ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και Υλοποίηση Συστήματος για Υπηρεσίες με Βάση το Προφίλ και τη Θέση σε μεγάλη κλίμακα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΙΩΑΝΝΗ Ε. ΚΟΛΤΣΙΔΑ
ΠΑΝΑΓΙΩΤΗ Ε. ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2006

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός και Υλοποίηση Συστήματος για Υπηρεσίες με Βάση το Προφίλ και τη Θέση σε μεγάλη κλίμακα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

των

ΙΩΑΝΝΗ Ε. ΚΟΛΤΣΙΔΑ
ΠΑΝΑΓΙΩΤΗ Ε. ΠΑΠΑΔΗΜΗΤΡΙΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Ιουλίου 2006.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Ιωάννης Βασιλείου	Τιμολέων Σελλής	Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επικ. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2006

Ιωάννης Ε. Κολτσίδας

Παναγιώτης Ε. Παπαδημητρίου

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

Περίληψη

Σκοπός της Διπλωματικής Εργασίας είναι η ανάπτυξη μίας Βασικής Πλατφόρμας για την Παροχή Υπηρεσιών σε χρήστες με βάση τη Θέση τους και τις ιδιαίτερες προτιμήσεις τους που συνοψίζονται στο Προφίλ τους. Στο σύστημα αλληλεπιδρούν πάροχοι περιεχομένου και χρήστες με τις δημοσιεύσεις των πρώτων να έχουν στόχο τους δεύτερους. Οι διάκριση των δύο ρόλων δεν είναι παρά μόνο τυπική, αφού ένας χρήστης του συστήματος κατέχει ταυτόχρονα και του δύο ρόλους.

Αναλυτικότερα, για την έκφραση των προφίλ των χρηστών χρησιμοποιείται η γλώσσα XPath ενώ οι δημοσιεύσεις των παρόχων περιεχομένου γίνονται μέσω XML εγγράφων και συνοδευτικές πληροφορίες σχετικά με το χωρικό εύρος στο οποίο μπορεί να βρίσκονται οι αποδέκτες (αποστολή σε μία περιοχή ή στον εγγύτερο γείτονα).

Η απαίτηση για λειτουργία του συστήματος σε μεγάλη κλίμακα (>100000 χρήστες) και πραγματικό χρόνο ικανοποιείται με την ανάπτυξη μίας νέας δομής ευρετηρίου κύριας μνήμης που ενοποιεί την ευρετηρίαση των χρηστών τόσο με βάση την τρέχουσα θέση όσο και το προφίλ τους. Το δέντρο του νέου ευρετηρίου συνδυάζει τη χωρική ευρετηρίαση των χρηστών με χρήση Quad Tree (επιλέχθηκε λόγω της αποδοτικότητας του στην ευρετηρίαση σημείων, στις ενημερώσεις και τα χωρικά ερωτήματα που μας ενδιαφέρουν) ενώ στα φύλλα του Quad Tree οι εγγραφές ευρετηριάζονται με τη βοήθεια του ΜΠΑ που παράγει η μηχανή για ταίριασμα XML εγγράφων YFilter (yfilter.cs.berkeley.edu). Για τη νέα δομή ευρετηρίου αναπτύχθηκε αποδοτικό πρωτόκολλο κλειδωμάτων που συνοδεύεται από θεωρητική απόδειξη της ορθότητάς του.

Το μοντέλα χρήσης του συστήματος που αναπτύχθηκαν περιλαμβάνουν Διεπαφή επικοινωνίας με το σύστημα μέσω Υπηρεσιών Ιστού (Web Services Interface). Μελετήθηκε και παρατίθεται η απόδοση του συστήματος σε πραγματικές συνθήκες μοντελοποιώντας τον ρυθμό των ενημερώσεων και των ερωτημάτων στο σύστημα με αφίξεις Poisson.

Λέξεις Κλειδιά: «Υπηρεσίες με Βάση τη Θέση, Ταίριασμα Προφίλ, χωρική ευρετηρίαση, κλειδώματα Quad Tree, πολυπλοκότητα χωρικού ευρετηρίου και ευρετηρίου προφίλ, Υπηρεσίες Ιστού, YFilter»

Abstract

The object of this Diploma Thesis is the development of a Basic Framework for the provision of Location and Profile Based Services to users. Although there are two roles for the users of this system: publishers and subscribers, the role discretion is just typical, after every user can occasionally act either as a publisher or as a user.

Specifically, subscribers use XPath language to express their profiles. Publishers publish XML documents and define spatial criteria that prospective receivers must fulfill. They can publish their documents within a certain area (range publication) or to the nearest user who is an eligible receiver.

The crucial requirement for our platform was the support of more than 100000 moving users. To satisfy this requirement, we developed a new indexing structure that efficiently unifies spatial and profile indexing of users. The new index is a main memory structure. Quad Tree is used for spatial indexing (it was selected due its efficiency at points indexing, updates and range and nearest neighbor queries). Users at the leaves of the Quad Tree are indexed by the NFA that is produced by YFilter, an XML matching engine (yfilter.cs.berkeley.edu). YFilter was modified to serve as an index. Furthermore, we present an efficient locking protocol for the new index and prove its correctness.

Our system is accessible through a Web Services Interface. We used this interface to study and evaluate the performance of our system. We present different diagrams that depict its performance, supposing that updates and documents sends follow Poisson distribution.

Keywords: «Location Based Services, profile matching, spatial indexing, Quad Tree lock, complexity of spatial and profile index, Web Services, YFilter»

Πίνακας περιεχομένων

1 *Εισαγωγή*

1.1 Αντικείμενο της διπλωματικής

Αντικείμενο αυτής της Διπλωματικής Εργασίας είναι η ανάπτυξη της Βασικής Πλατφόρμας για Υπηρεσίες με Βάση τη Θέση και το Προφίλ των χρηστών τους. Η στοχευμένη παροχή υπηρεσιών αποτελεί το ζητούμενο τόσο για τους παρόχους των υπηρεσιών που επιθυμούν τη μέγιστη δυνατή αξιοποίηση των πόρων που διαθέτουν για την παροχή τους, όσο και από τους χρήστες που επιθυμούν να αποφύγουν τον κατακλυσμό τους από προσφορές υπηρεσιών που δεν ενδιαφέρονται να αξιοποιήσουν.

Για την ανάπτυξη ενός τέτοιου συστήματος, ικανού να λειτουργεί σε μεγάλη κλίμακα, απαιτούνται ανάπτυξη σύνθετων δομών δεδομένων, επινόηση ή χρήση έξυπνων αλγορίθμων ταιριάσματος προφίλ και προσεχτικές αρχιτεκτονικές επιλογές. Η λύση που αναπτύσσεται σε αυτή τη διπλωματική βασίζεται στον αποδοτικό και καινοτόμο συνδυασμό τόσο ερευνητικών αποτελεσμάτων όσο και σύγχρονων τεχνολογικών δυνατοτήτων.

1.2 Οργάνωση του τόμου

Ο τόμος της Διπλωματικής Εργασίας οργανώνεται σε 6 κεφάλαια.

Στο $0^{ο}$ Κεφάλαιο γίνεται αρχικά μια μικρή παρουσίαση των Υπηρεσιών με Βάση τη Θέση και αναφέρονται οι διάφορες τεχνολογίες που χρησιμοποιούνται για την παροχή τους. Στη συνέχεια γίνεται επισκόπηση των λύσεων που χρησιμοποιούνται για τη διατύπωση προφίλ και αποδοτικών αλγορίθμων για το ταίριασμα προφίλ. Τέλος, παρουσιάζονται διάφορες δομές χωρικής ευρετηρίασης και αλγόριθμοι για την αποτίμηση ερωτημάτων περιοχής και εγγύτερης γειτονίας.

Στο $3^{ο}$ Κεφάλαιο γίνεται Ανάλυση των απαιτήσεων και η Σχεδίαση του συστήματος. Τίθενται με συστηματικό τρόπο οι προδιαγραφές του συστήματος και παρουσιάζονται οι βασικές συνι-

στώσες του. Εκτός από την έκθεση και η αιτιολόγηση των βασικών αρχιτεκτονικών επιλογών δίνεται και η περιγραφή των web services (με χρήση WSDL) που υλοποιούνται στη συνέχεια. Επίσης, δίνεται ο ορισμός της δομής ευρετηρίου που αναπτύξαμε για το σύστημα μας. Περιγράφεται η λειτουργικότητά της και δίνονται ακριβείς αλγόριθμοι για όλες τις πράξεις προσπέλασης σε αυτήν. Ακολουθεί το πρωτόκολλο κλειδωμάτων που αναπτύχθηκε ειδικά για το ευρετήριό μας με συστηματική απόδειξη για την ορθότητα της λειτουργίας του.

Στο 4ο Κεφάλαιο αναφέρονται τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του συστήματος και περιγράφεται η υλοποίηση του συστήματος και των συνοδευτικών εφαρμογών.

Στο 5ο Κεφάλαιο μελετάται η απόκριση του συστήματος κατά τη μεταβολή των διαφόρων συνιστωσών του με τη βοήθεια προσομοίωσης διάφορων μοντέλων χρήσης. Για την κατάστρωση των τελευταίων χρησιμοποιούνται στοιχεία από τη Θεωρία των Συστημάτων Αναμονής.

Στο 6ο Κεφάλαιο παρουσιάζονται προτάσεις για επέκταση του συστήματος και μελλοντική έρευνα.

1.3 Ευχαριστίες

Σε αυτό το σημείο θα θέλαμε να ευχαριστήσουμε όλους όσους συνέβαλαν στην διεκπεραίωση αυτής της Διπλωματικής Εργασίας που σηματοδοτεί το πέρας των σπουδών μας στο ΕΜΠ. Θερμές ευχαριστίες οφείλουμε στον καθηγητή Τίμο Σελλή για το ενδιαφέρον του για την εύρεση ενός θέματος που θα διέγειρε την ερευνητική μας διάθεση και την πρόθυμη και έμπρακτη βοήθειά του στη συνέχεια. Ευχαριστούμε, επίσης, τον καθηγητή Ιωάννη Βασιλείου αλλά και τον δρ. Κωνσταντίνο Αρκουμάνη για τη στενή καθοδήγησή του και τη πολύτιμη συμβολή του σε κρίσιμες καμπές κατά τη διάρκεια εκπόνησης αυτής της Δ.Ε.. Ευχαριστούμε, ακόμα, όλα τα μέλη του Εργαστηρίου Βάσεων Γνώσεων και Δεδομένων για την επιστημονική στήριξη που μας παρείχαν. Τέλος, ευχαριστούμε από τα βάθη της καρδιάς μας τα μέλη των οικογενειών μας και τα αγαπημένα μας πρόσωπα για την αγάπη τους και την εμπιστοσύνη τους που μας στήριξε και μας ενθάρρυνε καθ' όλη τη διάρκεια των σπουδών μας.

