεταβλητή με χρήση της δομής let, στην συνέχεια εφαρμόζεται ο έλεγχος με χρήση της δομής where και της built-in XQuery συνάρτηση count, για τον αριθμό των λύσεων. Αν οι λύσεις είναι ίσες η και περισσότερες σε αριθμό από τον αριθμό που ορίζει το OFFSET, τότε γίνεται επιστροφή αποτελεσμάτων ,ενώ σε αντίθετη περίπτωση όχι.

- **SPARQL query με OFFSET=n ⇨ Μετάφραση ⇨ XQuery**
- **let $Result := (XQuery)**

  where (count($Result) >= n)

  return ($Result)

Αλγόριθμος 13.4 : Μεθοδολογία Μετάφρασης του Τροποποιητή OFFSET

Παράδειγμα 13.4

Έστω η ερώτηση : " Επέστρεψε τον μισθό των εργαζομένων, εάν τα αποτελέσματα είναι πάνω από πέντε"

```
SELECT ?sal
        WHERE { ?x Salary ?sal }
OFFSET 5
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Result := (
      for  $x  in $doc/Persons/Employee
         for  $sal in  $x/Salary
         return (<Result>{ <sal>{string($sal)}</sal>}</Result>)
)
 where ( count($Result) >= 5 )
 return ( $Result )
```

13.6 ORDER BY

Ο τροποποιητής ORDER BY προσδιορίζει την σειρά της ακολουθίας των λύσεων. Η δομή ORDER BY, ακολουθείται από ένα σύνολο μεταβλητών με βάση των οποίων θα πραγματοποιηθεί η ταξινόμηση της ακολουθίας των λύσεων. Προαιρετικά υπάρχει η δυνατότητα, να προσδιοριστεί η φορά της ταξινόμησης, δηλώνοντας ASC ή DESC για αύξουσα ή φθίνουσα ταξινόμηση αντίστοιχα, στην περίπτωση που δεν προσδιορίζεται η φορά ταξινόμησης, εφαρμόζεται αύξουσα. Κατά την ταξινόμηση σύμφωνα με την σημασιολογία του τροποποιητή ORDER BY, οι unbound μεταβλητές έχουν χαμηλότερη προτεραιότητα έναντι των άλλων. Για την εφαρμογή του τροποποιητή ORDER BY ακολουθείται η παρακάτω μεθοδολογία. Εφόσον πραγματοποιηθεί η μετάφραση της ερώτησης με την μεθοδολογία που έχει αναλυθεί, η ακολουθία των λύσεων που δημιουργείται από την XQuery ερώτηση, ανατίθεται σε μια μεταβλητή με χρήση της δομής let, στην συνέχεια ορίζεται μια for δομή για την διάσχιση της ακολουθίας και την εφαρμογή της XQuery δομής order by, για την ταξινόμηση της ακολουθία των λύσεων με βάση τις μεταβλητές που ορίζονται στην δομή ORDER BY της SPARQL ερώτησης. Για τις δηλώσεις φοράς ταξινόμησης ακολουθείται η αντιστοιχία: για την ASC δήλωση της SPARQL ορίζεται η **ascending** της XQuery και για την DESC δήλωση της SPARQL ορίζεται η **descending** της XQuery. Επίσης για την εφαρμογή της χαμηλής προτεραιότητας των unbound μεταβλητών, ορίζεται στην order by δομή η δήλωση **empty least**.

- **SPARQL query με ORDER BY ?x DESC(?y) ⇨ Μετάφραση ⇨ XQuery**
- **let $Result := (XQuery)**

```
let $Ordered_Result := (
        for $iter in $Result
        order by $iter/x empty least , $iter/y  descending empty least
        return ( $iter)
    )
    return ( $Ordered_Result )
```

- **SPARQL ASC ⇨ XQuery ascending**
- **SPARQL DESC ⇨ XQuery descending**
- **SPARQL Χαμηλή προταιρεότητα των unbound μεταβλητών ⇨ XQuery empty least**

Αλγόριθμος 13.5 : Μεθοδολογία Μετάφρασης του Τροποποιητή ORDER BY

Παράδειγμα 13.5

Έστω η ερώτηση : " Επέστρεψε το μικρό όνομα και τον μισθό των εργαζομένων, ταξινομημένα πρώτα με αύξουσα σειρά ως προς το μικρό όνομα και στην συνέχεια με φθίνουσα σειρά ως προς τον μισθό"

```
SELECT ?fn ?sal
        WHERE {  ?x    FirsName   ?fn .
                       ?x Salary ?sal }
ORDER BY ?fn DESC(?sal)
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Result := (
        for  $x  in $doc/Persons/Employee
        for  $fn in  $x/FirstName
        for  $sal in  $x/Salary
        return (<Result>{ <fn>{string($fn)}</fn> ,
                          <sal>{string($sal)}</sal>}</Result>)
)
let $Ordered_Result := (
      for $iter in $Result
      order by $iter/fn empty least, $iter/salary descending empty least
      return ( $iter)
)
return ( $Ordered_Result )
```

13.7 Πολλαπλή Εφαρμογή Τροποποιητών - Σειρά Εφαρμογής Τροποποιητών

Υπάρχουν περιπτώσεις στις οποίες ορίζονται παραπάνω από ένας τροποποιητές στην ιδία ερώτηση, σε αυτές τις περιπτώσεις η σειρά εφαρμογής(και μετάφρασης) των τροποποιητών στην ακολουθία λύσεων είναι προκαθορισμένη, ώστε να επιτευχθεί η επιθυμητή ακολουθία λύσεων σύμφωνα με την σημασιολογία της γλώσσας SPARQL. Αρχικά εφαρμόζονται οι τροποποιητές DISTINCT και REDUCE, στην συνέχεια εφαρμόζεται ο τροποποιητής ORDER BY και τέλος οι τροποποιητές LIMIT και OFFSET. Για λόγους βελτιστοποίησης ο τροποποιητής OFFSET εφαρμόζεται άμεσος μετά τους DISTINCT και REDUCE

ώστε στην περίπτωση που η ακολουθία περιέχει λιγότερες λύσεις από αυτές που ορίζεται από τον LIMIT να μην εφαρμόζεται (άσκοπα) ο τροποποιητής ORDER BY.

1. **DISTINCT / REDUCE**
2. **LIMIT**
3. **ORDER BY**
4. **OFFSET**

Αλγόριθμος 13.6 : Σειρά Εφαρμογής Τροποποιητών

Παράδειγμα 13.6

Έστω η ερώτηση : " Επέστρεψε τους δέκα διαφορετικούς(distinct) και πιο υψηλούς μισθούς των εργαζομένων με φθίνουσα σειρά, η ερώτηση να επιστρέψει αποτελέσματα εφόσον βρεθούν πέντε αποτελέσματα για τις παραπάνω προϋποθέσεις".

```
SELECT DISTINCT ?sal
    WHERE { ?x Salary ?sal }
ORDER BY DESC(?sal)  LIMIT 10  OFFSET 5
```

Παράγεται το εξής XQuery ερώτημα:

```
let $Result := (
      for  $x  in $doc/Persons/Employee
      for  $sal in  $x/Salary
      return (<Result>{ <sal>{string($sal)}</sal>}</Result>)
)
let $dist_result := func:DISTINCT( $Result )
where ( count ($dist_result) >= 5 )
return (
            let  $Ordered_Result := (
                  for $iter in $dist_result
                  order by $iter/salary descending empty least
                  return ( $iter )
            )
            return ($Ordered_Result[position() <=10] )
)
```

Η συνάρτηση func:DISTINCT που θα δημιουργηθεί :

```
declare function func:DISTINCT ( $res as node()* )  as node()*{
                    let $dist_sal :=distinct-values($res/sal)
                    let $dist_res_sal:=(
                            for $dsal in  $dist_sal
                            let $res_sal:=$res[./sal=$dsal ]
                            return $res_sal[position()<=1]
                    )
                    return $dist_res_sal
    };
```

13.8 Επίλογος

Στο κεφάλαιο αυτό αναλύεται η διαδικασία μετάφρασης των τροποποιητών της ακολουθίας των λύσεων, παρουσιάζεται η διαδικασία μετάφραση των τροποποιητών: DISTINCT, REDUCE, LIMIT, OFFSET , ORDERBY. Για την προσομοίωση του τροποποιητή DISTINCT δημιουργείται μια XQuery συνάρτηση η οποία υλοποιεί την σημασιολογία του τροποποιητή, το ίδιο εφαρμόζεται και για τον τροποποιητή REDUCE. Για την προσομοίωση του τροποποιητή LIMIT γίνεται χρήση της XQuery συνάρτησης position, ενώ για τον τροποποιητή OFFSET γίνεται χρήση της XQuery συνάρτησης count. Τέλος για την προσομοίωση του τροποποιητή ORDERBY γίνεται χρήση της XQuery συνάρτησης orderby και των XQuery δηλώσεων empty least.

Επίσης αναλύεται η σειρά με την οποία πρέπει να μεταφραστούν και να εφαρμοστούν οι τροποποιητές λύσεων ώστε το αποτέλεσμα της εφαρμογής τους να είναι σύμφωνο με την σημασιολογία της γλώσσας SPARQL. Οι XQuery εκφράσεις που παράγει η διαδικασία ως έξοδο, παράγουν ακολουθίες λύσεων σύμφωνα με τους τροποποιητές λύσεων που εμφανίζονται στην SPARQL ερώτηση, αυτές οι εκφράσεις στην συνέχεια θα εμπλουτιστούν από την διαδικασία μετάφρασης με βάση την μορφή της ερώτησης(Κεφάλαιο 14) ώστε η μορφή των λύσεων να προσαρμοστεί ανάλογα με την μορφή της SPARQL ερώτησης.

14 Μετάφραση με Βάση την Μορφή των Ερωτήσεων

14.1 Εισαγωγή

Όπως έχει αναλυθεί και στην Παράγραφο 2.7.3 η γλώσσα SPARQL έχει τέσσερις μορφές ερωτήσεων, ανάλογα με την μορφή της ερώτησης είναι και η μορφή των αποτελεσμάτων, για τον λόγο αυτόν διαφοροποιείται και ο τρόπος μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους. Οι αλγόριθμοι και η μεθοδολογία που έχει αναλυθεί μέχρι αυτό το σημείο είναι ανεξάρτητη από την μορφή των ερωτήσεων και δεν διαφοροποιείται. Μια διαφοροποίηση εμφανίζεται στην σύνταξη της δομής return για τις ερωτήσεις ASK μορφής, όπως έχει αναφερθεί και στην υπό-ενότητα 11.5.2. Επίσης διαφοροποιείται ο τρόπος διαχείρισης των λύσεων ανάλογα με την μορφή της ερώτησης.

Στο παρόν κεφάλαιο περιγράφεται αναλυτικά ο τρόπος μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους. Η διαδικασία "μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους" δέχεται ως είσοδο

τις XQuery εκφράσεις που έχουν προκύψει από την μετάφραση τόσο της σχηματομορφής γράφου της ερώτησης όσο και των τροποποιητών λύσεων, τις επεξεργάζεται και ανάλογα με την μορφή της SPARQL ερώτησης τις εμπλουτίζει ώστε η μορφή των λύσεων να προσαρμοστεί ανάλογα με την μορφή της SPARQL ερώτησης.

Στην υπό-ενότητα 14.2 περιγράφεται η διαδικασία μετάφρασης SELECT SPARQL ερωτήσεων, στην υπό-ενότητα 14.3 περιγράφεται η διαδικασία μετάφρασης ASK SPARQL ερωτήσεων, στην υπό-ενότητα 14.4 περιγράφεται η διαδικασία μετάφρασης CONSTRUCT SPARQL ερωτήσεων και τέλος στην υπό-ενότητα 14.5 περιγράφεται η διαδικασία μετάφρασης DESCRIBE SPARQL ερωτήσεων.

14.2 SELECT

Οι ερωτήσεις SELECT επιστέφουν μια ακολουθία από λύσεις, οι οποίες περιέχουν τις τιμές που έχουν αντιστοιχηθεί με τις μεταβλητές. Οι μεταβλητές για τις οποίες θα επιστραφούν τιμές ορίζονται στην δομή SELECT.

Στις XQuery ερωτήσεις οι μεταβλητές για τις οποίες θα επιστραφούν τιμές προσδιορίζονται στην δομή return, ο προσδιορισμός των μεταβλητών που θα περιέχονται στο return γίνεται με βάση τις μεταβλητές που προσδιορίζονται στην SELECT δομή της SPARQL ερώτηση(βλέπε και υπό-ενότητα 11.3.5 για τις μεταβλητές που περιέχονται στην return δομή). Όπως έχει αναφερθεί στην υπό-ενότητα 11.5.1 τα αποτελέσματα των ερωτήσεων θα είναι σε μορφή XML, η εκτέλεση της ερώτησης επιστρέφει μια ακολουθία από XML στοιχεία. Για να αποτελούν τα αποτελέσματα ένα έγκυρο(valid) XML έγγραφο θα πρέπει να έχουν ένα ριζικό στοιχείο (root element). Που δεν ισχύει για τα αποτελέσματα που παράγονται από την εκτέλεση της ερώτησης, καθώς είναι μια ακολουθία από XML στοιχεία της μορφής <Result> ...</Result> , <Result> ...</Result> … <Result> ...</Result>. Επομένως είναι απαραίτητο η δημιουργία ενός ριζικού στοιχείου που θα περιέχει την ακολουθία των αποτελεσμάτων.

