ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εκτέλεση ερωτήσεων XPath με τη χρήση πολυδιάστατης αναζήτησης

Γιαννακάρας Γιώργος

Επιβλέπων Καθηγητής : Καθ. Σαμολαδάς Βασίλης
Εξεταστική Επιτροπή : Καθ. Σαμολαδάς Βασίλης
Καθ. Χριστοδουλάκης Σταύρος
Καθ. Πετράκης Ευριπίδης

Χανιά, 2005

Στους γονείς μου...

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα καθηγητή μου κ. Σαμολαδά Βασίλη για την ανάθεση αυτής της διπλωματικής εργασίας και για την καθοδήγησή του καθ' όλη την διάρκεια αυτής. Φυσικά ευχαριστίες θα ήθελα να δώσω στην οικογένεια μου, για την υποστήριξή που μου παρείχε όλα αυτά τα χρόνια των σπουδών μου στην Κρήτη.

Περιεχόμενα

Κεφάλαιο 1
ΕΙΣΑΓΩΓΗ

1.1 Αναγκαιότητα της XPath

Η XML (Extendible Markup Language) [1, 2, 11, 14] μας παρέχει τον τρόπο να δομήσουμε, να ανταλλάξουμε και να περιγράψουμε δεδομένα με ένα πλούσιο, ευέλικτο και αποτελεσματικό τρόπο. Αυτό το επιτυγχάνει με την χρήση περιγραφικών tags (ετικέτες) για τα δεδομένα. Σημαντικό είναι ακόμα και το γεγονός πως η δομή δεδομένων που υπόκειται ενός XML εγγράφου και είναι γνωστή ως δέντρο, είναι ιδανική για να συγκρατήσει την δομή διακριτών πηγών δεδομένων και αρκετά απλή για την επεξεργασία των δεδομένων που συγκρατεί, από αποδοτικούς και κομψούς αλγορίθμους (π.χ. αναδρομικούς).

Εντούτοις, παρά την ευελιξία που μας παρέχει για την περιγραφή και την ανταλλαγή δεδομένων και πληροφορίας, η XML δεν μας παρέχει τον τρόπο να εντοπίσουμε συγκεκριμένα κομμάτια δομημένων δεδομένων μέσα σε ένα έγγραφο. Έτσι για παράδειγμα, ένα XML έγγραφο που περιέχει δεδομένα σχετικά με διάφορα βιβλία κάποιου εκδοτικού οίκου, θα χρειαστεί να το ψάξουμε στοιχείο προς στοιχείο, προκειμένου να εντοπίσουμε κάποιο συγκεκριμένο βιβλίο που επιθυμούμε να βρούμε. Η διαδικασία αυτή όπως γίνεται εύκολα αντιληπτό, είναι χρονοβόρα και αναποτελεσματική, ιδίως όταν πρόκειται για μεγάλα XML έγγραφα.

Έχοντας μεγάλες ποσότητες δεδομένων να αναπαρίστανται ως XML έγγραφα, η ανάγκη να υπάρξει κάποιος αποτελεσματικός τρόπος αναζήτησης και εύρεσης των δεδομένων ήταν ιδιαιτέρως επιτακτική. Για τον λόγο αυτό το W3C (World Wide Web Consortium) [1, 14] πρότεινε την XPath (XML Path Language) [4, 11, 14].

Η XPath παρέχει την σύνταξη για τον αποτελεσματικό εντοπισμό συγκεκριμένων τμημάτων μέσα στο XML έγγραφο (π.χ. τις τιμές των attributes). Η XPath δεν είναι μια δομική γλώσσα όπως είναι η XML. Βασικά πρόκειται για μία γλώσσα από εκφράσεις με συγκεκριμένη σύνταξη που χρησιμοποιείται από άλλες τεχνολογίες της XML, όπως η XSLT (Extendible Stylesheet Language Transformations) [5, 11, 14] η οποία μετασχηματίζει ή μετατρέπει τα XML έγγραφα σε άλλες μορφές για παράδειγμα σε HTML [7, 14] και τον XPointer (XML Pointer

Language) [9] , ο οποίος περιγράφει μηχανισμούς για τον εντοπισμό συγκεκριμένων τμημάτων μέσα σε ένα XML έγγραφο.

1.2 Περιγραφή της διπλωματικής

Βασικός στόχος αυτής της εργασίας είναι η υλοποίηση ενός συστήματος, που εμείς ονομάσαμε XPath Query Engine, το οποίο θα μας δίνει την δυνατότητα, να κάνουμε αναζητήσεις μέσα σε XML έγγραφα, με την χρήση των εκφράσεων, που μας παρέχει η σύνταξη της XPath.

Για την υλοποίηση του συστήματός μας, στηριχτήκαμε σε μία σχετική εργασία, τον *XPath accelerator*, που έχει γίνει από μια ερευνητική ομάδα του πανεπιστήμιου της Konstanz [13]. Ο XPath accelerator είναι μία δομή δεικτοδότησης (index structure) που σχεδιάστηκε ειδικά για την εκτίμηση των XPath queries. Πρόκειται, για μία δομή που μπορεί να υποστηρίξει και τους 13 XPath άξονες π.χ. **ancestor**, **descendant**, **parent** κ.τ.λ. Η βασική υλοποίηση του συστήματός μας, με την οποία πραγματοποιούμε τις αναζητήσεις για την ανάκτηση των δεδομένων, βασίστηκε στην χρήση της πολυδιάστατης αναζήτησης, την οποία πραγματοποιήσαμε με την χρησιμοποίηση μιας δενδρικής δομής, που κρατάει πολυδιάστατα δεδομένα, το kd-tree.

1.3 Η οργάνωση του κειμένου

Το κείμενο της διπλωματικής μας περιλαμβάνει 5 κεφάλαια μαζί με την εισαγωγή. Στο δεύτερο κεφάλαιο παρέχουμε όλο το απαραίτητο θεωρητικό υπόβαθρο που πρέπει να διαθέτει ο αναγνώστης του κειμένου, για να παρακολουθήσει την εξέλιξη της διπλωματικής. Έτσι παρουσιάζουμε στοιχεία για τους XML parsers, κάνουμε μια παρουσίαση της XPath γλώσσας και παρουσιάζουμε αναλυτικά τον XPath accelerator. Στο τρίτο κεφάλαιο παρουσιάζουμε αναλυτικά την αρχιτεκτονική και την υλοποίηση του συστήματός μας, στο τέταρτο κεφάλαιο παρουσιάζουμε τις μετρήσεις που πραγματοποιήσαμε για να εκτιμήσουμε την απόδοση του συστήματος και τέλος στο πέμπτο κεφάλαιο κάνουμε μια σύνοψη και παραθέτουμε τα συμπεράσματα μας.

Κεφάλαιο 2

Θεωρητικό Υπόβαθρο

Σε αυτό το κεφάλαιο παραθέτουμε όλες τις απαραίτητες γνώσεις και το θεωρητικό υπόβαθρο που θα πρέπει να διαθέτει ο αναγνώστης αυτού του κειμένου, για την κατανόηση και την παρακολούθηση αυτής της διπλωματικής εργασίας. Ξεκινάμε, παραθέτοντας στοιχεία που αφορούν την γλώσσα XML, συνεχίζουμε εξετάζοντας τη δομή των XML εγγράφων, ακολουθεί αναφορά στους XML parsers ερευνώντας πιο ειδικά το DOM API, το οποίο χρησιμοποιήθηκε για την υλοποίηση μας, στην συνέχεια παρουσιάζουμε αναλυτικά την XPath επικεντρώνοντας στην σύνταξη των εκφράσεων της γλώσσας και τέλος παρουσιάζουμε στοιχεία της υλοποίησης του XPath accelerator, τα οποία χρησιμοποιήθηκαν και για την υλοποίηση του δικού μας συστήματος.

2.1 eXtensible Markup Language (XML)

2.1.1 Ιστορικά Στοιχεία

Η XML [1, 2, 11, 14] είναι ένα υποσύνολο της SGML (Structured Generalized Markup Language) [3] και προσδιορίστηκε το 1998 από το W3C (World Wide Web Consortium) [1]. Ονομάστηκε extensible, δηλαδή επεκτάσιμη, επειδή επιτρέπει στους χρήστες να ορίσουν το δικό τους σχήμα, σε αντίθεση με την HTML [7], η οποία είναι μια προκαθορισμένη γλώσσα. Ο λόγος που η XML μοιάζει τόσο πολύ με την HTML έγκειται στο γεγονός ότι και η HTML αποτελεί ένα υποσύνολο της SGML. Η XML όμως μοιάζει πολύ περισσότερο με την SGML απ' ότι η HTML, επειδή η HTML είναι ένα συγκεκριμένο υποσύνολο της SGML που χρησιμοποιείται για να περιγράψει ιστοσελίδες. Βασικά ένας από τους λόγους δημιουργίας της XML ήταν η απλοποίηση της SGML. Δεν είναι επομένως παράξενο το γεγονός πως το W3C λόγω της κομψότητας και της απλότητας που παρουσιάζει η XML αποφάσισε να επαναπροσδιορίσει την HTML δημιουργώντας την XHTML και παράλληλα να εμφανιστούν και άλλες διάλεκτοι όπως η WeatherML (γλώσσα περιγραφής του καιρού), η CellML (γλώσσα περιγραφής βιολογικών μοντέλων) ή η XMLPay που περιγράφει πληρωμές στο internet.

2.1.2 Γιατί XML

Υπάρχουν πολλοί λόγοι για τους οποίους μπορεί κανείς σήμερα να χρησιμοποιήσει την XML. Στην συγκεκριμένη παράγραφο όμως, δε θα σταθούμε στα πλεονεκτήματα που μπορεί να μας αποφέρει η χρήση της XML για συγκεκριμένες εφαρμογές όπως είναι το εμπόριο στο διαδίκτυο, τα μαθηματικά, η βιολογία, η χημεία κ.α. – αλλά θα εστιάσουμε σε συγκεκριμένες ιδιότητες της XML οι οποίες είναι ιδιαίτερα χρήσιμες για όλες τις εφαρμογές και καθιστούν την XML ως ένα από τους πιο δημοφιλείς τρόπους για την αναπαράσταση και την περιγραφή δεδομένων.

- Η XML είναι εύκολα αναγνώσιμη από ανθρώπους και μηχανές: Οι περισσότερες μορφές αποθήκευσης δεδομένων ήταν είτε κατάλληλες για μετάφραση από προγράμματα λογισμικού (π.χ. dBase, GIF), ή αναγνώσιμα από ανθρώπους (text ή CSV αρχεία). Η XML ορίζει ένα σύνολο από κανόνες που κάνουν την μετάφραση από υπολογιστή πολύ απλή. Έτσι ικανοποιούνται και οι δύο πλευρές, αφού τα XML έγγραφα διατηρούν ως βάση τους το κείμενο κι έτσι μπορεί εύκολα να τα χειριστεί ένας άνθρωπος.
- Η XML είναι φιλική στα αντικείμενα (object-friendly): Ενώ το σχεσιακό μοντέλο δεδομένων εμφανίζει μεγάλη επιτυχία για την επεξεργασία μεγάλων ποσοτήτων δεδομένων αποθηκευμένων σε πίνακες, ο χειρισμός άλλων τύπων δεδομένων - όπως είναι το hypertext (κείμενο με hyperlinks), πολυμέσα, γραφικά, μαθηματικές ή χημικές φόρμουλες, ιεραρχική πληροφορία – δεν είναι τόσο απλός. Η XML από την άλλη πλευρά είναι φιλική στα αντικείμενα, υπό την έννοια ότι είναι κατάλληλη για την περιγραφή αντικειμένων του πραγματικού κόσμου ή οποιουδήποτε αφαιρετικού προβλήματος μοντελοποιώντας τις ιδιότητες όπως ακριβώς είναι, αντί να χρειάζεται μια κανονικοποιημένη διάσπαση σε διάφορους πίνακες, με τους οποίους συνδέονται διάφορες σχέσεις. Αυτό κάνει τα XML έγγραφα περισσότερο κατανοητά κι έτσι μειώνεται ο χρόνος που απαιτείται για την σχεδίαση και υλοποίηση υπολογιστικών συστημάτων που βασίζονται στην XML.
- Η XML έχει ευρέως υιοθετηθεί από την βιομηχανία υπολογιστών: Η XML είναι ευρέως αποδεκτή και υλοποιείται από πολλές εταιρίες. Το γεγονός αυτό έχει ως αποτέλεσμα χαμηλότερο κόστος για όλα τα συστατικά του λογισμικού.

- Η XML είναι παγκόσμια: Για να γίνει ευκολότερα κατανοητός ο λόγος για τον οποίο η XML έγινε τόσο αποδεκτή, θα ήταν χρήσιμο να αναφερθούμε στον ASCII κώδικα (American Standard Code for Information Interchange), ο οποίος είναι επίσης ιδιαίτερα αποδεκτός. Παρόλο που ο ASCII κώδικας διαθέτει ένα συγκεκριμένο αλφάβητο και σύστημα γραφής, ήταν απαραίτητο να επιτρέψει την ελεύθερη ανταλλαγή δεδομένων μεταξύ διαφορετικών τύπων υπολογιστών και λειτουργικών συστημάτων. Η ιδέα του ASCII επεκτάθηκε σε τέτοιο βαθμό ώστε να συμπεριλάβει όλες τις γλώσσες και όλα τα συστήματα γραφής του κόσμου. Σήμερα θεωρούμε ως δεδομένο ότι οι υπολογιστές μπορούν να διαβάζουν και να επεξεργάζονται έγγραφα κειμένου που βασίζονται στον ASCII κώδικα. Η XML επεκτείνει αυτή τη προσέγγιση χρησιμοποιώντας το Unicode και ορίζοντας ένα καθολικό τρόπο για την περιγραφή δομημένων δεδομένων για κάθε διαφορετικό σκοπό. Όλα τα XML έγγραφα είναι εξ' ορισμού βασισμένα σε Unicode, αλλά μπορούν να αποθηκευτούν στο δίσκο ή να μεταδοθούν σε ένα δίκτυο με διάφορες κωδικοποιήσεις όπως ISO-8859-1 ή UTF-8. Αυτός είναι και ένας λόγος που μερικοί σήμερα καλούν την XML "ASCII του μέλλοντος".

2.1.3 XML Έγγραφα

Έχοντας πλέον δει κάποια στοιχεία που αφορούν τα χαρακτηριστικά και την ιστορία της XML μπορούμε πλέον να δώσουμε ένα συγκεκριμένο ορισμό για το τι είναι XML. Μπορούμε να ορίσουμε επομένως ότι : ***XML είναι ένας ξεκάθαρος και καθορισμένος τρόπος να δομήσουμε, να περιγράψουμε και να ανταλλάξουμε δεδομένα***. Μπορούμε π.χ. να χρησιμοποιήσουμε την XML για να περιγράψουμε μαθηματικές φόρμουλες, χημικά μείγματα, αστρονομική πληροφορία κ.α.

Για να αντιληφθούμε ακόμα καλύτερα τι είναι XML ας δούμε ένα απλό παράδειγμα ενός XML εγγράφου όπως αυτό παρουσιάζεται στην εικόνα 2.1.

```
<invoice due="2000-09-22">
  <product>
    <name>Apple</name>
    <price>0.10</price>
  </product>
  <product>
    <name>Orange</name>
    <price>0.08</price>
  </product>
  <product>
    <name>Strawberries</name>
    <price>0.20</price>
  </product>
  <product>
    <name>Banana</name>
    <price>0.14</price>
  </product>
  <total currency="US$">0.52</total>
</invoice>
```

Εικόνα 2.1 : Ένα απλό XML έγγραφο

Τα XML έγγραφα περιέχουν δομημένο κείμενο. Η δομή καθορίζεται από ειδικά διαμορφωμένο κείμενο γύρω από τα δεδομένα κειμένου και το ονομάζουμε markup tags. Τα όρια ενός tag προσδιορίζονται από τα σύμβολα ανισότητας < και >. Το κείμενο μεταξύ των συμβόλων ανισότητας περιέχει πληροφορία που προσδιορίζει το element. Το element αποτελείται από ένα tag που ανοίγει, τα περιεχόμενα του element και από ένα tag που κλείνει. Το κλείσιμο ενός element που έχει ανοίξει είναι υποχρεωτικό. Για να κλείσουμε ένα tag χρησιμοποιούμε το ίδιο όνομα με το tag που ανοίγει με την διαφορά ότι για να συμβολίσουμε ένα tag που κλείνει χρησιμοποιούμε τα σύμβολα </ και >. Τα elements μπορεί να περιέχουν κείμενο, άλλα elements ή ακόμα και συνδυασμό των δύο. Υπάρχουν ακόμα και τα άδεια elements τα οποία είτε έχουν ένα tag που κλείνει χωρίς περιεχόμενα **</element>**, ή συμβολίζονται ως **<element/>** προκειμένου να τα ξεχωρίζουμε από αντικείμενα που δεν έχουν κλείσει. Τα σχόλια στην XML εμφανίζονται μεταξύ των συμβόλων **<!- -** και **- ->**. Τα tags μπορούν ακόμα να περιέχουν προαιρετικά, επιπρόσθετη πληροφορία, τα attributes (χαρακτηριστικά γνωρίσματα). Τα attributes τοποθετούνται μέσα στο tag ενός αντικειμένου που ανοίγει και γράφονται με την μορφή *attribute_name = "value"*.

Το XML έγγραφο που απεικονίζεται στο εικόνα 2.1 περιγράφει ένα τιμολόγιο με 4 προϊόντα και το άθροισμα τους. Το **<invoice>** element έχει ένα attribute το **due** με τιμή "*2000-09-22*". Ακόμα περιλαμβάνει 5 elements εκ των οποίων τα 4 είναι **<product>** και το πέμπτο είναι το **<total>**. Βλέπουμε επομένως ότι elements στην XML όπως στην περίπτωση του **<product>** μπορούν να επαναλαμβάνονται. Το κάθε

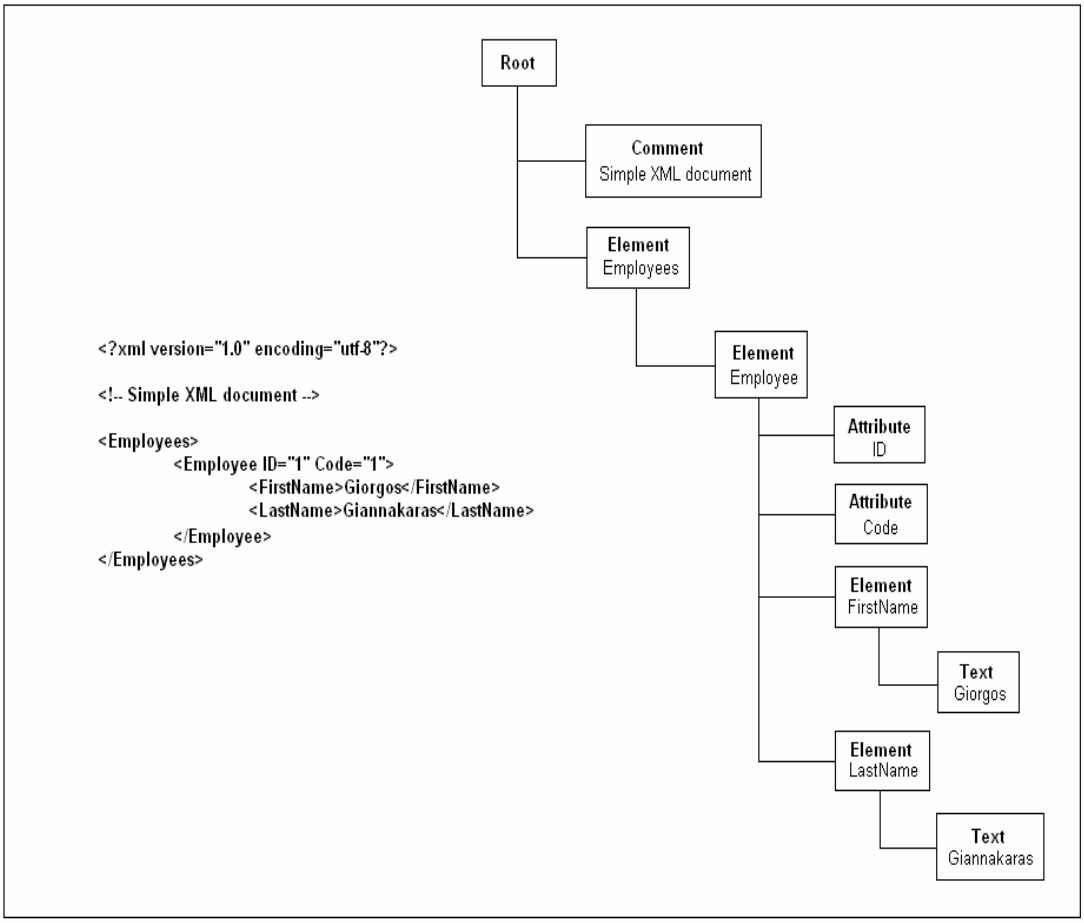
<product> περιέχει 2 elements τα **<name>** και **<price>**. Τα **<name>** και **<price>** περιέχουν πληροφορία σχετικά με το όνομα και την τιμή του κάθε προϊόντος. Έτσι για παράδειγμα το πρώτο προϊόν έχει όνομα **Apple** και τιμή **0.10**. Το element **<total>** περιλαμβάνει ένα attribute το **currency** με τιμή "*US$*" και την πληροφορία ότι το συνολικό κόστος είναι **0.52** δολάρια.

2.1.4 Δενδρική αναπαράσταση των XML εγγράφων

Ένα XML έγγραφο μπορεί να αναπαρασταθεί σαν ένα δέντρο, του οποίου οι κόμβοι περιέχουν τα δομικά συστατικά του εγγράφου, δηλαδή τα elements, το κείμενο, τα attributes, τα σχόλια και τις processing instruction. Η δενδρική αναπαράσταση ενός XML εγγράφου θυμίζει την δομή των αρχείων και των φακέλων στον σκληρό δίσκο ενός υπολογιστή. Οι κόμβοι είναι πανομοιότυποι με τα αρχεία και τους φακέλους. Έτσι όπως οι φάκελοι μπορεί να περιέχουν άλλα αρχεία και φακέλους, έτσι και ένας κόμβος μπορεί να περιέχει άλλους κόμβους.

Το δέντρο έχει ένα κόμβο αφετηρίας (root node) που περιέχει όλους τους υπόλοιπους κόμβους του δέντρου. Ο κόμβος της αφετηρίας περιέχει το element έγγραφο. Ο κόμβος που περιέχει το έγγραφο και οι κόμβοι που περιέχουν elements περιέχουν λίστες με τους κόμβους – παιδιά. Κάθε κόμβος μέσα στο δέντρο, εκτός από τον κόμβο αφετηρίας, έχει ένα κόμβο – γονέα και οι κόμβοι – γονείς μπορεί να έχουν ένα αριθμό από κόμβους παιδιά ή απογόνους. Οι κόμβοι που περιέχουν κείμενο, attributes και σχόλια, δεν έχουν απογόνους.