2 Σχετικά Πεδία

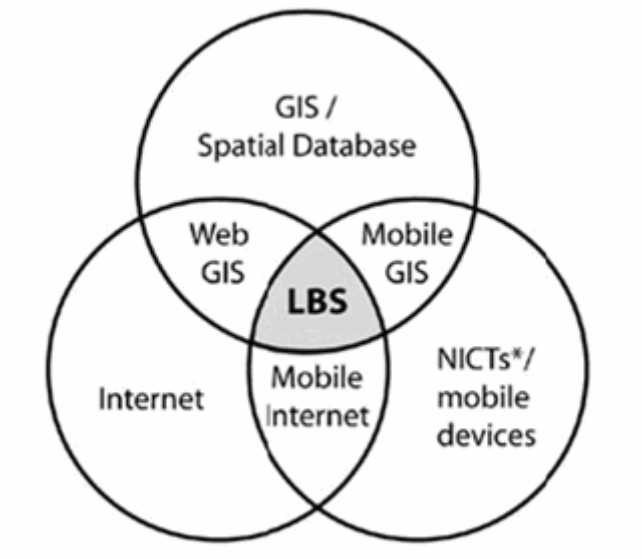
2.1 Υπηρεσίες με Βάση τη Θέση (Location Based Services)

2.1.1 Εισαγωγή

Η αλματώδης ανάπτυξη στον τομέα των τηλεπικοινωνιών και των δικτύων από τη μία, και της πληροφορικής από την άλλη, κατά τα τελευταία χρόνια έχουν επιφέρει πολλές αλλαγές στον τρόπο επικοινωνίας των ανθρώπων και αναζήτησης πληροφοριών για θέματα που τους αφορούν. Η κυριότερη ίσως επίδραση των αλλαγών αυτών είναι ότι πλέον όλο και περισσότερες φορητές συσκευές, όπως κινητά τηλέφωνα ή υπολογιστές παλάμης (PDA's) έχουν δυνατότητα σύνδεσης στο internet σε ικανοποιητικές ταχύτητες, με αποτέλεσμα οι κάτοχοί τους να είναι σε θέση να χρησιμοποιούν τις υπηρεσίες του οποτεδήποτε θέλουν, σε όποια θέση και αν βρίσκονται, με πολύ μικρή επιβάρυνση(τόσο χρονική, όσο και οικονομική). Οι Υπηρεσίες με Βάση τη Θέση (*Location Based Services, LBS*) είναι μια οικογένεια υπηρεσιών οι οποίες ικανοποιούν τις παραπάνω δυνατότητες.

Οι Υπηρεσίες με Βάση τη Θέση είναι προσωποκεντρικές υπηρεσίες που παρέχονται σε χρήστες μέσω φορητών συσκευών πάνω από ένα ασύρματο δίκτυο, κάνοντας χρήση της δυνατότητας που έχουν οι συσκευές αυτές ανά πάσα στιγμή να γνωρίζουν τη θέση τους και να μπορούν να την αναφέρουν μέσω του ασύρματης δικτυακής υποδομής. Γενικότερα, θα μπορούσαμε να πούμε ότι στην κατηγορία αυτή ανήκουν όλες οι ασύρματες υπηρεσίες που αξιοποιούν γεωγραφική πληροφορία ώστε να εξυπηρετήσουν κάποιο κινητό χρήστη. Όπως είναι εμφανές, οι Location Based Services αποτελούν την τομή τριών τεχνολογιών: α) των Γεωγραφικών Συστημάτων Πληροφοριών (GIS) και Χωρικών Βάσεων Δεδομένων(Spatial Databases), β) των τεχνολογιών Δικτύωσης και του Internet και γ) των Νέων Τεχνολογιών Πληροφόρησης και Επικοινωνίας(*New Information and Communication Technologies – NICT's*) και των φορητών

συσκευών με ικανότητες ασύρματης δικτύωσης, όπως φαίνεται στο διάγραμμα (Σχήμα 1) που ακολουθεί:



Σχήμα 1: Τεχνολογίες που συνθέτουν τα Location Based Services

Από ιστορικής πλευράς, η πληροφόρηση που παρέχεται με βάση τη θέση δεν είναι κάτι καινούριο, καθώς ως τέτοια μπορεί να θεωρηθεί η πληροφόρηση που δίνεται από συμβατικά μέσα πληροφόρησης στους δρόμους όπως ταμπέλες για την κυκλοφορία ή αφίσες που αναγγέλλουν κάποιο καλλιτεχνικό γεγονός. Ωστόσο, τα παραπάνω είναι μέσα μονόδρομης επικοινωνίας. Αντίθετα, τα Location Based Services δίνουν τη δυνατότητα αμφίδρομης επικοινωνίας και αλληλεπίδρασης. Ο χρήστης γνωστοποιεί στον πάροχο των πληροφοριών το είδος των πληροφοριών που χρειάζεται, τις θεματικές του προτιμήσεις και τη θέση του. Τούτο επιτρέπει στον πάροχο να εξατομικεύει τις πληροφορίες που στέλνει στον κάθε χρήστη, έτσι ώστε όχι μόνο εξυπηρετούν τις ανάγκες του, αλλά και να του παρουσιάζονται με τον τρόπο που τον εξυπηρετεί.

Όπως φαίνεται από τα παραπάνω, τα συστήματα Υπηρεσιών με Βάση τη Θέση έχουν πολλές ομοιότητες με τα Γεωγραφικά Συστήματα Πληροφοριών, με κυριότερη ότι και τα δύο χρησιμοποιούν χωρική πληροφορία και αξιοποιούν χωρικές σχέσεις μεταξύ των αντικειμένων για να απαντήσουν ερωτήματα διαφόρων τύπων, όπως «Πού βρίσκομαι;», «Τι βρίσκεται κοντά μου;» ή «Πώς μπορώ να πάω στο τάδε;». Η κυριότερη διαφορά από την άλλη, των δύο αυτών εφαρμογών είναι ότι τα Γεωγραφικά Συστήματα Πληροφοριών απευθύνονται κυρίως σε έμπειρους επαγγελματίες χρήστες, και κυρίως μεγάλη υπολογιστική ισχύ για την αξιοποίηση επαγγελματικών αναγκών τους με , ενώ τα Location Based Services απευθύνονται στη μεγάλη «μάζα» των απλών χρηστών για πιο περιορισμένο αριθμό εφαρμογών και έχουν εγγενείς περιορισμούς σε ότι αφορά το υλικό στο οποίο προορίζονται να εκτελεστούν, όπως χαμηλή υπολογιστική ισχύ, μικρή κατανάλωση ενέργειας και μικρές δυνατότητες γραφικής απεικόνισης.

2.1.2 Συστατικά ενός Συστήματος Location Based Services

Ένα τυπικό σύστημα για την παροχή υπηρεσιών με βάση τη θέση αποτελείται από τα ακόλουθα συστατικά (βλ. [SNE05]):

- *Φορητές Συσκευές (Mobile Devices)*

  Ένα εργαλείο για το χρήστη μέσω του οποίου να έχει τη δυνατότητα να ζητά και να λαμβάνει τις πληροφορίες που επιθυμεί. Οι πληροφορίες αυτές μπορεί να του δίνονται με κείμενο, εικόνες, φωνής, ή και video. Τέτοιου τύπου συσκευές είναι οι υπολογιστές παλάμης(PDA's), τα κινητά τηλέφωνα, οι φορητοί υπολογιστές ή ακόμα και ένας υπολογιστής(μονάδα πλοήγησης) αυτοκινήτου.

- *Δίκτυο Επικοινωνίας (Communication Network)*

  Ένα ασύρματο δίκτυο το οποίο μεταφέρει τα δεδομένα και τις προτιμήσεις του χρήστη από το τερματικό του στον πάροχο των υπηρεσιών και στη συνέχεια την απάντηση του παρόχου με τις ζητούμενες πληροφορίες στο χρήστη.

- *Σύστημα Εντοπισμού Θέσης (Positioning Component)*

  Για τη λειτουργία υπηρεσιών με βάση τη θέση του χρήστη απαιτείται να υπάρχει κάποιο σύστημα με το οποίο ο χρήστης θα γνωρίζει τη θέση του. Αυτό μπορεί να γίνει είτε μέσω του δικτύου κινητής τηλεφωνίας (υπολογίζοντας αποστάσεις από σταθμούς βάσης) είτε απευθείας χρησιμοποιώντας το GPS (Global Positioning System), του οποίου ο δέκτης μπορεί να είναι είτε ενσωματωμένος στη φορητή συσκευή είτε εξωτερικός. Για εφαρμογές που είναι πολύ στατικές από γεωγραφική πλευρά μπορεί ακόμα και ο ίδιος ο χρήστης να εισάγει τη θέση του χειρωνακτικά.

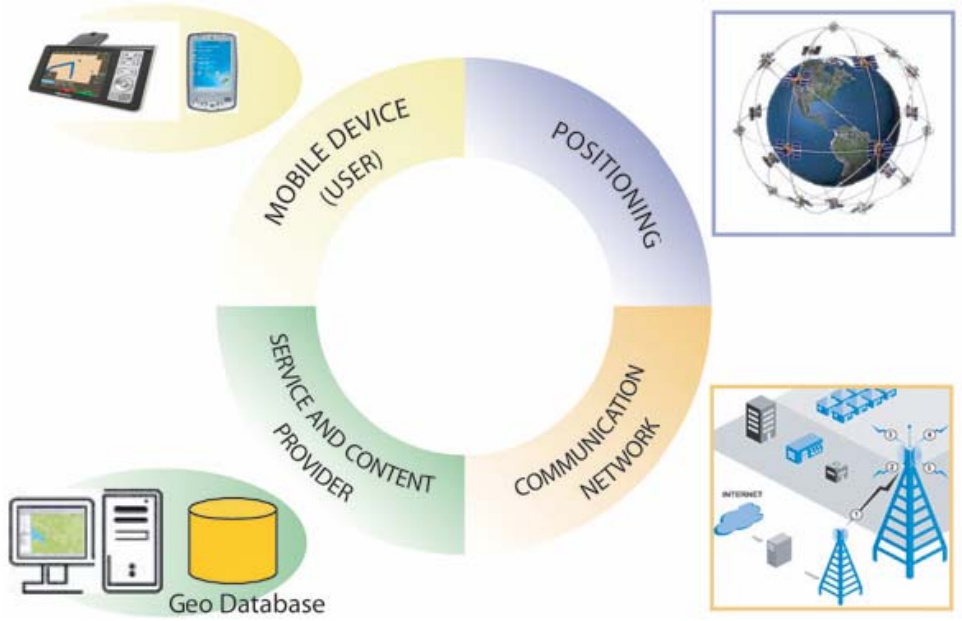
- *Πάροχος Υπηρεσίας και Εφαρμογών (Service and Application Provider)*

  Ο πάροχος της υπηρεσίας είναι υπεύθυνος για το γενικότερο συντονισμό και λειτουργία του συστήματος. Η κυριότερη υπηρεσία που προσφέρει είναι η επεξεργασία ερωτημάτων από τους χρήστες, όπως για παράδειγμα η εύρεση ή ο υπολογισμός μιας ζητούμενης θέσης, η εύρεση μιας διαδρομής προς κάποιο στόχο ή ο εντοπισμός ενός συγκεκριμένου τύπου υπηρεσίας σε σχέση με την τρέχουσα θέση(όπως το κοντινότερο βενζινάδικο).

- *Πάροχος Περιεχομένου και Δεδομένων (Data and Content Provider)*

  Οι πάροχοι της υπηρεσίας και των εφαρμογών που περιγράφονται παραπάνω συνήθως δεν είναι υπεύθυνοι για το περιεχόμενο που προσφέρεται μέσω των υπηρεσιών. Για το τελευταίο φροντίζουν οι πάροχοι δεδομένων και περιεχομένου, τους οποίους στη συνέχεια θα αποκαλούμε απλά παρόχους. Τέτοιοι πάροχοι είτε αποθηκεύουν και συντηρούν γεωγραφικά στοιχεία για διάφορες περιοχές και κατηγορίες υπηρεσιών, όπως υπηρεσίες καταλόγου κτλ.