Για την δημιουργία του ριζικού στοιχείου, ορίζεται μια δομή Let η οποία κάνει ανάθεση σε μια μεταβλητή την ερώτηση που έχει προκύψει από την μετάφραση. Στην συνέχεια ορίζεται μια return δομή η οποία επιστρέφει την μεταβλητή, περιβαλλομένη από δυο XML tags (<Results> </Results>). Επομένως πλέον τα αποτελέσματα αποτελούν έγκυρο XML έγγραφο της μορφής :

<Results>
<Result> <x>x_1</x> <y>y_1</y> <z>z_1</z> … </Result>
<Result> <x>x_2</x> <y>y_2</y> <z>z_2</z> … </Result>
<Result> <x>x_3</x> <y>y_3</y> <z>z_3</z> … </Result>
….
….
<Result> <x>x_n</x> <y>y_n</y> <z>z_n</z> … </Result>
<Results>

Τα αποτελέσματα στην συνέχεια μετασχηματίζονται σύμφωνα με την XML μορφοποίηση (SPARQL Query Results XML Format)[56] , που προτείνεται από τον W3C για την αναπαράσταση και αποθήκευση των αποτελεσμάτων των SPARQL ερωτήσεων. (για περισσότερες λεπτομέρειες βλέπε υπό-ενότητα 16.2)

- **SPARQL SELECT query ⇨ Μετάφραση ⇨ XQuery**
- **let $Results := (XQuery)**

  return (<Results> $Results </Results>)

Αλγόριθμος 14.1 : Μεθοδολογία Μετάφρασης των SELECT ερωτήσεων

Παράδειγμα 14.1

Έστω η ερώτηση : " Για όλα τα άτομα (και εργαζόμενοι) επέστρεψε το/α μικρό/ά όνομα/τα τους"

```
SELECT ?Fn
WHERE { ?x FirstName ?Fn . }
```

Με χρήση του αλγόριθμου BGP2XQuery παράγεται XQuery ερώτημα:

```
for $x in $doc/Persons/Person union $doc/Persons/Employee
for $Fn in $x/FirstName
return (<Result>{<Fn>{string($Fn)}</Fn>}</Result>)
```

Για την δημιουργία ριζικού στοιχείου :

```
let $Results := (
      for  $x  in $doc/Persons/Person union $doc/Persons/Employee
      for  $Fn in  $x/FirstName
      return (<Result>{<Fn>{string($Fn)}</Fn>}</Result>)
)
return (<Results> $Results </Results>)
```

14.3 ASK

Οι ερωτήσεις ASK δεν επιστρέφουν πληροφορίες για πιθανές λύσεις της ερώτησης, παρά μόνο αν η ερώτηση έχει λύση ή όχι, επιστρέφοντας yes ή no . Αυτός είναι και ο λόγος για τον οποίο οι return δομές των XQuery ερωτήσεων, που έχουν προκύψει από μετάφραση SPARQL ASK ερωτήσεων δεν περιέχουν μεταβλητές.

Όπως γίνεται αντιληπτό, μια ερώτηση που περιέχει στο return την συμβολοσειρά "yes" , θα επιστρέψει την συμβολοσειρά όσες φορές θα επαληθευθούν οι δομές της ερωτήσεις. Όμως σύμφωνα με την σημασιολογία των SPARQL ASK ερωτήσεων, επιστρέφεται ένα μόνο "yes" ή "no". Για να επιτευχθεί αυτό, πραγματοποιείται η ανάθεση των αποτελεσμάτων της ερώτησης XQuery που έχει προκύψει από την μετάφραση, σε μια μεταβλητή με χρήση της δομής let και στην συνέχεια ελέγχεται αν η ακολουθία είναι κενή. Αν δεν είναι κενή σημαίνει ότι η ερώτηση θα επέστρεψε αποτελέσματα επομένως επιστρέφεται ένα yes, στην αντίθετη περίπτωση που η ακολουθία είναι κενή επιστρέφεται no. Το αποτέλεσμα στην συνέχεια μετασχηματίζεται σύμφωνα με την XML μορφοποίηση (SPARQL Query Results XML Format)[56] , που προτείνεται από τον W3C για την αναπαράσταση και αποθήκευση των αποτελεσμάτων των SPARQL ερωτήσεων. (για περισσότερες λεπτομέρειες βλέπε υπό-ενότητα 16.2)

Σημείωση : *Όπως γίνεται αντιληπτό από τον τρόπο με τον οποίον γίνεται ο ελέγχου για την επιστροφή του αποτελέσματος, δεν ήταν απαραίτητο η διαφοροποίηση της δομής return από της άλλες ερωτήσεις, καθώς ελέγχεται αν έγινε επιστροφή αποτελεσμάτων, αυτό έγινε για λογούς βελτιστοποίησης της ερώτησης.*

- **SPARQL ASK query ⇨ Μετάφραση ⇨ XQuery**
- **let $Results := (XQuery)**

  return (if (empty($Results)) then "no" else "yes")

Αλγόριθμος 14.2: Μεθοδολογία Μετάφρασης των ASK ερωτήσεων

Παράδειγμα 14.2

Έστω η ερώτηση : " Υπάρχει λύση για την ερώτηση : για όλα τα άτομα (και εργαζόμενοι) επέστρεψε το/α μικρό/ά όνομα/τα τους"

```
ASK  ?Fn
WHERE { ?x FirstName ?Fn .  }
```

Με χρήση του αλγόριθμου BGP2XQuery παράγεται XQuery ερώτημα:

```
let  $x  := $doc/Persons/Person union $doc/Persons/Employee
let  $Fn :=  $x/FirstName
return ("yes")
```

Έλεγχος των αποτελεσμάτων για την επιστροφή ενός μόνο yes ή no :

```
let $Results :=(
      let  $x  := $doc/Persons/Person union $doc/Persons/Employee
      let  $Fn :=  $x/FirstName
      return ("yes")
  )
  return ( if ( empty($Results)) then "no" else "yes" )
```

14.4 CONSTRUCT

Οι ερωτήσεις CONSTRUCT επιστέφουν έναν RDF γράφο, η μορφή του οποίου προσδιορίζεται από έναν γράφο πρότυπο (graph pattern). Ο γράφος δημιουργείται περνώντας κάθε λύση από την ακολουθία λύσεων και αντικαθιστώντας τις τιμές των μεταβλητών, στις αντίστοιχες μεταβλητές των σχηματομορφών τριπλέτων του RDF γράφου. Ο τελικός γράφος παράγεται από την ένωση των σχηματομορφών τριπλέτων που έχουν δημιουργηθεί για όλες τις λύσεις της ακολουθίας των λύσεων. Στην περίπτωση που σε κάποια μεταβλητή μιας σχηματομορφής τριπλέτας δεν έχει ανατεθεί τιμή (δηλαδή η μεταβλητή είναι Unbound), τότε η τριπλέτα αυτή δεν περιλαμβάνεται στον τελικό γράφο. Υπάρχει η δυνατότητα ο γράφος πρότυπο να περιέχει και κενούς κόμβους (blank nodes), σε αυτή την περίπτωση για κάθε λύση από την ακολουθία λύσεων, θα δημιουργείται και ένας διαφορετικός κενός κόμβος. Για να πραγματοποιηθούν τα παραπάνω στις XQuery ερωτήσεις ακολουθείται η παρακάτω διαδικασία.

Τα αποτελέσματα της XQuery ερώτηση, που έχει προκύψει από την μετάφραση της SPARQL ερώτησης, ανατίθενται σε μια μεταβλητή με χρήση της δομής let. Στην συνέχεια ορίζεται μια δομή for για την διάσχιση της ακολουθίας λύσεων. Επίσης ορίζεται και μια return δομή η οποία περιέχει τον πρότυπο γράφο. Στην θέση κάθε μεταβλητής του πρότυπου γράφου τοποθετείται η αντίστοιχη τιμή από την διάσχιση της ακολουθίας των λύσεων. Για κάθε τριπλέτα σχηματομορφής του πρότυπου γράφου που περιέχει μεταβλητές, ελέγχεται αν αυτές οι μεταβλητές είναι unbound, στην περίπτωση που είναι unbound, η τριπλέτα δεν επιστρέφεται. Επίσης για τον χειρισμό των κενών κόμβων του πρότυπου γράφου, οι οποίοι πρέπει να έχουν διαφορετική τιμή για κάθε διαφορετική λύση από την ακολουθία λύσεων, ορίζεται μια μεταβλητή θέσης (positional variable) $iter στην For δομή, η οποία επισυνάπτεται στην συμβολοσειρά "_:bn" , δημιουργώντας με αυτό τον τρόπο διαφορετικό κενό κόμβο για κάθε λύση από την ακολουθία λύσεων.

Σημείωση: *Στην γλώσσα XQuery η μεταβλητή θέσης (positional variable) ορίζεται στην δομή For με χρήση της λέξης at. Η μεταβλητή αυτή, παίρνει ακέραιες τιμές ξεκινώντας από το ένα και αυξάνεται κάθε φορά που εκτελείται επανάληψη στην δομή for .*

- **SPARQL CONSTRUCT query ⇨ Μετάφραση ⇨ XQuery**
- **Έστω CONSTRUCT GRAPH Template : _:a iri:property ?x**

  _:a ?p ?y
- **let $Results := (XQuery)**

  for $res at $iter in $Results //επανάληψη στην ακολουθία λύσεων

  return (

  if (exists($res/x)) then //έλεγχος αν η μεταβλητή x είναι unbound

  //δημιουργία του triple του template graph με την τιμή της μεταβλητής

  (concat ("_:bn" , $iter), "iri:property" , string($res/x) , ".")

  else ( ) , // αν η μεταβλητή x είναι unbound το triple δεν επιστέφεται

  if (exists($res/p) and exists($res/y)) then

  (concat("_:bn" , $iter), string($res/p) , string($res/y) , ".")

  else ( )

)

Αλγόριθμος 14.3 : Μεθοδολογία Μετάφρασης των CONSTRUCT ερωτήσεων

Παράδειγμα 14.3

Έστω η ερώτηση που δημιουργεί έναν γράφο βασισμένο στο FOAF Vocabulary [55] , χρησιμοποιώντας το/α μικρό/α όνομα/τα και το/α επίθετο/α των ατόμων**.**

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

CONSTRUCT { _:v foaf:firstname   ?fn .
                _:v foaf:surname    ?ln }

WHERE { ?x FirstName ?fn .
        ?x LastName ?ln  }
```

Με χρήση των αλγόριθμων μετάφρασης παράγεται XQuery ερώτημα:

```
for  $x  in $doc/Persons/Person union $doc/Persons/Employee
for  $fn in  $x/FirstName
for  $ln in  $x/LastName
return (<Result>{<fn>{string($fn)}</fn>,
                 <ln>{string($ln)}</ln>}</Result>)
```

Δημιουργία του γράφου σύμφωνα με την σημασιολογία του CONSTRUCT :