Στο σημείο αυτό πρέπει να επισημάνουμε μια ιδιαιτερότητα που έχουν οι κόμβοι που περιέχουν attributes. Η σχέση που υπάρχει μεταξύ ενός πατρικού κόμβου και ενός κόμβου – παιδιού, είναι περιεκτική, δηλαδή ο κόμβος – γονιός περιέχει τον κόμβο – παιδί. Στην περίπτωση ενός κόμβου που περιέχει ένα attribute, παρόλο που ο κόμβος έχει ως γονιό ένα κόμβο που περιέχει ένα element, το attribute δεν θεωρείται παιδί του πατρικού κόμβου. Η σχέση που υπάρχει μεταξύ του κόμβου που περιέχει το attribute και του πατρικού του κόμβου είναι περιγραφική, δηλαδή το attribute περιγράφει τον πατρικό του κόμβο και δεν εμπεριέχεται σε αυτόν, όπως συμβαίνει με τους υπόλοιπους κόμβους που είναι παιδιά άλλων κόμβων. Επομένως οι κόμβοι που περιέχουν attributes δεν θεωρούνται κόμβοι – παιδιά, καθώς χρησιμοποιούνται για την παροχή πληροφορίας που περιγράφει τον πατρικό τους κόμβο. Στην εικόνα 2.2 μπορούμε να δούμε την δενδρική αναπαράσταση ενός XML εγγράφου.



Εικόνα 2.2 : Ένα απλό XML έγγραφο και η δενδρική του αναπαράσταση

Ο κόμβος αφετηρίας στην παραπάνω εικόνα περιέχει 2 κόμβους – παιδιά. Περιέχει το σχόλιο με την τιμή "*Simple XML document*" και ένα element το **Employees**. Το element **Employees** περιέχει ένα κόμβο – παιδί το element **Employee**. Ο κόμβος που περιέχει το element **Employee** είναι γονιός των attribute – κόμβων **ID** και **Code**, όμως οι attribute – κόμβοι **ID** και **Code** δεν είναι παιδιά του element – κόμβου **Employee**. Επιπλέον στα παιδιά – κόμβους του **Employee** συγκαταλέγονται και οι 2 element – κόμβοι **FirstName** και **LastName**. Οι 2 element – κόμβοι **FirstName** και **LastName** είναι γονείς των text – κόμβων με τις τιμές "*Giorgos*" και "*Giannakaras*" αντίστοιχα.

2.2 Document Object Model (DOM)

2.2.1 XML Parsers

Στην προηγούμενη παράγραφο, είδαμε πώς τα XML έγγραφα μπορούν να αναπαρασταθούν ως δέντρα. Εκμεταλλευόμενοι την δενδρική αυτή δομή των εγγράφων, μπορούμε να χρησιμοποιήσουμε τους κατάλληλους αλγορίθμους, προκειμένου να μπορέσουμε να επεξεργαστούμε τα δεδομένα που αναπαρίστανται σε αυτά.

Για τον σκοπό αυτό χρειαζόμαστε ένα XML parser, ο οποίος ουσιαστικά διαβάζει το έγγραφο και δημιουργεί μία ιεραρχική δενδρική δομή του εγγράφου στην μνήμη. Η δενδρική αυτή δομή περιέχει όλα τα συστατικά ενός XML εγγράφου, τα οποία είδαμε αναλυτικά νωρίτερα. Προκειμένου να λάβουμε ή να αναζητήσουμε την πληροφορία που ο parser ανακτά από το XML έγγραφο, χρησιμοποιούμε τις μεθόδους που είναι ορισμένες στο API (application programming interface) του parser.

Μία σημαντική απόφαση που θα χρειαστεί επομένως να πάρει κανείς όταν βρίσκεται στην αρχή ενός XML project, είναι ποιο API θα χρησιμοποιήσει. Τα πιο σημαντικά APIs για την επεξεργασία των XML εγγράφων με Java, είναι το SAX (Simple API for XML) [10, 11, 14] και το DOM (Document Object Model) [6, 11, 14]. Επειδή στην υλοποίησή μας χρησιμοποιήσαμε τον parser *Xerces* [12] που βασίζεται στο DOM API θα αναφερθούμε μόνο στο DOM.

2.2.2 Τι είναι το Document Object Model

Το Document Object Model, είναι ένα πολύπλοκο API που μοντελοποιεί ένα XML έγγραφο ως δέντρο. Με το DOM μπορούμε να αναλύσουμε έγγραφα, καθώς και να δημιουργήσουμε καινούρια. Το κάθε XML έγγραφο αναπαριστάνεται σαν ένα document object. Στα έγγραφα μπορούμε να κάνουμε αναζητήσεις και ενημερώσεις καλώντας τις μεθόδους και τα objects που περιέχει το Document object. Parsers που είναι βασισμένοι στο DOM υπάρχουν διαθέσιμοι σε μία ποικιλία από προγραμματιστικές γλώσσες και συνήθως είναι διαθέσιμοι χωρίς χρέωση. Πολλές εφαρμογές όπως για παράδειγμα ο Internet Explorer 5 διαθέτουν ενσωματωμένους parsers. Αναφορικά παραθέτουμε 6 διαφορετικούς parsers που βασίζονται στο DOM

και είναι οι εξής : JAXP [15], XML4J [16], Xerces [12], msxml [17], 4DOM [18] και XML::DOM [19].

2.2.3 DOM API

Σε αυτή την παράγραφο παρουσιάζουμε κάποιες από τις πιο σημαντικές DOM κλάσεις, interfaces και μεθόδους. Λόγω του ότι υπάρχει ένας μεγάλος αριθμός από DOM objects και από μεθόδους που είναι διαθέσιμες, εμείς παρέχουμε μόνο ένα μέρος από αυτά τα objects και τις μεθόδους.

Interface	Description
Document interface	Αναπαριστά τον κόμβο κορυφής του XML έγγραφου, που παρέχει πρόσβαση σε όλους τους κόμβους του εγγράφου – συμπεριλαμβανομένου και του root element.
Node interface	Αναπαριστά ένα κόμβο ενός XML εγγράφου.
NodeList interface	Αναπαριστά μία λίστα από **Node** objects.
Element interface	Αναπαριστά ένα element κόμβο. Παράγεται από το **Node.**
Attr interface	Αναπαριστά ένα attribute κόμβο. Παράγεται από το **Node.**
CharacterData Interface	Αναπαριστά δεδομένα χαρακτήρων. Παράγεται από το **Node.**
Text interface	Αναπαριστά ένα text κόμβο. Παράγεται από το **Characterdata.**
Comment interface	Αναπαριστά ένα comment κόμβο. Παράγεται από το **Characterdata.**
ProcessingInstruction interface	Αναπαριστά ένα processing instruction κόμβο. Παράγεται από το **Node.**

Πίνακας 2.1: DOM classes και interfaces.

Το **Document** interface αναπαριστά τον κόμβο κορυφής ενός XML εγγράφου στην μνήμη και μας παρέχει με μεθόδους με τις οποίες μπορούμε να δημιουργήσουμε και να ανακτήσουμε κόμβους. Στον πίνακα 2.2 βλέπουμε κάποιες από τις **Document** μεθόδους. Στον πίνακα 2.3 βλέπουμε μεθόδους της κλάσης **XmlDocument,** συμπεριλαμβανόμενες και μεθόδους, οι οποίες κληρονομούνται από το **Document,** ενώ στον πίνακα 2.4 βλέπουμε μεθόδους του interface **Node.**

Method Name	Description
createElement	Δημιουργεί ένα element κόμβο.
createAttribute	Δημιουργεί ένα attribute κόμβο.

createTextNode	Δημιουργεί ένα text κόμβο.
createComment	Δημιουργεί ένα comment κόμβο.
createProcessingInstruction	Δημιουργεί ένα processing instruction κόμβο.
getDocumentElement	Επιστρέφει το root element του εγγράφου.
appendChild	Προσθέτει ένα κόμβο – παιδί.
getChildNodes	Επιστρέφει τους κόμβους – παιδιά.

Πίνακας 2.3: Document methods

Method Name	Description
createXmlDocument	Κάνει parse ένα XML έγγραφο.
appendChild	Προσθέτει ένα κόμβο – παιδί.
cloneNode	Διπλασιάζει το κόμβο.
getAttributes	Επιστρέφει τα attributes ενός κόμβου.
getChildNodes	Επιστρέφει τους κόμβους – παιδιά του κόμβου.
getNodeName	Επιστρέφει το όνομα του κόμβου.
getNodeType	Επιστρέφει τον τύπο του κόμβου π.χ. element, attribute, text κ.τ.λ.
getNodeValue	Επιστρέφει την τιμή του κόμβου.
getParentNode	Επιστρέφει τον πατέρα του κόμβου.
hasChildNodes	Επιστρέφει **true** αν ο κόμβος έχει κόμβους – παιδιά.
removeChild	Μετακινεί ένα κόμβο – παιδί από τον κόμβο.
replaceChild	Αντικαθιστά ένα κόμβο – παιδί με ένα άλλο κόμβο.
setNodeValue	Ορίζει την τιμή του κόμβου.

Πίνακας 2.4: Node methods.

Στον πίνακα 2.5 παρουσιάζουμε κάποιους τύπους κόμβων, οι οποίοι επιστρέφονται από την μέθοδο **getNodetype.** Κάθε τύπος στον πίνακα 2.5 είναι ένα static final μέλος της κλάσης **Node**. Στον πίνακα 2.6 παρουσιάζουμε κάποιες **Element** μεθόδους.

Node Type	Description
Node.ELEMENT_NODE	Αναπαριστά ένα element κόμβο.
Node.ATTRIBUTE_NODE	Αναπαριστά ένα attribute κόμβο.
Node.TEXT_NODE	Αναπαριστά ένα text κόμβο.
Node.COMMENT_NODE	Αναπαριστά ένα comment κόμβο.
Node.PROCESSING_INSTRUCTION_NODE	Αναπαριστά ένα processing instruction κόμβο.

Πίνακας 2.5: Node Types.

Method Name	Description
getAttribute	Επιστρέφει την τιμή ενός attribute.
getTagName	Επιστρέφει το όνομα ενός element.
removeAttribute	Μετακινεί ενός element το attribute.
setAttribute	Ορίζει την τιμή ενός attribute.

Πίνακας 2.6: Element methods.

2.3 XML Path Language (XPath)

Έχοντας πλέον δει την δομή ενός XML εγγράφου αναλυτικά, εξετάζουμε τώρα πώς μπορούμε να χρησιμοποιήσουμε τη δομή αυτή, με τη βοήθεια της XPath, για να εντοπίσουμε συγκεκριμένα τμήματα του εγγράφου. Για την υλοποίηση των εκφράσεων της XPath εκμεταλλευόμαστε τη δενδρική δομή των XML εγγράφων, καθώς στην XPath αντιμετωπίζουμε τα XML έγγραφα σαν να είναι δέντρα στα οποία το κάθε τμήμα του εγγράφου παριστάνεται ως κόμβος. Η XPath υποστηρίζει τους τύπους κόμβων που παρουσιάσαμε νωρίτερα στο DOM. Επομένως υποστηρίζει τους εξής τύπους : **root, element, attribute, text, comment** και **processing instructio***n.*

2.3.1 Location Paths

Ένα location path (μονοπάτι τοποθεσίας) είναι μία έκφραση, με την οποία ορίζουμε τον τρόπο με τον οποίο θα κινηθούμε μέσα στο XML έγγραφο, από ένα κόμβο σε έναν άλλο. Το location path αποτελείται από τα location steps (βήματα τοποθεσίας), καθένα από τα οποία αποτελείται από ένα axis (άξονα), ένα node test (επιλογέα κόμβου) και ένα προαιρετικό predicate (φίλτρο). Για να μπορέσουμε να εντοπίσουμε ένα συγκεκριμένο κόμβο μέσα στο έγγραφο, χρησιμοποιούμε πολλαπλά location steps, καθένα από τα οποία κάνει την ερευνά μας περισσότερο συγκεκριμένη.

2.3.1.1 Axes

Το ψάξιμο στο XML έγγραφο ξεκινάει από ένα context node (κόμβο αναφοράς). Όλες οι διασχίσεις γίνονται με αφετηρία τον context node. Ένας άξονας καθορίζει ποιοι κόμβοι, που σχετίζονται με τον context node, θα συμπεριληφθούν στην έρευνά μας. Ο άξονας που επιλέγουμε για την έρευνά μας, μας δείχνει ακόμα

την σειρά των κόμβων στο έγγραφο. Έτσι οι άξονες που επιλέγουν κόμβους που ακολουθούν τον context node σε document order καλούνται forward axes. Οι άξονες που επιλέγουν κόμβους που προηγούνται σε σχέση με τον context node σε document order καλούνται reverse axes. Η XPath υποστηρίζει συνολικά 13 άξονες, τους οποίους παραθέτουμε στον πίνακα 2.7, καθώς και μια σύντομη περιγραφή του καθενός.

Axis Name	Ordering	Description
self	none	Ο ίδιος ο context node.
parent	reverse	Ο γονιός του context node, αν υπάρχει.
child	forward	Τα παιδιά του context node, αν υπάρχουν.
ancestor	reverse	Οι πρόγονοι του context node, αν υπάρχουν.
ancestor-or-self	reverse	Οι πρόγονοι του context node, καθώς και ο ίδιος.
descendant	forward	Οι απόγονοι του context node.
descendant-or-self	forward	Οι απόγονοι του context node, καθώς και ο ίδιος.
following	forward	Οι κόμβοι στο XML έγγραφο που ακολουθούν τον context node, χωρίς να συμπεριλαμβάνουμε τους απογόνους.
following-sibling	forward	Οι γειτονικοί κόμβοι που ακολουθούν τον context node.
preceding	reverse	Οι κόμβοι στο XML έγγραφο που προηγούνται του context node, χωρίς να συμπεριλαμβάνουμε τους προγόνους.
preceding-sibling	reverse	Οι γειτονικοί κόμβοι που προηγούνται του context node.
attribute	forward	Οι attribute κόμβοι του context node.
namespace	forward	Οι namespace κόμβοι του context node.

Πίνακας 2.7: XPath Axes.

Ο κάθε άξονας έχει ένα κύριο τύπο κόμβου που ανταποκρίνεται στον τύπο κόμβου που ο συγκεκριμένος άξονας μπορεί να επιλέξει. Έτσι για τον attribute άξονα ο κύριος τύπος κόμβου είναι attribute. Για τον namespace άξονα ο κύριος τύπος κόμβου είναι namespace. Όλοι οι υπόλοιποι άξονες έχουν τον element κύριο τύπο κόμβου.

2.3.1.2 Node Tests

Όπως είδαμε στην προηγούμενη παράγραφο ένας άξονας επιλέγει ένα σύνολο από κόμβους. Το σύνολο των κόμβων που επιλέξαμε με την χρήση του άξονα, μπορούμε να το κάνουμε πιο συγκεκριμένο με την χρήση των node tests. Τα node tests για την επιλογή των κόμβων, στηρίζονται στον κύριο τύπο κόμβου του άξονα. Στον πίνακα 2.8 παραθέτουμε τα node tests, καθώς και μία σύντομη περιγραφή τους.

Node Test	Description
*	Επιλογή όλων των κόμβων του ίδιου τύπου.
node()	Επιλογή όλων των κόμβων, ανεξάρτητα από τον τύπο τους .
text()	Επιλογή όλων των text nodes.
comment()	Επιλογή όλων των comment nodes.
processing-instruction()	Επιλογή όλων των processing-instruction nodes.
node name	Επιλογή όλων των κόμβων με το συγκεκριμένο *node name.*

Πίνακας 2.8: XPath Node Tests.

2.3.1.3 Location Paths χρησιμοποιώντας Axes και Node Tests

Τα location paths αποτελούνται από αλληλουχίες από location steps. Ένα location step περιλαμβάνει ένα άξονα και ένα node test, τα οποία διαχωρίζονται από μία διπλή άνω κάτω τελεία (::) και προαιρετικά από ένα predicate, το οποίο περικλείεται από αγκύλες ([ ]). Καλύτερα όμως ας δούμε μερικά παραδείγματα από location paths για τον εντοπισμό συγκεκριμένων elements σε ένα XML έγγραφο. Για τα παραδείγματα που ακολουθούν χρησιμοποιούμε το XML έγγραφο της εικόνας 2.3.

```
<?xml version="1.0" encoding="utf-8"?>
<Employees>
  <Employee ID="1">
    <FirstName>Klaus</FirstName>
    <LastName>Salchner</LastName>
    <PhoneNumber>410-727-5112</PhoneNumber>
    <EmailAddress>klaus_salchner@hotmail.com</EmailAddress>
    <WebAddress>http://www.enterprise-minds.com</WebAddress>
    <JobTitle>Sr. Enterprise Architect</JobTitle>
  </Employee>
  <Employee ID="2">
    <FirstName>Peter</FirstName>
    <LastName>Pan</LastName>
    <PhoneNumber>604-111-1111</PhoneNumber>
    <EmailAddress>peter.pan@fiction.com</EmailAddress>
    <JobTitle>Sr. Developer</JobTitle>
  </Employee>
</Employees>
```

Εικόνα 2.3 : XML έγγραφο, στο οποίο περιγράφονται τα στοιχεία των εργαζόμενων μιας εταιρίας

Το location step *child::** επιλέγει όλους τους element κόμβους του context node, αφού ο κύριος τύπος κόμβου για τον *child* άξονα, είναι το element. Όταν χρησιμοποιούμε τον *child* άξονα στην αρχή μιας XPath έκφρασης, αναφερόμαστε στο root του XML έγγραφου. Χρησιμοποιώντας τον **Employees** element – κόμβο από την εικόνα 2.3 ως context node, θα επιλεγούν οι 2 element – κόμβοι **Employee**. Το location step *child::text()* χρησιμοποιεί τον *child* άξονα και το node test *text(),* για να επιλέξει όλους τους κόμβους – παιδιά του context node που είναι τύπου *text()*. Μπορούμε να συνδυάσουμε τα 2 παραπάνω location steps για να σχηματίσουμε το location path *child::*/child::text()*, με το οποίο επιλέγουμε όλους τους text – κόμβους εγγόνια του context node. Σε αυτό το location path υπάρχουν 2 βήματα. Το πρώτο βήμα, *child::**, επιλέγει όλους τους element – κόμβους παιδιά του context node. Το δεύτερο βήμα *child::text()*, επιλέγει όλους τους text – κόμβους παιδιά του συνόλου των κόμβων που επιλέχθηκαν από το πρώτο βήμα. Χρησιμοποιώντας ξανά τον **Employees** element – κόμβο από την εικόνα 2.3 ως context node, δεν θα επιλεγεί κανένας κόμβος από το συγκεκριμένο location path, αφού ο **Employees** element – κόμβος δεν έχει text – κόμβους εγγόνια. Αν ο **Employee** element – κόμβος είχε τουλάχιστον ένα text – κόμβο παιδί, θα επιλεγόταν από το συγκεκριμένο location path.

Κάποια location paths μπορούν να γραφτούν εν συντομία, όπως αυτό φαίνεται στον πίνακα 2.9.

Location Path	**Description**
child::	Αυτό το location path χρησιμοποιείται ως προκαθορισμένο στην περίπτωση που δεν δηλώσουμε κάποιον άξονα και μπορεί εξ' αυτού να παραληφθεί.
attribute::	Ο attribute άξονας μπορεί να γραφεί εν συντομία ως **@**.
/descendant-or-self::node()/	Αυτό το location path μπορούμε να το γράψουμε εν συντομία ως (//).
self::node()	Τον context node μπορούμε να τον γράψουμε εν συντομία ως μία τελεία (**.**).
parent::node()	Τον γονέα του context node μπορούμε να τον γράψουμε εν συντομία ως 2 τελείες (**..**)

Πίνακας 2.9 : Συντομεύσεις για κάποια location paths.

Από τον παραπάνω πίνακα συμπεραίνουμε επομένως πως το location path *Employee* είναι ισοδύναμο με το location path *child::Employee* και θα επιλέξει όλα τα παιδιά που είναι element – κόμβοι του context node. Αν θελήσουμε να επιλέξουμε όλους τους **Employee** element – κόμβους σε ολόκληρο το έγγραφο, μπορούμε να χρησιμοποιήσουμε την συντόμευση *//Employee*, αντί του location path */descendant-or-self::node()/child::Employee*.

Αν θελήσουμε τώρα να βρούμε το επώνυμο του εργαζόμενου, ο οποίος έχει "**ID=1**", θα μπορούσαμε να χρησιμοποιήσουμε το location path : */Employees/Employee[@ID = 1]/LastName*. Αυτό το location path χρησιμοποιεί ένα απλό predicate, με το οποίο συγκρίνουμε την τιμή του attribute **ID**, με την τιμή " **1** ". Το predicate είναι μια boolean έκφραση που χρησιμοποιείται ως τμήμα ενός location path προκειμένου να φιλτράρουμε τους κόμβους από το ψάξιμο. Ανάλογα θα μπορούσαμε να χρησιμοποιήσουμε το location path */Employees/Employee[LastName = 'Pan']/@ID*, προκειμένου να βρούμε το **ID** του υπαλλήλου του οποίου το επίθετο είναι "**Pan**". Όπως μπορεί κανείς να διαπιστώσει από τα 2 τελευταία location path, μέσα σε ένα predicate μπορούμε να συμπεριλάβουμε όλους τους δυνατούς τελεστές σύγκρισης. Έτσι θα μπορούσαμε κάλλιστα να χρησιμοποιήσουμε τους τελεστές **<**, **<=**, **>**, **>=** και **!=** . Εκτός των τελεστών σύγκρισης μπορούμε ακόμα να συμπεριλάβουμε και τους λογικούς τελεστές σύγκρισης **and** και **or**. Επομένως αν θελήσουμε να βρούμε τον τίτλο εργασίας του εργαζομένου με επίθετο "Pan" και μικρό όνομα "Peter", θα μπορούσαμε να χρησιμοποιήσουμε το εξής location path : *//Employee[LastName = 'Pan' and FirstName = 'Peter']/JobTitle*.

2.3.2 Node-set Operators και Functions

Προηγουμένως παραθέσαμε τον τρόπο με τον οποίο μπορεί κανείς να επιλέξει σύνολα κόμβων από ένα XML έγγραφο χρησιμοποιώντας location paths. Η XPath μας επιτρέπει να χειριστούμε αυτά τα σύνολα κόμβων με τους node-set operators (τελεστές συνόλου κόμβων), προκειμένου να σχηματίσουμε άλλα σύνολα κόμβων. Επιπλέον μας παρέχει ένα σύνολο από συναρτήσεις, με τις οποίες μπορούμε να πραγματοποιήσουμε συγκεκριμένες πράξεις σε ένα σύνολο κόμβων που επιστρέφεται από ένα location path. Η περιγραφή των node-set operator και των συναρτήσεων γίνεται συνοπτικά στους πίνακες 2.10 και 2.11 που ακολουθούν.