Τα παραπάνω φαίνονται στο ακόλουθο σχήμα (Σχήμα 2):



Σχήμα 2: Συστατικά ενός συστήματος Location Based Services

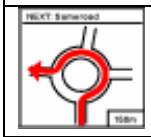
2.1.3 Push και Pull Services

Μπορούμε να διακρίνουμε τις υπηρεσίες που προσφέρονται στους χρήστες ανάλογα με το αν η προσφορά τους απαιτεί αλληλεπίδραση του χρήστη ή όχι. *Pull Services* ονομάζονται οι υπηρεσίες που προσφέρονται στο χρήστη μόνο αν αυτός τις ζητήσει ρητά(όπως στο internet ένας χρήστης που θέλει να έχει πρόσβαση σε μια ιστοσελίδα εισάγει το URL της στο browser που χρησιμοποιεί). Αντίθετα, οι *Push Services* διανέμουν πληροφορίες στους χρήστες είτε χωρίς ο χρήστης να τις έχει ζητήσει άμεσα είτε ακόμα και χωρίς να τις έχει καν ζητήσει. Τέτοιου είδους push υπηρεσίες ενεργοποιούνται από κάποιο γεγονός, όταν για παράδειγμα ένας χρήστης εισέλθει σε μια περιοχή ή όταν περάσει ένα χρονικό διάστημα μετά από κάποιο γεγονός. Push πληροφορίες που κάποιος χρήστης μπορεί να μην έχει ζητήσει άμεσα είναι για παράδειγμα η συνδρομή σε μια υπηρεσία ειδήσεων αναφορικά με την πόλη ή την περιοχή στην οποία βρίσκεται ο χρήστης (έτσι όταν αυτός εισέλθει σε μια άλλη πόλη/περιοχή λαμβάνει πληροφορίες ειδήσεων σε σχέση με αυτή την περιοχή). Push πληροφορίες που κάποιος χρήστης μπορεί να μην έχει ζητήσει μπορεί να είναι διαφημιστικά μηνύματα όταν ο χρήστης εισέλθει σε μια περιοχή ενδιαφέροντος για τους διαφημιστές ή προειδοποιητικά μηνύματα για αλλαγές του καιρού ή κυκλοφοριακά προβλήματα. Για να είναι αυτές οι push πληροφορίες χρήσιμες στο χρήστη, θα πρέπει το σύστημα να γνωρίζει (ή να μαντεύει) τα ενδιαφέροντα του χρήστη και τους θεματικούς τομείς προτίμησής του.

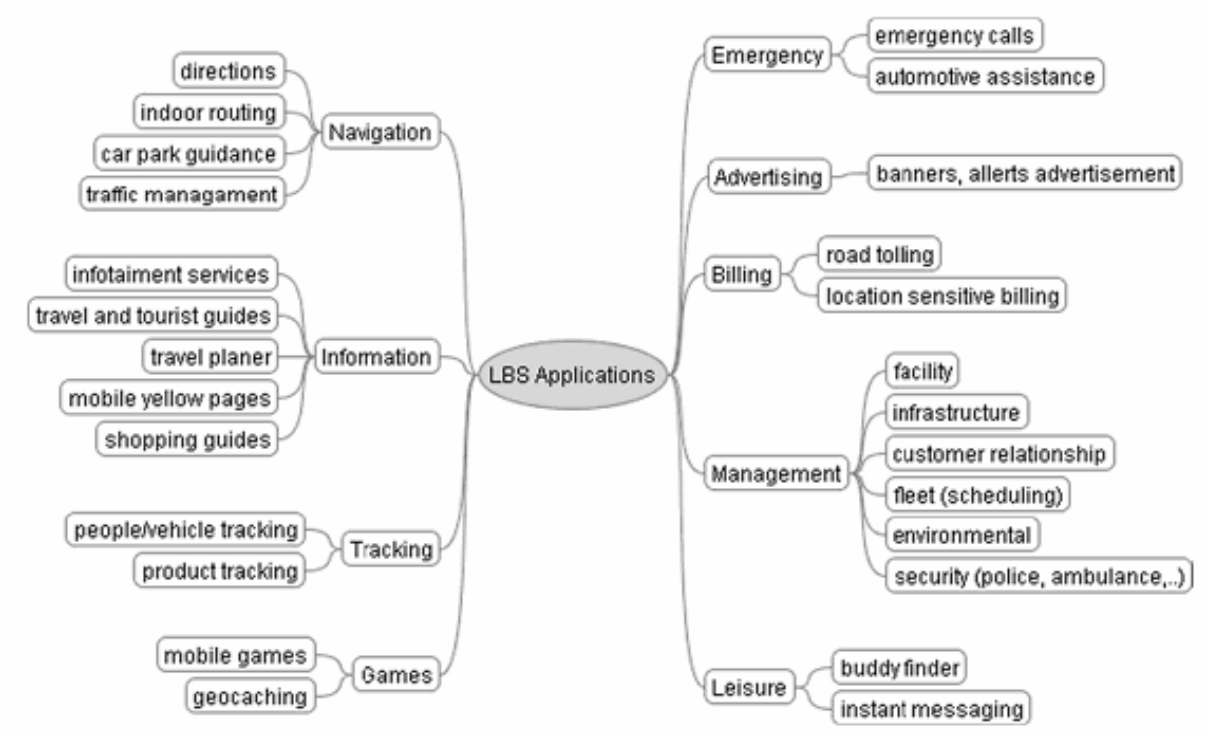
2.1.4 Χρησιμότητα των Location Based Services

Μελέτες σχετικά με την ανθρώπινη συμπεριφορά δείχνουν ότι άτομα βρίσκονται σε ένα άγνωστό τους περιβάλλον, η συμπεριφορά, οι αντιδράσεις και οι ανάγκες τους είναι κατά μεγάλο ποσοστό προβλέψιμες είτε βρίσκονται στην πατρίδα τους είτε στο εξωτερικό, είτε ταξιδεύουν με αυτοκίνητο είτε πεζοί. Οι πιο κοινές ανάγκες τους είναι το να βρουν κάποιο εστιατόριο για φαγητό, κάποιο κατάστημα για είδη πρώτης ανάγκης ή φαρμακείο, κάποιο τερματικό για εκτέλεση ηλεκτρονικών τραπεζικών συναλλαγών και κυρίως εκταμίευση μετρητών, η εύρεση ενός μέσου μεταφοράς (π.χ. ταξί), ή ακόμα η εύρεση ενός ξενοδοχείου. Για την περίπτωση που κάποιος ταξιδεύει οδηγώντας, μπορεί να υπάρχουν και κάποιες άλλες ανάγκες όπως η εύρεση της διαδρομής προς κάποιο σημείο ή η ενημέρωση σχετικά με τις κυκλοφοριακές ή/και καιρικές συνθήκες που θα συναντήσει.

Πιο συγκεκριμένα, οι ενέργειες και οι ερωτήσεις ενός χρήστη καθώς αυτός κινείται τείνουν να εμπεριέχουν χωρικές αναφορές. Η πιο βασική και προφανής ερώτηση ενός κινούμενου χρήστη είναι το να μάθει που βρίσκεται σε σχέση με κάτι ή με κάποιον άλλο (*εντοπισμός – locating*). Επίσης, οι χρήστες ενδέχεται να αναζητήσουν κάποιο άλλο πρόσωπο ή αντικείμενο ή συμβάν (*αναζήτηση – searching*). Μια άλλη τυπική ερώτηση σε ένα τέτοιο σύστημα είναι η ερώτηση *πλοήγησης* (*navigation*) δηλαδή η εύρεση της διαδρομής από ένα σημείο στο χώρο προς ένα άλλο, συνήθως κάνοντας χρήση του ωδικού δικτύου). Εκτός αυτών, ένας χρήστης μπορεί να ζητήσει πληροφορίες ή ιδιότητες μιας περιοχής (*αναγνώριση – identifying)* ή ακόμη να ελέγξει τι γεγονότα συμβαίνουν σε μια συγκεκριμένη περιοχή στο παρόν ή στο μέλλον (*έλεγχος – checking*). Τα παραπάνω συνοψίζονται στον πίνακα που ακολουθεί:

	Ενέργεια	***Ερώτηση***	***Λειτουργίες***
	Προσανατολισμός και Εντοπισμός	*«Πού είμαι;»* *«Πού είναι {ο | η | το};»*	Εύρεση θέσης, Geocoding
	Πλοήγηση	*«Πώς θα πάω στο {όνομα | διεύθυνση | περιοχή};»*	Εύρεση θέσης, Geocoding, Υπολογισμός Διαδρομής
	Αναζήτηση	*«Πού είναι το πιο {κοντινό | σχετικό} {πρόσωπο | αντικείμενο};»*	Εύρεση θέσης, Geocoding, Υπολογισμός Απόστασης, Εύρεση Συσχετίσεων
	Αναγνώριση	*«Ποιος είναι στο τάδε μέρος;» «Πόσο κοστίζει το τάδε;»*	Χρήση Καταλόγου, Επιλογή, Θεματική ή Χωρική Αναζήτηση
	Έλεγχος για Συμβάντα	*«Γίνεται κάτι εδώ κοντά;»* *«Τι γίνεται στο τάδε μέρος;»*	

Στο σχήμα που ακολουθεί (Σχήμα 3) φαίνονται διάφορες θεματικές περιοχές οι οποίες μπορούν να εξυπηρετηθούν με Location Based Services:

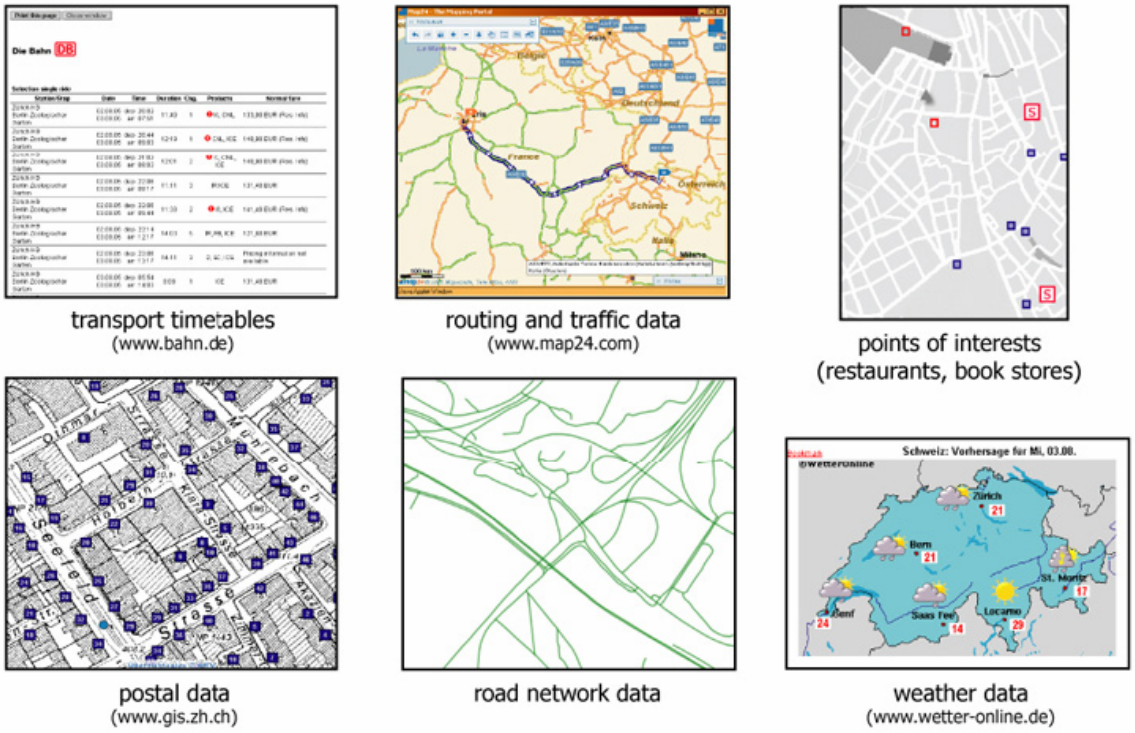


Σχήμα 3: Θεματικές περιοχές που υπηρετούν οι εφαρμογές Location Based Services

Συγκεκριμένα, υπηρεσίες με βάση τη θέση μπορούν να χρησιμοποιηθούν σε μια πληθώρα από εφαρμογές όπως είναι:

- Υπηρεσίες Επειγόντων Περιστατικών: Εντοπισμός με ακρίβεια ατόμων που βρίσκονται σε κατάσταση εκτάκτου ανάγκης
- Υπηρεσίες Πλοήγησης: Πλοήγηση κινητών χρηστών στο οδικό δίκτυο προς έναν επιθυμητό προορισμό.
- Εύρεση Πληροφοριών: Εντοπισμός της κοντινότερης υπηρεσίας ενός συγκεκριμένου τύπου, εύρεση πληροφοριών για την κίνηση στους δρόμους και τις καιρικές συνθήκες, ειδοποίηση προς το χρήστη ότι κοντά του βρίσκεται κάτι που θα τον ενδιέφερε θεματικά (π.χ. μουσείο) ή ακόμα και ξεναγήσεις μέσω του φορητού τερματικού για την περιοχή από την οποία περνάει ο χρήστης.
- Παρακολούθηση/Διαχείριση Κινούμενων Αντικειμένων: Παρακολούθηση ενός δέματος που στέλνεται κάπου, ώστε μια εταιρία να γνωρίζει που βρίσκεται κάθε προϊόν της, παρακολούθηση και διαχείριση στόλου ή προσωπικού μιας εταιρίας(π.χ. παρακολούθηση της θέσης όλων των ασθενοφόρων και ανάθεση του κοντινότερου σε ένα νέο περιστατικό).
- Υπηρεσίες Χρέωσης: Όταν για κάποια τρίτη υπηρεσία ο χρήστης χρεώνεται ανάλογα με τη θέση του, τα Location Based Services μπορούν να προσφέρουν μεγαλύτερη ακρίβεια στη χρέωσή του.