```
let $Results :=(
    for  $x  in $doc/Persons/Person union $doc/Persons/Employee
    for  $fn in  $x/FirstName
    for  $ln in  $x/LastName
    return (<Result>{<fn>{string($fn)}</fn>,
                     <ln>{string($ln)}</ln>}</Result>)
 )
for $res at $iter in $Results
return ( if ( exists( $res/fn) ) then
            (concat("_:bn" , $iter ), "foaf:firstname" , string($res/fn) , "." )
      else  ( ) ,
      if ( exists( $res/ln) )then
            (concat("_:bn" , $iter ), "foaf:surname" , string($res/ln) , ".")
      else  ( )
)
```

14.5 DESCRIBE

Οι ερωτήσεις αυτής της μορφής επιστέφουν έναν RDF γράφο ο οποίος περιλαμβάνει RDF δεδομένα τα οποία "περιγράφουν" τις πηγές (Resources). Αντίθετα με τις CONSTRUCT ερωτήσεις η μορφή του γράφου δεν προσδιορίζεται από την ερώτηση αλλά ορίζεται από το SPARQL query engine. Η σχηματομορφή (Pattern) του ερωτήματος χρησιμοποιείται για να δημιουργηθεί ένα σύνολο αποτελεσμάτων. Οι ερωτήσεις αυτής της μορφής για κάθε πηγή που εμφανίζεται στα αποτελέσματα ή για κάθε πηγή που προσδιορίζεται από IRI, δημιουργούν μια "Περιγραφή"(Description) βασισμένη στα

RDF δεδομένα. Τα αποτελέσματα που παράγει μια ερώτηση αυτής της μορφής δεν καθορίζονται από την τρέχουσα προδιαγραφή της γλώσσας, αλλά καθορίζονται από το SPARQL query engine που εκτελεί την ερώτηση.

Εφόσον τα αποτελέσματα τα οποία παράγονται από ερωτήσεις αυτής της μορφής, δεν καθορίζονται από την τρέχουσα προδιαγραφή της γλώσσας αλλά καθορίζονται από το engine που εκτελεί την ερώτηση, επιλέχτηκε σαν πρώτη προσέγγιση μετάφρασης αυτών των ερωτήσεων η παρακάτω. Οι ερωτήσεις μεταφράζονται σαν να ήταν ερωτήσεις μορφής SELECT, οι οποίες παράγουν ένα XML σύνολο αποτελεσμάτων. Στην συνέχεια εκτελείται η DESCRIBE SPARQL ερώτηση από κάποιο SPARQL query engine, παράγοντας έναν RDF γράφο με πληροφορίες από την δομή και σημασιολογία της οντολογίας για τα περιεχόμενα της σχηματομορφής της ερώτησης. Τα τελικά αποτελέσματα από την μετάφραση της ερώτησης, είναι μια ένωση των XML αποτελεσμάτων που παράγονται από την XQuery ερώτηση και του RDF γράφου που παράγεται από την εκτέλεση της DESCRIBE SPARQL ερώτησης και "περιγράφει" τα περιεχόμενα της σχηματομορφής της ερώτησης. Τα αποτελέσματα μετασχηματίζονται σε μια επέκταση της XML μορφοποίησης (SPARQL Query Results XML Format) [56] , που προτείνεται από τον W3C για την αναπαράσταση και αποθήκευση των αποτελεσμάτων των SPARQL ερωτήσεων. (για περισσότερες λεπτομέρειες βλέπε υπό-ενότητα 16.2)

Η μετάφραση των ερωτήσεων αυτής της μορφής, ώστε να παράγεται RDF γράφου ο οποίος θα δημιουργεί "περιγραφή" δεδομένων με χρήση των XML δεδομένων, αποτελεί μια μελλοντική επέκταση της παρούσας προσέγγισης. Όπως επίσης και η μετάφραση με βάση πιθανής μελλοντικής προδιαγραφής της γλώσσας SPARQL η οποία θα προσδιορίζει την σημασιολογία των αποτελεσμάτων των ερωτήσεων αυτής της μορφής.

14.6 Επίλογος

Σε αυτό το κεφάλαιο έγινε η περιγραφή του τρόπου μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους. Η διαδικασία "μετάφρασης των ερωτήσεων ανάλογα με την μορφή τους" δέχεται ως είσοδο τις XQuery εκφράσεις που έχουν προκύψει από την μετάφραση τόσο της σχηματομορφής γράφου της ερώτησης όσο και των τροποποιητών λύσεων, τις επεξεργάζεται και ανάλογα με την μορφή της SPARQL ερώτησης τις εμπλουτίζει ώστε η μορφή των λύσεων να προσαρμοστεί ανάλογα με την μορφή της SPARQL ερώτησης.

Στο επόμενο κεφάλαιο(Κεφάλαιο 15) παρουσιάζεται η μετάφραση των ενσωματωμένων(built-in) τελεστών της γλώσσας SPARQL και αναλύεται ο τρόπος προσομοίωση-μετάφρασης τους με την γλώσσα XQuery. Οι τελεστές αυτοί μπορούν να χρησιμοποιηθούν στις εκφράσεις των φίλτρων των SPARQL ερωτήσεων.

15 Μετάφραση των Ενσωματωμένων Τελεστών

15.1 Εισαγωγή

Στη παρούσα προδιαγραφή της γλώσσας SPARQL ορίζεται ένα σύνολο από ενσωματωμένους (built-in) τελεστές/συναρτήσεις. Οι τελεστές αυτοί μπορούν να χρησιμοποιηθούν στις εκφράσεις των φίλτρων. Στο παρόν κεφάλαιο θα αναλυθεί ο τρόπος προσομοίωση-μετάφρασης τους με την γλώσσα XQuery. Όπως είναι λογικό δεν είναι δυνατόν να προσομοιωθούν όλοι οι τελεστές, καθώς αναφέρονται σε RDF δεδομένα.

Οι παρακάτω προσομοιώσεις-μεταφράσεις των τελεστών εφαρμόζονται στις περιπτώσεις όπου τα ορίσματα τους μπορούν να μεταβληθούν κατά την εκτέλεση της ερώτησης, δηλαδή αποτελούν μεταβλητές ή τιμές επιστροφής άλλων τελεστών. Στην αντίθετη περίπτωση(που η τιμή των ορισμάτων

είναι σταθερή) η τιμή που προκύπτει από την εφαρμογή του τελεστή, υπολογίζεται πριν από την μετάφραση της ερώτησης .

Παρακάτω αναφέρονται μόνο οι τελεστές που εισάγει η γλώσσα SPARQL και όχι οι τελεστές της SPARQL που υλοποιούνται έμμεσα με τελεστές της XQuery, όπως πχ < ,> , >= ,<= , = ,!= κτλ .

15.2 bound

xsd:boolean bound (variable var)

Ο τελεστής bound ελέγχει αν στην μεταβλητή έχει ανατεθεί κάποια τιμή ή είναι unbound μεταβλητή. Ο τελεστής αυτός προσομοιώνεται με την ενσωματωμένη συνάρτηση της XQuery **fn:exists**.

15.3 isURI

xsd:boolean isIRI (RDF term term)

xsd:boolean isURI (RDF term term)

Οι παραπάνω τελεστές έχουν την ίδια λειτουργία και ελέγχουν αν ο RDF όρος είναι IRI. Στην παρούσα προσέγγιση ο διαχωρισμός μεταξύ IRI και σταθερών (literal) είναι εφικτός μόνο μετά την εκτέλεσης της ερώτησης και όχι κατά την διάρκεια εκτέλεσης της. Οπότε δεν είναι δυνατή η προσομοίωση αυτής της συνάρτησης στην γλώσσα XQuery.

15.4 isBlank

xsd:boolean isBlank (RDF term term)

Ο τελεστής isBlank ελέγχει αν ο RDF όρος είναι κενός κόμβος(blank node). Δεν μπορεί να προσομοιωθεί στην XQuery καθώς στα XML δεδομένα δεν υπάρχουν κενοί κόμβοι.

15.5 isLiteral

xsd:boolean isLiteral (RDF term term)

Ο τελεστής isLiteral ελέγχει αν ο RDF όρος είναι σταθερά (literal). Στην προσέγγιση μας ο διαχωρισμός μεταξύ IRI και σταθερών (literal) είναι εφικτός μόνο μετά την εκτέλεσης της ερώτησης και όχι κατά την διάρκεια εκτέλεσης της. Οπότε δεν είναι δυνατή η προσομοίωση αυτής της συνάρτησης στην γλώσσα XQuery.

15.6 str

simple literal str (literal ltrl)
simple literal str (IRI rsrc)

Ο τελεστής str επιστρέφει την lexical form μια σταθεράς ή ενός IRI . Στην περίπτωση που η είσοδος του τελεστή προέρχεται από τιμή μεταβλητής ή από τιμή επιστροφής τελεστή τότε προσομοιώνεται από την συνάρτηση **fn:string** . Ενώ σε αντίθετη περίπτωση που η είσοδος του τελεστή είναι κάτι σταθερό, που δεν μεταβάλλεται κατά την διάρκεια εκτέλεσης της ερώτησης (πχ μια συμβολοσειρά), το αποτέλεσμα του τελεστή δεν μεταβάλλεται και επόμενος μπορεί να υπολογιστεί πριν την μετάφραση της ερώτησης και στην συνέχεια να χρησιμοποιηθεί.

15.7 datatype

IRI datatype (typed literal typedLit)
IRI datatype (simple literal simpleLit)

Ο τελεστής datatype επιστρέφει το IRI του datatype της σταθεράς που δέχεται σαν είσοδο. Όπως γίνεται κατανοητό αυτός ο τελεστής δεν γίνεται να υλοποιηθεί από την XQuery, καθώς δεν είναι δυνατόν να προσδιοριστεί ο τύπος κάποιας μεταβλητής στις ερωτήσεις XQuery.

15.8 logical-or

xsd:boolean xsd:boolean left || xsd:boolean right

Το λογικό OR (||) , προσομοιώνεται με το **or** στην XQuery.

15.9 logical-and

xsd:boolean xsd:boolean left && xsd:boolean right

Το λογικό AND (&&), προσομοιώνεται με το **and** στην XQuery.

15.10 sameTerm

xsd:boolean sameTerm (RDF term term1, RDF term term2)

Ο τελεστής sameTerm ελέγχει αν οι δυο RDF όροι που δέχεται σαν είσοδο είναι ίδιοι. Στην περίπτωση που τουλάχιστον ένα από τα δυο ζορίσματα είναι σταθερά ο τελεστής προσομοιώνεται με την χρήση του τελεστή **=** στην XQuery. Σε αντίθετη περίπτωση (δηλαδή που τα ορίσματα είναι και τα δυο μεταβλητές) προσομοιώνεται με την συνάρτηση **func:sameTerm** (Εικόνα 15.1) . Ο έλεγχος πραγματοποιείται είτε ανάμεσα σε σύνθετα στοιχεία, ελέγχοντας την ισότητα τους με βάση το μονοπάτι τους σε συνδυασμό με αρίθμηση κόμβων, είτε μεταξύ απλών στοιχείων εφαρμόζοντας τον τελεστή = .

```
declare function func:sameTerm( $nodeA as node() ,$nodeB as node())  as xs:boolean {
if( (exists($nodeA/*) or  exists($nodeA/@*)) and (exists($nodeB/*) or  ex-ists($nodeB/@*)) )then
   (  func:nodeURI($nodeA) =func:nodeURI($nodeB)  )
else
   ($nodeA =$nodeB ))
};
```

Εικόνα 15.1 : Συνάρτηση func:sameTerm η οποία προσομοιώνει την SPARQL συνάρτηση sameTerm

15.11 RDFterm-equal

xsd:boolean RDF term term1 = RDF term term2

Ισχύουν τα ίδια με τον τελεστής sameTerm. Καθώς η διαφοροποίηση τους εμφανίζεται στην σύγκριση των typed literal, κάτι που δεν υπάρχει στην XML .

15.12 lang

simple literal lang（literal ltrl）

Ο τελεστής lang επιστρέφει το language tag μιας σταθεράς. Όπως είναι γνωστό στα RDF δεδομένα για την δήλωση του language tag χρησιμοποιείται το xml:lang χαρακτηριστικό. Επομένως όπως γίνεται αντιληπτό και στο επίπεδο των XML δεδομένων το language tag θα ισούται με το χαρακτηριστικό xml:lang. Για την προσομοίωση του SPARQL τελεστή lang δημιουργήθηκε η XQuery συνάρτηση **func:return_lang**.(δεν ονομάστηκε func:lang για να αποφευχθεί η σύγχυση με την built-in XQuery συνάρτηση fn:lang) Η **func:return_lang** (Εικόνα 15.2) δέχεται σαν όρισμα ένα σύνολο από κόμβους και επιστρέφει ένα σύνολο με τα πιθανά xml:lang τους χαρακτηριστικά. Στην περίπτωση που ένα κόμβος δεν είναι φίλο(δηλαδή περιέχει και άλλους κόμβους), ελέγχεται αν το χαρακτηριστικό περιέχεται και στους βαθύτερους κόμβους.

Τα παραπάνω ισχύουν στην περίπτωση που η είσοδος του τελεστή προέρχεται από τιμή μεταβλητής ή από τιμή επιστροφής τελεστή . Σε αντίθετη περίπτωση που η είσοδος του τελεστή είναι κάτι σταθερό (πχ μια συμβολοσειρά), το αποτέλεσμα του τελεστή δεν μεταβάλλεται και επόμενος μπορεί να υπολογιστεί πριν την μετάφραση της ερώτησης και στην συνέχεια να χρησιμοποιηθεί.