Node-set Operators	Description
pipe (|)	Υλοποιεί την ένωση 2 συνόλων από κόμβους
slash (/)	Διαχωρίζει τα location steps.
double – slash (//)	Συντόμευση για το location path */descendant-or-self::node()/*

Πίνακας 2.10: Node-set operators.

Node-set Functions	Description
last()	Επιστρέφει την θέση του τελευταίου από όλους τους επιλεγμένους κόμβους.
position()	Επιστρέφει τον αριθμό θέσης του τρέχων κόμβου στο σύνολο κόμβων που ελέγχουμε.
count (*node-set* **)**	Επιστρέφει τον αριθμό των κόμβων στο σύνολο από κόμβους.

Πίνακας 2.11: Κάποιες από τις node-set functions.

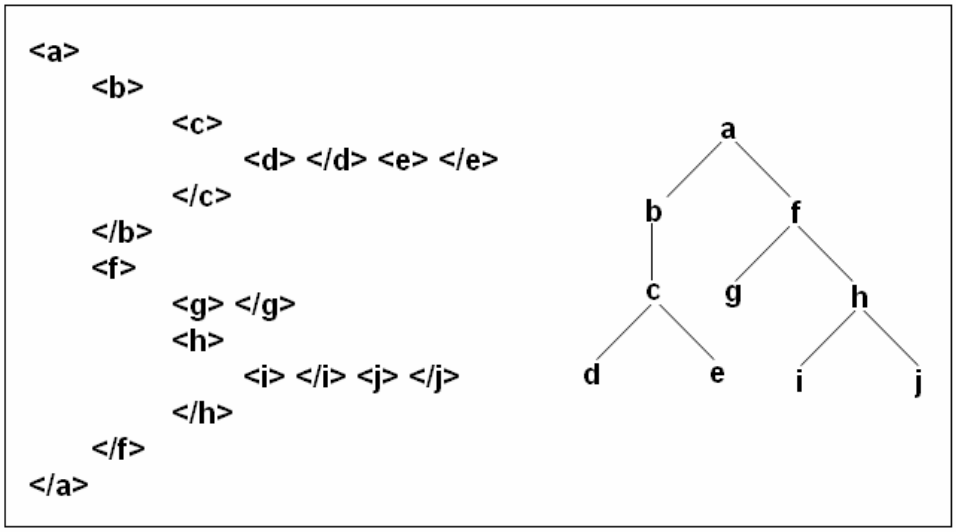
Οι node – set operators και οι συναρτήσεις μπορούν να συνδυαστούν προκειμένου να σχηματίσουμε location path εκφράσεις. Έτσι για παράδειγμα με την έκφραση **FirstName** | **LastName** επιλέγουμε όλους τους **FirstName** και **LastName** element κόμβους που είναι παιδιά του context node. Επιλέγοντας το location path */Employees/Employee[last()]* χρησιμοποιούμε την συνάρτηση **last()** για να μας επιστραφεί ο τελευταίος **Employee** element κόμβος που περιέχεται στον **Employees** element κόμβο. Χρησιμοποιώντας το προηγούμενο location path για το έγγραφο της εικόνας 2.3, θα μας επιστραφεί ο **Employee** με attribute το **ID** = **2**. Το location path *//Employee[position() = 1]* είναι ίδιο με την συντόμευση *//Employee[1]* και μας επιλέγει τον πρώτο **Employee** δηλαδή τον **Employee** με attribute το **ID** = **1**. Τέλος το location path *count(*)* θα μας επιστρέψει τον συνολικό αριθμό των παιδιών του context node που είναι elements.

2.4 XPath Accelerator

Ο *XPath accelerator* [13] είναι μία δομή δεικτοδότησης που σχεδιάστηκε στο πανεπιστήμιο της Konstanz, για να μπορούμε να υποστηρίξουμε τις XPath εκφράσεις. Για την υλοποίηση αυτής της διπλωματικής στηριχτήκαμε στον XPath accelerator, οπότε είναι απαραίτητο να παρουσιάσουμε τις βασικές ιδέες και την θεωρητική προσέγγιση που πραγματοποίησαν οι δημιουργοί του, καθώς και τα θεωρητικά κομμάτια που χρησιμοποιήσαμε για την δική μας υλοποίηση.

2.4.1 XPath άξονες και περιοχές των XML εγγράφων

Στην προηγούμενη παράγραφο είδαμε πώς οι εκφράσεις της XPath εφαρμόζονται σε δέντρα, των οποίων οι κόμβοι αποτελούνται από τα δομικά στοιχεία των XML εγγράφων. Στην εικόνα 2.4 βλέπουμε ένα XML έγγραφο, που θα αποτελέσει και το έγγραφο – αναφορά για το υπόλοιπο του κεφαλαίου, καθώς και την δενδρική του αναπαράσταση. Προς διευκόλυνσή μας, θεωρούμε πως οι κόμβοι του δέντρου είναι τύπου element και αγνοούμε attributes, text nodes κ.τ.λ.



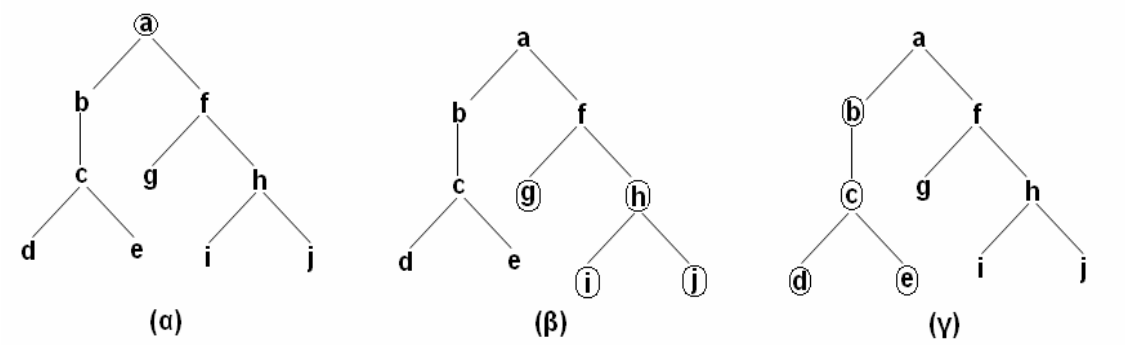
Εικόνα 2.4 : Ένα XML έγγραφο και η δενδρική του αναπαράσταση.

Υπενθυμίζουμε πως οι XPath εκφράσεις ορίζουν την διάσχιση του δέντρου με την χρήση 2 παραμέτρων : (1) ένα context node, ο οποίος αποτελεί και το σημείο εκκίνησης της διάσχισης, (2) και μία αλληλουχία από location steps, τα οποία συντακτικά διαχωρίζονται από / και η εκτίμησή τους γίνεται από αριστερά προς τα δεξιά. Έχοντας ένα context node ως δεδομένο, ένας XPath άξονας (πίνακας 2.7) δημιουργεί ένα υποσύνολο από κόμβους (μία περιοχή στο έγγραφο). Το σύνολο αυτών των κόμβων παρέχει τους context nodes για το επόμενο βήμα.

2.4.1.1 Τμηματοποίηση των XML εγγράφων

Από τους 13 άξονες (πίνακας 2.7) που υποστηρίζει η XPath, τέσσερις είναι οι άξονες πρωταρχικού ενδιαφέροντος. Οι άξονες αυτοί είναι οι : **descendant**, **ancestor**, **following** και **preceding**. Από δω και στο εξής θα καλούμε τους τέσσερις αυτούς άξονες *κύριους* για να τους ξεχωρίζουμε από τους υπόλοιπους. Η εικόνα 2.5

απεικονίζει τα σύνολα κόμβων που προκύπτουν με την χρήση των κύριων αξόνων επιλέγοντας ως context node τον **f** της εικόνας 2.4.



Εικόνα 2.5 : Οι κυκλωμένοι κόμβοι είναι οι κόμβοι που προκύπτουν αν επιλεγούν οι (α) ancestor::*, (β) descendant::* και (γ) preceding::* άξονες αντίστοιχα, με context node τον f.

Για οποιοδήποτε context node n, οι 4 κύριοι άξονες τμηματοποιούν το XML έγγραφο. Το σύνολο κόμβων

n/descendant U n/ancestor U
n/following U n/preceding U {n}

περιέχει κάθε κόμβο του εγγράφου ακριβώς μία φορά. Άλλωστε την ιδιότητα αυτή υλοποιούμε και στην εικόνα 2.5 έχοντας ως context node τον κόμβο f (το βήμα *f/following* δεν επιστρέφει κόμβους, για το συγκεκριμένο έγγραφο).

Η **ιδέα κλειδί** για την υλοποίηση αυτής της εργασίας είναι να προσδιορίσουμε μία δομή με δείκτες τέτοια ώστε, για οποιοδήποτε context node, να μπορούμε με τρόπο αποτελεσματικό να προσδιορίσουμε τα σύνολα των κόμβων στις τέσσερις περιοχές του εγγράφου, όπως αυτές προκύπτουν με την χρήση των κύριων αξόνων.

Οι υπόλοιποι άξονες που υποστηρίζει η XPath (parent, child, descendant-or-self, ancestor-or-self, following-sibling και preceding-sibling) ορίζουν συγκεκριμένα υπερσύνολα ή υποσύνολα αυτών των συνόλων και είναι εύκολο να προσδιοριστούν.

2.4.2 Κωδικοποίηση των περιοχών των XML εγγράφων

Έχουμε πλέον μείνει με την πρόκληση να βρούμε την κατάλληλη κωδικοποίηση για την δενδρικού σχήματος ιεραρχία κόμβων ενός XML εγγράφου, τέτοια ώστε

1. να διατηρεί την έννοια των περιοχών όπως αυτές προσδιορίζονται από τους τέσσερις κύριους XPath άξονες, και

2. να μπορεί να υποστηριχτεί αποτελεσματικά από την ήδη υπάρχουσα τεχνολογία των βάσεων δεδομένων.

Σε αυτό το σημείο θα ήταν χρήσιμο να υπενθυμίσουμε ότι ο preceding άξονας περιλαμβάνει όλους τους κόμβους στο έγγραφο που προηγούνται του context node σε document order, εξαιρώντας τους ancestor, τους attribute και τους namespace κόμβους και ο following άξονας περιλαμβάνει όλους τους κόμβους στο έγγραφο που ακολουθούν τον context node σε document order εξαιρώντας τους ancestor, τους attribute και τους namespace κόμβους.

Όταν λέμε document order εννοούμε την σειρά με την οποία εμφανίζονται οι κόμβοι καθώς διασχίζουμε το έγγραφο από πάνω προς τα κάτω. Ένας πιο σαφής ορισμός του document order, είναι ότι το document order προσδιορίζεται από μια **preorder** (προθεματική) διάσχιση του εγγράφου – δέντρου. Σε μία preorder διάσχιση ενός δέντρου πρώτα επισκεπτόμαστε τον κόμβο – ρίζα και στην συνέχεια τα παιδιά του από αριστερά προς τα δεξιά. Έτσι για το έγγραφο της εικόνας 2.4 το document order θα είναι a < b < c < d < e < f < g < h < i < j.

Σε μία **postorder** (επιθεματική) διάσχιση ενός δέντρου, πρώτα επισκεπτόμαστε τα παιδιά ενός κόμβου από αριστερά προς τα δεξιά και στην συνέχεια επισκεπτόμαστε τον κόμβο – ρίζα. Για το έγγραφο της εικόνας 2.4 η postorder σειρά των κόμβων θα είναι d < e < c < b < g < i < j < h < f < a.

Στον πίνακα 2.12 αναθέτουμε σε κάθε έναν από τους κόμβους του εγγράφου της εικόνας 2.12 δύο χαρακτηριστικές τιμές με τις οποίες μπορούμε να προσδιορίσουμε την θέση του κόμβου μέσα στο δέντρο : τον preorder **pre(n)** και τον postorder **post(n)** βαθμό του κόμβου n, ανάλογα με την διάσχιση που εφαρμόσαμε στο δέντρο.

	pre	**post**
a	0	9
b	1	3
c	2	2
d	3	0
e	4	1
f	5	8
g	6	4
h	7	7

i	8	5
j	9	6

Πίνακας 2.12: Οι preorder και postorder βαθμοί των κόμβων του εγγράφου της εικόνας 2.4

Όπως θα μπορούσε κανείς να παρατηρήσει απ' όλα τα παραπάνω, θα μπορούσαμε να χρησιμοποιήσουμε τα pre(n) και post(n) ώστε να χαρακτηρίσουμε τους descendants n΄ του n. Έχουμε επομένως ότι :

n΄ είναι descendant του n

$\Leftrightarrow$

pre(n) < pre(n΄) και post(n) > post(n΄)

Το παραπάνω μπορεί να διαβαστεί ως εξής : κατά τη διάρκεια ενός σειριακού διαβάσματος του XML εγγράφου, είδαμε το tag που υποδηλώνει άνοιγμα <n> πριν το tag < n΄> και το tag που υποδηλώνει κλείσιμο </n> μετά το </ n΄>. Με άλλα λόγια μπορούμε να πούμε πως το element που αντιστοιχεί στο n΄ περιέχεται στο element που αντιστοιχεί στο n.

Παρόμοια μπορούμε να χρησιμοποιήσουμε τα pre(n) και post(n΄) για να χαρακτηρίσουμε και τους υπόλοιπους 3 από τους *κύριους* άξονες. Αντίστοιχα μπορούμε να διατυπώσουμε τους εξής κανόνες :

1. Οι preceding κόμβοι του context node n έχουν μικρότερο preorder και μικρότερο postorder βαθμό από τον n. Άρα

   n΄ είναι preceding του n

   $\Leftrightarrow$

   pre(n) < pre(n΄) και post(n) < post(n΄)

2. Οι following κόμβοι του context node n έχουν μεγαλύτερο preorder και μεγαλύτερο postorder βαθμό από τον n. Άρα

   n΄ είναι following του n

   $\Leftrightarrow$

   pre(n) > pre(n΄) και post(n) > post(n΄)

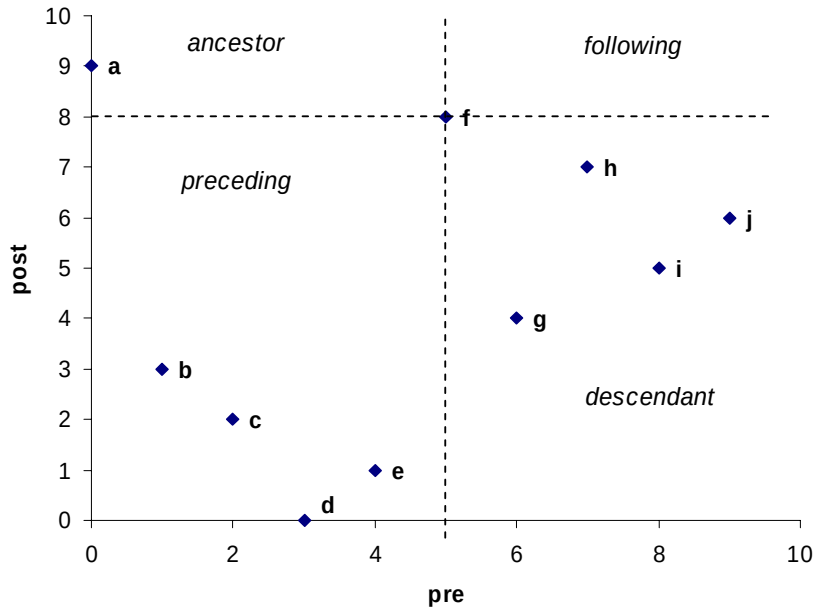
3. Οι ancestor κόμβοι του context node n έχουν μικρότερο preorder και μεγαλύτερο postorder βαθμό από τον n. Άρα

$$n' \text{ είναι ancestor του } n$$

$$\Leftrightarrow$$

$$pre(n) < pre(n') \text{ και } post(n) > post(n')$$

Στην εικόνα 2.6 απεικονίζεται η κατανομή των κόμβων του εγγράφου της εικόνας 2.4 σε ένα διάγραμμα με τις preorder και postorder τιμές τους.



Εικόνα 2.6 : Κατανομή κόμβων σε ένα preorder/postorder διάγραμμα και οι περιοχές του XML εγγράφου όπως αυτές ορίζονται από τον context node f (- -).

Παίρνοντας τον κόμβο **f** ως context node, από το παραπάνω διάγραμμα βλέπουμε πως το XML έγγραφο τμηματοποιείται σε 4 περιοχές :

1. το κάτω και δεξί τμήμα περιλαμβάνει όλους τους descendants του f,
2. στο πάνω και αριστερό τμήμα συναντάμε όλους τους ancestors του f,
3. το κάτω και αριστερό τμήμα φιλοξενεί τους preceding κόμβους του f και τέλος,
4. το πάνω και δεξί τμήμα περιλαμβάνει τους following κόμβους του f (στην προκειμένη περίπτωση το f δεν έχει following κόμβους).

Η τμηματοποίηση του εγγράφου στις 4 περιοχές **ancestor**, **descendant**, **following** και **preceding** υφίσταται για οποιοδήποτε κόμβο και να επιλέξουμε ως context node αντί του **f**. Αυτό σημαίνει πως μπορούμε να επιλέξουμε οποιοδήποτε κόμβο στο pre/post διάγραμμα και να χρησιμοποιήσουμε την τοποθεσία του για να ξεκινήσουμε μία XPath διάσχιση, γεγονός που αποδεικνύεται ως ιδιαίτερα σημαντικό χαρακτηριστικό για την υλοποίηση των XPath ερωτήσεων.

2.4.3 Άξονες και Query Windows

Έχοντας δει στην προηγούμενη παράγραφο πώς μπορούμε να κωδικοποιήσουμε τους 4 *κύριους* άξονες **ancestor**, **descendant**, **following** και **preceding,** αυτό που μας έχει απομείνει είναι να δούμε πως θα υποστηρίξουμε τους υπόλοιπους XPath άξονες και τους έλεγχους ονομάτων.

Με τους άξονες ancestor-or-self και descendant-or-self δεν έχουμε ιδιαίτερο πρόβλημα, καθώς έχοντας ως context node τον n, απλά προσθέτουμε τον κόμβο n στις περιοχές ancestor και descendant αντίστοιχα. Ο κόμβος n αναγνωρίζεται εύκολα, εφόσον ο preorder βαθμός pre(n) είναι μοναδικός. Όσον αφορά τους άξονες following – sibling και preceding – sibling θα χρειαστεί να κρατήσουμε τον preorder βαθμό του πατέρα **par(n)** του κάθε κόμβου n, αφού οι γειτονικοί κόμβοι έχουν όλοι τον ίδιο πατέρα. Εξάλλου το par(n) είναι πληροφορία που θα μας χρειαστεί και για τους άξονες child και parent επίσης. Για την υποστήριξη του attribute άξονα θα χρησιμοποιήσουμε την boolean τιμή **att(n)**, που μας πληροφορεί αν ο κόμβος n περιλαμβάνει ή όχι attributes. Τέλος οι έλεγχοι ονομάτων υλοποιούνται με την χρήση της string τιμής **tag(n)**, στην οποία αποθηκεύουμε το element tag ή το όνομα του attribute.

Έτσι ολοκληρώνουμε την κωδικοποίηση και πλέον ο κάθε κόμβος *n* ενός XML εγγράφου περιγράφεται από ένα διάνυσμα 5 διαστάσεων :

$$desc(n) = [\ pre(n), post(n), par(n), att(n), tag(n)\ ].$$

Σύμφωνα με όσα γράψαμε προηγουμένως, ένας XPath άξονας αντιστοιχεί σ' ένα συγκεκριμένο query window στο διάστημα όπως αυτό ορίζεται από τα διανύσματα 5 διαστάσεων. Στον πίνακα 2.18 παρουσιάζονται τα query windows μαζί με τους αντίστοιχους άξονες που υλοποιούν.

Axis a	Query Window window(a, n)				
	pre	post	par	att	tag
child	(pre(n), ∞)	[0, post(n))	pre(n)	false	*
descendant	(pre(n), ∞)	[0, post(n))	*	false	*
descendant-or-self	[pre(n), ∞)	[0, post(n)]	*	false	*
parent	[par(n), par(n)]	(post(n), ∞)	*	false	*
ancestor	[0, pre(n))	(post(n), ∞)	*	false	*
ancestor-or-self	[0, pre(n)]	[post(n), ∞)	*	false	*
following	(pre(n), ∞)	(post(n), ∞)	*	false	*
preceding	(0, pre(n))	(0, post(n))	*	false	*
following-sibling	(pre(n), ∞)	(post(n), ∞)	par(n)	false	*
preceding-sibling	(0, pre(n))	(0, post(n)]	par(n)	false	*
attribute	(pre(n), ∞)	[0, post(n)]	par(n)	true	*

Πίνακας 2.18: XPath άξονες και τα αντίστοιχα παράθυρα ερωτήσεων window(a, n) (context node n).

Θα θεωρούμε πως ένας κόμβος *n* βρίσκεται μέσα σ' ένα query window αν το διάνυσμα *desc(n)*, που αναπαριστά τον κόμβο, ταιριάζει στοιχείο προς στοιχείο με το query window. Κάποια κελιά του πίνακα περιέχουν την τιμή *. Με τον τρόπο αυτό υποδηλώνουμε ότι η συγκεκριμένη τιμή δεν μας ενδιαφέρει. Για να γίνουμε ακόμα πιο συγκεκριμένοι στον ορισμό των query windows, για ένα κόμβο *n΄* που είναι παιδί του *n* θα ήταν χρήσιμο να ελέγξουμε την συνθήκη *par(n΄) = pre(n)* και έτσι ορίζουμε πως το αντίστοιχο query window θα έχει την τιμή :

*window(child, n) = [*, *, pre(n), false, *]*

Φυσικά εφόσον ο κόμβος *n΄* είναι παιδί του context node *n* έχουμε την επιπρόσθετη πληροφορία ότι περιέχεται στην περιοχή των descendants κι έτσι θα ισχύει ότι :

pre(n) < pre(n΄) και post(n΄) < post(n)

2.4.4 Χρήση πολυδιάστατων δομών με δείκτες

Η θεωρητική προσέγγιση που έγινε νωρίτερα μας οδήγησε στην κωδικοποίηση των κόμβων των XML εγγράφων με την βοήθεια ενός διανύσματος 5 διαστάσεων τον XPath accelerator. Δομές πολλαπλών διαστάσεων όπως είναι σε αυτή την περίπτωση ο XPath accelerator έχει βρεθεί πως υποστηρίζονται αποτελεσματικά με την βοήθεια πολυδιάστατων δομών με δείκτες όπως είναι τα R-trees και τα B-trees. Στο

πανεπιστήμιο της Konstanz η υλοποίηση των εκφράσεων της XPath έγινε με χρήση και των 2 δομών. Τα αποτελέσματα των μετρήσεων που έγιναν για την εκτίμηση των XPath ερωτήσεων, έδειξαν πως τόσο τα R-trees όσο και τα B-trees αποδίδουν αρκετά καλά. Όπως θα δούμε στο επόμενο κεφάλαιο για την δικιά μας υλοποίηση αποφασίσαμε να στηριχτούμε στην χρήση μιας άλλης πολυδιάστατης δομής, το kd-tree.