- Διαφήμιση: Κάθε επιχείρηση (π.χ. εστιατόριο) μπορεί να διαφημίζεται εξατομικευμένα σε υποψήφιους πελάτες ανάλογα με τη θέση και τα ενδιαφέροντά τους
- On-line δημοπρασίες: Δημοπρασίες μεταξύ χρηστών (ή και παρόχων) που ανταγωνίζονται για κάποιο αντικείμενο σε μια συγκεκριμένη χωρικά περιοχή.



Σχήμα 4: Τύποι εφαρμογών Location Based Services

Ένα από τα σημαντικότερα χαρακτηριστικά των Location Based Services είναι ότι μπορούν να λειτουργήσουν εξατομικευμένα, δηλαδή γνωρίζοντας ποιον χρήστη εξυπηρετούν και υπό ποιες περιστάσεις (χρονικές) και για ποιο σκοπό. Επίσης, είναι δυνατό οι υπηρεσίες να προσφέρονται με βάση και άλλες πληροφορίες, όπως η ηλικία του χρήστη, τις καιρικές συνθήκες που επικρατούν στην περιοχή που βρίσκεται (π.χ. αν βρέχει) ή που θέλει να πάει. Ειδικότερα τα Location Based Services μπορούν να εξατομικευτούν σε κάθε χρήστη ή κατάσταση, αρκεί το σύστημα να γνωρίζει μια σειρά από παράγοντες, οι κυριότεροι από τους οποίους είναι:

- **Η ταυτότητα του χρήστη**

  Για παράδειγμα, η ηλικία και το φύλο του καθώς και προτιμήσεις του, αλλά και τι γλώσσες γνωρίζει ή προτιμά να χρησιμοποιεί, ποιοι άλλοι χρήστες είναι φίλοι του κτλ.
- **Η θέση του χρήστη**

  Το πιο σημαντικό στοιχείο για τα Location Based Services, καθώς η γνώση της θέσης επιτρέπει την εντοπιότητα των υπηρεσιών
- **Η τοπική ώρα**

  Για παράδειγμα αν είναι πρωί, απόγευμα ή βράδυ, τι εποχή του χρόνου είναι κτλ.

- **Η κατεύθυνση(κίνησης) του χρήστη**

  Δηλαδή, το προς τα πού κινείται ο χρήστης ώστε το σύστημα αφενός να γνωρίζει αν ο χρήστης κινείται προς τη σωστή κατεύθυνση για το στόχο που έχει βάλει και αφ' ετέρου να γνωρίζει ποια αντικείμενα βρίσκονται στα δεξιά του, αριστερά του, μπροστά ή πίσω του και να τον ενημερώνει αναλόγως.

- **Το ιστορικό του χρήστη**

  Δηλαδή, ποιες περιοχές ή τοποθεσίες έχει επισκεφθεί ο χρήστης, ώστε όχι μόνο να διαμορφώνεται από το σύστημα ένα προφίλ με τις προτιμήσεις του χρήστη, αλλά και να γνωρίζει το σύστημα αν ο χρήστης βρίσκεται σε μια άγνωστη γι' αυτόν περιοχή και να τον συμβουλεύει αναλόγως

- **Ο σκοπός χρήσης του συστήματος**

  Για παράδειγμα αν ο χρήστης χρησιμοποιεί το σύστημα επαγγελματικά, για ενημέρωση, ή σαν τουριστικό οδηγό και να προσαρμόζει αναλόγως τον τρόπο παρουσίασης των πληροφοριών και το είδος των πληροφοριών που θα παρουσιάζει στο χρήστη.

- **Κοινωνικές σχέσεις του χρήστη**

  Για παράδειγμα, το σύστημα είναι χρήσιμο να γνωρίζει τους φίλους του χρήστη, ή την κοινωνική του συμπεριφορά (π.χ. αν του αρέσουν μέρη που πηγαίνει η πλειοψηφία των ανθρώπων ή όχι)

- **Το (φυσικό) περιβάλλον του χρήστη**

  Το φυσικό περιβάλλον του χρήστη περιλαμβάνει για παράδειγμα τις καιρικές συνθήκες που επικρατούν στην περιοχή, αν υπάρχει ορατότητα ή όχι, αν υπάρχει θόρυβος κτλ., ώστε να προσαρμόζει τους τρόπους παρουσίασης των πληροφοριών.

- **Το είδος του φορητού τερματικού του**

  Πληροφορίες για το είδος της φορητής συσκευής – τερματικού που χρησιμοποιεί ο χρήστης, για τον τρόπο χρέωσης και το εύρος ζώνης της δικτυακής σύνδεσης του τερματικού, καθώς και την ποιότητα, εγκυρότητα και ακρίβεια των δεδομένων σχετικά με τη θέση του χρήστη που λαμβάνονται από το τερματικό.

Με βάση τις παραπάνω πληροφορίες για το χρήστη και το περιβάλλον του, το σύστημα μπορεί να εξατομικεύει τις υπηρεσίες που προσφέρει σε πολλά επίπεδα:

- **Προσφερόμενη Πληροφορία**

  Το περιεχόμενο των πληροφοριών προσαρμόζεται αναλόγως. Για παράδειγμα, φιλτράρονται οι πληροφορίες που δεν ανήκουν θεματικά στα ενδιαφέροντα του χρήστη, ή είναι ακατάλληλες για την ηλικία του, ή προσαρμόζεται το επίπεδο των λεπτομερειών με τις οποίες περιγράφονται οι εν λόγω πληροφορίες.

- **Χρησιμοποιούμενες Τεχνολογίες**

Ανάλογα με τη φορητή συσκευή που χρησιμοποιεί ο χρήστης και τον τρόπο με τον οποίο αυτή συνδέεται στο internet, μπορεί να χρησιμοποιούνται περισσότερες φωνητικές οδηγίες προς το χρήστη ή περισσότερα γραφικά και κινούμενες εικόνες.

- **Διαπροσωπεία Χρήστη (User Interface)**

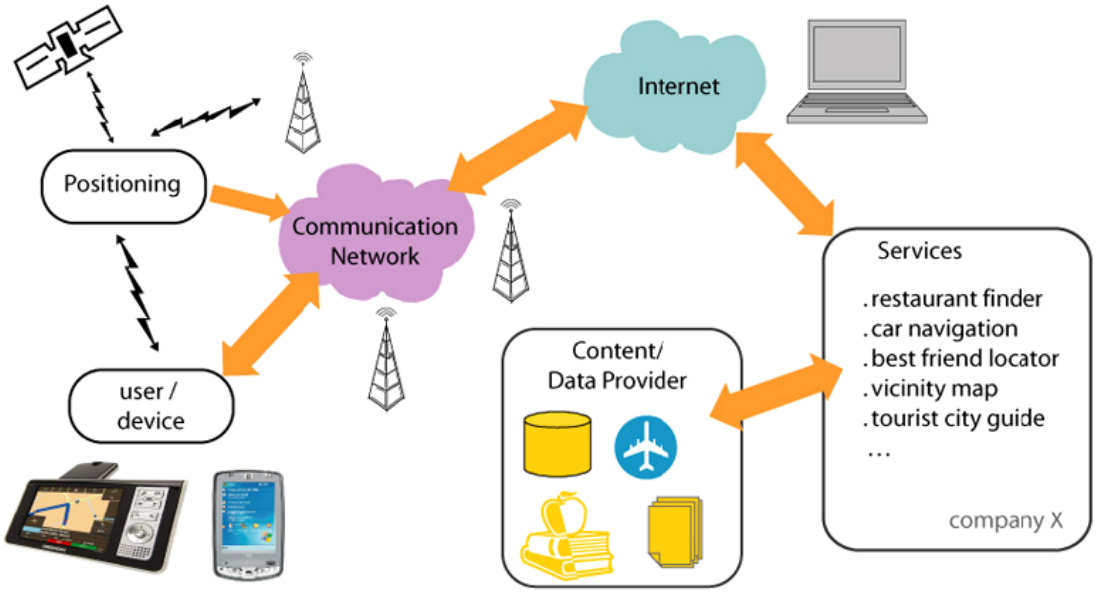
  Για παράδειγμα εστίαση σε ένα χάρτη όταν ο χρήστης κινείται με μικρή ταχύτητα και το αντίθετο όταν η ταχύτητά του αυξάνει ή περιστροφή του χάρτη καθώς ο χρήστης αλλάζει κατεύθυνση

- **Παρουσίαση**

  Ο τρόπος με τον οποίο διαφορετικές θεματικά πληροφορίες παρουσιάζονται στο χρήστη. Για παράδειγμα, αν ο χρήστης έχει δηλώσει ιδιαίτερη προτίμηση στα κινέζικα εστιατόρια, τότε αυτά εμφανίζονται στο χάρτη με πιο έντονα διακριτικά από τα υπόλοιπα.

2.1.5 Επεξεργασία ενός Ερωτήματος

Έστω ότι ένας χρήστης, όπως παραπάνω, αναζητά ένα κινέζικο εστιατόριο και μάλιστα όντας σε μια θέση ζητά από το σύστημα τη διαδρομή που θα τον οδηγήσει από τη θέση αυτή στο εστιατόριο. Τότε στο τερματικό του ο χρήστης επιλέγει κάτι του τύπου "Menu → New Route To → Search → Restaurants → Chinese Cuisine". Από τη στιγμή που το ερώτημα αυτό υποβάλλεται στη συσκευή από το χρήστη, η ακολουθία των γεγονότων και η ροή των πληροφοριών που ακολουθεί φαίνεται στο παρακάτω σχήμα (Σχήμα 5) και περιγράφεται στη συνέχεια.