```
declare function func:return_lang ( $nodes as node()* )  as xs:string * {
  ($nodes/@xml:lang ,$nodes/*/@xml:lang)
} ;
```

Εικόνα 15.2 : Συνάρτηση func:return_lang η οποία προσομοιώνει την SPARQL συνάρτηση lang

15.13 regex

xsd:boolean regex (simple literal text, simple literal pattern)

xsd:boolean regex (simple literal text, simple literal pattern, simple literal flags)

Ο τελεστής regex καλεί την XQuery συνάρτηση fn:matches , ώστε να γίνει έλεγχος ταιριάσματος του κειμένου text (το πρώτο όρισμα της συνάρτησης) έναντι στην κανονική έκφραση pattern (το δεύτερο όρισμα της συνάντησης). Οπότε για αυτόν τον τελεστής υπάρχει απόλυτη αντιστοιχία με την συνάρτηση **fn:matches** .

15.14 Επίλογος

Στο παρόν κεφάλαιο αναλύθηκε ο τρόπος προσομοίωση-μετάφρασης των ενσωματωμένων (built-in) τελεστών/συναρτήσεων που περιέχονται στην παρούσα προδιαγραφή της γλώσσας SPARQL με χρήση της γλώσσας XQuery. Όπως είναι λογικό δεν ήταν δυνατόν να προσομοιωθούν όλοι οι τελεστές, καθώς αναφέρονται σε RDF δεδομένα, όμως αυτό επιτεύχθηκε για το μεγαλύτερο μέρος των τελεστών.

Στο επόμενο κεφάλαιο(Κεφάλαιο 16) αναλύεται η διαδικασία "Μετασχηματισμού Αποτελεσμάτων" (Result Transformation), η οποία δέχεται ως είσοδο τα XML αποτέλεσμα που προκύπτουν από την εκτέλεση των XQuery ερωτήσεων, πραγματοποιεί τον μετασχηματισμό τους ώστε τα αποτελέσματα να είναι έγκυρα (valid) ως προς την XML μορφή αναπαράστασης των αποτελεσμάτων που προτείνεται από το W3C και ως έξοδο παράγει ένα XML έγγραφο που περιέχει τα αποτελέσματα της ερώτησης.

16 Μετασχηματισμός Αποτελεσμάτων

16.1 Εισαγωγή

Όπως είναι γνωστό η XML προσφέρει **συντακτική** και **δομική διαλειτουργικότητα**, για τον λόγο αυτό αποτελεί το κυρίαρχο πρότυπο ανταλλαγής δεδομένων στο σημερνό διαδίκτυο, το ίδιο ισχύει και για την ανταλλαγή δεδομένων στο Σημασιολογικό Ιστό (Semantic Web). Για τον λόγο αυτό έχει προταθεί από το W3C η XML μορφή αναπαράστασης των αποτελεσμάτων, που προέρχονται από SELECT και ASK SPARQL ερωτήσεις. Με την χρήση της XML μορφής αποτελεσμάτων, επιτυγχάνεται συντακτική και δομική διαλειτουργικότητα, δίνοντας την δυνατότητα σε χρήστες, εφαρμογές και πράκτορες (agents) να επεξεργάζονται και να αποθηκεύουν τα δεδομένα που προέρχονται από τις SPARQL ερωτήσεις.

Η διαδικασία **"Μετασχηματισμού Αποτελεσμάτων" (Result Transformation)**, δέχεται ως είσοδο τα XML αποτέλεσμα που προκύπτουν από την εκτέλεση των XQuery ερωτήσεων, πραγματοποιεί τον μετασχηματισμό τους ώστε τα αποτελέσματα να είναι έγκυρα (valid) ως προς την XML μορφή αναπαράστασης των αποτελεσμάτων που προτείνεται από το W3C και ως έξοδο παράγει ένα XML έγγραφο που περιέχει τα αποτελέσματα της ερώτησης και είναι έγκυρο ως προς την XML μορφή αναπαράστασης των αποτελεσμάτων που προτείνεται από το W3C.

Στο παρόν κεφάλαιο περιγράφεται η XML μορφή των SPARQL αποτελεσμάτων, όπως αυτή προτείνεται από το W3C (υπό-ενότητα 16.2), καθώς και μια επέκταση της για την αναπαράσταση αποτελεσμάτων που προέρχονται από DESCRIBE ερωτήσεις (υπό-ενότητα 16.2.2). Τέλος περιγράφεται η διαδικασία μετασχηματισμού των αποτελεσμάτων που επιστρέφονται από τις XQuery ερωτήσεις, ώστε να ακολουθούν την XML μορφή που προτείνεται για τα SPARQL αποτελέσματα(υπό-ενότητα 16.3).

16.2 XML Μορφή Αποτελεσμάτων

Για την αποθήκευση και διακίνηση των αποτελεσμάτων που προέρχονται από ερωτήσεις SELECT και ASK μορφής, το W3C RDF Data Access Working Group (DAWG) [58] που αποτελεί μέρος του W3C Semantic Web Activity [59] έχει αναπτύξει και αποτελεί W3C σύσταση (Recommendation) από της 15 Ιανουαρίου 2008, ένα SPARQL έγγραφο αποτελεσμάτων (SPARQL Result Document) [56].

> **Ορισμός 16.1** [56] **SPARQL έγγραφο αποτελεσμάτων (SPARQL Result Document)** Ένα SPARQL έγγραφο αποτελεσμάτων (SPARQL Result Document) είναι ένα έγγραφο έγκυρο (valid) ως προς το RELAX NG XML[60] ή ως προς το W3C XML Schema [19][20] .

Όπως έχει αναφερθεί το SPARQL έγγραφο αποτελεσμάτων αναπαριστά αποτελέσματα τα οποία προέρχονται από ASK και SELECT SPARQL ερωτήσεις. Λόγω της ιδιαιτερότητας των αποτελεσμάτων των DESCRIBE ερωτήσεων στην παρούσα προσέγγιση, ήταν δυνατή η περιγραφή τους με XML μορφή. Για τον λόγω αυτό πραγματοποιηθεί επέκταση της XML μορφής που προτείνεται από το W3C, ώστε να είναι περιγράφονται και τα αποτελεσμάτων από DESCRIBE ερωτήσεις.

16.2.1 Περιγραφή της Προτεινόμενης από το W3C XML μορφή

Η γραφική απεικόνιση του XML σχήματος φαίνεται στην Εικόνα 16.2. Το SPARQL έγγραφο αποτελεσμάτων έχει ως ριζικό στοιχείο το στοιχείο SPARQL το οποίο ορίζεται στο http://www.w3.org/2005/sparql-results# namespace. Το SPARQL στοιχείο περιέχει δυο στοιχεία, το head και το στοιχείο των αποτελεσμάτων που είναι είτε το results είτε το boolean. Το στοιχείο των αποτελεσμάτων εμφανίζεται ακόμα και όταν είναι άδειο.

16.2.1.1 Το head Στοιχείο

Το στοιχείο head είναι το πρώτο στοιχείο του στοιχείου sparql. Για τις SELECT ερωτήσεις, περιέχει μια ακολουθία από κενά στοιχεία variable με ένα χαρακτηριστικό name. Για κάθε μεταβλητή που επιστρέφεται από την ερώτηση, αντιστοιχεί ένα στοιχείο variable και η τιμή του χαρακτηριστικού name, ισούται με το όνομα της μεταβλητής. Για τις ASK ερωτήσεις το στοιχείο head δεν περιέχει στοιχεία variable. Επίσης το στοιχείο head εναλλακτικά μπορεί να περιέχει ένα στοιχείο link το οποίο έχει ένα χαρακτηριστικό href που περιέχει ένα σχετικό URI που περιέχει κάποια επιπρόσθετα μετά-δεδομένα για τα αποτελέσματα της ερώτησης. Παράδειγμα του στοιχείου head (SELECT ερώτησης) φαίνεται στην Εικόνα 16.1

16.2.1.2 Το results Στοιχείο

Το στοιχείο results περιέχει το στοιχείο result. Για κάθε λύση από την ακολουθία λύσεων που επιστέφεται από την ερώτηση, προστίθεται και ένα στοιχείο result. Κάθε στοιχείο result περιέχει ένα στοιχείο binding για κάθε μεταβλητή που της έχει αντιστοιχηθεί τιμή στη συγκεκριμένη λύση. Το στοιχείο binding έχει ένα χαρακτηριστικό name που έχει τιμή, το όνομα της μεταβλητής που αντιστοιχεί το συγκεκριμένο binding και στο περιεχόμενο του έχει την τιμή της μεταβλητής στην συγκεκριμένη λύση. Η τιμή της μεταβλητής που περιλαμβάνεται μέσα στο στοιχείο binding αναπαρίσταται ανάλογα με τον τύπο της τιμής.

- **RDF URI Reference *U***
  <binding><uri>*U*</uri></binding>
- **RDF Literal *S***
  <binding><literal>*S*</literal></binding>
- **RDF Literal *S* with language *L***
  <binding><literal xml:lang="*L*">*S*</literal></binding>
- **RDF Typed Literal *S* with datatype URI *D***

<binding><literal datatype="*D*">*S*</literal></binding>

- **Blank Node label *I***

  <binding><bnode>*I*</bnode></binding>

Στην προσέγγιση μας έχουμε μόνο τους δυο πρώτους τύπους αποτελεσμάτων. Παράδειγμα των στοιχείων results, result και binding φαίνονται στην Εικόνα 16.1

16.2.1.3 Το boolean Στοιχείο

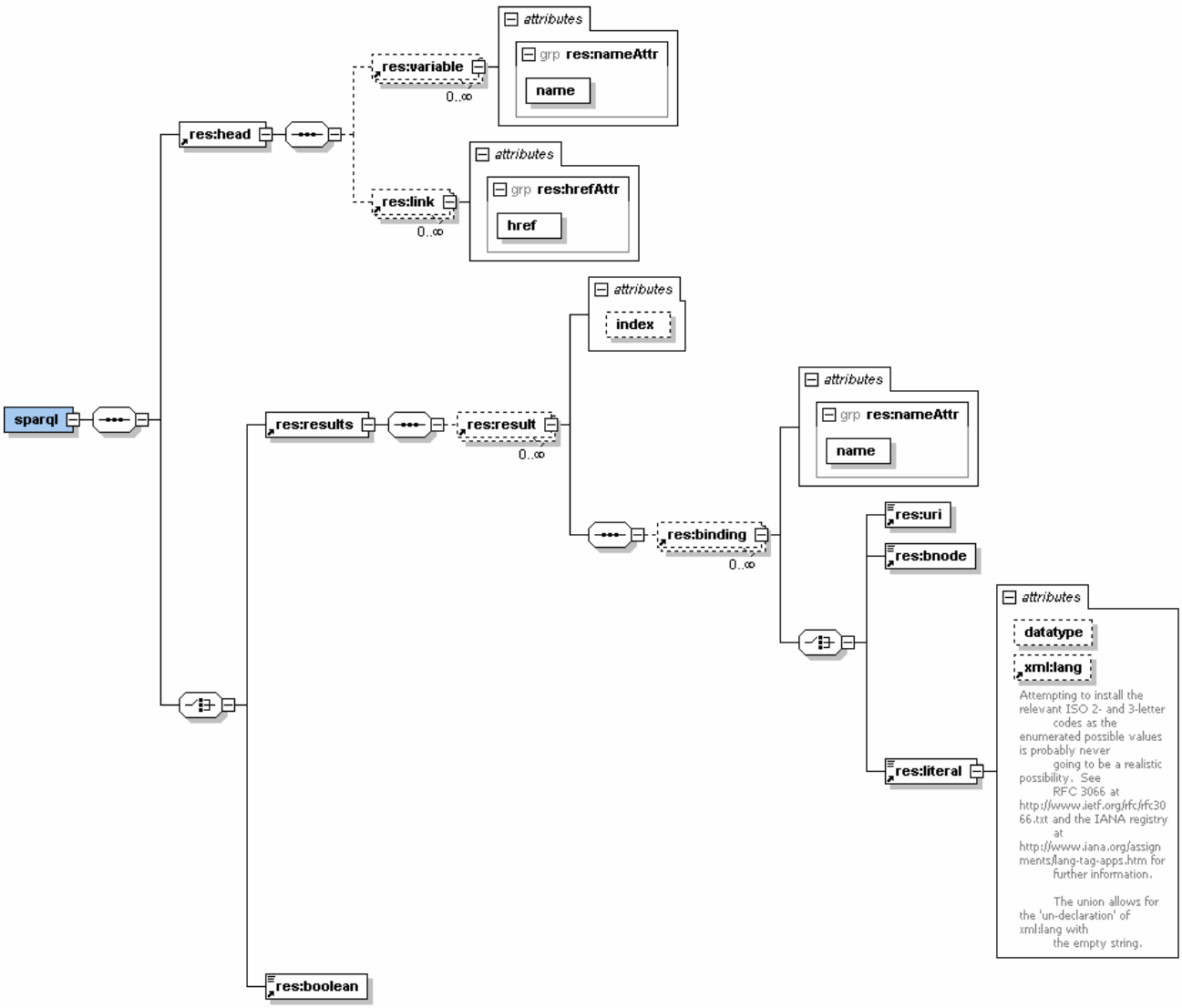
Το boolean στοιχείο αντικαθιστά το στοιχείο results, όταν πρόκειται για ASK ερωτήσεις. Είναι στοιχείο παιδί (child element) του στοιχείου sparql και εμφανίζεται αμέσως μετά το στοιχείο head. Το περιεχόμενο του είναι true ή false σε αντιστοιχία με την τιμή yes ή no που επέστρεψε η ASK ερώτηση.

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

  <head>
    <variable name="x"/>
    <variable name="hpage"/>
    <variable name="name"/>
    <variable name="age"/>
    <variable name="mbox"/>
    <variable name="friend"/>
  </head>

  <results>

    <result>
      <binding name="x">
   <bnode>r2</bnode>
      </binding>
      <binding name="hpage">
   <uri>http://work.example.org/bob/</uri>
      </binding>
      <binding name="name">
   <literal xml:lang="en">Bob</literal>
      </binding>
      <binding name="age">
   <literal
datatype="http://www.w3.org/2001/XMLSchema#integer">30</literal>
      </binding>
      <binding name="mbox">
   <uri>mailto:bob@work.example.org</uri>
      </binding>
    </result>

    ...
```

```
  </results>
</sparql>
```

Εικόνα 16.1 : Παράδειγμα XML Εγγράφου Αποτελεσμάτων

Όπως φαίνεται στο παράδειγμα της Εικόνας 16.1, το στοιχείο head περιέχει μια ακολουθία από στοιχεία variable τα οποία αναπαριστούν τις μεταβλητές και σαν τιμή του χαρακτηριστικού τους name έχουν το όνομα της εκάστοτε μεταβλητής(x, hpagem name, age, mboxm friend). Στην συνέχεια φαίνεται το στοιχείο results που αναπαριστά την ακολουθία λύσεων και περιέχει μια ακολουθία από στοιχεία result που αναπαριστούν τις λύσεις. Το πρώτο στοιχείο result περιέχει τις λύσεις: για την μεταβλητή x περιέχει την τιμή r2 και προσδιορίζει ότι πρόκειται για κενό κόμβο(blank node), για την μεταβλητή hpage περιέχει την τιμή http://work.example.org/bob και προσδιορίζει ότι πρόκειται για Uri τιμή και τέλος για την μεταβλητή name που περιέχει την τιμή Bob και προσδιορίζει ότι πρόκειται για σταθερά (literal)



Εικόνα 16.2 : Η Γραφική Απεικόνιση του XML Σχήματος που Προτείνεται από το W3C

16.2.2 Επέκταση του XML Schema της W3C XML μορφής

Όπως έχει περιγραφεί και στην Ενότητα 14.5, στην παρούσα προσέγγιση τα αποτελέσματα των DESCRIBE ερωτήσεων, αποτελούνται από τα απολεσματα που θα προέκυπταν από την ερώτηση αν ήταν SELECT μορφής, "εμπλουτισμένα" με έναν RDF γράφο που προκύπτει από την αποτίμηση της DESCRIBE SPARQL ερώτησης πάνω στην οντολογία.

Επομένως για την αναπαράσταση των αποτελεσμάτων που προέρχονται από DESCRIBE ερωτήσεις απαιτείται μια επέκταση της μορφής που έχει περιγραφεί έως αυτό το σημείο. Στο

στοιχείο result προστίθεται ένα προαιρετικό στοιχείο παιδί describe_graph, το οποίο περιέχει τον RDF γράφο που προέκυψε από την DESCRIBE SPARQL ερώτηση. Το στοιχείο describe_graph είναι τύπου xs:string, με minOccurs=0 και maxOccurs=1 και εμφανίζεται μετά από τα στοιχεία result. Ένα παράδειγμα εγγράφου αποτελεσμάτων DESCRIBE ερωτήσεων, φαίνεται στην Εικόνα 16.3 .

```
 <?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

  <head>
    <variable name="x"/>
    ...
  </head>

  <results>

    <result>
      ...
    </result>

    <result>
      ...
    </result>

    ...

    <describe_graph>
      ...
    </describe_graph>

  </results>


</sparql>
```

Εικόνα 16.3 : Παράδειγμα Επέκτασης XML Εγγράφου Αποτελεσμάτων

16.3 Μετασχηματισμός Αποτελεσμάτων

Ο μετασχηματισμός των αποτελεσμάτων πραγματοποιείται για SELECT, ASK και DESCRIBE ερωτήσεις, εφόσον οι ερωτήσεις CONSTRUCT επιστρέφουν έναν RDF γράφο και δεν απαιτούν κάποιον μετασχηματισμό.

16.3.1 Μετασχηματισμός Αποτελεσμάτων SELECT Ερωτήσεων

Η ανάλυση της μορφής των αποτελεσμάτων των XQuery ερωτήσεων, που προέρχονται από μετάφραση SELECT SPARQL ερωτήσεων, έχει πραγματοποιηθεί στην ενότητα 14.2. Τα αποτελέσματα αυτά αποτελούν ένα έγκυρο XML έγγραφο της παρακάτω μορφής :

<Results>
<Result> <x>x_1</x> <y>y_1</y> <z>z_1</z> ... </Result>
<Result> <x>x_2</x> <y>y_2</y> <z>z_2</z> ... </Result>
<Result> <x>x_3</x> <y>y_3</y> <z>z_3</z> ... </Result>
....
....
<Result> <x>x_n</x> <y>y_n</y> <z>z_n</z> ... </Result>
<Results>

Γίνεται αντιληπτό ότι το παραπάνω έγγραφο δεν διαφέρει κατά πολύ στον τρόπο δόμησης και αναπαράστασης των αποτελεσμάτων, σε σχέση με το έγγραφο που περιγράφηκε στην προηγούμενη ενότητα.

Οι αντιστοιχήσεις μεταξύ του XML εγγράφου που δημιουργείται από την XQuery ερώτηση και του εγγράφου των αποτελεσμάτων των SPARQL ερωτήσεων είναι οι εξής. Το στοιχείο **Results** αντιστοιχεί στο στοιχείο **results**, το στοιχείο **Result** αντιστοιχεί στο στοιχείο **result**. Και τα στοιχεία με όνομα το όνομα της μεταβλητής αντιστοιχούν στα στοιχεία **binding**, στα οποία η τιμή του χαρακτηριστικού **name** ταυτίζεται με το όνομα του αντιστοίχου στοιχείου.

Το περιεχόμενο του στοιχείου binding καθορίζεται από το περιεχόμενο του αντιστοίχου στοιχείου, αν το αντίστοιχο στοιχείο περιέχει το στοιχείο IRI, το περιεχόμενο του binding θα είναι της μορφής <binding><uri>*U*</uri></binding> . Ενώ αν το περιεχόμενο του αντιστοίχου στοιχείου δεν περιέχει το στοιχείο IRI, το περιεχόμενο του binding θα είναι της μορφής <binding><literal>*S*</literal></binding>. Καθώς όπως έχει αναλυθεί τα αποτελέσματα στην παρούσα προσέγγιση είναι είτε IRI είτε Literal, καθώς στα XML δεδομένα δεν περιέχονται κενοί κόμβοι, typed literal κτλ .

Το περιεχόμενο του στοιχείου head, δηλαδή τα ονόματα των μεταβλητών για τις οποίες επιστέφονται αποτελέσματα, μπορούν πολύ εύκολα να προσδιοριστούν από την ανάλυση(Parse) της SPARQL ερώτησης.