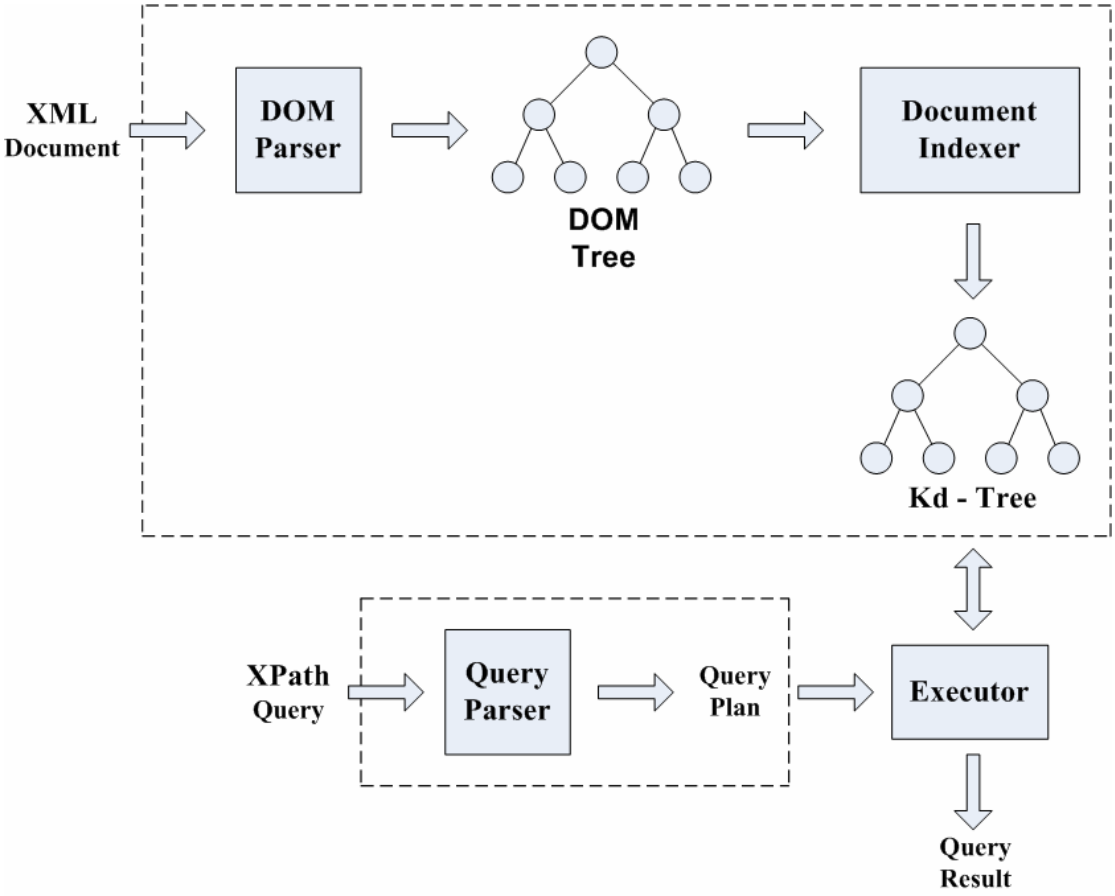
Κεφάλαιο 3

Υλοποίηση του δικού μας Συστήματος

Σε αυτό το κεφάλαιο παρουσιάζουμε τον τρόπο με τον οποίο έγινε η υλοποίηση του δικού μας συστήματος. Εξετάζουμε αναλυτικά την αρχιτεκτονική του συστήματος παρουσιάζοντας και επεξηγώντας με λεπτομέρεια τα διαγράμματα, τις δομές και τους αλγορίθμους, που χρησιμοποιήθηκαν.

3.1 Αρχιτεκτονική του Συστήματος

Προκειμένου να μπορέσουμε να κάνουμε εκτίμηση των XPath queries, παρουσιάζουμε την υλοποίηση του *XPath Query Engine*, η οποία στηρίχτηκε στην υλοποίηση του XPath Accelerator, όπως αυτή παρουσιάστηκε στο προηγούμενο κεφάλαιο. Η αρχιτεκτονική του συστήματός μας παρουσιάζεται στην εικόνα 3.1.



Εικόνα 3.1 : Η αρχιτεκτονική του XPath Query Engine Συστήματος

Το σύστημά μας όπως αυτό παρουσιάζεται στην εικόνα 3.1 αποτελείται ουσιαστικά από 2 τμήματα :

- Στο πρώτο τμήμα του συστήματος παίρνουμε ως είσοδο ένα XML έγγραφο. Το έγγραφο το κάνουμε parsing με ένα DOM Parser, οπότε προκύπτει μία ιεραρχική δενδρική δομή του εγγράφου στην μνήμη, το DOM tree. Από το DOM tree και με την κατάλληλη επεξεργασία, δημιουργούμε μία πολυδιάστατη δενδρική δομή το kd-tree. Χρησιμοποιώντας το kd-tree μπορούμε να κάνουμε πιο γρήγορες και πιο αποτελεσματικές αναζητήσεις δεδομένων μέσα στο XML έγγραφο συγκριτικά με το DOM tree.
- Στο δεύτερο τμήμα του συστήματος παίρνουμε ως είσοδο ένα XPath query. Το XPath query το περνάμε μέσα από ένα query parser, με τον οποίο αναλύουμε και ελέγχουμε λεκτικά και συντακτικά την XPath έκφραση. Έχοντας πιστοποιήσει πως η έκφρασή μας είναι σωστή, τόσο λεκτικά όσο και συντακτικά και σε συνδυασμό με το kd-tree, που έχουμε δημιουργήσει από το πρώτο τμήμα του συστήματος, μπορούμε να κάνουμε εκτίμηση του XPath Query και να παράγουμε το τελικό αποτέλεσμα.

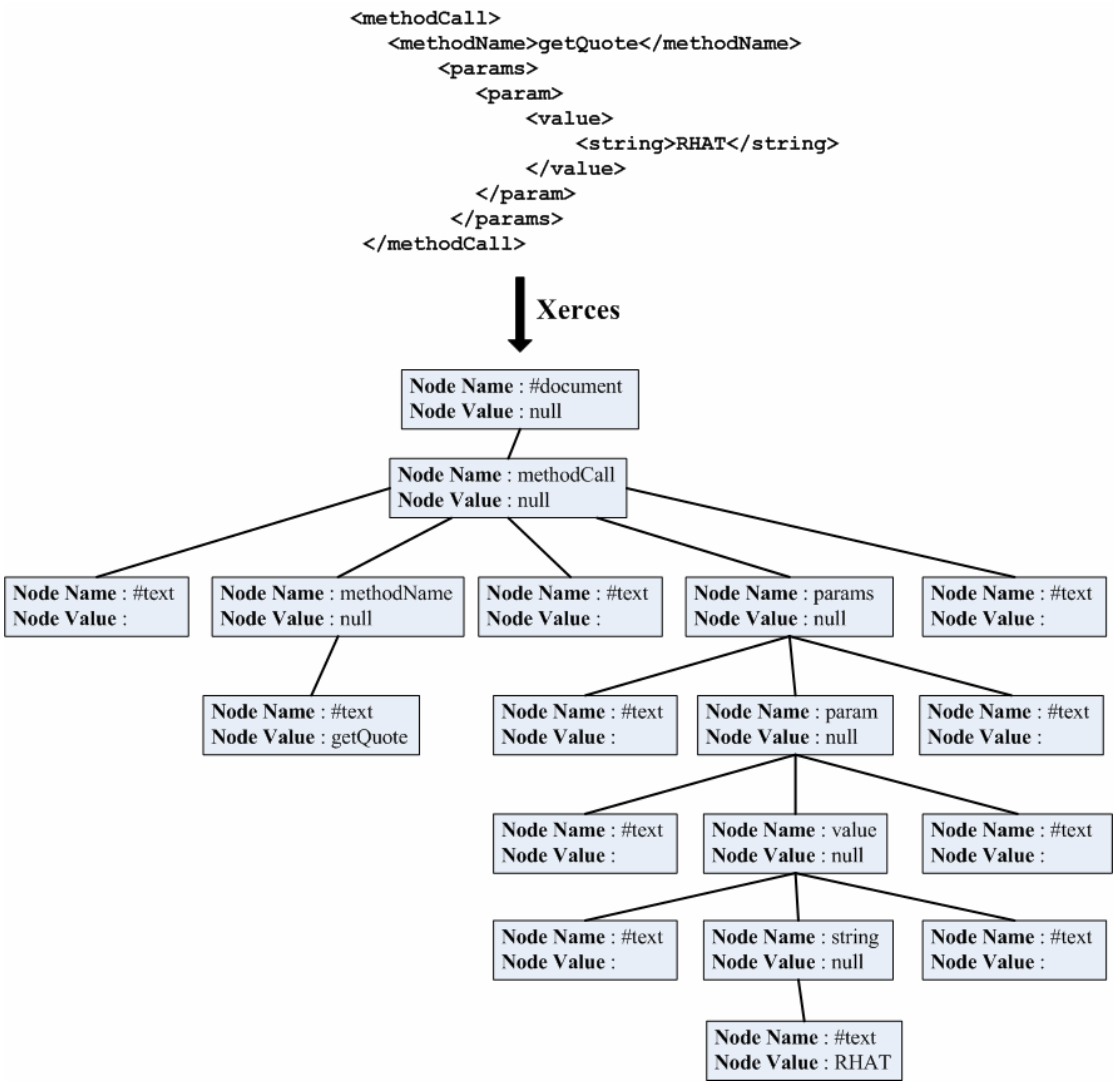
Στις επόμενες παραγράφους που ακολουθούν θα ασχοληθούμε με την περιγραφή του κάθε κομματιού του συστήματος αναλυτικά, καθώς και με την περιγραφή των πιο σημαντικών αλγορίθμων που χρησιμοποιήθηκαν για την υλοποίηση μας.

3.2 DOM Parser και DOM tree

Στο προηγούμενο κεφάλαιο είδαμε πως ένα σημαντικό πλεονέκτημα που έχουν τα XML έγγραφα, είναι η δυνατότητά τους να αναπαρίστανται ως δέντρα. Η δενδρική αναπαράσταση των XML εγγράφων μας παρέχει την δυνατότητα να αξιοποιήσουμε μια ποικιλία από αλγορίθμους που εφαρμόζονται σε δέντρα, προκειμένου να κάνουμε διασχίσεις και να εντοπίσουμε συγκεκριμένα τμήματα των εγγράφων που μας ενδιαφέρουν.

Ένας XML parser παίρνει ένα XML έγγραφο ως είσοδο και δημιουργεί την δενδρική αναπαράσταση του εγγράφου στην μνήμη. Η διάσχιση και η επεξεργασία του δέντρου γίνεται με τις μεθόδους οι οποίες είναι ορισμένες στο API του parser που χρησιμοποιούμε. Για την υλοποίηση του δικού μας συστήματος αποφασίσαμε να χρησιμοποιήσουμε το DOM API. Για την ενεργοποίηση του DOM API χρειάστηκε

να επιλέξουμε ένα parser. Ο parser που επιλέξαμε για αυτή την υλοποίηση είναι ο *Xerces* [12]. Ο Xerces 1.4.4 που χρησιμοποιήσαμε, μας δίνει την δυνατότητα να υποστηρίξουμε τα DOM level 1 και SAX version 1 API's. Στην εικόνα 3.2 βλέπουμε ένα XML έγγραφο, το οποίο θα χρησιμοποιήσουμε ως έγγραφο αναφοράς για το υπόλοιπο του κεφαλαίου, και την αναπαράστασή του ως DOM tree.



Εικόνα 3.2 : Η μετατροπή ενός XML εγγράφου σε DOM tree με την χρήση του parser Xerces.

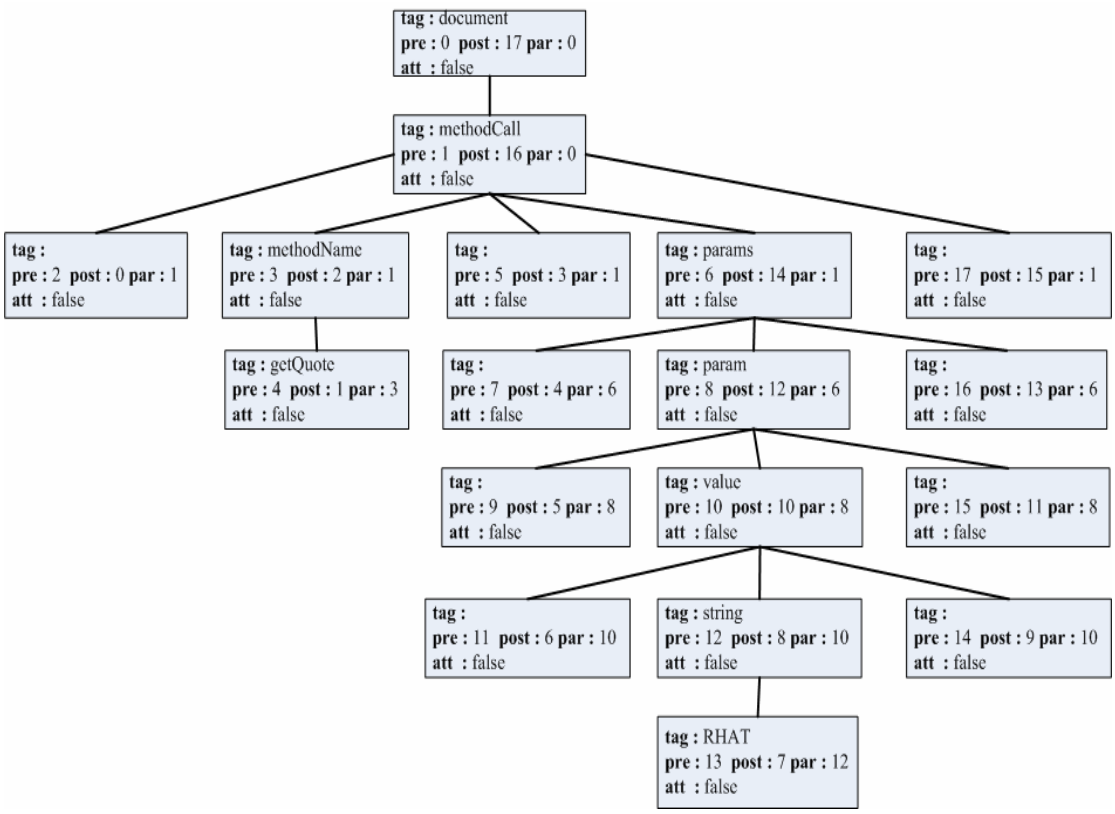
Για την ανάκτηση των τιμών **Node Name** και **Node Value,** οι οποίες ορίζονται για κάθε κόμβο του δέντρου της παραπάνω εικόνας, χρησιμοποιήσαμε 2 μεθόδους από το DOM API, όπως αυτό ενεργοποιείται μετά από το parsing του εγγράφου με τον parser Xerces. Οι μέθοδοι αυτές είναι οι **getNodeName()** και **getNodeValue()** και περιγράφηκαν στο δεύτερο κεφάλαιο ως κομμάτι των μεθόδων του **Node** (Πίνακας 2.4).

Αξιοσημείωτο στο παραπάνω διάγραμμα είναι πως κάποιοι text κόμβοι έχουν ως Node Value το κενό διάστημα, σε αντίθεση με τους υπόλοιπους text κόμβους που έχουν ως Node Value κάποια συγκεκριμένη τιμή, όπως συμβαίνει με τους text κόμβους που έχουν Node Value τις τιμές *"getQuote"* και *"RHAT"*. Το κενό διάστημα περιλαμβάνεται στους text κόμβους, ακόμα κι αν αυτό ουσιαστικά δεν μας παρέχει κάποια πληροφορία και κατά τα άλλα θα πρέπει να το αγνοούμε.

3.3 Document Indexer

Έχοντας αποθηκευμένο στην μνήμη το DOM tree, το οποίο προκύπτει από το XML έγγραφο που χρησιμοποιούμε ως είσοδο (Εικόνα 3.2), το επόμενο βήμα στην υλοποίηση μας είναι να εκμεταλλευτούμε την κωδικοποίηση των περιοχών των XML εγγράφων, όπως αυτή περιγράφηκε στο προηγούμενο κεφάλαιο στις παραγράφους 2.4.1.1, 2.4.2 και 2.4.3.

Υπενθυμίζουμε πως ως βασικό στόχο μας σε αυτό το βήμα, έχουμε την δημιουργία του πενταδιάστατου διανύσματος *desc(n) = [pre(n), post(n), par(n), att(n), tag(n)]* (παράγραφος 2.4.3), για κάθε έναν από τους κόμβους του DOM tree. Στην εικόνα 3.3 παρουσιάζουμε τους κόμβους του DOM tree της εικόνας 3.2 με την αντίστοιχη κωδικοποίησή τους ως πενταδιάστατα διανύσματα.



Εικόνα 3.3 : Αναπαράσταση των κόμβων του DOM tree της εικόνας 3.2 με την αντίστοιχη κωδικοποίηση τους ως πενταδιάστατα διανύσματα.

Το pre(n) είναι ο preorder βαθμός του κόμβου n, το post(n) είναι ο postorder βαθμός του κόμβου n, το par(n) είναι ο preorder βαθμός του πατέρα του κόμβου n, το att(n) είναι μια boolean τιμή που μας πληροφορεί αν ο κόμβος n έχει ή δεν έχει attributes και το tag(n) αφορά το όνομα του κόμβου n.

Με την συγκεκριμένη κωδικοποίηση των κόμβων του DOM tree, οι αναζητήσεις μας για την ανάκτηση πληροφορίας μέσα από XML έγγραφα, γίνονται ιδιαίτερα αποτελεσματικές. Για να μπορέσουμε όμως να εκμεταλλευτούμε την κωδικοποίηση των κόμβων και να προχωρήσουμε στο επόμενο βήμα που είναι η δημιουργία ενός kd-tree, παράλληλα με την διαδικασία της κωδικοποίησης των κόμβων δημιουργούμε μία λίστα, η οποία κρατάει τους κωδικοποιημένους κόμβους μόλις αυτοί δημιουργούνται. Στην επόμενη παράγραφο παρουσιάζουμε τον βασικό αλγόριθμο πάνω στον οποίο στηριχτήκαμε για την κωδικοποίηση των κόμβων του DOM tree και την δημιουργία της λίστας με τους κωδικοποιημένους κόμβους.

3.3.1 Αλγόριθμος δημιουργίας λίστας με κωδικοποιημένους κόμβους

Για την κωδικοποίηση των κόμβων του DOM tree χρειαζόμαστε τον preorder και τον postorder βαθμό του κάθε κόμβου. Μία απλοϊκή προσέγγιση που μπορούμε να χρησιμοποιήσουμε για να υπολογίσουμε τους 2 βαθμούς, είναι να κάνουμε αρχικά μία preorder διάσχιση του δέντρου και να καταχωρήσουμε τους preorder βαθμούς των κόμβων και στην συνέχεια να κάνουμε μία postorder διάσχιση του δέντρου και να καταχωρήσουμε τους postorder βαθμούς των κόμβων. Μία τέτοια προσέγγιση όμως, όπως γίνεται εύκολα αντιληπτό δεν είναι ιδιαίτερα αποτελεσματική καθώς απαιτεί 2 διασχίσεις του δέντρου και αποδεικνύεται ιδιαίτερα χρονοβόρα, ιδιαίτερα όταν έχουμε να ασχοληθούμε με μεγάλα XML έγγραφα. Πρέπει επομένως με μία διάσχιση να υπολογίζουμε και τους 2 βαθμούς για κάθε κόμβο.

Ο αλγόριθμος που χρησιμοποιήσαμε για τον υπολογισμό των preorder και postorder βαθμών των κόμβων με μία διάσχιση, χρησιμοποιεί ως βοηθητική δομή μια στοίβα. Η στοίβα θα περιέχει ζευγάρια τιμών (n, p) όπου n είναι ο κόμβος του δέντρου και p είναι ένας "pop – μετρητής" και κρατάει πόσες φορές έγινε ο κόμβος pop από τη στοίβα. Οι τιμές που μπορεί να πάρει ο p είναι 0,1,2 ως τον μέγιστο αριθμό των παιδιών που διαθέτει ο κόμβος. Στην εικόνα 3.4 παρουσιάζουμε τα βήματα του αλγορίθμου.

1. Βρίσκουμε τον μέσο m ως προς τις preorder τιμές αν το βάθος είναι άρτιο ή ως προς τις postorder τιμές αν το βάθος είναι περιττό και δημιουργούμε την ρίζα.

2. Όσο η στοίβα δεν είναι άδεια κάνε

2.0 Κάνουμε pop από την στοίβα, και παίρνουμε (n, p).

2.1 Αν p = 0, κάνουμε push (n, p + 1) **στη στοίβα και μετά κάνουμε push στη στοίβα (αν υπάρχει) το πιο αριστερό παιδί του κόμβου n** (first child of n, 0).

2.2 Αν p < μέγιστου αριθμού των παιδιών του n, κάνουμε push (n, p + 1) **στη στοίβα και μετά κάνουμε push στη στοίβα το p σε σειρά παιδί του κόμβου n** (p child of n, 0).

2.3 Αν p = μέγιστο αριθμό των παιδιών του n, βγάζουμε τον κόμβο n από την στοίβα.

Εικόνα 3.4 : Αλγόριθμος για τον υπολογισμό των preorder και postorder βαθμών των κόμβων του DOM tree.