Σχήμα 5: Επεξεργασία ενός ερωτήματος σε σύστημα Location Based Services

1. Αρχικά λαμβάνεται από το τερματικό του χρήστη η πληροφορία για τη θέση του. Όπως περιγράφηκε και παραπάνω αυτό μπορεί να γίνει είτε απευθείας από τη συσκευή (π.χ. ένα κινητό τηλέφωνο μπορεί να υπολογίσει την απόστασή του από τους σταθμούς βάσης του δικτύου) είτε κάνοντας χρήση ενός εξωτερικού δέκτη GPS. Όταν βρεθεί η θέση του χρήστη, η συσκευή στέλνει μέσω του ασύρματου δικτύου τη θέση αυτή μαζί με το αντικείμενο της αναζήτησης του χρήστη στην πύλη (gateway) του συστήματος.
2. Ρόλος της πύλης είναι να ανταλλάσει μηνύματα μεταξύ του ασύρματου δικτύου και του internet. Ως εκ τούτου γνωρίζει τις διευθύνσεις στο internet των κατάλληλων εξυπηρετητών που ανήκουν στον πάροχο της υπηρεσίας και στέλνει την αίτηση σε κάποιον από αυτούς. Επίσης η πύλη θυμάται ποιος χρήστης έκανε το συγκεκριμένο ερώτημα προς το σύστημα.
3. Ο εξυπηρετητής του παροχέα περιεχομένου λαμβάνει την αίτηση εξυπηρέτησης και εξετάζει το είδος του ερωτήματος – στη συγκεκριμένη περίπτωση είναι ένα ερώτημα χωρικής αναζήτησης, και κατόπιν ενεργοποιεί την κατάλληλη υπηρεσία για την καθαυτό επεξεργασία του ερωτήματος (εδώ μια χωρική βάση δεδομένων).
4. Το σύστημα που αναλαμβάνει την ανάλυση του ερωτήματος αποφασίζει σχετικά με το τι επιπλέον πληροφορίες απαιτούνται για την απάντηση στο ερώτημα. Στη συγκεκριμένη περίπτωση αποφασίζει ότι απαιτούνται δεδομένα από μια υπηρεσία καταλόγου σχετικά με εστιατόρια ενός συγκεκριμένου τύπου. Γι' αυτό το λόγο ο εξυπηρετητής απευθύνεται σε κάποιο παροχέα περιεχομένου και του ζητάει τα δεδομένα που του λείπουν.
5. Στη συνέχεια η υπηρεσία θα αναζητήσει πληροφορία σχετικά με το οδικό δίκτυο και τους τρόπους με τους οποίους μπορεί να προσεγγιστεί το εστιατόριο από το χρήστη.
6. Έχοντας συγκεντρώσει όλες τις απαραίτητες πληροφορίες ο εξυπηρετητής θα εκτελέσει μια χωρική αναζήτηση και θα αποφασίσει ποια εστιατόρια είναι πιο κατάλληλα για το συγκεκριμένο χρήστη.
7. Στη συνέχεια, θα δημιουργήσει την απάντηση στη μορφή που τη θέλει ο χρήστης και θα τη στείλει μέσω του internet στην πύλη, η οποία με την σειρά της θα την προωθήσει στο χρήστη μέσω του ασύρματου δικτύου.
8. Τελικά τα εστιατόρια θα εμφανιστούν στην οθόνη του χρήστη, είτε ως κείμενο είτε ως σημεία στο χάρτη και ο χρήστης επιλέγοντας ένα από αυτά θα μπορεί να δει τη συντομότερη διαδρομή που πρέπει να ακολουθήσει για να φτάσει σε αυτό.

2.1.6 Φορητές Συσκευές

Οι φορητές συσκευές που χρησιμοποιούν οι χρήστες ως τερματικά μπορεί να είναι είτε γενικού σκοπού είτε να προορίζονται αποκλειστικά για μια συγκεκριμένη υπηρεσία. Συσκευές γενικού σκοπού είναι τα κινητά τηλέφωνα, τα PDA's, οι φορητοί υπολογιστές(laptops) και τα Tablet PC's και μπορούν να χρησιμεύσουν στο χρήστη ως τερματικά για μια μεγάλη ποικιλία από υ-

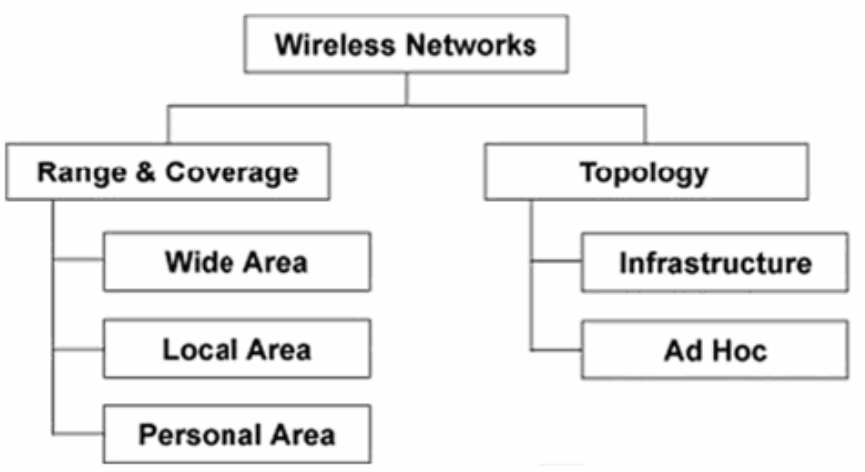
πηρεσίες που προσφέρονται ως Location Based Services. Από την άλλη, υπάρχουν και οι λεγόμενες συσκευές ειδικού σκοπού, όπως ένα τερματικό για πλοήγηση ενός αυτοκινήτου είτε μια συσκευή-πομπός για την κλήση σωστικών συνεργείων κτλ. Διάφορες τέτοιες συσκευές φαίνονται στο σχήμα (Σχήμα 6) που ακολουθεί:

Σχήμα 6: Είδη τερματικών για Location Based Services

Όλες οι παραπάνω συσκευές επιβάλλουν κάποιους περιορισμούς στις εφαρμογές για τις οποίες μπορούν να χρησιμοποιηθούν εξαιτίας τόσο του μεγέθους τους, όσο και της ενεργειακής αυτονομίας την οποία πρέπει να έχουν. Οι περιορισμοί αυτοί είναι κυρίως η μικρή επεξεργαστική ισχύ και η μικρή χωρητικότητα σε μνήμη. Ως εκ τούτου στο τερματικό του χρήστη δεν μπορεί να γίνονται εκτεταμένοι υπολογισμοί, ούτε να αποτιμώνται πολύπλοκα χωρικά ερωτήματα ή λειτουργίες εύρεσης συντομότερης διαδρομής και δημιουργίας χάρτη εξατομικευμένου στο χρήστη. Επιπλέον περιορισμοί επιβάλλονται από τις μικρές δυνατότητες απεικόνισης των συσκευών αυτών και της μικρής κατανάλωσης σε ενέργεια που πρέπει να έχουν. Προβλήματα μπορεί να δημιουργηθούν επίσης από καιρικές συνθήκες(π.χ. σε μεγάλη ηλιοφάνεια δυσχεραίνεται η χρήση της οθόνης). Φυσικά, για κάθε τύπο υπηρεσίας είναι πολύ πιθανό να προκύπτουν και περιορισμοί σχετικά με τον τύπο δικτύωσης της συσκευής και το εύρος ζώνης του δικτύου. Ωστόσο, όλοι οι παραπάνω περιορισμοί γίνονται όλο και πιο ασήμαντοι με το πέρασμα του χρόνου, καθώς η επεξεργαστική ισχύς των συσκευών αυτών αυξάνεται συνεχώς, μαζί με το μέγεθος της διαθέσιμης μνήμης τους και την ενεργειακή τους αυτονομία.

2.1.7 Ασύρματα Δίκτυα

Τα ασύρματα δίκτυα μπορούν να διακριθούν είτε με βάση τη χωρική τους *εμβέλεια*, που καθορίζεται από το σκοπό για τον οποίο είναι φτιαγμένο το δίκτυο και τις φυσικές ιδιότητες των ραδιοκυμάτων, και από την *τοπολογία* τους, δηλαδή αν υπάρχει κάποια δομή στο δίκτυο (κάποιοι ακίνητοι και στατικοί κόμβοι μέσω των οποίων συνδέονται όλοι οι υπόλοιποι στο δίκτυο) ή αν οι ίδιοι κινητοί χρήστες αποτελούν τους κόμβους μέσω τον οποίων συνδέονται άλλοι χρήστες(ad-hoc περιβάλλον), όπως φαίνεται στο Σχήμα 7:

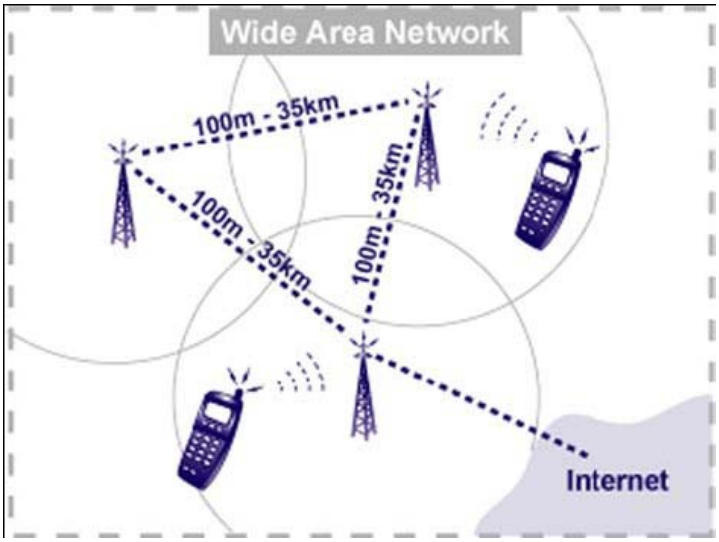


Σχήμα 7: Κατηγοριοποίηση Ασυρμάτων Δικτύων

Σε σχέση με την εμβέλειά τους τα ασύρματα δίκτυα μπορούν να χωριστούν σε **Ασύρματα Δίκτυα Ευρείας Περιοχής (Wireless Wide Area Networks – WWAN's)**, για παράδειγμα δίκτυα GSM και UMTS, **Τοπικά Ασύρματα Δίκτυα (Wireless Local Area Networks – WLAN's)**, όπως το IEEE 802.11 και τα **Προσωπικά Ασύρματα Δίκτυα (Wireless Personal Area Networks – WPAN's)**, όπως ένα δίκτυο με Bluetooth ή υπέρυθρες(IrDA).

2.1.7.1 Wireless Wide Area Networks – WWAN's

Οι κυψέλες των δικτύων του τύπου αυτού καλύπτουν αποστάσεις από 100 m έως και 35 km, ενώ το φάσμα των συχνοτήτων που χρησιμοποιείται δεν είναι συνήθως ελεύθερο, πράγμα που σημαίνει ότι απαιτείται άδεια για τη λειτουργία τους(και ως εκ τούτου δεν μπορεί να το χρησιμοποιεί και κάποιος άλλος). Η πρώτη γενιά τέτοιων δικτύων(αναλογικά G1) προορίζονταν κυρίως για μεταφορά φωνής και ως εκ τούτου ο ρυθμός μετάδοσης δεδομένων ήταν πολύ χαμηλός(4.8 kbps). Ακολούθησε η δεύτερη γενιά δικτύων (ψηφιακά G2) και συγκεκριμένα τα δίκτυα GSM και GPRS που μπορούν να υποστηρίξουν μεγαλύτερους ρυθμούς μετάδοσης δεδομένων(9.6 – 14 kbps για το GSM και 20 – 115 kbps για το GPRS). Ακόμη όμως και αυτοί οι ρυθμοί μετάδοσης δεν είναι κατάλληλοι για εφαρμογές πολυμέσων (multimedia).

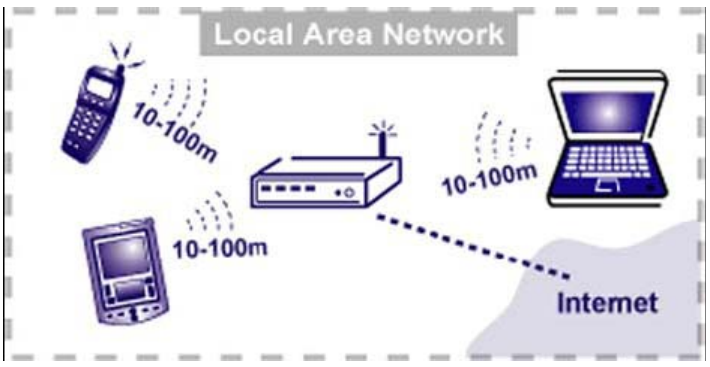


Σχήμα 8: Παράδειγμα ασύρματου δικτύου ευρείας περιοχής

Γι' αυτό το λόγο αναπτύχθηκαν τα λεγόμενα δίκτυα τρίτης γενιάς, όπως είναι το UMTS, που μπορεί να υποστηρίξει μετάδοση δεδομένων σε ρυθμούς που φτάνουν τα 2 Mbps. Τα τελευταία έχουν το μειονέκτημα ότι η εμβέλεια των κυψελών τους είναι μικρότερη (καθώς η κάλυψη που παρέχουν εξαρτάται από την πυκνότητα των χρηστών και τη χρήση που κάνουν) με αποτέλεσμα να απαιτούνται περισσότεροι σταθμοί βάσης και ως εκ τούτου το κόστος της δικτυακής υποδομής είναι μεγαλύτερο. Από την άλλη όμως, εκτός από τους μεγαλύτερους ρυθμούς μετάδοσης, μπορούν να υποστηρίξουν καλύτερη κρυπτογράφηση των δεδομένων (128 bits) και μεγαλύτερη ασφάλεια, ενώ παρέχουν και δυνατότητες για πιο ακριβή εύρεση της θέσης ενός τερματικού.