16.3.2 Μετασχηματισμός Αποτελεσμάτων ASK Ερωτήσεων

Η ανάλυση της μορφής των αποτελεσμάτων των XQuery ερωτήσεων, που προέρχονται από μετάφραση ASK SPARQL ερωτήσεων, έχει πραγματοποιηθεί στην ενότητα 14.3. Τα αποτελέσματα αυτά αποτελούνται από ένα yes ή no. Κατά την διαδικασία μετασχηματισμού των αποτελεσμάτων δημιουργείται ένα στοιχείο head που δεν περιέχει καμιά μεταβλητή και ένα στοιχείο boolean που περιέχει την τιμή true ή false ανάλογα με το αποτέλεσμα της ερώτησης. Συνεπώς η μετατροπή αποτελεσμάτων για τις ASK ερωτήσεις αποτελεί μια πολύ απλή διαδικασία.

16.3.3 Μετασχηματισμός Αποτελεσμάτων DESCRIBE Ερωτήσεων

Η ανάλυση της μορφής των αποτελεσμάτων των XQuery ερωτήσεων, που προέρχονται από μετάφραση ASK SPARQL ερωτήσεων, έχει πραγματοποιηθεί στην ενότητα 14.4 Για τα στοιχεία **head**, **result** και **binding** ακολουθείται η ίδια διαδικασία που ακολουθείται για της SELECT ερωτήσεις (βλέπε Ενότητα 16.3.1). Περιεχόμενο του στοιχείου **describe_graph** αποτελεί ο RDF γράφος που επιστρέφεται από την εκτέλεση της DESCRIBE SPARQL ερώτησης από ένα SPARQL engine.

16.4 Παραδείγματα

Παράδειγμα 16.1

SPARQL Ερώτηση :

```
PREFIX ns: <http://www.music.tuc.gr/ontology.owl#>
SELECT ?x ?n
WHERE {?x  ns:FirsName ?n .  }
```

Έστω τα αποτέλεσμα που επιστρέφει η XQuery ερώτηση :

```
<Results>
    <Result>
        <x><IRI>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Employ
            ee[1]/FirstName[1]) </IRI></x>
        <n>George</n>
    </Result>

    <Result>
        <x><IRI>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Person[
            1]/FirstName[1]) </IRI></x>
        <n>John</n>
    </Result>

</Results>
```

Αποτελέσματα μετά από μετασχηματισμό :

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
     <head>
       <variable name="x"/>
       <variable name="n"/>
     </head>

     <results>
          <result>
              <binding name="x">
                   <uri>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Employee[1]/
                        FirstName[1]) </uri>
              </binding>
              <binding name="n"><literal>George</literal></binding>
          </result>

          <result>
              <binding name="x">
                   <uri>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Person[1]/Firs
                        tName[1]) </uri>
              </binding>
              <binding name="n"><literal>John</literal></binding>
          </result>
     </results>

</sparql>
```

Παράδειγμα 16.2

SPARQL Ερώτηση :

```
PREFIX ns: <http://www.music.tuc.gr/ontology.owl#>
ASK
WHERE {?x  ns:FirsName "John".  }
```

Έστω τα αποτέλεσμα που επιστέφει η XQuery ερώτηση :

yes

Αποτελέσματα μετά από μετασχηματισμό :

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">

      <head>
      </head>

      <boolean> true </boolean>

</sparql>
```

Παράδειγμα 16.3

SPARQL Ερώτηση :

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ns: <http://www.music.tuc.gr/ontology.owl#>
DESCRIBE  ?x ?y
WHERE { ?x  ns:FirsName "John".
          ?y  rdfs:domain ns:Person_Type .  }
```

Έστω τα αποτέλεσμα που επιστέφει η XQuery ερώτηση :

```
<Results>
    <Result>
        <x><IRI>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Employ
            ee[1]/FirstName[1]) </IRI></x>
        <n>George</n>
    </Result>

    <Result>
        <x><IRI>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Person[
            1]/FirstName[1]) </IRI></x>
        <n>John</n>
    </Result>
</Results>
```

Έστω τα αποτέλεσμα που επιστέφει η SPARQL ερώτηση :

<ModelCom {http://www.music.tuc.gr/Age @rdf:type owl:DatatypeProperty;
http://www.music.tuc.gr/Age @rdfs:domain :PersonType; http://www.music.tuc.gr/FirstName
@rdf:type owl:DatatypeProperty; http://www.music.tuc.gr/ FirstName @rdfs:domain
:PersonType; http://www.music.tuc.gr/LastName @rdf:type owl:DatatypeProperty;
http://www.music.tuc.gr/LastName@rdfs:domain :PersonType;
http://www.music.tuc.gr/Telephone @rdf:type owl:DatatypeProperty;
http://www.music.tuc.gr/Telephone @rdfs:domain :PersonType;
http://www.music.tuc.gr/NickName @rdf:type owl:DatatypeProperty;
http://www.music.tuc.gr/NickName @rdfs:domain :PersonType;
http://www.music.tuc.gr/Address @rdf:type owl:ObjectProperty;
http://www.music.tuc.gr/Address @rdfs:domain :PersonType;} | >

Αποτελέσματα μετά από μετασχηματισμό :

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
     <head>
       <variable name="x"/>
       <variable name="n"/>
     </head>

     <results>
          <result>
              <binding name="x">
                   <uri>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Employee[1]/
                        FirstName[1]) </uri>
              </binding>
              <binding name="n"><literal>George</literal></binding>
          </result>

          <result>
              <binding name="x">
                   <uri>www.music.tuc.gr/xmlDoc_1.xml#xpointer(Persons/Person[1]/Firs
                        tName[1]) </uri>
              </binding>
              <binding name="n"><literal>John</literal></binding>
          </result>
              <describe_graph>
                   ns:Age rdf:type owl:DatatypeProperty.
                   ns:Age rdfs:domain ns:PersonType.
                   ns:FirstName rdf:type owl:DatatypeProperty.
                   ns:FirstName rdfs:domain ns:PersonType.
                   ns:LastName rdf:type owl:DatatypeProperty.
                   ns:LastName rdfs:domain ns:PersonType.
                   ns:Telephone rdf:type owl:DatatypeProperty.
                   ns:Telephone rdfs:domain ns:PersonType.
                   ns:NickName rdf:type owl:DatatypeProperty.
                   ns:NickName rdfs:domain ns:PersonType.
                   ns:Address rdf:type owl:ObjectProperty.
```

```
                    ns:Address  rdfs:domain ns:PersonType.
        </describe_graph>

    </results>
```

16.5 Περίληψη

Στο κεφάλαιο αυτό, έγινε η περιγραφή της XML μορφής που προτείνεται από το W3C για την αποθήκευση και διακίνηση των αποτελεσμάτων που προέρχονται από SPARQL ερωτήσεις SELECT και ASK μορφής, επίσης ορίστηκε μια επέκταση της συγκεκριμένης XML μορφής για την αποθήκευση και διακίνηση των αποτελεσμάτων που προέρχονται από DESCRIBE SPARQL ερωτήσεις.

Στην συνέχεια περιγράφηκε η διαδικασία μετασχηματισμού των XQuery αποτελεσμάτων ανάλογα με την μορφή της ερώτησης. Όπως γίνεται κατανοητό ότι η επιθυμητή XML μορφή SPARQL αποτελεσμάτων, δεν διαφέρει κατά πολύ από την μορφή των αποτελεσμάτων που δημιουργούνται από τις XQuery ερωτήσεις της παρούσας προσέγγισης. Επομένως ο μετασχηματισμός των αποτελεσμάτων αποτελεί μια απλοϊκή διαδικασία.

Στο επόμενο κεφάλαιο (Κεφάλαιο 17) παρουσιάζεται η ανακεφαλαίωση της παρούσας εργασίας, αναλύεται ο τρόπος αξιολόγησης του πλαισίου SPARQL2XQuery και τέλος προτείνονται κάποιες μελλοντικές επεκτάσεις.

17 Ανακεφαλαίωση, Αξιολόγηση & Μελλοντικές Επεκτάσεις

17.1 Ανακεφαλαίωση

Στα πλαίσια της παρούσας εργασίας αντικείμενο έρευνας αποτέλεσε η επίτευξη διαλειτουργικότητας μεταξύ του Σημασιολογικού και του XML Περιβάλλοντος. Αποτέλεσμα αυτής, είναι η ανάπτυξη του πλαισίου (framework) **SPARQL2XQuery**, το οποίο υποστηρίζει την **διαλειτουργικότητα** μεταξύ του **Σημασιολικού** και **XML περιβάλλοντος**, επιτρέποντας σημασιολογικές **SPARQL** ερωτήσεις να αποτιμώνται μέσω **XQuery** διεπαφών (interfaces) σε **XML βάσεις δεδομένων**. Της ανάπτυξη του πλαισίου προηγήθηκε θεωρητική τεκμηρίωση μεθόδων και αλγορίθμων , αυστηρή αναπαράσταση της σημασιολογίας και απόδειξης των σημασιολογικών ισοδυναμιών.

Το πλαίσιο **SPARQL2XQuery** πραγματοποιεί την **μετάφραση SPARQL** ερωτήσεων, σε σημασιολογικά ισοδύναμες **XQuery**, επιτρέπει την συνεργασία με το πλαίσιο **XS2OWL**[17] για την **ανακάλυψη** (discover) και την **αυτόματη παραγωγή** και **αποθήκευση** των

αντιστοιχήσεων. Και τέλος υποστηρίζει τον **μετασχηματισμό** των **αποτελεσμάτων** που προκύπτουν από τις XQuery ερωτήσεις. σύμφωνα με την XML μορφή των SPARQL αποτελεσμάτων, που προτείνεται από το W3C [56].

Η συνεισφορά της παρούσα εργασίας είναι :

- Ανάπτυξη μια **γενικής μεθοδολογία** και **αλγορίθμων**, για την **μετάφραση SPARQL** ερωτήσεων, σε **σημασιολογικά ισοδύναμες XQuery**, κατά την οποία επιτεύχθηκαν τα εξής : α) Η διαδικασία μετάφρασης είναι ανεξάρτητη από τον **τρόπο ορισμού** και **αποθήκευσης** τον **αντιστοιχήσεων** (**mappings**) μεταξύ της οντολογίας και του XML σχήματος β) Μετάφραση **όλων των πιθανών ερωτήσεων**, καλύπτοντας όλους τους δυνατούς συνδυασμούς της γραμματικής της γλώσσας SPARQL. γ) **Αυστηρή** τήρηση της σημασιολογίας της γλώσσας κατά την διαδικασία της μετάφρασης. δ) Οι **ακολουθίες λύσεων** που προκύπτουν από το μεταφρασμένο XQuery ερώτημα **είναι οι επιθυμητές** και δεν απαιτούν κάποια περεταίρω επεξεργασία (από API ή Software), επομένως **ανεξαρτησία** από το query engine και το περιβάλλον εκτέλεσης της XQuery ερώτησης. ε) Ανάπτυξη **εύκολα κατανοητής** διαδικασίας και αλγορίθμων μετάφρασης. ζ) Παραγωγή όσο το δυνατόν **μικρότερων** και **λιγότερο πολύπλοκων** εκφράσεων(expressions) XQuery.η) Ανάπτυξη και σύνταξη των μεταφρασμένων ερωτήσεων με τρόπο ώστε οι "**αντιστοιχίες**" μεταξύ των δυο ερωτήσεων(SPARQL-XQuery) και ο **τρόπος μετάφρασης** να γίνονται **εύκολα αντιληπτά**. θ) Σε συνδυασμό με όλα τα παραπάνω όσο το δυνατόν **βελτιστοποίηση** (Optimization) των XQuery ερωτήσεων. ι) Η παραγωγή **μιας και μόνο XQuery** ερώτησης για κάθε μία SPARQL ερώτηση που δίνεται προς μετάφραση χωρίς την εκτέλεση ενδιάμεσων ερωτήσεων.
- **Συνεργασία** με το πλαίσιο **XS2OWL**[17], το οποίο παράγει OWL οντολογίες από XML σχήματα. Στην περίπτωση αυτή, το πλαίσιο **SPARQL2XQuery** πραγματοποιεί την **ανακάλυψη** (discover) και την **αυτόματη παραγωγή** και **αποθήκευση** των **αντιστοιχήσεων**. Με την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων επιτυγχάνεται μια **πλήρως αυτοματοποιημένη διαδικασία**, χωρίς να είναι απαραίτητη η παρέμβαση ανθρώπινου παράγοντα. Επίσης επιτυγχάνεται η **πλήρης αντιστοίχηση** όλων των στοιχείων τόσο της οντολογίας όσο και του XML σχήματος, **με αντιστοιχήσεις απολύτως σημασιολογία ορθές**, **χωρίς ύπαρξη αβεβαιότητα (Uncertainty)** και **πιθανότητας σφάλματος** κατά την δημιουργία τους, προβλήματα που εμφανίζονται στην περίπτωση "χειροκίνητου"(manual) ορισμού των αντιστοιχήσεων από εξειδικευμένο χρήστη.
- **Μετασχηματισμός** των **αποτελεσμάτων** που προκύπτουν από τις XQuery ερωτήσεις, σύμφωνα με την XML μορφή των SPARQL αποτελεσμάτων(SPARQL Query Results XML Format), που προτείνεται από το W3C [56].

- Υπηρεσία Διαδικτύου (Web Service) η οποία παρέχει την δυνατότητα μετάφρασης SPARQL ερωτήσεων σε XQuery.