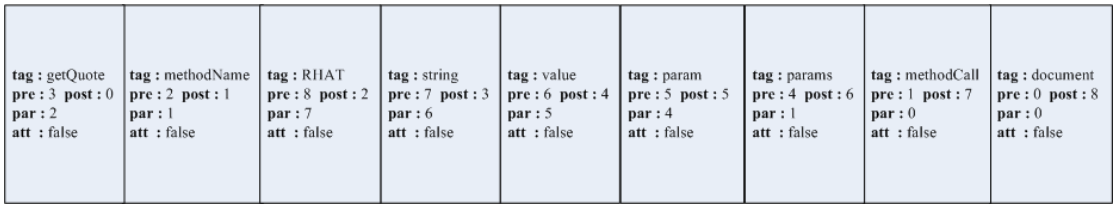
Ο υπολογισμός της preorder τιμής ενός κόμβου γίνεται στο βήμα 2.0 του παραπάνω αλγορίθμου, όταν ο κόμβος γίνεται pop για πρώτη φορά, ενώ ο υπολογισμός της postorder τιμής του κόμβου γίνεται στο βήμα 2.3 του παραπάνω αλγορίθμου, όταν ουσιαστικά ο κόμβος απομακρύνεται από την στοίβα. Όταν ο κόμβος απομακρυνθεί από την στοίβα, η κωδικοποίηση του ως πενταδιάστατο διάνυσμα έχει τελικά ολοκληρωθεί, αφού η preorder και η postorder τιμή του κόμβου είναι πλέον γνωστές και τα υπόλοιπα 3 πεδία του διανύσματος, δηλαδή οι τιμές *par(n), att(n)* και *tag(n)*, έχουν ήδη υπολογιστεί κατά τη διάρκεια του αλγορίθμου. Συνεπώς όταν στο βήμα 2.3 απομακρύνουμε τον κόμβο από την στοίβα, το πενταδιάστατο διάνυσμα που αναπαριστά τον συγκεκριμένο κόμβο, το καταχωρούμε σε μία λίστα, στην οποία κρατάμε τις κωδικοποιημένες πενταδιάστατες αναπαραστάσεις των κόμβων του DOM tree και η οποία θα χρησιμοποιηθεί στην συνέχεια για την δημιουργία του kd-tree. Όπως γίνεται εύκολα αντιληπτό, εφόσον η κωδικοποίηση ολοκληρώνεται στο βήμα 2.3 του αλγορίθμου, το βήμα κατά το οποίο υπολογίζουμε και την postorder τιμή του κόμβου, η σειρά με την οποία γίνονται οι καταχωρήσεις στην λίστα ακολουθεί την postorder σειρά διάσχισης του DOM tree.

3.3.2 Διαγραφή των text κόμβων με τα κενά διαστήματα

Στο DOM tree της εικόνας 3.2 είδαμε πως κάποιοι text κόμβοι περιλαμβάνουν το κενό διάστημα. Από τους συνολικά 11 text κόμβους που περιέχονται στο XML έγγραφο της ίδιας εικόνας, παρατηρούμε πως μόνο οι 2 κόμβοι περιέχουν ουσιαστική πληροφορία κειμένου (text κόμβοι με Node Value *getQuote* και *RHAT*), ενώ οι υπόλοιποι 9 περιλαμβάνουν το κενό διάστημα. Δεδομένου ότι το έγγραφο περιέχει συνολικά 18 κόμβους, το γεγονός πως οι μισοί κόμβοι δεν περιλαμβάνουν αξιοποιήσιμη πληροφορία, όπως αντιλαμβανόμαστε, θα επιβαρύνει ιδιαίτερα την απόδοση του συστήματος, αφού κατά τις αναζητήσεις μας αργότερα θα χρειαστεί να ψάχνουμε και σε κόμβους που δεν μας παρέχουν καμία πληροφορία. Το πρόβλημα γίνεται μεγαλύτερο, ιδίως όταν θα έχουμε να κάνουμε αναζητήσεις σε πολύ μεγαλύτερα XML έγγραφα, που περιέχουν πολύ περισσότερους κόμβους.

Για τον λόγο αυτό κατά τη δημιουργία της λίστας των πενταδιάστατων διανυσμάτων που περιγράφουν τους κόμβους του DOM tree, όταν διασχίζουμε το DOM tree, ελέγχουμε αν ένας κόμβος είναι τύπου text και αν είναι τύπου text ελέγχουμε αν περιλαμβάνει το κενό διάστημα. Στην περίπτωση που ένας text κόμβος

περιλαμβάνει το κενό διάστημα, παραλείπουμε την κωδικοποίησή του ως πενταδιάστατο διάνυσμα και προχωράμε στον επόμενο κόμβο του δέντρου επαναλαμβάνοντας τον έλεγχο. Στην εικόνα 3.5 βλέπουμε την λίστα των πενταδιάστατων διανυσμάτων που θα προκύψει τελικά από το DOM tree της εικόνας 3.2.



Εικόνα 3.5 : Η λίστα με τους κωδικοποιημένους κόμβους όπως αυτή προκύπτει από το DOM tree της εικόνας 3.2.

3.4 Kd-tree

Στην προηγούμενη παράγραφο περιγράψαμε τον τρόπο υλοποίησης μιας λίστας από διανύσματα 5 διαστάσεων, με τα οποία κωδικοποιούμε τους κόμβους του DOM tree. Σύμφωνα με την υλοποίηση του XPath accelerator, την κωδικοποίηση των κόμβων του DOM tree, ως διανύσματα 5 διαστάσεων, μπορούμε να την εκμεταλλευτούμε για τις αναζητήσεις μας αποτελεσματικά, σε περιπτώσεις που το σύστημά μας υποστηρίζει δομές όπως είναι τα R-trees και τα B-trees. Για την δική μας υλοποίηση αποφασίσαμε να χρησιμοποιήσουμε τα kd-trees.

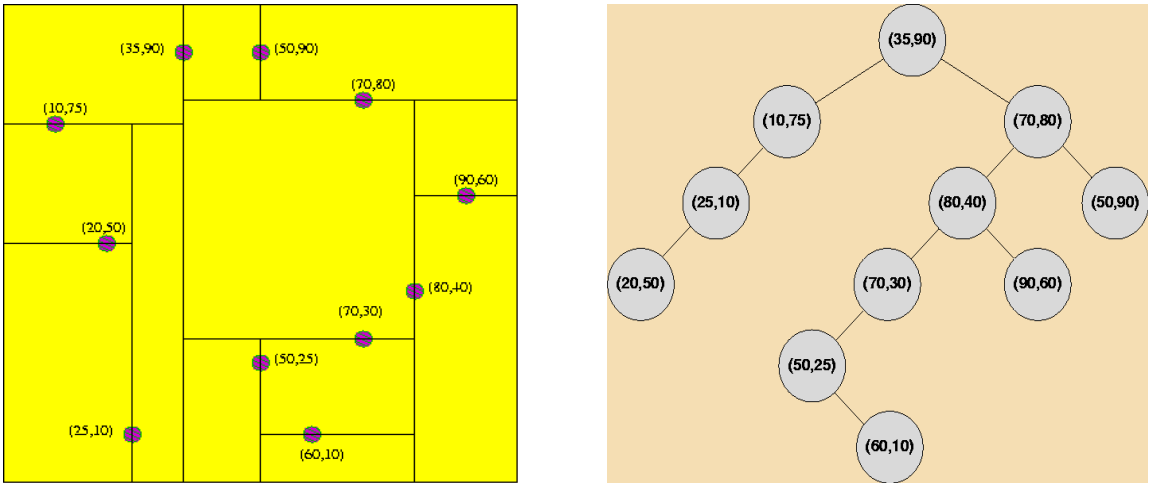
3.4.1 Εισαγωγή στα kd-trees

Ένα kd-tree [22, 23, 24, 25] είναι ένα δυαδικό δέντρο και σχεδιάστηκε για να μπορούμε να χειριζόμαστε πολυδιάστατα δεδομένα με απλό τρόπο. Το kd-tree χρησιμοποιείται για την αποθήκευση ενός πεπερασμένου συνόλου σημείων πολλών διαστάσεων. Η διάσταση των δεδομένων είναι κάθε φορά k, έτσι αν έχουμε σημεία 2 διαστάσεων θα έχουμε 2d-tree, με 3 διαστάσεις 3d-tree κ.τ.λ. Η βασική ιδέα κατασκευής ενός kd-tree, του οποίου τα σημεία είναι 2 διαστάσεων (x, y), είναι η εξής :

- Σε κάθε βήμα επιλέγουμε μία από τις συντεταγμένες των σημείων ως βάση για να διαιρέσουμε τα υπόλοιπα σημεία.

- Για παράδειγμα για την ρίζα του δέντρου επιλέγουμε την x συντεταγμένη ως βάση.
    - Εφόσον το kd-tree είναι στην ουσία ένα δυαδικό δέντρο, όλοι οι κόμβοι από τ' αριστερά της ρίζας θα έχουν την x συντεταγμένη μικρότερη από αυτή της ρίζας .
    - Όλοι οι κόμβοι από τα δεξιά της ρίζας θα έχουν την x συντεταγμένη μεγαλύτερη από αυτή της ρίζας.
- Επιλέγουμε την y συντεταγμένη ως βάση για την διάκριση των παιδιών της ρίζας.
- Επιλέγουμε ξανά την x συντεταγμένη για τα εγγόνια της ρίζας κ.ο.κ.

Έστω ότι έχουμε τις εξής συντεταγμένες : (35, 90), (70, 80), (10, 75), (80, 40), (50, 90), (70, 30), (90, 60), (50, 25), (25, 10), (20, 50) και (60, 10). Έχοντας ως δεδομένο τις παραπάνω συντεταγμένες κατασκευάζουμε το kd-tree που φαίνεται στην εικόνα 3.6 β).



Εικόνα 3.6 : α) Ο τρόπος με τον οποίο το kd-tree της διπλανής εικόνα τέμνει το x,y επίπεδο και β) Το kd-tree που προκύπτει από τις συντεταγμένες της εικόνας α).

Στην εικόνα 3.6 α), βλέπουμε τον τρόπο με τον οποίο οι εισαγωγές στο kd-tree τέμνουν το x, y επίπεδο.

3.4.2 Ισοζυγισμένα kd-trees

Το πρόβλημα με το kd-tree της εικόνας 3.6 β) είναι πως δεν είναι ισοζυγισμένο. Οι αναζητήσεις μας στο δέντρο θα είναι πιο γρήγορες όταν το δέντρο είναι

ισοζυγισμένο, δηλαδή όταν το ύψος των αριστερών και των δεξιών υποδέντρων είναι το ίδιο. Η απόδοση όταν το δέντρο μας είναι ισοζυγισμένο θα είναι πάντα O(logN), ενώ στην περίπτωση που το δέντρο μας δεν είναι ισοζυγισμένο η απόδοση θα είναι O(N), στην χειρότερη των περιπτώσεων.

Για να έχουμε επομένως καλύτερη απόδοση στην υλοποίηση του συστήματός μας, κατασκευάζουμε ένα ισοζυγισμένο kd-tree από την λίστα με τα διανύσματα 5 διαστάσεων, χρησιμοποιώντας ως σημεία εισαγωγής τις τιμές preorder και postorder των διανυσμάτων. Για την κατασκευή ενός ισοζυγισμένου kd-tree στηριχτήκαμε στον αναδρομικό αλγόριθμο της εικόνας 3.7.

1. Βρίσκουμε τον μέσο m ως προς τις preorder τιμές αν το βάθος είναι άρτιο ή ως προς τις postorder τιμές αν το βάθος είναι περιττό και δημιουργούμε την ρίζα.

- **Η εύρεση του μέσου χωρίζει την λίστα σε 2 τμήματα :**
  - **Το τμήμα αριστερά του μέσου περιλαμβάνει στοιχεία με preorder τιμές < m (αν το βάθος είναι άρτιο) ή στοιχεία με postorder τιμές < m (αν το βάθος είναι περιττό).**
  - **Το τμήμα δεξιά του μέσου περιλαμβάνει στοιχεία με preorder τιμές > m (αν το βάθος είναι άρτιο) ή στοιχεία με postorder τιμές > m (αν το βάθος είναι περιττό).**

2. Αναδρομικά επιλέγουμε το αριστερό παιδί από τα στοιχεία αριστερά του μέσου και το δεξί παιδί από τα στοιχεία δεξιά του μέσου.

Εικόνα 3.7 : Αλγόριθμος κατασκευής ισοζυγισμένου kd-tree.

Για την εύρεση του μέσου χρησιμοποιούμε έναν αλγόριθμο [26] που θυμίζει τον αλγόριθμο ταξινόμησης **quicksort**. Υπενθυμίζουμε πως ο μέσος αριθμός μιας λίστας n αριθμών, είναι ο n/2, μικρότερος αριθμός σε σειρά στην λίστα μας. Παρουσιάζουμε τον αλγόριθμο εύρεσης του μέσου στην εικόνα 3.8.

1. Επιλέγουμε έναν αριθμό στην τύχη ως μέσο και το καλούμε pivot.

2. Χωρίζουμε την λίστα μας σε 2 τμήματα : το πρώτο τμήμα αποτελείται από αριθμούς <= pivot, ενώ το άλλο τμήμα αποτελείται από αριθμούς > pivot.

3. Αν το τμήμα της λίστας που περιέχει τους μικρότερους από τον μέσο αριθμούς, έχει μέγεθος ίσο με n – 1 τότε τελειώσαμε.

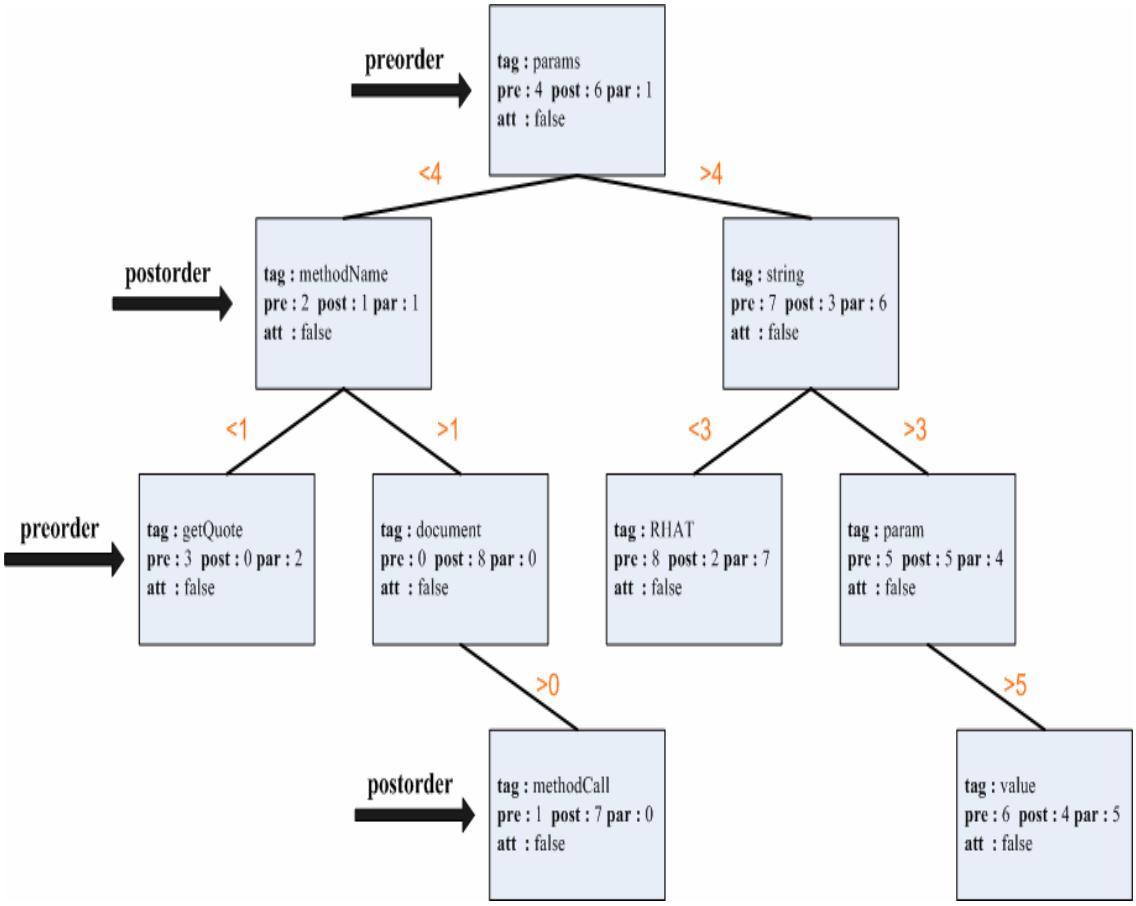
4. Αν όχι, τότε :

α) Αν το μέγεθος του τμήματος της λίστας που περιέχει τους μικρότερους από τον μέσο αριθμούς έχει μέγεθος >= n επαναλαμβάνουμε τα βήματα 1 μέχρι 4 για το συγκεκριμένο τμήμα.

β) Αλλιώς επαναλαμβάνουμε τα βήματα 1 μέχρι 4 για το τμήμα της λίστας με τους αριθμούς μεγαλύτερους του μέσου και n = n – (μέγεθος του τμήματος της λίστας με τους μικρότερους του μέσου αριθμούς) – 1, ως καινούριο n.

Εικόνα 3.8 : Αλγόριθμος εύρεσης του μέσου μιας λίστας αριθμών

Με την χρήση των αλγορίθμων που απεικονίζονται στις εικόνες 3.7 και 3.8 και σε συνδυασμό με την λίστα της εικόνας 3.5, προκύπτει το ισοζυγισμένο kd-tree που απεικονίζεται στην εικόνα 3.9 για το XML έγγραφο της εικόνας 3.2.



Εικόνα 3.9 : Το ισοζυγισμένο kd-tree που προκύπτει από την λίστα της εικόνας 3.5 σε συνδυασμό με την χρήση των αλγορίθμων στις εικόνες 3.7 και 3.8.

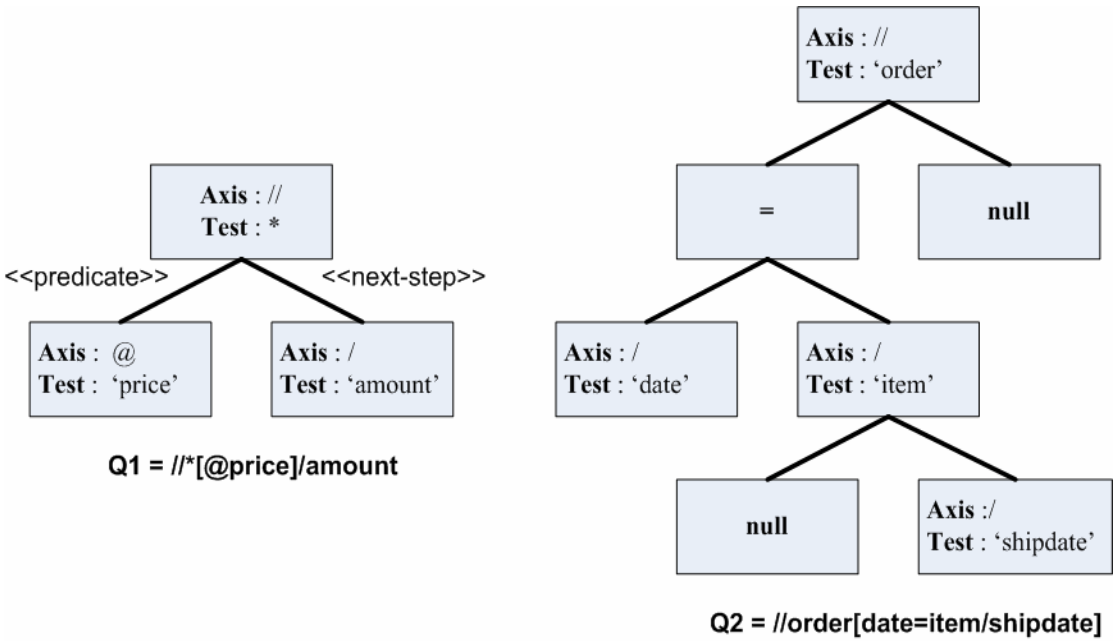
Στο kd-tree της εικόνας 3.9 βλέπουμε πως στο πρώτο επίπεδο επιλογή αριστερού και δεξιού παιδιού γίνεται ως προς την preorder τιμή, στο δεύτερο επίπεδο η επιλογή γίνεται ως προς την postorder τιμή και στο τρίτο επίπεδο η επιλογή γίνεται ξανά ως προς την preorder τιμή. Αν το δέντρο είχε πέμπτο επίπεδο η επιλογή θα γινόταν ως προς την postorder τιμή.

3.5 Query Parser

Όπως ακριβώς χρειάζεται να περάσουμε ένα XML έγγραφο από κάποιον XML parser της επιλογής μας και να δημιουργήσουμε την δενδρική αναπαράσταση του εγγράφου στην μνήμη, ώστε με τον τρόπο αυτό να μπορέσουμε επεξεργαστούμε τα δεδομένα που περιέχει το έγγραφο, για τον ίδιο ακριβώς λόγο χρειαζόμαστε ένα Query Parser, ο οποίος θα παίρνει ως είσοδο μία XPath έκφραση και θα την αναλύει με τέτοιο τρόπο, ώστε σε συνδυασμό με το kd-tree που παρουσιάσαμε στην προηγούμενη παράγραφο, να μπορέσουμε να πάρουμε το τελικό αποτέλεσμα της XPath έκφρασης.

3.5.1 XPS-Trees (XPath Step Trees)

Με την χρήση του parser δημιουργούμε ένα δυαδικό δέντρο, το XPS-tree [27], που αποτελεί την δενδρική αναπαράσταση της XPath έκφρασης που δίνουμε για είσοδο στον parser. Ο κάθε κόμβος στο XPS δέντρο αναπαριστά ένα βήμα στην XPath έκφραση. Όπως έχουμε ήδη δει από το δεύτερο κεφάλαιο, αυτό αποτελείται από ένα άξονα π.χ. descendant, child…, ένα node test και ένα predicate. Τα δύο παιδιά ενός κόμβου είναι τότε ο XPS κόμβος που αναπαριστάνει το επόμενο βήμα και ο XPS κόμβος που αναπαριστάνει το predicate ή null στην περίπτωση που κάποιο από τα προαναφερθέντα δεν υπάρχει. Στην εικόνα 3.10 βλέπουμε την αναπαράσταση δύο XPath εκφράσεων ως XPS-trees.



Εικόνα 3.10 : 2 XPath Queries και η αναπαράστασή τους ως XPS δέντρα

Το XPS-tree, με το οποίο αναπαριστάνουμε το query Q1 στην εικόνα 3.10 αποτελείται από 3 κόμβους. Βλέπουμε πως ένα predicate μπορεί να αποτελέσει ένα XPath βήμα από μόνο του. Γι' αυτό άλλωστε είναι το αριστερό παιδί του κόμβου που αναπαριστάνει το βήμα *//**. Το βήμα */amount* που είναι το επόμενο βήμα στην XPath έκφραση το αναπαριστάνουμε ως το δεξί παιδί του πρώτου βήματος. Το XPS-tree, με το οποίο αναπαριστάνουμε το query Q2 αποτελείται από ένα απλό βήμα (χωρίς predicate) και από ένα predicate. Εφόσον δεν υπάρχει επόμενο βήμα στην XPath έκφραση το δεξί παιδί του πρώτου βήματος θα είναι null. Ως αριστερό παιδί του

πρώτου βήματος χρησιμοποιούμε τον operator '=' που βρίσκεται μέσα στο predicate. Ως αριστερό παιδί του operator επιλέγουμε την αριστερή πλευρά της ισότητας και ως δεξί παιδί την δεξιά πλευρά της ισότητας. Η αριστερή πλευρά της ισότητας είναι ένα απλό βήμα, ενώ η δεξιά πλευρά της ισότητας αποτελείται από 2 απλά βήματα χωρίς predicate και γι' αυτό το αριστερό παιδί του κόμβου που συμβολίζει το βήμα */item* είναι null. Το δεξί παιδί συμβολίζει το επόμενο βήμα και επομένως αναπαριστάνει το βήμα */shipdate*.