2.1.7.2 Wireless Local Area Networks – WLAN's

Τα WLAN's μπορούν να καλύψουν αποστάσεις από 10 ως 150 m(ή ακόμα και 300 m σε εξωτερικούς χώρους), ενώ το φάσμα συχνοτήτων που χρησιμοποιούν δεν απαιτεί νομική άδεια χρήσης.



Σχήμα 9: Παράδειγμα ασύρματου τοπικού δικτύου

Επίσης, οι ρυθμοί μετάδοσης δεδομένων είναι πολύ μεγαλύτεροι από αυτούς της προηγούμενης κατηγορίας δικτύων, καθώς φτάνουν ακόμα και τα 100 Mbps, ενώ είναι τα πλέον κατάλληλα για μεταφορά δεδομένων, καθώς αποτελούν επέκταση των κλασσικών δικτύων υπολογιστών. Επιπλέον, τα τερματικά δε συνδέονται σε σταθμούς βάσης(όπως στα WWAN's), αλλά στα λεγόμενα σημεία πρόσβασης(*access points*) που είναι πολύ πιο απλά και φθηνά, ενώ μπορούν να λειτουργήσουν ως ad-hoc δηλαδή ένα τερματικό να συνδέεται απευθείας σε κάποιο άλλο γειτονικό του τερματικό.

2.1.7.3 Wireless Personal Area Networks – WPAN's

Η κατηγορία αυτή των δικτύων παρέχει συνδεσιμότητα μικρής εμβέλειας (της τάξης των 10 – 20 m) κυρίως για συσκευές όπως ασύρματα ακουστικά. Χρησιμοποιεί ελεύθερο φάσμα συχνοτήτων και οι ρυθμοί μετάδοσης κυμαίνονται γύρω στο 0.5 Mbps, ενώ δεν υπάρχει συγκεκριμένη δομή στο δίκτυο(οι συσκευές συνδέονται ad-hoc). Τα περισσότερα δίκτυα του συγκεκριμένου τύπου βασίζονται στο πρότυπο Bluetooth και σε σχέση με τα WLAN's προσφέρουν μεγαλύτερη ασφάλεια και καλύτερη υποστήριξη για μεταφορά φωνής.

Όλα τα παραπάνω μπορούν να συνοψιστούν ως εξής:

Network technology		Average range	Data Rate (Mbps)	frequency domain
WWAN	GSM (G2)	base station distance 100m-35km	0.009-0.014	~ 900 MHz, licensed Spectrum
	GPRS		0.160	
	UTMS (G3)		2.0	
WLAN	Ultra-Wideband	10m	100	~ 2.4 & 5 GHz, not licensed Spectrum
	IEEE 802.11a	50m	54	
	IEEE 802.11b	100m	11	
WPAN	Bluetooth	10m	1	~ 2.4 GHz, not licensed Spectrum
	HomeRF	50m	10	
	IrDA (infra-red)	1-1.5m, needs line of sight	1-16	not licensed spectrum

Σχήμα 10: Ιδιότητες ασυρμάτων δικτύων συνοπτικά

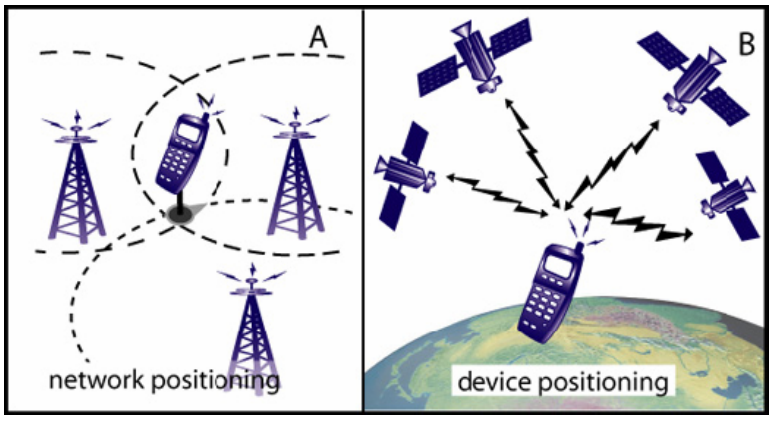
Από τα παραπάνω προκύπτει ότι δίκτυα τύπου WLAN και WPAN είναι κατάλληλα για εφαρμογές που απαιτούν μεγάλη χωρική διακριτικότητα(granularity) και κυρίως για εσωτερικούς χώρους όπως είναι ένα δημόσιο κτίριο, ένα μουσείο ή ένα πολυκατάστημα. Από την άλλη μεριά τα WWAN's μπορούν να υποστηρίξουν εφαρμογές σε μεγαλύτερη κλίμακα(όπως η διαχείριση στόλου και η τηλεματική) και περισσότερο εφαρμογές με μικρό θεματικό εύρος.

2.1.8 Μέθοδοι Εύρεσης Θέσης και Ακρίβεια

Παρακάτω θεωρούμε ότι ο χρήστης δεν εισάγει χειρωνακτικά τη θέση του στο σύστημα, αλλά η θέση του καταγράφεται αυτόματα. Για την εύρεση της θέσης ενός χρήστη στο χώρο, υπάρχουν δύο προσεγγίσεις:

α) Εύρεση θέσης μέσω του τηλεπικοινωνιακού δικτύου. Με βάση αυτή την προσέγγιση, το τερματικό είτε ανιχνεύεται αυτόματα από το δίκτυο, είτε στέλνει κάποιο σήμα, που επιτρέπει στο δίκτυο να βρει τη θέση του.

β) Εύρεση θέσης απευθείας από το τερματικό. Σε αυτή την περίπτωση το ίδιο το τερματικό υπολογίζει τη θέση του με βάση σήματα που φθάνουν σε αυτό από σταθμούς βάσης. Το πιο γνωστό παράδειγμα τέτοιου τύπου εύρεσης θέσης είναι το Global Positioning System(GPS), στο οποίο οι σταθμοί βάσης είναι δορυφόροι.



Σχήμα 11: Τρόποι εύρεσης της θέσης του χρήστη στο χώρο

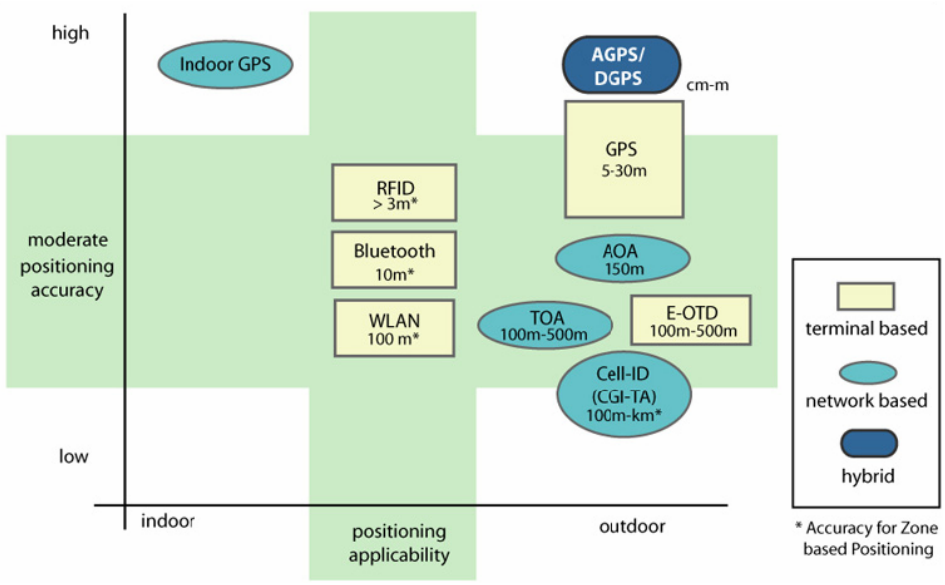
Με βάση τις δύο παραπάνω τεχνικές έχουν προκύψει πολλές καινούριες που τις συνδυάζουν. Οι βασικές αρχές στις οποίες στηρίζονται όλες οι τεχνικές εντοπισμού θέσης είναι:

- Οι σταθμοί βάσης έχουν γνωστή θέση
- Πληροφορία που προέρχεται από κάποιο τηλεπικοινωνιακό σήμα μεταφράζεται σε χωρικές αποστάσεις
- Ο υπολογισμός της απόλυτης θέσης του χρήστη γίνεται συνδυάζοντας τις παραπάνω σχετικές χωρικές αποστάσεις με την γνωστή απόλυτη θέση των σταθμών βάσης.

Οι πιο ευρέως χρησιμοποιούμενες τεχνικές είναι οι ακόλουθες:

- **Cell of Origin – COO** : Το αναγνωριστικό του κελιού του δικτύου είναι συνήθως αυτό που καθορίζει τον κοντινότερο σταθμό βάσης(π.χ. η κοντινότερη κεραία ενός δικτύου κινητής τηλεφωνίας). Με αυτή την τεχνική η θέση του χρήστη ορίζεται ως ένας κύκλος ή ένα κελί γύρω από τη γνωστή θέση των σταθμών βάσης..
- **Time of Arrival – TOA** : Με δεδομένο ότι τα ηλεκτρομαγνητικά κύματα διαδίδονται με την ταχύτητα του φωτός, όταν γνωρίζει κάποιος το χρονικό διάστημα που μεσολαβεί από την αποστολή ως τη λήψη ενός σήματος, τότε η απόσταση που το κύμα διάνυσε μπορεί εύκολα να υπολογιστεί. Το μειονέκτημα είναι ότι εξαιτίας της πολύ μεγάλης ταχύτητας διάδοσης του φωτός, ο χρόνος το παραπάνω χρονικό διάστημα είναι εξαιρετικά μικρό, με αποτέλεσμα να απαιτείται πολύ ακριβής μέτρηση του χρόνου.
- **Time Difference of Arrival(TDOA), Enhanced Observed Time Difference (E-OTD)** : Αυτές οι τεχνικές επίσης υπολογίζουν αποστάσεις μετρώντας το χρόνο διάδοσης κυμάτων, αλλά χρησιμοποιούν τη χρονική διαφορά μεταξύ σημάτων που προέρχονται από τρεις(συνήθως) διαφορετικούς σταθμούς βάσης, με αποτέλεσμα ο υπολογισμός της θέσης να μπορεί να γίνεται με μεγαλύτερη ακρίβεια. Στην περίπτωση του TDOA ο υπολογισμός της απόστασης γίνεται από τον παροχέα του δικτύου, ενώ στο E-OTD γίνεται από την ίδια τη φορητή συσκευή.
- **Angle of Arrival (AOA), Direction of Arrival (DOA)** : Σε περιπτώσεις που χρησιμοποιούνται κεραίες με κατευθυντικά χαρακτηριστικά, η γωνία άφιξης του κύματος στη συσκευή μπορεί να βρεθεί και έτσι στη συνέχεια να βρεθεί και η θέση. Εξαιτίας της διαρκούς κίνησης όμως των χρηστών και των τερματικών τους, η μέθοδος αυτή δε δίνει πολύ ακριβή αποτελέσματα