Η έμμεση συνεισφορά της παρούσας εργασίας με την υποστήριξη σημασιολογικών SPARQL ερωτήσεων σε XML βάσεις δεδομένων είναι:

- Έ**ρευνα** των **σημασιολογικών ισοδυναμιών** και της **διαδικασίας μετάφρασης** των δυο γλωσσών ερωτήσεων SPARQL και XQuery. Καθώς από όσο γνωρίζουμε δεν υπάρχει κάποια σχετική εργασία η οποία να έχει ασχοληθεί είτε με την αντιστοιχία , είτε με την μετάφραση, είτε με την σημασιολογική ισοδυναμία, μεταξύ των δυο γλωσσών ερωτήσεων, γεγονός το οποίο κάνει την παρούσα εργασία **καινοτόμα**.
- Επίτευξη **διαλειτουργικότητας των εφαρμογών** του Σημασιολογικού και XML περιβάλλοντος
- **Δημοσίευση** και **διαχείριση** αποθηκεμένων XML δεδομένων από τον Σημασιολογικό Ιστό μέσω της γλώσσας σημασιολογικών ερωτήσεων SPARQL.
- Η τελικοί χρήστες εκφράζουν σημασιολογικές ερωτήσεις (semantic-based queries), οι οποίες είναι **ποιο κατανοητές** και ποιο **κοντά στην διαίσθηση** τους, έναντι των βασισμένων σε δομή ερωτήσεων (structure-based queries), όπως η γλώσσα ερωτήσεων XQuery.
- Δυνατότητα ερωτήσεων **βασισμένων** σε **τύπους δεδομένων (data types)**, το οποίο δεν είναι εφικτό με την γλώσσα ερωτήσεων XQuery. Όπως για παράδειγμα, επέστεψε τα δεδομένα τα οποία είναι τύπου Person.
- Δυνατότητα ερωτήσεων βασισμένων σε **ιεραρχίες τύπων δεδομένων (data types hierarchies)**, το οποίο επίσης δεν εφικτό με την γλώσσα ερωτήσεων XQuery. Όπως για παράδειγμα, επέστεψε τα δεδομένα τα οποία ο τύπος τους είναι "πιο εξειδικευμένος" από τον τύπου Person. (Εκμεταλλευόμενοι τις IS-A σχέσεις οι οποίες προσφέρονται από τις οντολογίες, παρόμοια σχέση υποστηρίζεται και από το xml schema μέσω της δήλωσης extension, όμως δεν μπορεί να γίνει "εκμετάλλευση" του από την γλώσσα XQuery)

17.2 Αξιολόγηση

Για την αξιολόγηση του πλαισίου SPARQL2XQuery δεν εφαρμόστηκε κάποια ποσοτική μέθοδος, καθώς δεν γνωρίζουμε κάποια με την οποία θα μπορούσαμε να επαληθεύσουμε και παράλληλα αποδείξουμε την ορθότητα του πλαισίου. Η αξιολόγηση του πραγματοποιήθηκε με την διεξαγωγή ενός πολύ μεγάλου αριθμού εξονυχιστικών ελέγχων και δοκιμών. Πιο συγκεκριμένα :

Για το στοιχείο λογισμικού **Mapping Generator** :

Για την αξιολόγηση και τον έλεγχο ορθότητας της λειτουργίας του στοιχείου Mapping Generator, έγινε χρήση του πλαισίου XS2OWL σε έναν μεγάλο αριθμό XML σχημάτων, τόσο σε ευρέως αποδεκτά πρότυπα όπως MPEG-7 MDS και την MPEG-21 DIA Architecture στο πεδίο των πολυμέσων, τα IEEE LOM και SCORM στο πεδίο της ηλεκτρονικής εκπαίδευσης και το METS στο πεδίο των ψηφιακών βιβλιοθηκών, όσο και σε άλλα XML σχήματα τα οποία προσομοίωναν ειδικές περιπτώσεις. Στην συνέχεια έγινε χρήση του στοιχείου Mapping Generator για την ανακάλυψη και αυτόματη παραγωγή των αντιστοιχήσεων μεταξύ της οντολογίας που παράχθηκε από το σύστημα XS2OWL και του αντίστοιχου XML σχήματος. Τέλος πραγματοποιήθηκε εκτεταμένος έλεγχος και επαληθεύτηκε η ορθότητα και η συνέπεια των αντιστοιχήσεων που είχαν παραχθεί από το στοιχείο Mapping Generator.

Για το στοιχείο λογισμικού **Query Translator** :

Για την αξιολόγηση και τον έλεγχο ορθότητας της λειτουργίας του στοιχείου Query Translator, έγινε χρήση ενός πολύ μεγάλου αριθμού SPARQL ερωτήσεων. Η επιλογή των ερωτήσεων έγινε με στόχο να καλυφθούν όλες οι πιθανές περιπτώσεις τόσο ως προς την SPARQL σύνταξη, καλύπτοντας όλους τους πιθανούς συνδυασμούς της γραμματικής της γλώσσας, όπως επίσης καλύπτοντας όλες τις διαφορετικές περιπτώσεις κατά την διαδικασία της μετάφρασης. Ένα αντιπροσωπευτικό μέρος ερωτήσεων περιέχονται στο Παράρτημα Α. Κατά την αξιολόγηση του στοιχείου Query Translator, πραγματοποιήθηκε εκτεταμένος έλεγχος τόσο στην XQuery ερώτηση η οποία παρήγαγε αν ήταν η επιθυμητή και σημασιολογικά ισοδύναμη της SPARQL, όσο και στα αποτελέσματα τα οποία επέστρεφε από την αποτίμηση της στα XML δεδομένα.

Τέλος αξίζει να σημειωθεί ότι κατά την διάρκεια των ελέγχων που πραγματοποιηθήκαν, έγινε μέτρηση του χρόνου που απαιτείται για την μετάφραση των SPARQL ερωτήσεων σε XQuery, ο χρόνος αυτός κυμαίνεται από 500 ms έως 1000ms και εξαρτάται από την μορφή και την πολυπλοκότητα της SPARQL ερωτήσεις, όπως επίσης και την πολυπλοκότητα και αριθμό των αντιστοιχήσεων.

Για το στοιχείο λογισμικού **Result Transformer** :

Το στοιχείο λογισμικού Result Transformer ελεήθηκε και αξιολογήθηκε κατά την διάρκεια των ελέγχων του στοιχείου Query Translator. Δεν ήταν αναγκαίος ο παράταιρο έλεγχος, λόγο της μη πολύπλοκης λειτουργικότητας του.

17.3 Μελλοντικές Επεκτάσεις

Η παρούσα εργασία επικεντρώθηκε στην μελέτη της σημασιολογικής διαλειτουργικότητας μεταξύ της γλωσσών ερωτήσεων SPARQL και XQuery. Συγκεκριμένα, ακολουθώντας αυστηρά την σημασιολογία της γλώσσας SPARQL, την ανάπτυξη γενικής μεθοδολογίας και αλγορίθμων για την μετάφραση των SPARQL ερωτήσεων σε σημασιολογικά ισοδύναμες XQuery.

Οι μελλοντικές επεκτάσεις μέρος των οποίων πραγματοποιούνται είδη από το εργαστήριο Διανεμημένων πληροφοριακών Συστημάτων και Εφαρμογών Πολυμέσων – MUSIC είναι :

- Υποστήριξη των Identity Constraints που προσφέρει το XML σχήμα.
- Υποστήριξη πιο σύνθετων αντιστοιχήσεων (complex mappings) και αντιστοιχήσεων υπό συνθήκη (conditional mappings), μεταξύ του XML σχήματος και της οντολογίας .
- Δυνατότητα αντιστοιχήσεων στιγμιότυπων (Individual) της οινολογίας, με έννοιες του XML σχήματος και κατ' επέκταση υποστήριξη ερωτήσεων που περιέχουν αναφορές σε στιγμιότυπα .
- Μελέτη των πληροφοριών που αποθηκεύονται κατά την αντιστοίχηση οντολογίας και XML σχήματος , ώστε να επιτυγχάνεται βελτιστοποίηση (Optimization) της διαδικασίας μετάφρασης .
- Εφαρμογή τεχνικών βελτιστοποίησης (Optimization) στις XQuery ερωτήσεις που προκύπτουν από την μετάφραση
- Υποστήριξη των SPARQL DESCRIBE ερωτήσεων, σύμφωνα με πιθανό μελλοντικό προσδιορισμό της σημασιολογίας αυτών των ερωτήσεων από την προδιαγραφή της γλώσσας SPARQL
- Συνεργασία του πλαισίου SPARQL2XQuery, με πλαίσιο το οποίο επιτρέπει την αντιστοίχηση οντολογιών και τον μετασχηματισμό (Reformulation) SPARQL ερωτήσεων.

Αναφορές

[1] OWL Web Ontology Language Reference. W3C Recommendation, 10 February 2004. M. Dean and G.Schreiber (Eds.). http://www.w3.org/TR/2004/REC-owl-ref-20040210/.

[2] RDF Primer. W3C Recommendation, 10 February 2004. F. Manola and E. Miller (Eds.). http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ .

[3] RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 10 February 2004.D. Brickley and R.V. Guha (Eds.). http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[4] Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation,10 February 2004. G. Klyne, J. J. Carroll, and B. McBride (Eds.). http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

[5] SPARQL Query Language for RDF. W3C Recommendation 15 January 2008 . http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

[6] XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2007 . http://www.w3.org/TR/2007/REC-xquery-20070123/

[7] XQuery 1.0 and XPath 2.0 Functions and Operators . W3C Recommendation 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-functions-20070123/

[8] XML Path Language (XPath) 2.0 W3C Recommendation 23 January 2007 http://www.w3.org/TR/2007/REC-xpath20-20070123/

[9] XQuery 1.0 and XPath 2.0 Formal Semantics W3C Recommendation 23 January 2007 . http://www.w3.org/TR/2007/REC-xquery-semantics-20070123/

[10] XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999 http://www.w3.org/TR/1999/REC-xpath-19991116

[11] J. P´erez, M. Arenas, C. Gutierrez. Semantics and Complexity of SPARQL. 5th International Semantic Web Conference (ISWC-06), November 2006.

[12] J. Perez, M. Arenas, and C. Gutierrez. Semantics of SPARQL. http://ing.utalca.cl/»jperez/papers/sparql semantics.pdf.

[13] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170. 2005. http://www.hpl.hp.com/techreports/2005/HPL-2005-170.html.

[14] Axel Polleres: From SPARQL to rules (and back). WWW 2007: 787-796

[15] Tsinaraki C., Christodoulakis S. Support for Interoperability between OWL based and XML Schema based Applications 2nd DELOS Conference On Digital Libraries, Tirrenia, Italy, December 2007

[16] Tsinaraki C., Christodoulakis S. XS2OWL: A Formal Model and a System for enabling XML Schema Applications to interoperate with OWL-DL Domain Knowledge and Semantic Web Tools , In the proceedings of the 1st DELOS Conference, pp. 124-136, Tirrenia, Italy, February 2007

[17] Tsinaraki C., Christodoulakis S. Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools . In the proceedings of the 6th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2007), OTM Conferences (1) 2007: 850-869, Vilamoura, Algarve, Portugal, November 27-29 2007

[18] Tsinaraki C . A Semantic Based Framework for Multimedia Management and Interoperability . PHD Thesis Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2008

[19] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures - W3C Working Draft 20 June 2008 . http://www.w3.org/TR/2008/WD-xmlschema11-1-20080620/

[20] W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes - W3C Working Draft 20 June 2008. http://www.w3.org/TR/2008/WD-xmlschema11-2-20080620/

[21] OWL Web Ontology Language Semantics and Abstract Syntax - W3C Recommendation 10 February 2004 http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[22] OWL Web Ontology Language Guide - W3C Recommendation 10 February 2004. http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[23] FunctX XQuery Functions - http://www.xqueryfunctions.com/xq/

[24] B. Amann, C. Beeri, I. Fundulaki, and M. Scholl. Ontology-Based Integration of XML Web Resources. In Proceedings of the 1st International Semantic Web Conference (ISWC 2002),pages 117–131, 2002.

[25] B. Amann, I. Fundulaki, M. Scholl, C. Beeri, and A. Vercoustre. Mapping XML Fragments to Community Web Ontologies. In Proceedings of the 4th International Workshop on the Web and Databases (WebDB 2001), pages 97–102, 2001

[26] Bernd Amann , Catriel Beeri, Irini Fundulaki , Michel Scholl. Querying XML sources using an Ontology-based Mediator

[27] Huiyong Xiao , Isabel F. Cruz. Integrating and Exchanging XML Data using Ontologies

[28] Huiyong Xiao Isabel F. Cruz Feihong Hsu. Semantic Mappings for the Integration of XML and RDF Sources

[29] Huiyong Xiao Isabel F. Cruz Feihong Hsu. An Ontology-based Framework for XML Semantic Integration

[30] HUIYONG XIAO. QUERY PROCESSING FOR HETEROGENEOUS DATA INTEGRATION USING ONTOLOGIES- PHD Thesis

[31] FEIHONG HSU .SEMANTIC INTEGRATION OF XML USING A GLOBAL RDF MEDIATOR – Master's Thesis

[32] Ioanna Kofinana, Giorgos Serfiotis, Vassilis Christophides, Val Tannen,Alin Deutsch.Integrating XML Data Sources using RDF/S Schemas: The ICS-FORTH Semantic Web Integration Middleware (SWIM) Extended Abstract

[33] Ioanna Koffina . Integrating XML data sources using RDF/S Schemas:The ICS-FORTH Semantic Web Integration Middleware (SWIM) - Master's Thesis

[34] V.Christophides ,G.Karvounarakis,I. Koffina, G. Kokkinidis , A.Magkanaraki ,D. Plexousakis, G. Serfiotis, V. Tannen .The ICS-FORTH SWIM : A Powerful Semantic Web Integration Middleware

[35] An Y., Borgida A., Mylopoulos J.: "Constructing Complex Semantic Mappings Between XML Data and Ontologies". In the proceedings of the International Semantic Web Conference 2005: 6-20.

[36] Toni Rodrigues, Pedro Rosa, Jorge Cardoso MAPPING XML TO EXISTING OWL ONTOLOGIES

[37] Hannes Bohring , Soren Auer. Mapping XML to OWL Ontologies

[38] García R., Celma O.: "Semantic Integration and Retrieval of Multimedia Metadata". In the prceedings of the Knowledge Markup and Semantic Annotation Workshop, Semannot'05. CEUR,2005.

[39] Extensible Markup Language (XML) 1.0 - W3C Recommendation 16 August 2006. http://www.w3.org/TR/2006/REC-xml-20060816/

[40] Σαράφης Δημήτριος, "Implementation of an efficient XML Filtering Mechanism with XPath Expressions Based on Xtrie ", Διπλωματική εργασία Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2004

[41] Oracle BerkeleyDB XML . http://www.oracle.com/database/berkeley-db/xml/index.html

[42] Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net

[43] http://en.wikipedia.org/wiki/Jena_%28framework%29

[44] B. McBride Jena IEEE Internet Computing, July/August, 2002.

[45] J.J. Carroll Pulling XML Events to Parse RDF, submitted ISWC 03.