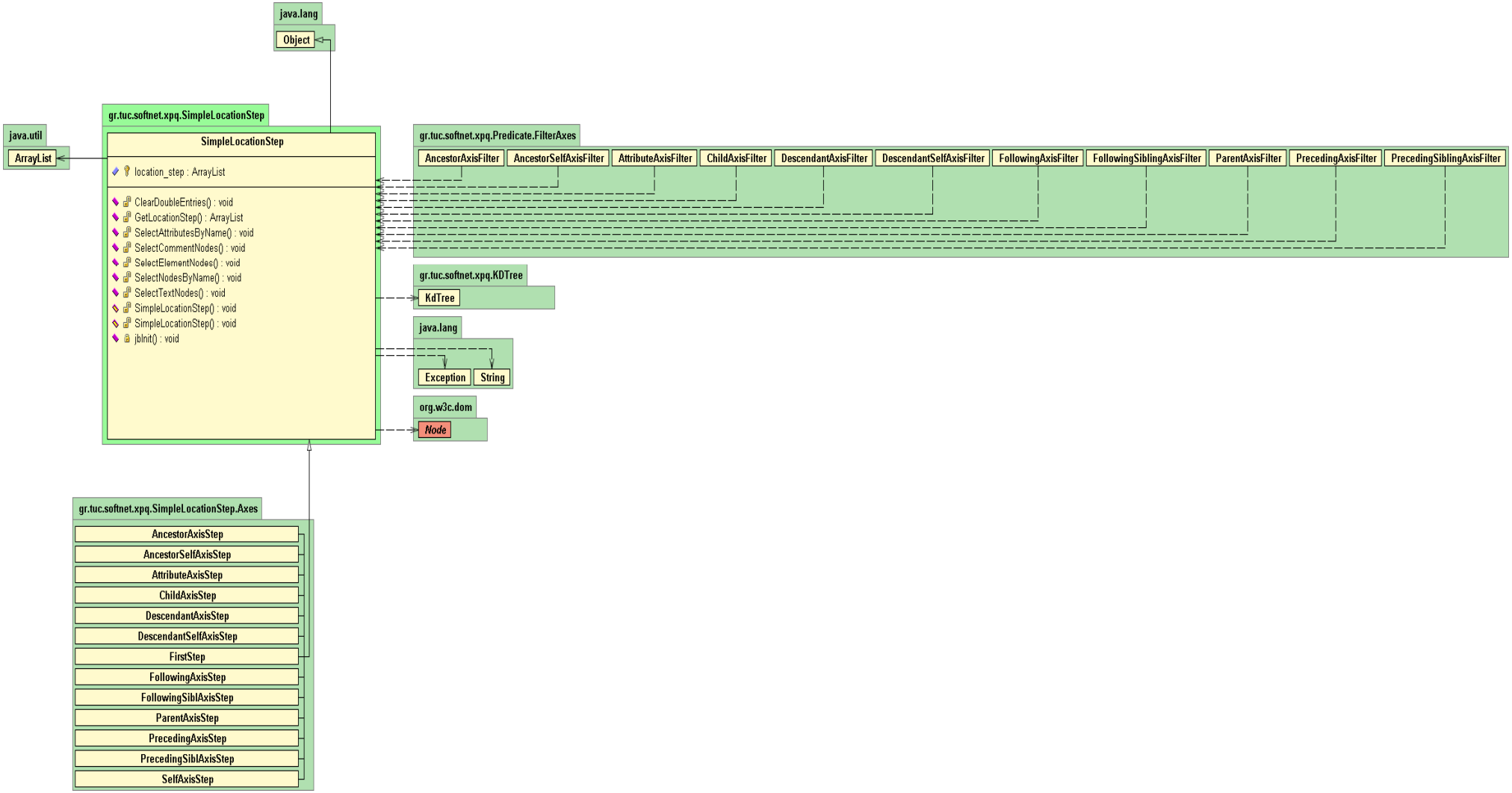
3.6 Executor

Το επόμενο και τελευταίο βήμα στην υλοποίησή μας είναι να μπορέσουμε να χρησιμοποιήσουμε το XPS-tree που παράγεται από τον Query Parser και παρουσιάστηκε στην προηγούμενη παράγραφο, με το kd-tree που δημιουργείται από το πρώτο τμήμα της υλοποίησης μας (Εικόνα 3.1), προκειμένου να παράγουμε το ζητούμενο, που είναι το αποτέλεσμα της XPath έκφρασης. Παρακάτω παραθέτουμε διαγράμματα σε UML που περιγράφουν τις κλάσεις και τις μεθόδους που χρησιμοποιήθηκαν για να μπορέσουμε να συνδέσουμε το kd tree με την ανάλυση της XPath έκφρασης κατά τον χρόνο εκτέλεσης.

3.6.1 Υλοποίηση των απλών XPath εκφράσεων

Στηριζόμενοι στα XPS trees (παράγραφος 3.4.1) και στον τρόπο με τον οποίο αναλύουν τις XPath εκφράσεις, δημιουργήσαμε μια κλάση την *SimpleLocationStep*, με την οποία περιγράφουμε μια απλή XPath έκφραση, χωρίς predicate. Σκοπός μας με την κλάση SimpleLocationStep είναι να υποστηρίξουμε εκφράσεις της μορφής *descendant :: **, εκφράσεις δηλαδή που περιέχουν ένα XPath axis και ένα node test. Το UML διάγραμμα της κλάσης SimpleLocationStep και των υπόλοιπων κλάσεων που χρησιμοποιούμε για την υλοποίηση των απλών XPath εκφράσεων φαίνεται στην εικόνα 3.11, στην επόμενη σελίδα.

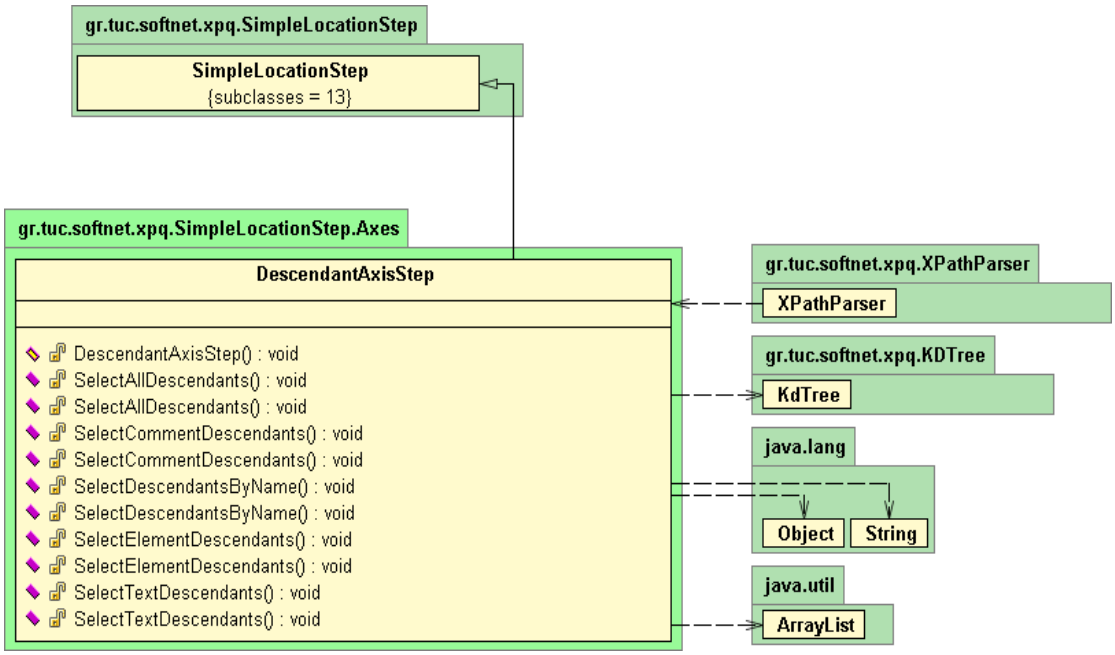
Η κλάση μας, περιλαμβάνει τις μεθόδους SelectCommentNodes(), SelectElementNodes(), SelectNodesByName() και SelectTextNodes() με τις οποίες



Εικόνα 3.11 : UML διάγραμμα της κλάσης SimpleLocationStep που περιγράφει μια XPath έκφραση χωρίς predicate

υλοποιούμε τα node tests. Με την μέθοδο SelectCommentNodes() επιλέγουμε τους κόμβους – σχόλια του XML εγγράφου, με την μέθοδο SelectElementNodes() επιλέγουμε τους element – κόμβους, με την μέθοδο SelectNodesByName() επιλέγουμε element – κόμβους του εγγράφου με συγκεκριμένο όνομα και τέλος με τη μέθοδο SelectTextNodes() επιλέγουμε τους text – κόμβους του εγγράφου.

Την υλοποίηση μιας XPath έκφρασης με συγκεκριμένο XPath axis και κάποιο node test την πραγματοποιούμε με τις κλάσεις που περιέχονται στο πακέτο **gr.tuc.softnet.xpq.SimpleLocationSteps.Axes**. Το πακέτο περιέχει τις εξής κλάσεις : AncestorAxisStep, AncestorSelfAxisStep, AttributeAxisStep, ChildAxisStep, DescendantAxisStep, DescendantSelfAxisStep, FirstStep, FollowingAxisStep, FollowingSiblAxisStep, ParentAxisStep, PrecedingAxisStep, PrecedingSiblAxisStep και SelfAxisStep. Όλες οι κλάσεις του πακέτου κληρονομούν από την κλάση SimpleLocationStep και όπως φανερώνει η ονομασία τους, καθεμία υλοποιεί από μία απλή XPath έκφραση με τον αντίστοιχο XPath axis. Στην εικόνα 3.12 βλέπουμε το UML διάγραμμα της κλάσης DescendantAxisStep. Οι υπόλοιπες κλάσεις του πακέτου είναι παρόμοιες.



Εικόνα 3.12 : UML διάγραμμα της κλάσης DescendantAxisStep, που υλοποιεί μια απλή XPath έκφραση με τον descendant axis.

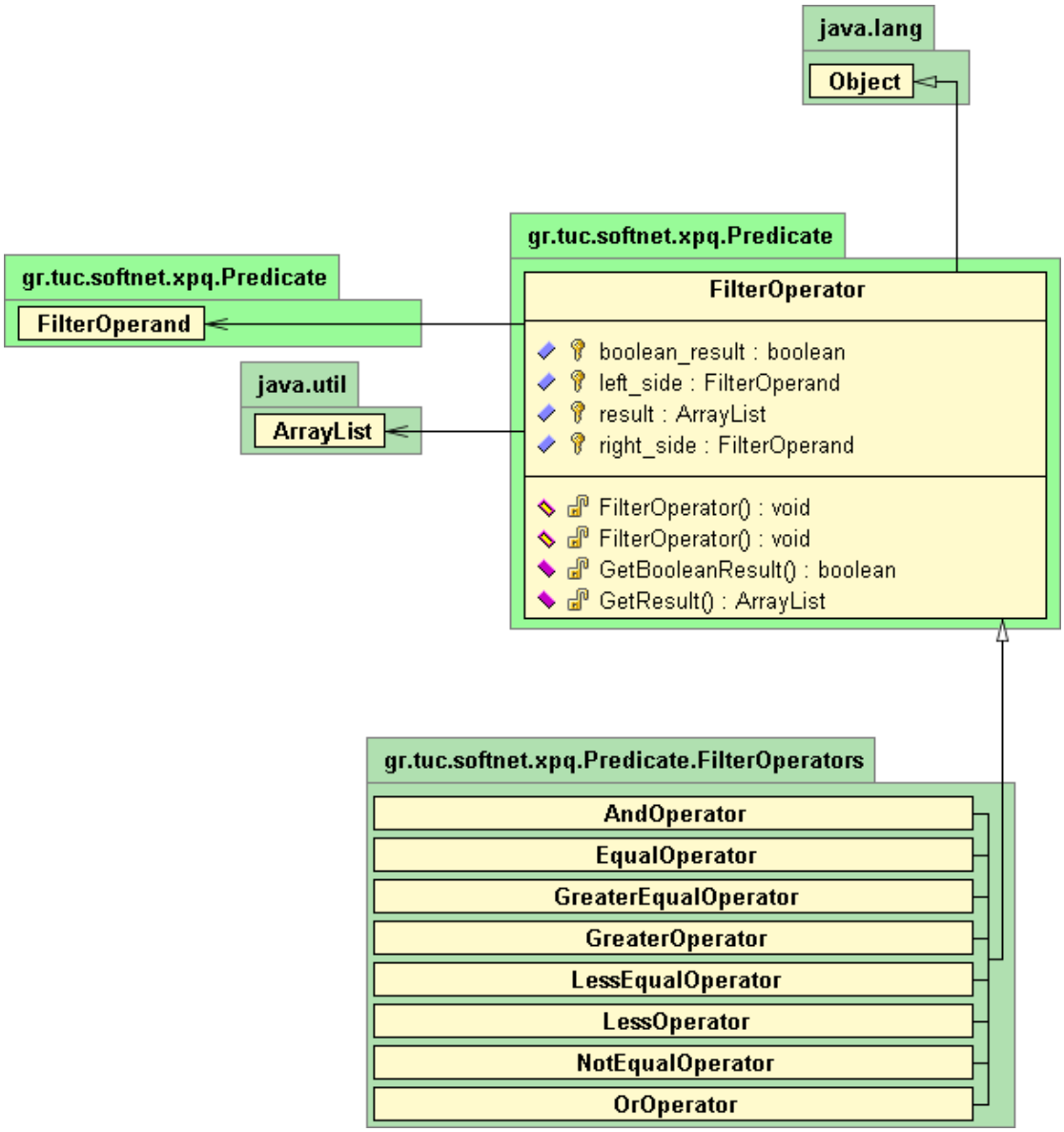
Η κλάση DescendantAxisStep, όπως βλέπουμε από το παραπάνω διάγραμμα είναι τύπου SimpleLocationStep και με τις μεθόδους που έχουμε ορίσει, υλοποιούμε

απλές XPath εκφράσεις που περιλαμβάνουν τον descendant axis. Με την μέθοδο SelectAllDescendants() υποστηρίζουμε την XPath έκφραση *descendant :: node()*, με την μέθοδο SelectCommentDescendants() την έκφραση *descendant :: comment()*, με την μέθοδο SelectDescendantsByName() την έκφραση *descendant :: name*, με την μέθοδο SelectElementDescendants() την έκφραση *descendant :: ** και τέλος με την έκφραση SelectTextDescendants() την έκφραση *descendant :: text()*.

Το πακέτο **gr.tuc.softnet.xpq.Predicate.FilterAxes** στην εικόνα 3.11 περιλαμβάνει κλάσεις που περιγράφουν τις ίδιες XPath εκφράσεις, όπως και οι κλάσεις του πακέτου **gr.tuc.softnet.xpq.SimpleLocationStep** που μόλις περιγράψαμε. Η διαφορά είναι πως οι κλάσεις του πρώτου πακέτου περιγράφουν απλές XPath εκφράσεις μέσα σε predicates δηλαδή εκφράσεις της μορφής : *[descendant :: *]*, ενώ οι κλάσεις στο δεύτερο πακέτο όπως είδαμε περιγράφουν XPath εκφράσεις της μορφής *descendant :: ** χωρίς predicate. Ο διαχωρισμός αυτός γίνεται επειδή στην πρώτη περίπτωση, από τους κόμβους που χρησιμοποιούμε ως είσοδο για το predicate, επιλέγουμε τους κόμβους που έχουν descendant κόμβους που είναι τύπου element, ενώ στην δεύτερη περίπτωση επιλέγουμε τους descendant κόμβους, των κόμβων που χρησιμοποιούμε ως αναφορά.

3.6.2 Υλοποίηση των λογικών τελεστών και των τελεστών σύγκρισης

Μέσα στα predicates πολλές φορές συναντάμε λογικούς τελεστές και τελεστές σύγκρισης. Για την υποστήριξη επομένως εκφράσεων της μορφής : *[@id = 1]* ή *[/FirstName = 'Giorgos' and @id < 4]* , υλοποιήσαμε κλάσεις που υποστηρίζουν τους λογικούς τελεστές **and** και **or** και κλάσεις που υποστηρίζουν τους τελεστές σύγκρισης **<**, **>**, **<=**, **>=**, **=** και **!=**. Στην εικόνα 3.13 βλέπουμε το UML διάγραμμα της κλάσης *FilterOperator*.

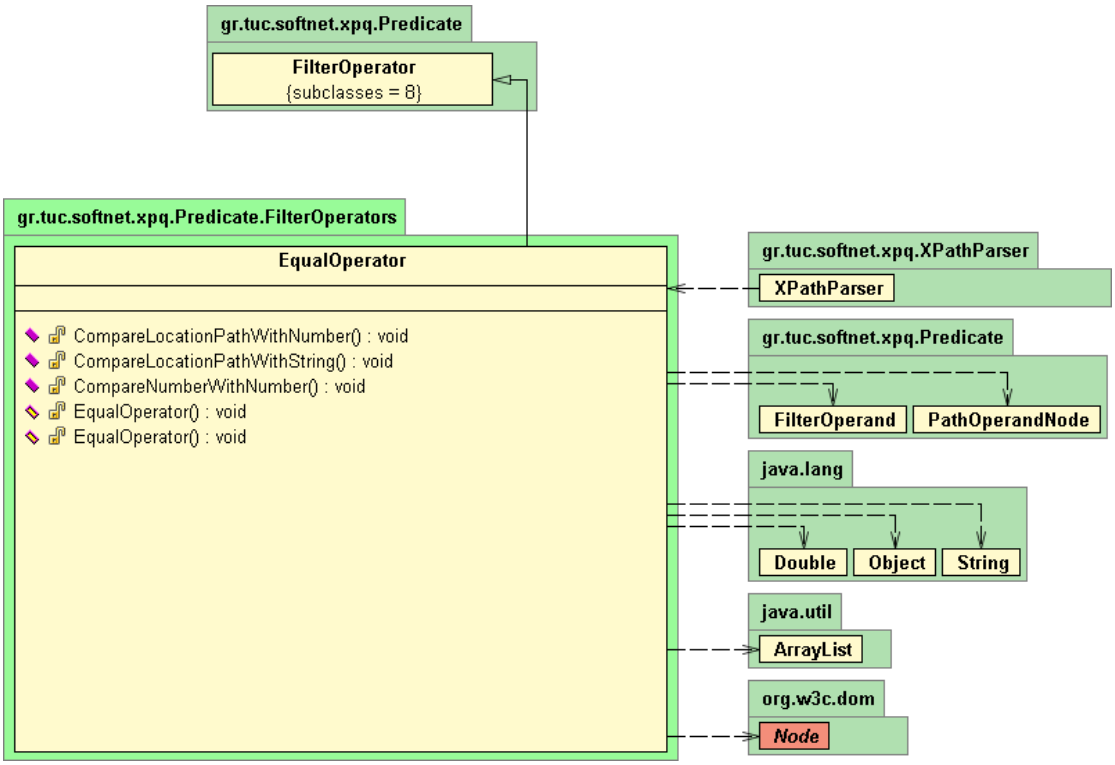


Εικόνα 3.13 : UML διάγραμμα της κλάσης FIlterOperator, που περιγράφει τους λογικούς τελεστές και τους τελεστές σύγκρισης.

Η κλάση FilterOperator δίνει μια γενική περιγραφή των τελεστών σύγκρισης και των λογικών τελεστών. Η κλάση μας περιλαμβάνει 2 objects, το *left_side* και το *right_side*, που είναι τύπου FilterOperand. Η κλάση FilterOperand περιγράφει τον τύπο των τελεστέων που μπορεί κανείς να χρησιμοποιήσει μέσα σε ένα predicate και μπορεί να είναι είτε String, είτε αριθμός, είτε μια απλή XPath έκφραση. Για παράδειγμα το predicate *[/FirstName = 'Giorgos' and @id < 4]* περιλαμβάνει τους εξής τελεστέους : 2 XPath εκφράσεις τις */FirstName* και *@id*, ένα String το '*Giorgos*' και έναν αριθμό το *4*.

Οι κλάσεις στο πακέτο **gr.tuc.softnet.xpq.Predicate.FilterOperators** AndOperator, EqualOperator, GreaterEqualOperator, GreaterOperator,

LessEqualOperator, LessOperator, NotEqualOperator και OrOperator είναι τύπου FilterOperator και όπως τα ονόματα τους φανερώνουν καθεμία υλοποιεί την λειτουργικότητα του τελεστή που υποδηλώνει. Στην εικόνα 3.14 βλέπουμε το UML διάγραμμα της κλάσης *EqualOperator*. Οι υπόλοιπες κλάσεις του πακέτου είναι παρόμοιες.