Οι πιο διαδεδομένες τεχνικές εντοπισμού θέσης είναι το GPS(που όπως αναφέρθηκε οι σταθμοί βάσης είναι δορυφόροι) και η εκτίμηση της θέσης με βάση το Cell-ID του κοντινότερου σταθμού βάσης κινητής τηλεφωνίας. Μεταξύ των δύο αυτών το GPS μπορεί να δώσει σαφώς ακριβέστερη πληροφορία(αστοχία της τάξης μόλις των 5 m), ενώ η δεύτερη μέθοδος μας δίνει ακρίβεια 100 m. Παρ' όλα αυτά, το GPS δεν είναι κατάλληλο για εσωτερικούς χώρους, όπου τεχνολογίες WLAN και WPAN μπορούν να μας δώσουν πολύ καλύτερης ποιότητας αποτελέσματα. Όπως εξηγείται καλύτερα από το σχήμα που έπεται, ο γενικός κανόνας είναι ότι οι τεχνικές εντοπισμού θέσης μέσω του δικτύου προτιμώνται όταν για τα Location Based Services που μας ενδιαφέρουν η ακριβής γνώση της θέσης δεν είναι υψηλής σημασίας. Αντίθετα, όταν οι υπηρεσίες απαιτούν μεγάλη ακρίβεια στη γνώση της θέσης του χρήστη, προτιμώνται τεχνικές εντοπισμού θέσης απευθείας από το τερματικό:



Σχήμα 12: Ακρίβεια τεχνικών εντοπισμού θέσης

2.1.9 Απαιτήσεις για ένα σύστημα Location Based Services

Όπως έχει επισημανθεί και παραπάνω, ένα σύστημα που προσφέρει υπηρεσίες με βάση τη θέση προορίζεται συνήθως να εξυπηρετεί ένα μεγάλο αριθμό χρηστών. Γι' αυτό το λόγο κάθε αρχιτεκτονική συστήματος για Location Based Services πρέπει να ικανοποιεί κάποιες απαιτήσεις, οι σημαντικότερες από τις οποίες είναι:

- **Υψηλές Επιδόσεις** : Τα ερωτήματα των χρηστών πρέπει να απαντώνται σε πραγματικό χρόνο, καθώς συνήθως ο κινούμενος χρήστης συνήθως πρέπει να πάρει μια γρήγορη απόφαση με βάση την απάντηση στο ερώτημά του.
- **Scalable Αρχιτεκτονική** : Το σύστημα θα πρέπει να μπορεί να λειτουργήσει εξίσου αποδοτικά με μεγάλο αριθμό χρηστών, καθώς τέτοιου είδους υπηρεσίες προορίζονται για χρήση σε μεγάλη πληθυσμιακή κλίμακα.
- **Αξιοπιστία** : Όπως και κάθε on-line σύστημα, έτσι και ένα σύστημα που προσφέρει υπηρεσίες με βάσης τη θέση πρέπει να δουλεύει αξιόπιστα, λειτουργώντας 24 ώρες το 24ωρο.
- **Ενημερωμένο** : Ένα τέτοιο σύστημα πρέπει να παρέχει πληροφορίες τις οποίες αποκτά δυναμικά και τις οποίες ανανεώνει συνεχώς ώστε να είναι ενημερωμένο.
- **Ανοιχτής Αρχιτεκτονικής** : Το σύστημα πρέπει να χρησιμοποιεί τεχνολογίες που βασίζονται σε κοινά και ευρέως αποδεκτά πρότυπα, όπως για παράδειγμα το Hypertext Transfer Protocol(HTTP), Wireless Markup Language(WML), Extensible Markup Language(XML) κτλ.
- **Ασφαλές** : Το σύστημα πρέπει να υιοθετεί τεχνικές για την ασφάλεια των δεδομένων και της πληροφορίας που αποθηκεύει και διανέμει στους χρήστες, καθώς και να κρυπτογραφεί

δεδομένα που αφορούν τους χρήστες και τις προτιμήσεις τους, ώστε να εξασφαλίζει ότι οι πληροφορίες που μεταφέρονται από και προς τους χρήστες είναι αφενός σωστές και αφ' ετέρου κρυφές προς οποιονδήποτε τρίτο.

2.2 Ταίριασμα Προφίλ (Profile Matching & Publish/Subscribe Systems)

2.2.1 Γενικά

Σε ένα σύστημα που προσφέρει Υπηρεσίες με Βάση το Προφίλ και τη Θέση, μια από τις πιο σημαντικές συνιστώσες είναι η τεχνική με βάση την οποία αποφασίζεται ποιες από τις πληροφορίες (*content*) που προσφέρει ο κάθε πάροχος φθάνουν σε κάθε χρήστη. Όπως αναφέρθηκε και παραπάνω, κάθε χρήστης θέλει να λαμβάνει πληροφορίες και δεδομένα μόνο για τις θεματικές κατηγορίες τις οποίες έχει επιλέξει. Ταυτόχρονα κάθε πάροχος επιθυμεί οι πληροφορίες του να διαχέονται μόνο σε χρήστες που πληρούν κάποια κριτήρια (διαφορετικά για κάθε πάροχο). Έτσι, οι πληροφορίες ξεκινούν από τους παρόχους και φτάνουν στους χρηστές μέσω ενός δικτύου διασύνδεσης περνώντας διάφορα στάδια φιλτραρίσματος. Οι πληροφορίες που κατευθύνουν αυτήν την επιλεκτική διαδικασία είναι οι προτιμήσεις των χρηστών και των παρόχων σε συνδυασμό με τη γεωγραφική τους θέση. Στο εξής αναφερόμαστε στο σύνολο των ιδιαιτέρων προτιμήσεων και επιλογών του κάθε χρήστη ως *προφίλ* (profile) του χρήστη. Αντίστοιχα υπάρχει προφίλ και για κάθε πάροχο, το οποίο περιγράφει τις προτιμήσεις του παρόχου σχετικά με το τι πληροφορίες θέλει να προσφέρει στο δίκτυο και σε ποιους χρήστες. Ως εκ τούτου για να αποφασίσει το σύστημα σε ποιους χρήστες θα στείλει ποια πληροφορία, αρκεί να αποφασίσει ποια προφίλ παροχέων ταιριάζουν με προφίλ χρηστών(που βρίσκονται σε κατάλληλη γεωγραφικά περιοχή). Η διαδικασία αυτή ονομάζεται *Ταίριασμα Προφίλ (Profile Matching)*.Η συνιστώσα του συστήματος που διαχειρίζεται τις δηλώσεις γνωρισμάτων των χρηστών και αναλαμβάνει το ταίριασμα με τα προφίλ χρηστών που ορίζονται από τους παρόχους στο εξής θα αναφέρεται ως *διαχειριστής των προφίλ* (*profile manager*).

Στην κατασκευή ενός τέτοιου συστήματος με τα παραπάνω χαρακτηριστικά υπάρχουν δύο αντικρουόμενες επιδιώξεις. Αφ' ενός η επιδίωξη προσωποποιημένης παροχής των υπηρεσιών που επιτυγχάνεται με την εκφραστική ικανότητα (expressiveness) στον καθορισμό των χαρακτηριστικών των χρηστών και των επιθυμητών προφίλ τους από την πλευρά των παρόχων των υπηρεσιών. Αυτή η επιδίωξη επιτυγχάνεται με τη υιοθέτηση σύνθετων μοντέλων δεδομένων που επιτρέπουν τη χρήση υπολογιστικά ακριβών αλλά και αποτελεσματικών τεχνικών φιλτραρίσματος. Αφ' ετέρου η επιδίωξη της υλοποίησης σε μεγάλη κλίμακα - που αφορά την περιοχή κάλυψης, τον αριθμό των χρηστών αλλά και των παρόχων – και της επεκτασιμότητας ενός τέ-

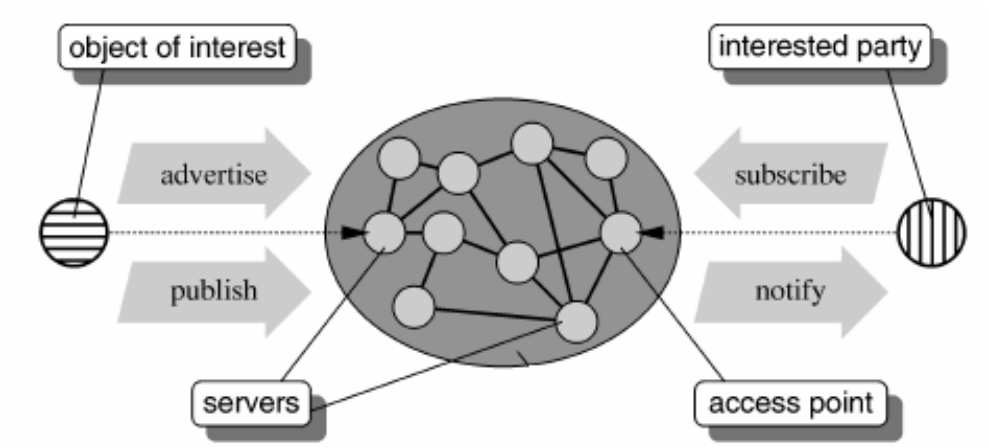
τοιου συστήματος θέτουν όρια σχετικά με τις υπολογιστικές απαιτήσεις που πρέπει να έχει οποιαδήποτε επιλεκτική διαδικασία που μεσολαβεί μεταξύ του παρόχου μιας υπηρεσίας και του τελικού χρήστη.

Όπως εξηγήθηκε και παραπάνω, μια από τις πιο σημαντικές παραμέτρους είναι η ανάγκη για αποκρίσεις του συστήματος σε πραγματικό χρόνο, γεγονός που επιτάσσεται από την κίνηση των χρηστών αλλά και τη φύση των υπηρεσιών που παρέχονται. Αν και ο ρυθμός αλλαγής γνωρισμάτων των χρηστών είναι αργός, η συνεχής μεταβολή της θέσης τους που λαμβάνεται η υπόψη για την παροχή των υπηρεσιών καθιστά αναγκαία την επεξεργασία σε πραγματικό χρόνο ρευμάτων δεδομένων (data streams) μεγάλου όγκου πληροφοριών.

Το μοντέλο ενός τέτοιου συστήματος φαίνεται στο σχήμα που ακολουθεί. Στο μοντέλο αυτό παρουσιάζονται οι βασικές συνιστώσες και εργασίες του συστήματος. Αυτές είναι:

- Η εγγραφή του χρήστη (subscription) ως αποδέκτη υπηρεσιών με ταυτόχρονη δήλωση των χαρακτηριστικών του γνωρισμάτων. Από εκεί στο εξής βέβαια η ροή πληροφοριών από το χρήστη προς το σύστημα είναι συνεχής ώστε να είναι δυνατός ο εντοπισμός του
- Η δημοσιοποίηση (publish) υπηρεσιών από τους παρόχους με ταυτόχρονη δήλωση του προφίλ των αποδεκτών των υπηρεσιών
- Οι υπηρεσίες που παρέχονται στους χρήστες
- Το δίκτυο διασύνδεσης

Ακριβώς επειδή σε ένα τέτοιο σύστημα οι πάροχοι δημοσιεύουν (*publish*) υπηρεσίες και οι χρήστες εγγράφονται (*subscribe*) σε αυτές, τα συστήματα αυτά αναφέρονται και ως *Publish/Subscribe Systems*.