[46] L. Miller, A. Seaborne, and A. Reggiori Three Implementations of SquishQL, a Simple RDF Query Language, 2002

[47] T. Berners-Lee et al. Primer: Getting into RDF & Semantic Web using N3, http://www.w3.org/2000/10/swap/Primer.html

[48] J. Grant, D. Beckett, RDF Test Cases, 2003

[49] J.J. Carroll, Unparsing RDF/XML, WWW2002

[50] F. van Harmelen, P. F. Patel-Schneider I. Horrocks, Reference description of the DAML+OIL (March 2001)ontology markup language, http://www.daml.org/2001/03/reference

[51] XPath over XML Schema . http://xpath-on-schema.sourceforge.net/

[52] Apache Foundation, "XMLBeans XML Binding Framework", http://xmlbeans.apache.org/, 2003

[53] ARQ - A SPARQL Processor for Jena , http://jena.sourceforge.net/ARQ/

[54] Representing vCard Objects in RDF/XML W3C Note 22 February 2001 http://www.w3.org/TR/vcard-rdf

[55] FOAF Vocabulary Specification 0.91 Namespace Document 2 November 2007 - http://xmlns.com/foaf/spec/

[56] SPARQL Query Results XML Format , W3C Recommendation 15 January 2008 http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115/

[57] XML Pointer Language (XPointer) W3C Working Draft 16 August 2002 http://www.w3.org/TR/2002/WD-xptr-20020816/

[58] W3C RDF Data Access Working Group (DAWG) http://www.w3.org/2001/sw/DataAccess/

[59] W3C Semantic Web Activity http://www.w3.org/2001/sw/

[60] RELAX NG Specification http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[61] Patrick Lehti, Peter Fankhauser . XML Data Integration with OWL: Experiences & Challenges, Proceedings of the 2004 International Symposium on Applications and the Internet (SAINT'04)

[62] H.Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann and S. Hubner, Ontology-Based Integration of Information — A Survey of Existing Approaches, IJCAI-01 Workshop, 2001

[63] Data Integration Projects World-Wide - http://www.ifi.uzh.ch/~pziegler/IntegrationProjects.html

[64] Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proceedings of the 11th International World Wide Web Conference (WWW), Honolulu, Hawaii (2002)

[65] Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the Semantic Web Through RVL Lenses. In: Proceedings of the Second International Semantic Web Conference (ISWC'03), Sanibel Island, Florida, USA, 20-23 October. (2003)

[66] RDQL - A Query Language for RDF, W3C Member Submission 9 January 2004 http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[67] Semantic Web Road map, http://www.w3.org/DesignIssues/Semantic.html

[68] THE SEMANTIC WEB: AN INTERVIEW WITH TIM BERNERS-LEE , http://consortiuminfo.org/bulletins/semanticweb.php

[69] The Semantic Web-A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, http://www-personal.si.umich.edu/~rfrost/courses/SI110/readings/In_Out_and_Beyond/Semantic_Web.pdf

[70] Guarino N. (1998): "Formal Ontology and Information Systems". In the proceedings of the first International Conference Formal Ontology in Information Systems (FOIS '98), pp. 3-15. June 6-8 1998, Trento, Italy, Ed. Guarino N., IOS Press

[71] W3C - The World Wide Web Consortium , www.w3c.org

[72] Natalya F. Noy and Deborah L. McGuinness , Ontology Development 101: A Guide to Creating Your First Ontology

[73] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini1, and Heiner Stuckenschmidt, C-OWL: Contextualizing Ontologies

[74] Zhengxiang Pan , Jeff Heflin , DLDB: Extending Relational Databases to Support Semantic Web Queries

[75] Jırı Dokulil , Evaluation of SPARQL queries using relational databases

[76] Jing Mei, Li Ma, Yue Pan , Ontology Query Answering on Databases

[77] Cristian P´erez de Laborda and Stefan Conrad, Querying Relational Databases with RDQL

[78] Cristian P´erez de Laborda Matth¨aus Zloch Stefan Conrad, RDQuery_ Querying Relational Databases on-the-fly with RDFQL

[79] Cristian P´erez de Laborda , Stefan Conrad Bringing Relational Data into the SemanticWeb using SPARQL and Relational.OWL

[80] Andrew Newman , Querying the Semantic Web using a Relational Based SPARQL

[81] Artem Chebotko, Shiyong Lu, Hasan M. Jamil and Farshad Fotouhi, Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns

[82] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi, Semantics Preserving SPARQL-to-SQL Translation

[83] Steve Harris, SPARQL query processing with conventional relational database systems

[84] Zhaohui Wu, Huajun Chen, Heng Wang, Yimin Wang, Yuxin Mao, Jinmin Tang, and Cunyin Zhou ,Dartgrid: a Semantic Web Toolkit for Integrating Heterogeneous Relational Databases

[85] Maksym Korotkiy and Jan L. Top, From Relational Data to RDFS Models

[86] Jesús Barrasa, Óscar Corcho, Asunción Gómez-Pérez R2O, an Extensible and Semantically Based Database to ontology Mapping Language

[87] Andy Seaborne, Christian Bizer , D2RQ – Treating Non-RDF Databases as Virtual RDF Graphs

[88] Jesús Barrasa, Oscar Corcho, Asunción Gómez-Pérez Fund Finder: A case study of database-to-ontology mapping

[89] Marcelo Arenas, Claudio Gutierrez, Bijan Parsia, Jorge P´erez, Axel Polleres, and Andy Seaborne SPARQL – Where are we? Current state, theory and practice , European Semantic Web Conference 2007 Tutorial

[90] Sandro Daniel Camillo, Ronaldo dos Santos Mello,Carlos Alberto Heuser , Querying Heterogeneous XML Sources through a Conceptual Schema

[91] Heping Chen, Jinguang Gu, Xiaohui Li, and Hongping Fang, An XML Query Mechanism with Ontology Integration

[92] Wei Sun and Da-Xin Liu, Using Ontologies for Semantic Query Optimization of XML Database

[93] Heping Chen, Jinguang Gu, Xiaohui Li and Hongping Fang An XML query mechanism with ontology integration

[94] Sophie Cluet, Pierangelo Veltri, Dan Vodislav , Views in a large-scale XML repository

[95] Λίνα Μπουντούρη και Μανόλης Γεργατσούλης, Σημασιολογική Ολοκλήρωση με χρήση Οντολογιών, 15ο Πανελλήνιο Συνέδριο Ακαδημαϊκών Βιβλιοθηκών, 2006

[96] Μαρια Ειρηνη Πολυτου, Πρότυπο Σύστημα Ολοκλήρωσης Σχημάτων με Υποστήριξη Οντολογιών για Κατανεμημένα Περιβάλλοντα Αυτόνομων Κόμβων Διπλωματική εργασία Εθνικο Μετσοβιο Πολυτεχνειο, Τμήμα ΗΜΜΥ, 2007

[97] Στυλιανάκης Γεώργιος "Design and Development of a Repository for Educational, Learning and Evaluation objects based on the METS Digital Library Model" , Διπλωματική εργασία Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2008

[98] Ντούσιας Αλέξανδρος "A System for Semantic Interoperability Between Cultural and Multimedia Applications", Διπλωματική εργασία Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2008

[99] Θεοδωράκης Γεώργιος "Design and Development of LEARNING DESIGN EDITOR: A Tool for managing abstract learning scenarios" , Διπλωματική εργασία Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2007

[100] Συντζανάκη Αθηνά "Development of an MPEG-7 Metadata Repository and Retrieval support based on the MP7QL language" Διπλωματική εργασία Πολυτεχνείο Κρήτης, Τμήμα ΗΜΜΥ, Χανιά 2007

[101] Chang S.F., Sikora T., Puri A.: "Overview of the MPEG-7 standard". IEEE Transactions on Circuits and Systems for Video Technology 11, pp. 688–695.

[102] Pereira F.: "The MPEG-21 standard: Why an open multimedia framework?". In Proceedings of the 8th International Workshop on Interactive Distributed Multimedia Systems (IDMS 2001), LNCS 2158, Lancaster, September 2001, Springer-Verlag, Heidelberg, pp. 219–220.

[103] IEEE LTSC: "IEEE 1484.12.1-2002 – Learning Object Metadata Standard." http://ltsc.ieee.org/wg12/.

[104] ADL Technical Team. Sharable Content Object Reference Model (SCORM). 2004.

[105] Metadata Encoding and Transmission Standard (METS) Official Website, http://www.loc.gov/standards/mets/.

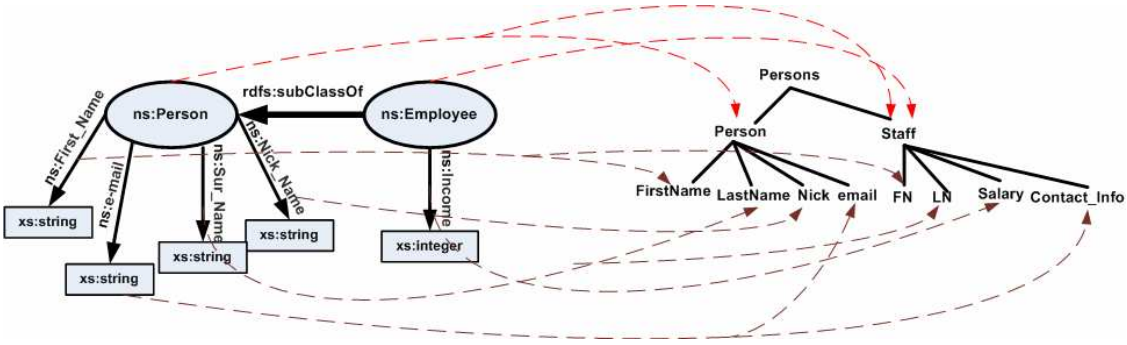
[106] Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P. F.: "The Description Logic Handbook". Theory, Implementation, and Applications. Cambridge University Press, Cambridge, 2002.

[107] Kay M. (ed.): "XSL Transformations (XSLT) Version 2.0". W3C Recommendation. http://www.w3.org/TR/xslt20/

ΠΑΡΑΡΤΗΜΑ Α

Στο παρών παράρτημα παρουσιάζεται ένας αντιπροσωπευτικός αριθμός παραδειγμάτων. Η οντολογία δεν έχει προκύψει με βάση την μεθοδολογία XS2OWL όπως τα παραδείγματα που έχουν παρουσιαστεί έως αυτό το σημείο. Τα παραδείγματα των παρακάτω ερωτήσεων έχουν προσεκτικά επιλεχτεί με σκοπό να καλύψουν όλες τις πιθανές περιπτώσεις και παραλλαγές στην διαδικασία της μετάφρασης.

Στην Εικόνα Α.1 φαίνονται οι αντιστοιχήσεις μεταξύ των εννοιών(κλάσεων) και συσχετίσεων(ιδιοτήτων) της οντολογίας και των στοιχείων του XML εγγράφου.



Εικόνα Α.1 : Αντιστοιχήσεις μεταξύ οντολογίας και XML

Οι αντιστοιχήσεις που προκύπτουν μεταξύ εννοιών και συσχετίσεων της οντολογίας και πιθανών μονοπατιών του XML εγγράφου είναι οι :

ns:Person={/Persons/Person, /Persons/Staff}
ns:Employee={/Persons/Staff}
ns:First_Name={/Persons/Person/FirstName , /Persons/Staff/FN}
ns:Sur_Name={/Persons/Person/LastName , /Persons/Staff/LN}
ns:Nick_Name={/Persons/Person/Nick}
ns:e-mail={ Persons/Person/email , /Persons/Staff/Contact_Info}
ns:Income={/Persons/Staff/Salary}

SPARQL Query 1 :

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?x rdf:type ns:Person . }
```

Translated XQuery Query 1 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 2 :

```
PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?y rdfs:subClassOf ns:Person .
               ?x rdf:type ?y . }
```

Translated XQuery Query 2 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 3 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?name
        WHERE{ ?x ns:First_Name ?name . }
```

Translated XQuery Query 3 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $name in $x/FirstName union $x/FN
        return(<Result><name>{ fn:string($name)}</name></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 4 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?x ns:First_Name ?name . }
```

Translated XQuery Query 4 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        let $name := $x/FirstName union $x/FN
        where( fn:exists($name) )
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 5 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?x ns:First_Name ?name .
               ?x ns:Income ?inc . }
```

Translated XQuery Query 5 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        let $name := $x/FN
        let $inc := $x/Salary
        where( fn:exists($name) and fn:exists($inc) )
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 6 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?name ?inc
        WHERE{ ?x ns:First_Name ?name .
               ?x ns:Income ?inc . }
```

Translated XQuery Query 6 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $name in $x/FN
        for $inc in $x/Salary
        return(<Result><name>{ fn:string($name)}</name>,<inc>{ fn:string($inc)}</inc></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 7 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?x ns:First_Name ?name .
               ?x ns:Income ?inc .
               ?x ns:Sur_Name ?sname .
               ?x ns:e-mail  ?email . }
```

Translated XQuery Query 7 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $name in $x/FN
        for $inc in $x/Salary
        for $sname in $x/LN
        for $email in $x/Contact_Info
        return(<Result><x>{func:CIVT($x)}</x>,<name>{ fn:string($name)}</name>,<inc>{
                fn:string($inc)}</inc>,<sname>{ fn:string($sname)}</sname>,<email>{
                fn:string($email)}</email></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 8 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?x ns:First_Name ?name .
               ?x ns:Income ?inc .
               ?x ns:Sur_Name ?sname .
               ?x ns:e-mail  ?email . }
```

Translated XQuery Query 8 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        let $name := $x/FN
        let $inc := $x/Salary
        let $sname := $x/LN
        let $email := $x/Contact_Info
        where( fn:exists($name) and fn:exists($inc) and fn:exists($sname) and fn:exists($email) )
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 9 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?sname
        WHERE{ ?x ns:First_Name "John" .
               ?x ns:Sur_Name ?sname .}
```

Translated XQuery Query 9 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person[./FirstName= "John"] union $doc/Persons/Staff[./FN= "John"]
        for $sname  in $x/LastName union $x/LN
        return(<Result><x>{func:CIVT($x)}</x>,<sname>{ fn:string($sname)}</sname>Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 10 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?inc
        WHERE{ ?x ns:Income ?inc .
                FILTER(?inc>1000) }
```

Translated XQuery Query 10 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $inc in $x/Salary
        where( $inc>1000 )
        return(<Result><x>{func:CIVT($x)}</x>,<inc>{ fn:string($inc)}</inc></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 11 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?inc
        WHERE{ ?x ns:Income ?inc .
                FILTER(?inc>1000 || ?inc<500) }
```

Translated XQuery Query 11 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $inc in $x/Salary
        where( $inc>1000 or $inc<500 )
        return(<Result><x>{func:CIVT($x)}</x>,<inc>{ fn:string($inc)}</inc></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 12 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?name
        WHERE{ ?x ns:First_Name ?name .
                FILTER ( regex(?name , "^ali") )}
```

Translated XQuery Query 12 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $name in $x/FirstName union $x/FN
        where( fn:matches($name, "^ali" ) )
        return(<Result><x>{func:CIVT($x)}</x>,<name>{ fn:string($name)}</name></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 13 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?s ?p ?o . }
```