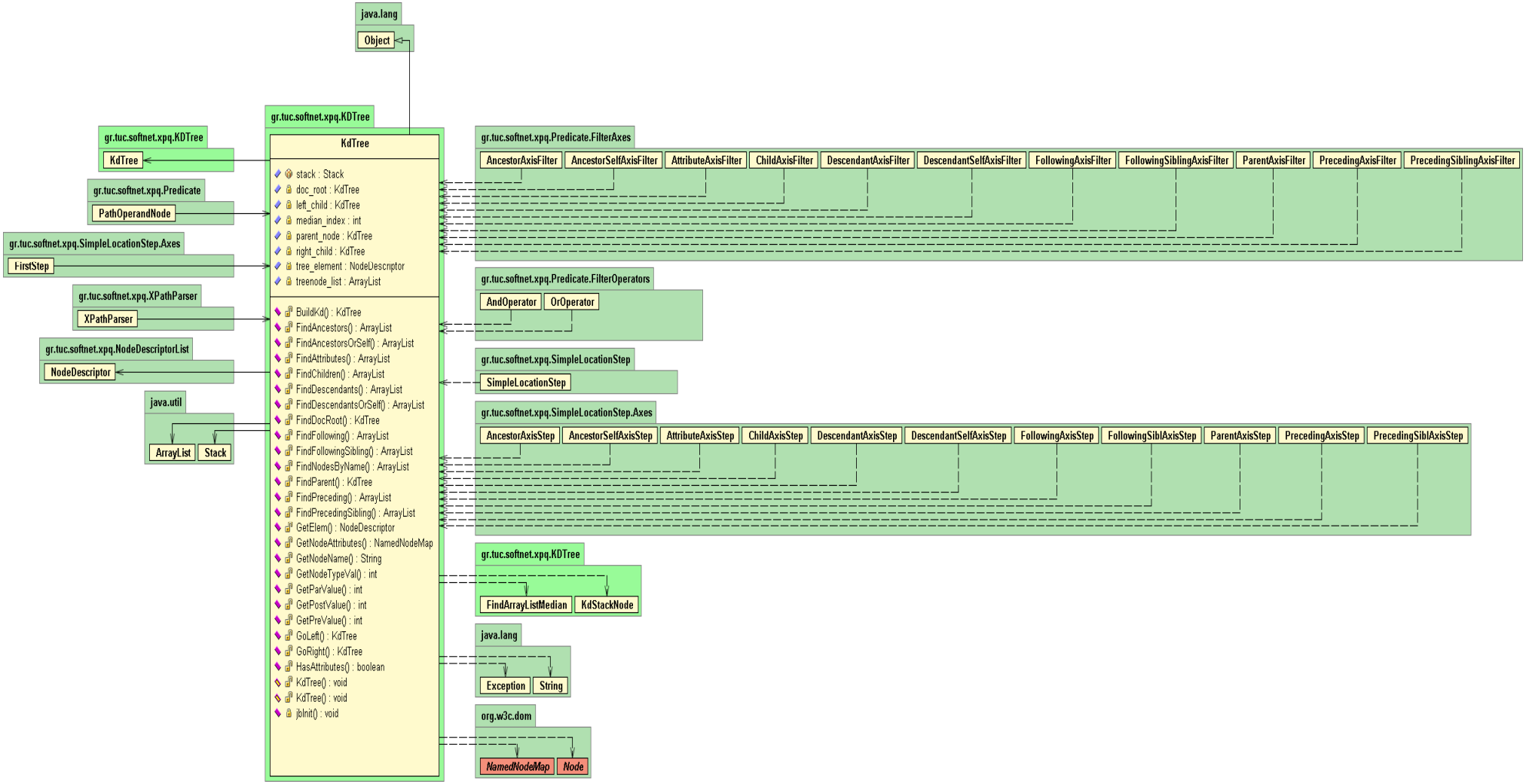
Εικόνα 3.14 : UML διάγραμμα της κλάσης EqualOperator, που περιγράφει την λειτουργικότητα του τελεστή '='.

Η κλάση EqualOperator περιγράφει την λειτουργικότητα του τελεστή ισότητας μέσα σε ένα predicate. Η κλάση κληρονομεί από την κλάση FilterOperator. Με την μέθοδο CompareLocationPathWithNumber() υποστηρίζουμε εκφράσεις της μορφής *[@id = 1]*, δηλαδή μια απλή XPath έκφραση με έναν αριθμό, με την μέθοδο CompareLocationPathWithString() εκφράσεις της μορφής *[/FirstName = 'Giorgos']*, δηλαδή μια απλή XPath έκφραση με ένα String και με την μέθοδο CompareNumberWithNumber() εκφράσεις της μορφής *[2 = 1]*, δηλαδή εξετάζει ως προς την ισότητα 2 αριθμούς. Για την τελευταία περίπτωση αν η έκφραση μέσα στο predicate είναι αληθής επιστρέφουμε τους κόμβους που χρησιμοποιούμε ως είσοδο στο predicate, αλλιώς δεν επιστρέφουμε τίποτα.

3.6.3 Υλοποίηση των αναζητήσεων με το kd-tree

Μέχρι τώρα είδαμε τον τρόπο με τον οποίο γίνεται η ανάλυση των XPath Queries από τον Parser. Παράλληλα όμως με την ανάλυση των XPath εκφράσεων, κατά την οποία δημιουργούνται αντικείμενα των κλάσεων που περιγράψαμε προηγουμένως, τα αντικείμενα που δημιουργούνται και αναπαριστούν τα απλά βήματα από τα οποία αποτελείται η XPath έκφραση, βρίσκονται σε μία διαρκή επικοινωνία με το kd-tree, το οποίο το χρησιμοποιούμε για τον εντοπισμό του συνόλου των κόμβων που αντιστοιχούν στον εκάστοτε XPath axis. Η αναπαράσταση σε UML της κλάσης *KdTree* φαίνεται στην εικόνα 3.15, στην επόμενη σελίδα.

Η κλάση KdTree που φαίνεται στο διάγραμμα περιγράφει την λειτουργικότητα ενός kd tree, έτσι περιέχει τις μεθόδους GoLeft() και GoRight() με τις οποίες επισκεπτόμαστε το αριστερό και το δεξί παιδί ενός κόμβου. Με τις μεθόδους FindAncestors(), FindAncestorsOrSelf(), FindAttributes(), FindChldren(), FindDescendants(), FindDescendantsOrSelf(), FindParent(), FindFollowing(), FindFollowingSibling(), FindPreceding() και FindPrecedingSibling() πραγματοποιούμε τις αναζητήσεις μας μέσα στο δέντρο. Έτσι για παράδειγμα με την μέθοδο FindDescendants() εντοπίζουμε όλους τους κόμβους μέσα στο kd tree που είναι απόγονοι ενός κόμβου, τον οποίο χρησιμοποιούμε ως κόμβο αναφοράς. Οι κλάσεις που περιέχονται στο πακέτο **gr.tuc.softnet.xpq.SimpleLocationStep.Axes** και στο πακέτο **gr.tuc.softnet.xpq.Predicate.FilterAxes** και περιγράφηκαν στις προηγούμενες παραγράφους, καλούν τις παραπάνω μεθόδους για τον εντοπισμό των κόμβων που μας ενδιαφέρουν. Έτσι για παράδειγμα η κλάση *DescendantAxisStep* που περιγράφει τον descendant axis, μέσα στις μεθόδους τις καλεί την μέθοδο FindDescendants() της κλάσης KdTree, για τον εντοπισμό των απογόνων του κόμβου αναφοράς.



Εικόνα 3.15 : UML διάγραμμα τηε κλάσης KdTree, που περιγράφει την λειτουργικότητα ενός kd tree και τις εξαρτήσεις των υπόλοιπων κλάσεων από τις μεθόδους της κλάσης.

Κεφάλαιο 4

Μετρήσεις Απόδοσης του Συστήματος

Σε αυτό το κεφάλαιο κάνουμε εκτίμηση της απόδοσης του συστήματος που υλοποιήσαμε, παρουσιάζοντας τις μετρήσεις που πραγματοποιήσαμε. Για την καλύτερη εκτίμηση της απόδοσης του συστήματος συγκρίνουμε τη δικιά μας υλοποίηση με ένα άλλο εμπορικό σύστημα, το Xindice, που επίσης υποστηρίζει την XPath ως γλώσσα ερωτήσεων.

4.1 Apache Xindice

Για να μπορέσουμε να κάνουμε μια καλή εκτίμηση για την απόδοση του συστήματός μας, επιλέξαμε το σύστημα Xindice version 1.0 της Apache [28], που χρησιμοποιεί την XPath ως γλώσσα ερωτήσεων. Βέβαια τα δύο συστήματα είναι διαφορετικά, αφού το Xindice είναι μία βάση δεδομένων, που έχει σχεδιαστεί για την αποθήκευση XML εγγράφων. Όταν κάνουμε μια XPath ερώτηση, το Xindice ελέγχει όλα τα XML έγγραφα που έχει αποθηκευμένα στην βάση του για να επιστρέψει το αποτέλεσμα, οπότε η απάντηση μπορεί να προέρχεται από παραπάνω από ένα XML έγγραφα. Για να επιτύχουμε ίδιες συνθήκες λειτουργίας για τα 2 συστήματα, εφόσον πρόκειται για διαφορετικές υλοποιήσεις και το δικό μας σύστημα παίρνει ως είσοδο μόνο ένα XML αρχείο, στην βάση δεδομένων του Xindice αποθηκεύαμε κάθε φορά ένα XML έγγραφο, στο οποίο θέταμε τα XPath ερωτήματα.

4.2 Το περιβάλλον των δοκιμών

Οι μετρήσεις για την απόδοση των 2 συστημάτων, του Xindice και της δικιάς μας υλοποίησης, πραγματοποιήθηκαν σε ένα υπολογιστή με επεξεργαστή Athlon XP της AMD στα 2.8 GHz, με μνήμη 512 MB. Το λειτουργικό σύστημα του υπολογιστή ήταν Windows XP και το πρόγραμμα που χρησιμοποιήθηκε για την καταγραφή των μετρήσεων, ήταν ο JBuilder 2005.

4.2.1 XML έγγραφα των μετρήσεων

Για την εκτίμηση της απόδοσης των 2 συστημάτων, χρησιμοποιήσαμε XML έγγραφα που ενεργοποιούνταν με τον XMLgen. Ο XMLgen που σχεδιάστηκε για το XMark benchmark project [29], πρόκειται για ένα εκτελέσιμο πρόγραμμα με το οποίο

μπορούμε να παράγουμε XML αρχεία. Το μέγεθος των XML εγγράφων που παράγονται, το ορίζουμε εμείς, με την χρήση μιας συγκεκριμένης παραμέτρου του προγράμματος. Ο XMLgen μας δίνει επομένως την δυνατότητα να παράγουμε με τρόπο αποτελεσματικό μεγάλα XML αρχεία, που αποτελούν ένα πολύ καλό μέτρο σύγκρισης για την αξιολόγηση συστημάτων που χρησιμοποιούν XML. Στον πίνακα 4.1 παρουσιάζουμε τα χαρακτηριστικά των XML εγγράφων που ενεργοποιήσαμε για τους ελέγχους της απόδοσης των 2 συστημάτων.

Παράγοντας	**Μέγεθος XML εγγράφου**
0	26.5KB
0.001	115KB
0.002	210KB
0.003	318KB
0.005	567KB
0.007	817KB
0.01	1,12MB

Πίνακας 4.1 : Χαρακτηριστικά των XML εγγράφων που χρησιμοποιήσαμε για τους ελέγχους απόδοσης.

Στον παραπάνω πίνακα, παράγοντας είναι η τιμή της μεταβλητής που δίνουμε στον XMLgen, για την ενεργοποίηση XML εγγράφων, των οποίων το μέγεθός αντιστοιχεί στην τιμή του παράγοντα που εισάγουμε.

4.2.2 XPath ερωτήματα των μετρήσεων

Για την επιλογή των XPath ερωτημάτων που χρησιμοποιήθηκαν για τις μετρήσεις της απόδοσης των δύο συστημάτων, στηριχτήκαμε στο XPathMark [30]. Το XPathMark είναι ένα XPath benchmark για το XMark. Αποτελείται από ένα σύνολο από XPath ερωτήματα, τα οποία σχεδιάστηκαν ειδικά για τα XML έγγραφα που ενεργοποιούνται με τον XMLgen του XMark. Τα ερωτήματα που προτείνονται από το XPathMark καλύπτουν ένα ευρύ φάσμα της XPath γλώσσας και αποτελούν πολύ καλό σημείο αναφοράς, για την αξιολόγηση συστημάτων που χρησιμοποιούν την XPath ως γλώσσα αναζήτησης.

Για την αξιολόγηση των δύο συστημάτων επιλέξαμε τα εξής 7 XPath ερωτήματα :

- **Q1 : Όλα τα items.**

  /site/regions//item*

- **Q2 : Τα keywords που βρίσκονται μέσα σε items.**

  /descendant::listitem/descendant::keyword

- **Q3 : Τα items που προέρχονται είτε από την βόρεια είτε από την Νότια Αμερική.**

  /site/regions//item[parent::namerica or parent::samerica]*

- **Q4 : Τα listitems που περιέχουν keyword.**

  //keyword/ancestor::listitem

- **Q5 : Τα initial που προηγούνται σε document order των current.**

  //descendant::current/preceding::initial

- **Q6 : Τα items που ακολουθούν σε document order ένα συγκεκριμένο item.**

  /site/regions//item[@id='item0']/following::item*

- **Q7 : Τα elements για τα οποία ορίστηκε ως attribute το id.**

  //[@id]*

Τα παραπάνω 7 XPath ερωτήματα επιλέχτηκαν ακριβώς όπως αυτά διατυπώνονται στο XPathMark και καλύπτουν διαφορετικούς άξονες, διαφορετικά σε αριθμό βήματα, διαφορετικά node tests και διαφορετικές εκφράσεις μέσα στα predicates. Η επιλογή έγινε έτσι, ώστε να μπορέσουμε να αποκτήσουμε μια καθολική άποψη, του τρόπου απόκρισης των συστημάτων, σε ερωτήματα της XPath γλώσσας.

4.3 Οι μετρήσεις

Στους πίνακες 4.2 και 4.3 παρουσιάζουμε τα αποτελέσματα των μετρήσεων που πήραμε για τα δύο συστήματα.

	XINDICE						
Μέγεθος XML αρχείου(KB)	**Q1(ms)**	**Q2(ms)**	**Q3(ms)**	**Q4(ms)**	**Q5(ms)**	**Q6(ms)**	**Q7(ms)**
26.5	532	484	532	453	516	469	500
115	656	640	657	516	595	579	813
210	718	641	734	594	1125	688	906
318	766	728	780	750	1844	988	1231
567	1009	1000	1235	926	2956	1389	1738
817	1218	1109	1456	1190	4231	1829	2531
1120	1609	1235	1485	1375	5246	2231	3250

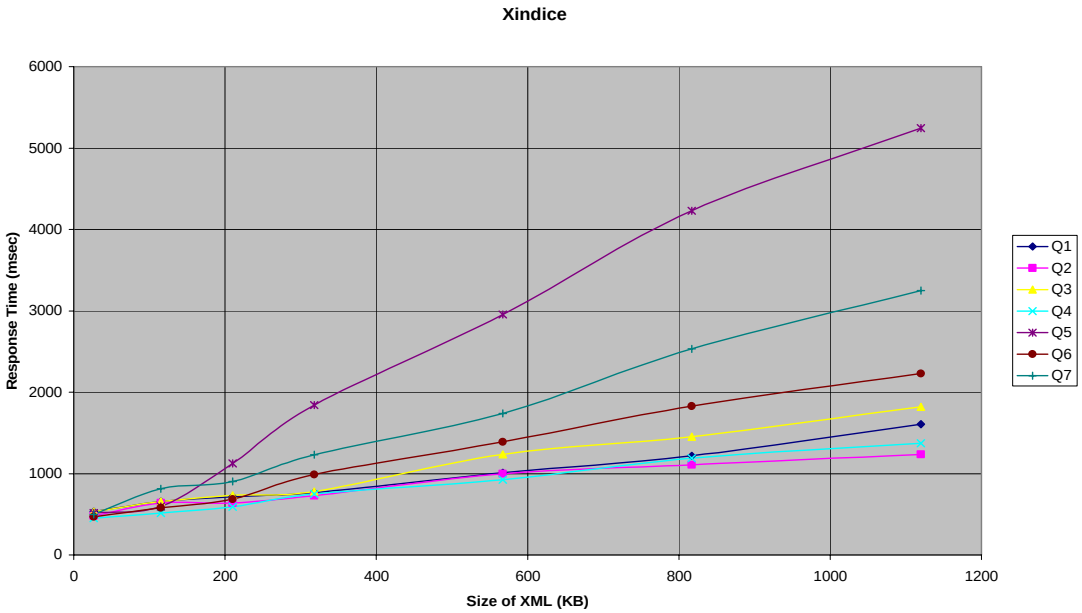
Πίνακας 4.2 : Αποτελέσματα των μετρήσεων για το Xindice.

	XPATH QUERY ENGINE						
Μέγεθος XML αρχείου(KB)	**Q1(ms)**	**Q2(ms)**	**Q3(ms)**	**Q4(ms)**	**Q5(ms)**	**Q6(ms)**	**Q7(ms)**
26.5	418	406	422	418	490	437	500
115	3009	2801	3080	2926	3626	3190	3650
210	9629	8123	10164	9071	12328	11165	12410
318	30813	23557	33541	28120	41915	35728	42194
567	129415 ≈ 2 min	91872 ≈ 1.5 min	144226 ≈ 2.4 min	115292 ≈ 1.9 min	251490 ≈ 4.2 min	150057 ≈ 2.5 min	177214 ≈ 2.9 min
817	543543 ≈ 9 min	459360 ≈ 7.6 min	562481 ≈ 9.3 min	461168 ≈ 7.7 min	754470 ≈ 12.5 min	600228 ≈ 10 min	708856 ≈ 11.8 min
1120	2174172 ≈ 36 min	1929312 ≈ 32 min	2249924 ≈ 37.5 min	1936905 ≈ 32.5 min	2942433 ≈ 49 min	2264136 ≈ 37 min	2322510 ≈ 38.7 min

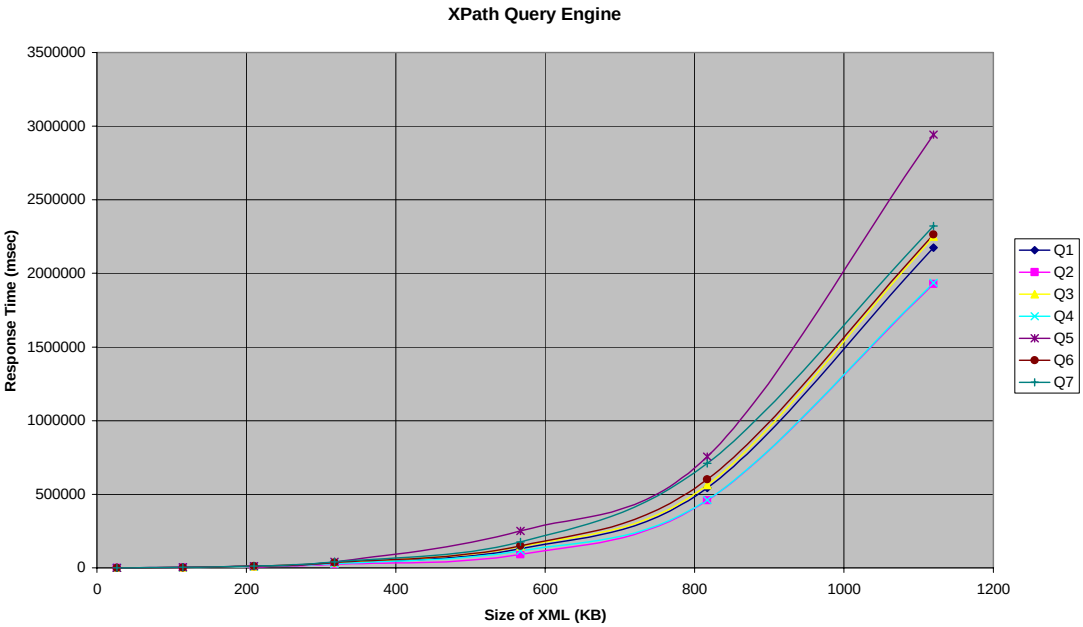
Πίνακας 4.3 : Αποτελέσματα των μετρήσεων για το δικό μας σύστημα.

Όπως φαίνεται από τους 2 παραπάνω πίνακες, καταγράψαμε τους χρόνους που χρειάζονται τα δύο συστήματα, να υπολογίσουν τα 7 XPath Queries που επιλέξαμε και παρουσιάσαμε στην προηγούμενη παράγραφο, για τα διάφορα μεγέθη των XML εγγράφων. Στις εικόνες 4.1 και 4.2 παρουσιάζουμε τα διαγράμματα των χρόνων

αποκρίσεων των 2 συστημάτων, σε συνάρτηση με τα διάφορα μεγέθη των XML εγγράφων που χρησιμοποιήσαμε, για τα 7 XPath ερωτήματα που δοκιμάσαμε.



Εικόνα 4.1 : Διάγραμμα της απόκρισης χρόνου του Xindice, σε συνάρτηση με το μέγεθος των XML αρχείων.



Εικόνα 4.1 : Διάγραμμα της απόκρισης χρόνου του δικού μας συστήματος, σε συνάρτηση με το μέγεθος των XML αρχείων.

Από τα 2 παραπάνω διαγράμματα, τα οποία μας παρουσιάζουν ουσιαστικά το πόσο γρήγορα μπορούν να απαντούν XPath ερωτήματα τα 2 συστήματα μπορούμε να κάνουμε τις ακόλουθες εκτιμήσεις :

- Είναι προφανές κι από τις μετρήσεις άλλωστε, πως το Xindice, το οποίο πρόκειται για ένα εμπορικό πρόγραμμα, μπορεί και απαντά τα XPath ερωτήματα με χρόνους απόκρισης που είναι τάξεις μικρότεροι συγκριτικά με την δική μας υλοποίηση. Μόνο στην περίπτωση που το XML αρχείο είναι μικρό (26,5 KB), οι χρόνοι απόκρισης της δικής μας υλοποίησης είναι παρόμοιοι με το Xindice και σε κάποιες περιπτώσεις πιο μικροί, αλλά η διαφορά είναι αμελητέα, αφού για την περίπτωση των 26,6 KB μιλάμε για τάξη των msec.
- Η αύξηση των χρόνων απόκρισης των ερωτημάτων που εκτιμώνται από το Xindice γίνεται σχεδόν γραμμικά σε συνάρτηση με το μέγεθος των XML αρχείων, ενώ στην περίπτωση της δικιάς μας υλοποίησης, η αύξηση των χρόνων απόκρισης, γίνεται με τρόπο εκθετικό. Η γραμμικότητα που εμφανίζεται στην πρώτη περίπτωση οφείλεται στο ότι οι διαφορές μεταξύ των αποκρίσεων χρόνων ενός ερωτήματος όταν αυξάνουμε το μέγεθος του XML αρχείου, είναι πολύ μικρές, σε αντίθεση με την δική μας υλοποίηση, που οι αντίστοιχες διαφορές μεταξύ των αποκρίσεων χρόνων είναι μεγαλύτερες. Αποτέλεσμα της εκθετικής αυτής αύξησης των χρόνων απόκρισης ήταν να μην μπορέσουμε να θέσουμε ερωτήματα σε XML αρχεία που είχαν μέγεθος μεγαλύτερο από 1,12 MB. Στην περίπτωση που προσπαθήσαμε να αυξήσουμε τον παράγοντα ενεργοποίησης XML εγγράφων στον XMLgen από 0.01 (1,12 MB) σε 0.02 (2,27 MB), το σύστημα κατέρρευσε. Πρέπει να πούμε εδώ πως μετά τα 2.27 MB συνεχίσαμε να τροφοδοτούμε το Xindice με XML αρχεία αυξανόμενου μεγέθους για να δοκιμάσουμε τα όρια του και είδαμε πως οι χρόνοι απόκρισης του Xindice παρέμεναν ικανοποιητικοί για το μέγεθος των αρχείων π.χ. στα 4,61 MB οι χρόνοι απόκρισης ήταν περίπου στα 4 με 5 sec, αλλά για αρχεία πάνω από 5 MB, το σύστημα αρνήθηκε την φόρτωση του αρχείου. Η εξήγηση είναι πως το Xindice πρόκειται για βάση που κρατάει πολλά XML αρχεία, μικρών και μεσαίων μεγεθών και δεν σχεδιάστηκε για να αποθηκεύει αρχεία τεραστίων διαστάσεων.