Translated XQuery Query 13 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let  $properties_xpaths :=( "/Persons/Person/FirstName", "/Persons/Staff/FN",
                "/Persons/Person/LastName" , "/Persons/Staff/LN", "/Persons/Person/Nick"
                ,"Persons/Person/email" , "/Persons/Staff/Contact_Info", "/Persons/Staff/Salary" )
let $Results :=(
        for $s in $doc/Persons/Person union $doc/Persons/Staff
        for $p in $s/* union $s/@*
        let  $o := $p
        where ( func:xpath($p)= $properties_xpaths )
        return(<Result><s>{func:CIVT($s)}</s>,<p>{ func:PVT ($p)}</p>,<o>{ func:UVT
                ($o)}</o></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 14 :

```
PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ns:     <http://example.com/ns#>
SELECT  *
        WHERE{ ?s ?p ?o .
               ?p rdfs:domain ns:Employee . }
```

Translated XQuery Query 14 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $s in $doc/Persons/Staff
        for $p in $s/FN union $s/LN union $s/Contact_Info union $s/Salary
        let  $o := $p
        return(<Result><s>{func:CIVT($s)}</s>,<p>{ func:PVT ($p)}</p>,<o>{ func:UVT
               ($o)}</o></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 15 :

```
PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ns:     <http://example.com/ns#>
SELECT  ?x
        WHERE{ ?y rdfs:subClassOf ns:Person .
               ?p rdfs:domain ?y.
               ?x ?p ?o. }
```

Translated XQuery Query 15 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        let  $p := $x/FN union $x/LN union $x/Contact_Info union $x/Salary
        let  $o := $p
        where( fn:exists($o) )
        return(<Result><x>{func:CIVT($x)}</x></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 16 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?s ?p "John". }
```

Translated XQuery Query 16 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let  $properties_xpaths :=( "/Persons/Person/FirstName", "/Persons/Staff/FN",
                    "/Persons/Person/LastName" , "/Persons/Staff/LN", "/Persons/Person/Nick"
                    ,"Persons/Person/email" , "/Persons/Staff/Contact_Info", "/Persons/Staff/Salary" )

let $Results :=(
        for $s in $doc/Persons/Person union $doc/Persons/Staff
        for $p in $s/*[.="John" ] union $s/@*[.="John" ]
        where ( func:xpath($p)= $properties_xpaths )
        return(<Result><s>{func:CIVT($s)}</s>,<p>{ func:PVT ($p)}</p></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 17 :

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?y
        WHERE{ ?x rdf:type ns:Employee .
               ?y rdf:type ns:Employee .
               ?x ?p "john".
               ?y ?p "George". }
```

Translated XQuery Query 17 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let  $prop_1 :=( "/Persons/Person/FirstName", "/Persons/Staff/FN")
let  $prop_2 :=( "/Persons/Person/LastName" , "/Persons/Staff/LN")
let  $prop_3 :=( "/Persons/Person/Nick" ,"Persons/Person/email")
let  $prop_4 :=("/Persons/Staff/Contact_Info")
let  $prop_5 :=( "/Persons/Staff/Salary" )
let $Results :=(
        for $x in $doc/Persons/Staff
        for $y in $doc/Persons/Staff
        let $p_1 := $x/FN[.="John" ]  union $x/LN[.="John" ]  union $x/Contact_Info[.="John" ]   union
                    $x/Salary[.="John" ]
        let $p_2 := $x/FN[.="George" ]  union $x/LN[.="George" ]  union $x/Contact_Info[.="George" ]
                    union $x/Salary[.="George" ]
        where ( (func:xpath($p_1)= $prop_1 and func:xpath($p_2)= $prop_1 ) or
                 (func:xpath($p_1)= $prop_2 and func:xpath($p_2)= $prop_2) or
                (func:xpath($p_1)= $prop_3 and func:xpath($p_2)= $prop_3) or
                (func:xpath($p_1)= $prop_4 and func:xpath($p_2)= $prop_4) or
                (func:xpath($p_1)= $prop_5 and func:xpath($p_2)= $prop_5) )
        return(<Result><x>{func:CIVT($x)}</x>,<y>{ func:CIVT($y)}</y></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 18 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?n
        WHERE{ ?x ns:First_Name ?n.
               ?x ns:Sur_Name ?n .}
```

Translated XQuery Query 18 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $n in ($x/FirstName union $x/FN) [.= ($x/LastName union $x/LN)]
        return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 19 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?x ns:First_Name ?n.
               OPTIONAL{ ?x ns:e-mail ?email .} }
```

Translated XQuery Query 19 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
    let $BGP_1 :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $n in $x/FirstName union $x/FN
        return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
    )
    let $BGP_2 :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $email in $x/email union $x/Contact_Info
        return(<Result><x>{func:CIVT($x)}</x>,<email>{ fn:string($email)}</email></Result>)
    )
    return (func:OPTIONAL($BGP_1, $BGP_2))
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 20 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?x ns:First_Name ?n.
                  OPTIONAL{ ?x ns:e-mail ?email .}
                  OPTIONAL{ ?x ns:Nick_Name ?nn .} }
```

Translated XQuery Query 20 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
     let $P_1 :=(
          let $BGP_1 :=(
               for $x in $doc/Persons/Person union $doc/Persons/Staff
               for $n in $x/FirstName union $x/FN
               return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
          )
          let $BGP_2 :=(
               for $x in $doc/Persons/Person union $doc/Persons/Staff
               for $email in $x/email union $x/Contact_Info
               return(<Result><x>{func:CIVT($x)}</x>,<email>{ fn:string($email)}</email></Result>)
          )
          return (func:OPTIONAL($BGP_1, $BGP_2))
     )
     let $BGP_3 :=(
               for $x in $doc/Persons/Person
               for $nn in $x/Nick
               return(<Result><x>{func:CIVT($x)}</x>,<nn>{ fn:string($nn)}</nn></Result>)
     )
     return (func:OPTIONAL($P_1, $BGP_3))

)
return (<Results>{$Results }</Results>)
```

SPARQL Query 21 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?x ns:First_Name ?n.
                  OPTIONAL{ ?x ns:e-mail ?email .
                                OPTIONAL{ ?x ns:Nick_Name ?nn .} }}
```

Translated XQuery Query 21 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
     let $BGP_1 :=(
          for $x in $doc/Persons/Person union $doc/Persons/Staff
          for $n in $x/FirstName union $x/FN
          return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
     )
     let $P_1 :=(
          let $BGP_2 :=(
               for $x in $doc/Persons/Person union $doc/Persons/Staff
               for $email in $x/email union $x/Contact_Info
               return(<Result><x>{func:CIVT($x)}</x>,<email>{ fn:string($email)}</email></Result>)
          )
          let $BGP_3 :=(
                    for $x in $doc/Persons/Person
                    for $nn in $x/Nick
                    return(<Result><x>{func:CIVT($x)}</x>,<nn>{ fn:string($nn)}</nn></Result>)
          )
          return (func:OPTIONAL($BGP_2, $BGP_3))
     )
     return (func:OPTIONAL($BGP_1, $P_1))
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 22 :

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?n ?nn
        WHERE{{?x ns:First_Name ?n.
                 ?x rdf:type ns:Employee. }
                 UNION
               {?x ns:First_Name ?nn.
                 ?x rdf:type ns:Person. }}
```

Translated XQuery Query 22 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        let $Union_1:=(
          for $x in $doc/Persons/Staff
          for $n in $x/FN
          return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
        )
        let $Union_2 :=(
          for $x in $doc/Persons/Person union $doc/Persons/Staff
          for $nn in $x/FirstName union $x/FN
          return(<Result><x>{func:CIVT($x)}</x>,<nn>{ fn:string($nn)}</nn></Result>)
        )
        return ($Union_1, $Union_2)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 23 :

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?fn ?ln ?inc
        WHERE{ {{?x ns:First_Name ?fn. }
                  UNION
                  {?x ns:Last_Name ?ln. }}
                ?x ns:Income ?inc. }
```

Translated XQuery Query 23 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        let $Union_1:=(
          for $x in $doc/Persons/Staff
          for $fn in $x/FN
          for $inc in $x/Salary
          return(<Result><x>{func:CIVT($x)}</x>,<fn>{ fn:string($fn)}</fn>,<inc>{
                fn:string($inc)}</inc></Result>)
        )
        let $Union_2 :=(
          for $x in $doc/Persons/Staff
          for $ln in $x/LN
          for $inc in $x/Salary
          return(<Result><x>{func:CIVT($x)}</x>,<ln>{ fn:string($ln)}</ln>,<inc>{
                fn:string($inc)}</inc></Result>)
        )
        return ($Union_1, $Union_2)
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 24 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?x ?y ?n
        WHERE{{?x ns:First_Name "John".
                        OPTIONAL{ ?y ns:Nick_Name ?n} }
                ?y  ns:First_Name "George". }
```

Translated XQuery Query 24 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
    let $P_1:=(
        let $BGP_1:=(
            for $x in $doc/Persons/Person[./FirstName= "John"] union $doc/Persons/Staff[./FN= "John"]
            return(<Result><x>{func:CIVT($x)}</x></Result>)
        )
        let $BGP_2:=(
            for $y in $doc/Persons/Person
            for $n in $y/nick
            return(<Result><y>{func:CIVT($y)}</y>,<n>{ fn:string($n)}</n></Result>)
        )
        return (func:OPTIONAL($BGP_1, $BGP_2))
    )
    let $BGP_3:=(
        for $y in $doc/Persons/Person[./FirstName= "George"] union $doc/Persons/Staff[./FN= "George"]
        return(<Result><y>{func:CIVT($y)}</y></Result>)
    )
    return (func:AND($P_1, $BGP_3))
)
return (<Results>{$Results }</Results>)
```

SPARQL Query 25 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?inc
        WHERE{ ?x ns:Income ?inc}
        LIMIT 10
```

Translated XQuery Query 25 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $inc in $x/Salary
        return(<Result><inc>{fn:string($inc)}</inc></Result>)
)
return (<Results>{$Results[fn:position()<=10] }</Results>)
```

SPARQL Query 26 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?inc
        WHERE{ ?x ns:Income ?inc}
        OFFSET 5
```

Translated XQuery Query 26 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $inc in $x/Salary
        return(<Result><inc>{fn:string($inc)}</inc></Result>)
)
where( fn:count ($Results ) >= 5 )
return (<Results>{$Results}</Results>)
```

SPARQL Query 27 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  DISTINCT ?n
        WHERE{ ?x ns:First_Name ?n}
```

Translated XQuery Query 27 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $n in $x/FirstName union $x/FN
        return(<Result><n>{fn:string($n)}</n></Result>)
)
return (<Results>{ func:DISTINCT($Results)}</Results>)
```

SPARQL Query 28 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  REDUCE ?n
        WHERE{ ?x ns:First_Name ?n}
```

Translated XQuery Query 28 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $n in $x/FirstName union $x/FN
        return(<Result><n>{fn:string($n)}</n></Result>)
)
return (<Results>{ func:REDUCE($Results)}</Results>)
```

SPARQL Query 29 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  ?sn ?inc
        WHERE{ ?x ns:Sur_Name ?sn.
               ?x ns:Income ?inc.}
        ORDER BY ?sn  DESC(?inc)
```

Translated XQuery Query 29 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Staff
        for $sn in $x/LN
        for $inc in $x/Salary
        return(<Result><sn>{fn:string($sn)}</sn>,<inc>{fn:string($inc)}</inc></Result>)
)
let $Ordered_Results :=(
        for $iter in $Results
        order by$iter/sn empty least , $iter/inc descending empty least
        return ($iter)
)
return (<Results>{$Ordered_Results}</Results>)
```

SPARQL Query 30 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  DISTINCT ?inc
        WHERE{ ?x ns:Income ?inc.}
         ORDER BY DESC(?inc)  LIMIT 30 OFFSET 10
```

Translated XQuery Query 30 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Modified_Results :=(
    let $Results :=(
        for $x in $doc/Persons/Staff
        for $inc in $x/Salary
        return(<Result><inc>{fn:string($inc)}</inc></Result>)
    )
    let $Distinct_Results := func:DISTINCT( $Results )
    where (fn:count ($Distinct_Results) >= 10 )
    return( let $Ordered_Results :=(
                for $iter in $ Distinct_Results
                order by $iter/inc descending empty least
                return ($iter)
            )
            return($Ordered_Results[fn:position()<=30])
    )
)
return(<Results>{$Modified_Results}</Results>)
```

SPARQL Query 31 :

```
PREFIX ns:      <http://example.com/ns#>
ASK
        WHERE{ ?x ns:Income ?inc.
                FILTER(?inc>1000) }
```

Translated XQuery Query 31 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        let $x := $doc/Persons/Staff
        let $inc := $x/Salary
        where( $inc>1000 )
        return("yes")
)
return ( if(fn:empty($Results)) then "no" else "yes")
```

SPARQL Query 32 :

```
PREFIX ns:      <http://example.com/ns#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

CONSTRUCT {  _:v foaf:firstname   ?fn .
             _:v foaf:surname     ?sn }
            WHERE{ ?x Sur_Name ?sn .
                   ?x fn:First_Name ?fn . }
```

Translated XQuery Query 32 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
        for $x in $doc/Persons/Person union $doc/Persons/Staff
        for $sn in $x/LastName union $x/LN
        for $fn in $x/FirstName union $x/FN
        return(<Result><fn>{ fn:string($fn)}</fn>,<sn>{ fn:string($sn)}</sn></Result>)
)
for $res at $iter in $ Results
return ( if (fn:exists($res/fn) ) then
                (fn:concat ("_:bn" , $iter ) , "foaf:firstname" , fn:string($res/fn) , "." )
        else ( ) ,
        if (fn:exists($res/sn) ) then
                (fn:concat ("_:bn" , $iter ) , "foaf:surname" , fn:string($res/sn) , "." )
        else ( )
)
```

SPARQL Query 33 :

```
PREFIX ns:      <http://example.com/ns#>
SELECT  *
        WHERE{ ?x ns:First_Name ?n.
                  OPTIONAL{ ?x ns:e-mail ?email .}
                  FILTER( ! bound ( ?email) )}
```

Translated XQuery Query 33 :

```
declare namespace func = "http://www.music.tuc.gr/SPARQL2XQuery/funcs";
let $doc:= collection(http://www.music.tuc.gr/...)
let $Results :=(
     let $P_1 :=(
          let $BGP_1 :=(
               for $x in $doc/Persons/Person union $doc/Persons/Staff
               for $n in $x/FirstName union $x/FN
               return(<Result><x>{func:CIVT($x)}</x>,<n>{ fn:string($n)}</n></Result>)
          )
          let $BGP_2 :=(
               for $x in $doc/Persons/Person union $doc/Persons/Staff
               for $email in $x/email union $x/Contact_Info
               return(<Result><x>{func:CIVT($x)}</x>,<email>{ fn:string($email)}</email></Result>)
          )
          return (func:OPTIONAL($BGP_1, $BGP_2))
     )
     for $res in $P_1
     where (fn:exists($res/email)=false)
     return ($res )
)
return (<Results>{$Results }</Results>)
```