- Αν και οι διαφορές των χρόνων απόκρισης μεταξύ των 2 συστημάτων είναι τεράστιες, μπορούμε εντούτοις να διαπιστώσουμε από τα 2 διαγράμματα παραπάνω, πως τα 2 συστήματα επεξεργάζονται τα ερωτήματα με ανάλογη καθυστέρηση. Με αυτό, εννοούμε πως και στα 2 συστήματα, από τα 7 XPath ερωτήματα που δοκιμάσαμε, η μεγαλύτερη καθυστέρηση εμφανίζονται στο ερώτημα Q5 : *//descendant::current/preceding::initial*, ακολουθεί το ερώτημα Q7 : *//*[@id]*, συνεχίζουμε με το ερώτημα Q6 : */site/regions/*/item[@id='item0']/following::item*, έπειτα το ερώτημα Q3 : *//keyword/ancestor::listitem*, το Q1 : */site/regions/*/item*, το ερώτημα Q4 : */site/regions/*/item[parent::namerica or parent::samerica]* και τέλος την μικρότερη καθυστέρηση την εμφανίζει το ερώτημα Q2 : */descendant-or-self::listitem/descendant-or-self::keyword.* Αυτό άλλωστε μπορούμε να τα διαπιστώσουμε υπολογίζοντας την μέση ταχύτητα απόκρισης για το κάθε ερώτημα. Αρχικά υπολογίζουμε την ταχύτητα απόκρισης για το κάθε ερώτημα που ορίζεται ως εξής : *query response speed = size of document / query response time* (MB/sec). Η μέση ταχύτητα απόκρισης για το κάθε ερώτημα, θα είναι ο μέσος όρος των ταχυτήτων των αποκρίσεων που υπολογίσαμε για το ερώτημα, για όλα τα XML έγγραφα. Στους 2 παρακάτω πίνακες παρουσιάζουμε τις ταχύτητες απόκρισης των ερωτημάτων για όλα τα μεγέθη των XML εγγράφων.

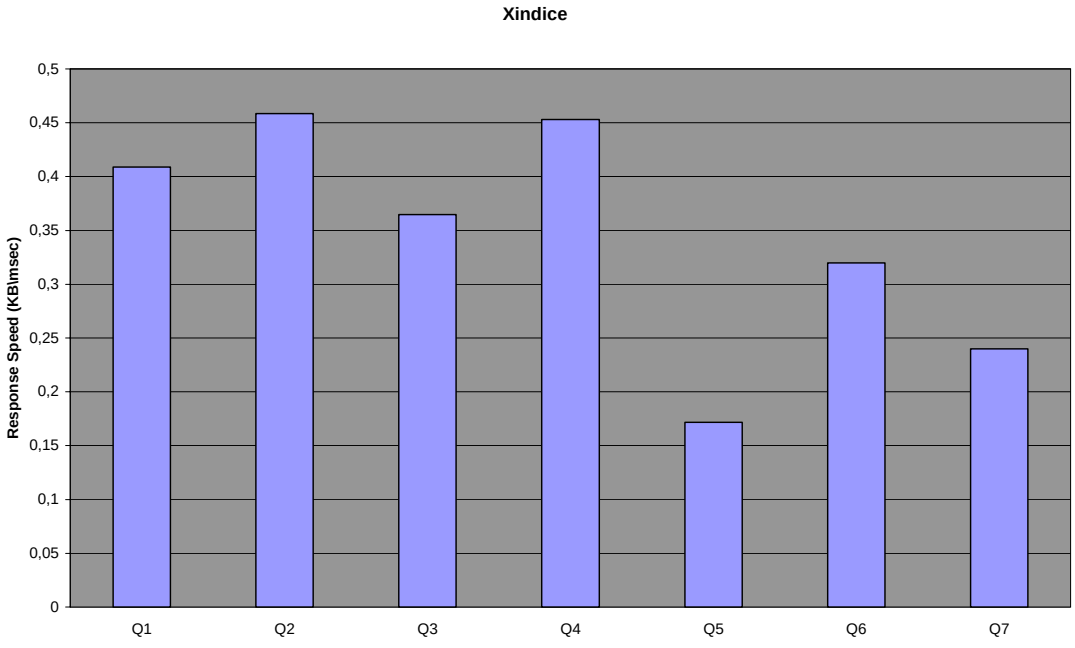
	XINDICE						
Μέγεθος XML αρχείου(KB)	**Q1 Response Speed (KB/ms)**	**Q2(ms) Response Speed (KB/ms)**	**Q3(ms) Response Speed (KB/ms)**	**Q4(ms) Response Speed (KB/ms)**	**Q5(ms) Response Speed (KB/ms)**	**Q6(ms) Response Speed (KB/ms)**	**Q7(ms) Response Speed (KB/ms)**
26.5	0,049812	0,054752	0,049812	0,058499	0,051357	0,056503	0,053
115	0,175305	0,179688	0,175038	0,222868	0,193277	0,198618	0,141451
210	0,292479	0,327613	0,286104	0,353535	0,186667	0,305233	0,231788
318	0,415144	0,436813	0,407692	0,424	0,172451	0,321862	0,258327
567	0,561943	0,567	0,459109	0,612311	0,191813	0,408207	0,326237
817	0,670772	0,7367	0,561126	0,686555	0,193099	0,446692	0,322797
1120	0,696085	0,906883	0,615047	0,814545	0,213496	0,502017	0,344615
Μέσος Όρος	0,408791	0,458493	0,364847	0,453188	0,171737	0,319876	0,239745

Πίνακας 4.3 : Ταχύτητες απόκρισης του Xindice.

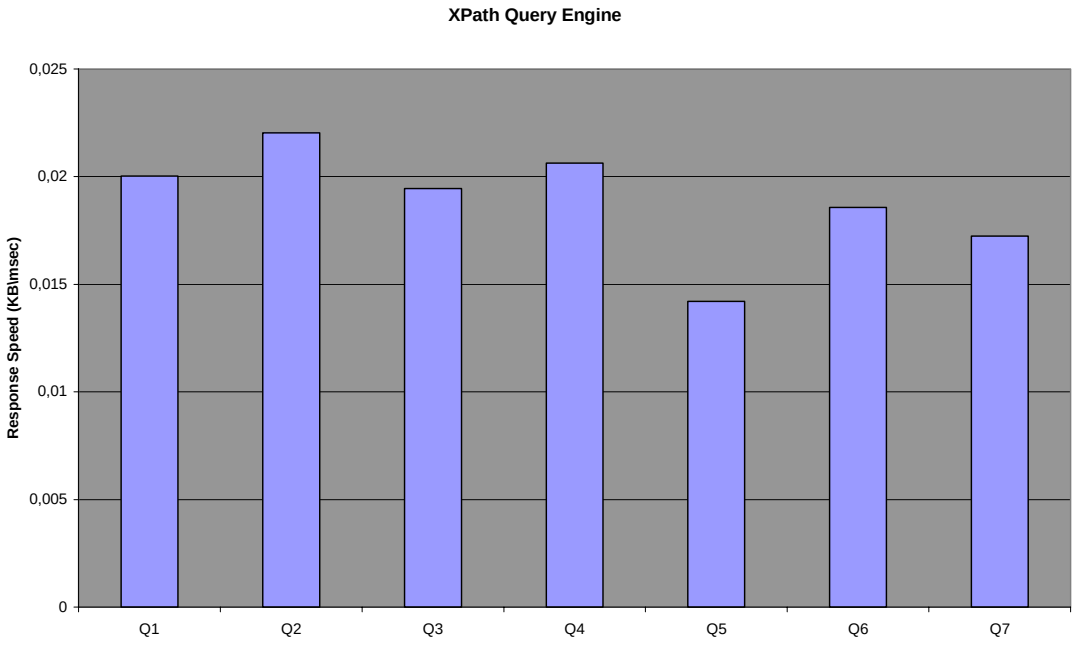
	XPATH QUERY ENGINE						
Μέγεθος XML αρχείου(KB)	**Q1 Response Speed (KB/ms)**	**Q2(ms) Response Speed (KB/ms)**	**Q3(ms) Response Speed (KB/ms)**	**Q4(ms) Response Speed (KB/ms)**	**Q5(ms) Response Speed (KB/ms)**	**Q6(ms) Response Speed (KB/ms)**	**Q7(ms) Response Speed (KB/ms)**
26.5	0,063397	0,065271	0,062796	0,063397	0,054082	0,060641	0,053
115	0,038219	0,041057	0,037338	0,039303	0,031715	0,03605	0,031507
210	0,021809	0,025853	0,020661	0,023151	0,017034	0,018809	0,016922
318	0,01032	0,013499	0,009481	0,011309	0,007587	0,008901	0,007537
567	0,004381	0,006172	0,003931	0,004918	0,002255	0,003779	0,0032
817	0,001503	0,001779	0,001452	0,001772	0,001083	0,001361	0,001153
1120	0,000515	0,000581	0,000498	0,000578	0,000381	0,000495	0,000482
Μέσος Όρος	0,020021	0,02203	0,019451	0,020632	0,016305	0,018576	0,016257

Πίνακας 4.4 : Ταχύτητες απόκρισης του δικού μας συστήματος.

Στις εικόνες 4.3 και 4.4 παρουσιάζουμε τα διαγράμματα των μέσων τιμών των ταχυτήτων απόκρισης στα 7 XPath ερωτήματα για τα 2 συστήματα.



Εικόνα 4.3 : Μέσες τιμές των ταχυτήτων απόκρισης για το Xindice.



Εικόνα 4.3 : Μέσες τιμές των ταχυτήτων απόκρισης για το δικό μας σύστημα.

Από τα 2 παραπάνω διαγράμματα μπορούμε να συμπεράνουμε το πόσο γρήγορα εκτελείται το κάθε ένα από τα XPath ερωτήματα. Ένα συμπέρασμα που μπορεί να βγει από την μέση ταχύτητα εκτέλεσης των ερωτήσεων είναι πως γενικά και τα 3 συστήματα καθυστερούν πιο πολύ στις περιπτώσεις που το query περιλαμβάνει άξονες που αφορούν τους προγόνους των κόμβων, όπως για παράδειγμα ο preceding άξονας (Q5) και ο ancestor άξονας (Q3). Μεγάλη καθυστέρηση επίσης συναντάμε στην περίπτωση του Q7 που περιλαμβάνει τον attribute άξονα. Τέλος ο μεγάλος αριθμός από location steps επίσης επιβραδύνει την εκτέλεση ενός query (Q6).

Κεφάλαιο 5

Συμπεράσματα και μελλοντικές βελτιώσεις

Σε αυτό το κεφάλαιο, αρχικά κάνουμε μια σύνοψη της δουλειάς που πραγματοποιήσαμε, για την ολοκλήρωση αυτής της διπλωματικής εργασίας και παραθέτουμε τα συμπεράσματα μας. Επιπλέον, προτείνουμε τρόπους, με τους οποίους μπορούμε να επεκτείνουμε και να βελτιώσουμε την δουλειά μας.

5.1 Σύνοψη και Συμπεράσματα

Δεδομένου του ότι η XML έχει πλέον καθιερωθεί ως standard για την ανταλλαγή δεδομένων και πληροφορίας μέσω του διαδικτύου, η ανάκτηση πληροφορίας από XML έγγραφα είναι επιταχτική. Η διπλωματική αυτή εργασία είχε ως βασικό στόχο την υλοποίηση ενός συστήματος, που εμείς ονομάσαμε XPath Query Engine, το οποίο θα μας δίνει την δυνατότητα, να κάνουμε αναζητήσεις μέσα σε XML έγγραφα, με την χρήση των εκφράσεων, που μας παρέχει η σύνταξη της XPath.

Για να μπορέσουμε να επεξεργαστούμε τα δεδομένα που περιέχουν τα έγγραφα της XML, χρησιμοποιήσαμε το DOM API, που μας παρείχε ένας δημοφιλής XML parser, ο Xerces. Η βασική υλοποίηση του συστήματός μας, με την οποία πραγματοποιούμε τις αναζητήσεις για την ανάκτηση των δεδομένων, βασίστηκε στην χρήση της πολυδιάστατης αναζήτησης, την οποία πραγματοποιήσαμε με την χρησιμοποίηση μιας δενδρικής δομής, που κρατάει πολυδιάστατα δεδομένα, το kd-tree. Σημαντικό επίσης κομμάτι στην υλοποίηση, αποτέλεσε και ο query parser που χρησιμοποιήσαμε για την συντακτική και λεκτική ανάλυση των XPath εκφράσεων. Επιπλέον με τη χρήση του query parser, οι XPath εκφράσεις αναλύονται σε δενδρικές δομές τα XPS-trees, τα οποία συγκρατούν στους κόμβους τους τα βήματα από τα οποία αποτελούνται οι XPath εκφράσεις. Στηριζόμενοι στον τρόπο με τον οποίο αναλύουν τις XPath εκφράσεις τα XPS-trees, δημιουργήσαμε τις ανάλογες κλάσεις, τις οποίες σε συνδυασμό με τα kd-trees, τις χρησιμοποιήσαμε για να εξάγουμε το τελικό αποτέλεσμα, που είναι η ανάκτηση της πληροφορίας που επιθυμούμε από το XML έγγραφο που χρησιμοποιούμε ως είσοδο για το σύστημα.

Για να μπορέσουμε να εκτιμήσουμε την απόδοση του συστήματος, χρησιμοποιήσαμε 7 XPath ερωτήματα, όπως αυτά προτείνονται από το XPathMark, για την αναζήτηση σε XML έγγραφα διαφόρων μεγεθών, τα οποία ενεργοποιούνται

με τον XMLgen εργαλείο που προτείνεται από το XMark. Τόσο τα XML έγγραφα που ενεργοποιούνται με τον XMLgen, όσο και τα XPath ερωτήματα που χρησιμοποιήσαμε, αποτελούν πρότυπα για την εκτίμηση της απόδοσης συστημάτων που χρησιμοποιούν XML. Οι μετρήσεις που πήραμε αφορούσαν τον χρόνο που χρειάζονταν τα ερωτήματα να απαντηθούν και στην συνέχεια υπολογίσαμε τη μέση ταχύτητα απόκρισης για το κάθε ερώτημα ξεχωριστά. Ανάλογες μετρήσεις πραγματοποιήσαμε στο εμπορικό σύστημα της Apache τον Xindice, που επίσης υποστηρίζει την XPath ως γλώσσα αναζήτησης σε XML έγγραφα. Επειδή τα δύο συστήματα είναι διαφορετικά, καθώς ο Xindice πρόκειται για βάση δεδομένων XML αρχείων, οι συνθήκες καταγραφής των μετρήσεων φροντίσαμε να είναι τέτοιες, ώστε τα 2 συστήματα να είναι συγκρίσιμα.

Τα αποτελέσματα των μετρήσεων που καταγράψαμε μας έδειξαν πως ο Xindice μπορεί και επεξεργάζεται XPath ερωτήματα με πολύ μεγαλύτερη ταχύτητα, συγκριτικά με την δικιά μας υλοποίηση. Στην περίπτωση του Xindice, ο χρόνος απόκρισης των ερωτημάτων αυξάνεται με τρόπο γραμμικό, ενώ στην περίπτωση της δικιάς μας υλοποίησης η αύξηση είναι εκθετική. Συγκριτικά επομένως το σύστημά μας, με ένα εμπορικό σύστημα όπως είναι ο Xindice, δεν είναι αποδοτικό για μεγάλα XML αρχεία.

Φυσικά υπάρχουν περιθώρια βελτιστοποίησης του συστήματός μας, αλλά δεν πρέπει να ξεχνάμε πως η συγκεκριμένη υλοποίηση διεξήχθη στα πλαίσια μιας διπλωματικής εργασίας εξαμήνου, οπότε οι απαιτήσεις μας σε απόδοση δεν είναι δυνατό να φτάνουν σε μέγεθος, αυτές μιας εμπορικής εφαρμογής, η οποία ούτως ή άλλως προορίζεται να είναι ανταγωνιστική συγκριτικά με παρόμοιες εφαρμογές που κυκλοφορούν επισήμως από άλλες εταιρίες.

5.2 Μελλοντικές βελτιώσεις και επεκτάσεις

Η λειτουργικότητα του συστήματος που θέλαμε να επιτύχουμε παρέχεται ήδη. Έτσι το σύστημά μας υποστηρίζει όλους τους XPath άξονες, όλα τα node tests, XPath εκφράσεις μέσα σε predicates, τους λογικούς τελεστές and, or και τους τελεστές σύγκρισης μέσα στα predicates. Βέβαια η XPath παρέχει και ένα σύνολο από συναρτήσεις, τις οποίες δεν υποστηρίζουμε στην υλοποίηση μας. Οι συναρτήσεις παρέχουν μεγάλη ευελιξία σε κάποιον που θέλει να κάνει αναζητήσεις μέσα σε XML

έγγραφα, οπότε μια καλή προέκταση του συστήματος θα ήταν η υποστήριξη των συναρτήσεων της XPath.

Δεδομένου του ότι η απόδοση του συστήματός μας δεν ήταν ιδιαίτερα ικανοποιητική, μελέτη θα μπορούσε να αφιερωθεί στην βελτιστοποίηση των χρόνων απόκρισης του συστήματος. Αυτό θα μπορούσε να γίνει με την χρησιμοποίηση εναλλακτικών πολυδιάστατων δομών αναζήτησης, που μπορούν να υποστηρίξουν δομές, όπως είναι το πενταδιάστατο διάνυσμα που περιγράφει τους κόμβους των XML εγγράφων.

5.3 Εργαλεία ανάπτυξης του συστήματος

Για τέλος αφήσαμε την περιγραφή των εργαλείων που χρησιμοποιήσαμε για την υλοποίηση του συστήματός μας. Έτσι, για την εφαρμογή μας επιλέξαμε την γλώσσα προγραμματισμού Java, με την χρήση του εργαλείου Jbuilder 2005.Ο query parser που χρησιμοποιήσαμε, είναι γραμμένος σε Java κώδικα και η ενεργοποίηση του, έγινε με την χρήση του εργαλείου JavaCC [31, 32, 33]. Η γραμματική για το εργαλείο JavaCC, που υλοποιεί τις εκφράσεις τις XPath, διατίθεται στο εξής link : *http://www.cobase.cs.ucla.edu/pub/javacc/*. Η σύνδεση του parser με την υπόλοιπη εφαρμογή έγινε με την ενσωμάτωση Java κώδικα στην γραμματική του parser.

Βιβλιογραφία – Αναφορές

[1] W3C Home Page. http://www.w3.org

[2] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation 04. February 2004. http://www.w3.org/TR/REC-xml

[3] Dan Conolly. Overview of SGML Resources. Nov 1995. http://www.w3.org/Markup/SGML/

[4] James Clark, Steve DeRose. XML Path Language (XPath) Version 1.0 W3C Recommendation 16 November 1999. http://www.w3.org/TR/xpath

[5] Michael Kay. XSL Transformations (XSLT) Version 2.0 W3C Working Draft 15 September 2005. http://www.w3.org/TR/XSLT20

[6] Vidur Apparao, Steve Byrne, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood. Document Object Model (DOM) Level 1 Specification Version 1.0 W3C Recommendation *1* October, 1998. http://www.w3.org/TR/REC-DOM-Level-1/

[7] Dave Ragett, Arnaud Le Hors. HTML 4.0 Specification. December 1997. http://www.w3.org/TR/WD-html40/

[8] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, Jérôme Siméon. XQuery 1.0: An XML Query Language W3C Working Draft 15 September 2005. http://www.w3.org/TR/xquery

[9] Steve DeRose, Eve Maler, Ron Daniel Jr., Interwoven. XML Pointer Language (XPointer) Version 1.0. 8 January 2001. http://www.w3.org/TR/WD-xptr

[10] SAX (Simple API for XML). http://sax.sourceforge.net/

[11] Elliotte Rusty Harold. Processing XML with Java.
http://www.cafeconleche.org/books/xmljava/

[12] Xerces Java Parser 1.4.4. http://xml.apache.org/xerces-j/

[13] Torsten Grust. *Accelerating XPath Location Steps.* In Proc. SIGMOD Intl. Conf on Management of Data (SIGMOD 2002). 109-120. 2002.

[14] H. M. Deitel, P. J. Deitel, T. R. Nieto, T. M. Lin, P. Sadhu. *XML How to program.* 2001.

[15] Sun Microsystem's Java API for XML Parsing (JAXP). java.sun.com/xml

[16] IBM's XML Parser for Java (XML4J). www.alphaworks.ibm.com/tech/xml4j

[17] Microsoft's XML parser (msxml) version 2.0. msdn.microsoft.com/xml

[18] 4DOM parser for the Python programming language
fourthought.com/4Suite/4DOM

[19] XML::DOM is a Perl module to manipulate XML documents using Perl. www.4.ibm.com/software/developer/library/xml-perl2

[20] Yi Chen, Susan B. Davidson, Yifeng Zheng, *BLAS : An efficient XPath Processing System.* In Proc. SIGMOD Intl. Conf on Management of Data (SIGMOD '04). 2004.

[21] Raghav Kaushik, Rajasekar Krishnamurthy, Jeffrey F Naughton, Raghu Ramakrishnan. *On the integration of structure indexes and inverted lists.* In Proc. SIGMOD Intl. Conf on Management of Data (SIGMOD '04). 2004.

[22] Andrew W. Moore, *An introductory tutorial on kd-trees.* Extract from Andrew Moore's PhD Thesis: Efficient Memory-based Learning for Robot Control. 1991.

[23] Fred Merkle. KD-Trees. 1997. http://www.me.unm.edu/~bgreen/QEM97/FRED/KD-TREES.HTM

[24] Simonas Saltenis. *Advanced Algorithm Design and Analysis.* Lecture 11. 2004.

[25] Nancy Amato. *Orthogonal range searching.* Lecture 9. 2003.

[26] secret901-ga. Finding the median. June 2002. http://answers.google.com/answers/threadview?id=20120

[27] Andrey Balmin, Kevin S. Beyer. Roberta J. Cochrane, Hamid Pirahesh. *A framework for using materialized XPath Views in XML Query Processing.* In Proc. of the 30th VLDB (Very Large Databeses) Conference. 60-71. 2004.

[28] Apache Xindice. June 2005. http://xml.apache.org/xindice/

[29] Albert Schmidt. XMark — An XML Benchmark Project. June 2003. http://monetdb.cwi.nl/xml/index.html

[30] Massimo Franceschet. XPathMark An XPath Benchmark for XMark. http://staff.science.uva.nl/~francesc/xpathmark/

[31] Java Compiler Compiler [tm] (JavaCC [tm]) - The Java Parser Generator. https://javacc.dev.java.net/

[32] Howard Katz. JavaCC, parse trees and the Xquery grammar. December 2002. http://www-128.ibm.com/developerworks/xml/library/x-javacc1/?n-x-12192

[33] Implementing the XQuery grammar. 2002. http://www.fatdog.com/Extreme.html