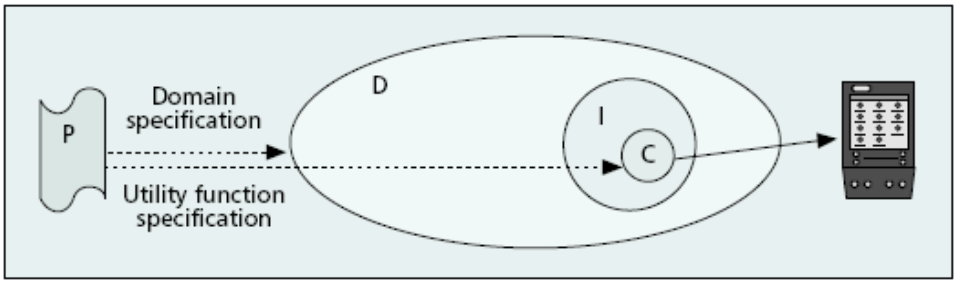
Σχήμα 13: Δομή ενός Συστήματος Δημοσιεύσεων / Εγγραφών (Publish/Subscribe System)

Παρακάτω μελετούμε τη γλώσσα με την οποία μπορούν να δηλωθούν τα γνωρίσματα και τα προφίλ των χρηστών και κυρίως εστιάζουμε σε τεχνικές για αποδοτικό ταίριασμα των προφίλ μέσα σε ένα τέτοιο σύστημα.

2.2.2 Γλώσσα περιγραφής γνωρισμάτων

Πριν αναφερθούμε στις γλώσσες περιγραφής γνωρισμάτων και επιλογής προφίλ χρηστών θα απεικονίσουμε πως οι ιδιαιτερότητες στον καθορισμό των προφίλ επηρεάζουν τη συμπεριφορά του διαχειριστή των προφίλ (*profile manager*). Στο επόμενο σχήμα απεικονίζεται ένα προφίλ P στα πλαίσια ενός συστήματος για Παροχή Υπηρεσιών με Βάση τη Θέση και το Προφίλ. Τα σύνολα D, I και C αποτελούν προοδευτικά μικρότερα σύνολα χρηστών τα οποία καθορίζονται από το P και τον διαχειριστή προφίλ που επεξεργάζεται το P. Το σύνολο D αποτελεί το χώρο των χρηστών που μπορούν να περιγράφουν με βάση τη γλώσσα γνωρισμάτων που χρησιμοποιείται. Αποτελεί ένα ιδεατό σύνολο που περιλαμβάνει όλους τους χωρικά διασπαρμένους χρήστες που δυνητικά μπορούν να εκφράσουν τη διαφορετικότητά τους μέσω της γλώσσας δήλωσης γνωρισμάτων.

Το σύνολο I αποτελεί το σύνολο των χρηστών οι οποίοι τελικά αποτελούν την «είσοδο» του διαχειριστή προφίλ. Σε περιπτώσεις που δεν χρησιμοποιείται κανενός είδους βελτιστοποίηση αυτοί οι χρήστες είναι το σύνολο των χρηστών που είναι παρόντες σε μια δεδομένη στιγμή στο σύστημα. Ο έξυπνος περιορισμός αυτού του συνόλου αποτελεί κρίσιμο παράγοντα για την επεκτασιμότητα του συστήματος. Τέλος, το σύνολο C αποτελεί το σύνολο των χρηστών που επιλέγονται τελικά από το διαχειριστή των προφίλ και στους οποίους είναι διαθέσιμες οι υπηρεσίες του παρόχου.



Σχήμα 14: Χώροι προφίλ

Ο χώρος D καθορίζεται πλήρως από τη γλώσσα που χρησιμοποιείται για την έκφραση των γνωρισμάτων ενός χρήστη. Η γλώσσα αυτή έχει προφανώς δηλωτικό χαρακτήρα. Σε απλουστευμένη μορφή η δήλωση των γνωρισμάτων μπορεί να επιτευχθεί με ζεύγη της μορφής:

<*γνώρισμα*> - <*τιμή*> || <*τιμές*>

Όλα τα γνωρίσματα των χρηστών σε αυτήν την περίπτωση έχουν ένα τύπο, ένα όνομα και μία τιμή. Τα ονόματα είναι απλές συμβολοσειρές, ενώ οι τύποι ανήκουν σε ένα προκαθορισμένο σύνολο πρωταρχικών τύπων (primitive types) όπως οι τύποι μία γλώσσας προγραμματισμού. Αυτή η απλουστευμένη μορφή δήλωσης διευκολύνει την αποθήκευση των γνωρισμάτων των χρηστών ως εγγραφές σε έναν πίνακα μίας σχεσιακής ΒΔ. Το σύνολο D είναι σε αυτήν την περίπτωση το εξωτερικό γινόμενο των συνόλων τιμών των πεδίων της σχέσης. Αν και με τη χρή-

ση μια τέτοιας γλώσσας θα μπορούσαμε να χρησιμοποιήσουμε τα ισχυρά εργαλεία που έχουμε στη διάθεσή μας για τη διαχείριση των ΒΔ, ένα τέτοιο μοντέλο δεδομένων για τον ορισμό των γνωρισμάτων των χρηστών παρουσιάζει εγγενείς αδυναμίες.

Κατ' αρχήν η έκφραση των γνωρισμάτων των χρηστών περιορίζεται από το σχήμα της ΒΔ. Το σύνολο των γνωρισμάτων των χρηστών είναι εκ των προτέρων γνωστό και οι χρήστες καλούνται απλά να δώσουν τιμές σε αυτά τα γνωρίσματα. Αυτό το μοντέλο δίνει προτεραιότητα στις τιμές των γνωρισμάτων (δεδομένα), ενώ σε έναν ορισμό ενός προφίλ παρουσιάζουν εξ ίσου ενδιαφέρον και τα μετα-δεδομένα (γνωρίσματα) στη δήλωση που κάνει ο κάθε χρήστης. Επίσης, το μοντέλο αυτό υφίσταται τους περιορισμούς που επιβάλλει το σχεσιακό μοντέλο στην αναπαράσταση της δομής των γνωρισμάτων.

Ένας εναλλακτικός τρόπος για τη δήλωση των γνωρισμάτων ενός χρήστη είναι μέσω XML εγγράφων. Αν και για να γίνει στη συνέχεια αποδοτικά το ταίριασμα προφίλ απαιτείται η χρήση ενός DTD εγγράφου, η χρήση XML δίνει μεγαλύτερη ελευθερία στο χρήστη. Τα γνωρίσματα του οργανώνονται στο XML έγγραφο απεικονίζοντας τη φυσική τους οργάνωση. Επίσης, η απουσία εξάρτησης από ένα καθολικό σχήμα μίας ΒΔ παρέχει τη δυνατότητα δήλωσης γνωρισμάτων που εκφράζουν μεν το χρήστη αλλά μία δεδομένη στιγμή δεν παρουσιάζουν ενδιαφέρον για το σύστημα. Ο ορισμός ενός νέου προφίλ χρηστών στη συνέχεια από έναν πάροχο μπορεί να αξιοποιήσει χωρίς αλλαγές στο σύστημα αυτήν την πληροφορία.

Αν και η χρηστική του αξία για ένα σύστημα με τέτοιες υπολογιστικές απαιτήσεις είναι μηδαμινή αξίζει να αναφερθεί το μοντέλο της απλής δήλωσης του προφίλ του χρήστη με τη μορφή κειμένου. Σε αυτό το μοντέλο ο χρήστης έχει πλήρη ελευθερία να δηλώσει τα προσωπικά του γνωρίσματα και τις προτιμήσεις. Από πλευράς εκφραστικότητας το μοντέλο αυτό υπερέχει των παραπάνω. Ωστόσο η άναρχη και μη δομημένη δήλωση των γνωρισμάτων δεν διευκολύνει την επεξεργασία και ορισμό των προφίλ χρηστών.

2.2.3 Γλώσσα για τον Ορισμό Προφίλ

Η γλώσσα που ορίζει τα προφίλ των χρηστών είναι κατ' ουσίαν μια γλώσσα που αναλαμβάνει την κατηγοριοποίηση τους (*classification*) με βάση τα γνωρίσματά τους. Πρόκειται για μία γλώσσα ερωτημάτων που είναι άμεσα εξαρτημένη από τη γλώσσα ορισμού των γνωρισμάτων. Παρακάτω παρουσιάζονται τέτοιες γλώσσες για κάθε μία από τις περιπτώσεις που εξετάστηκαν παραπάνω:

2.2.3.1 Εκτεταμένη SQL

Στην περίπτωση που οι προτιμήσεις και τα γνωρίσματα των χρηστών αποθηκεύονται σε πίνακες μίας βάσης δεδομένων η γλώσσα που χρησιμοποιείται για τον ορισμό των προφίλ αποτελεί μία

επέκταση της SQL [SCZ05]. Η επέκταση είναι απαραίτητη καθώς στον ορισμό των προφίλ περιέχονται και χωροχρονικοί περιορισμοί που δεν υποστηρίζονται από την κλασική SQL.

Κάθε τύπος ενός γνωρίσματος έχει ένα καθορισμένο σύνολο τελεστών που μπορούν να εφαρμοστούν σε αυτόν. Ένα κατηγόρημα μπορεί να είναι σύνθετο ή απλό (ισότητας, ανισότητας, εγγύτητας κτλ.) εκμεταλλευόμενο την ισχύ των τελεστών του συγκεκριμένου τύπου. Τελικά κάθε προφίλ είναι ένας συνδυασμός σταθερών κατηγορημάτων (constant predicates) κατηγορημάτων ένωσης (join predicates) σε πίνακες της σχεσιακής βάσης δεδομένων. Παράδειγμα ενός προφίλ χρηστών για τη διαφήμιση ενός δίσκου που πωλείται από ένα δισκοπωλείο στη θέση Θέση_Δισκοπωλείου είναι το εξής:

$$age > 14 \text{ AND } age < 18 \text{ AND } gender = female \text{ AND}$$

$$distance(current\_location, Θέση\_Δισκοπωλείου) < 1\ km$$

Η χρονική παράμετρος υπεισέρχεται στο παραπάνω προφίλ χρηστών από την επιλογή του χρόνου και τη διάρκεια δημοσίευσης ενός τέτοιου προφίλ.

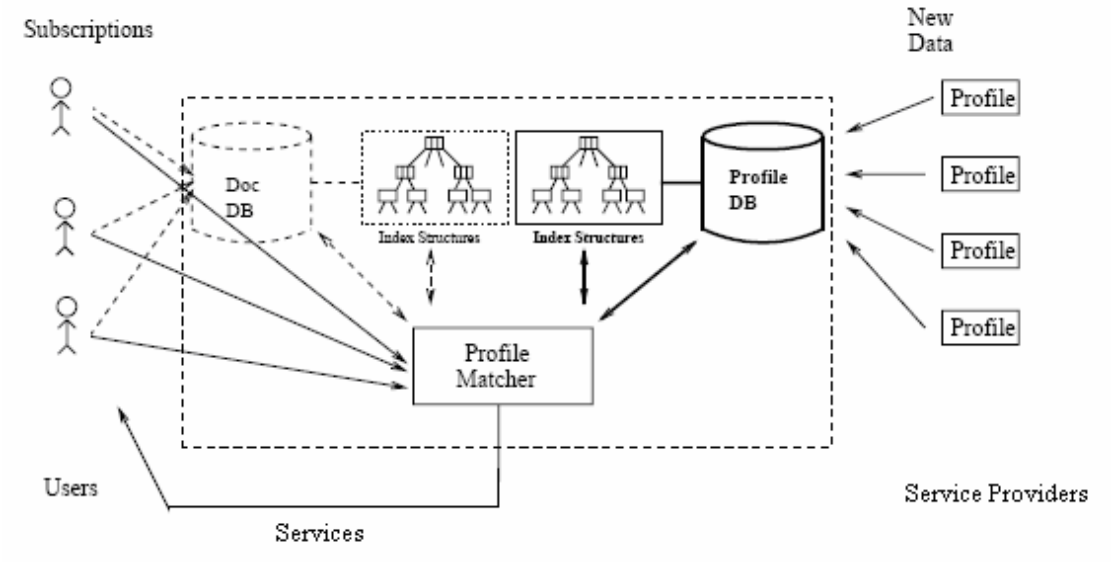
Σε αυτήν την περίπτωση η αποδοτική υλοποίηση ταιριάσματος προφίλ ανάγεται στην αποδοτική υλοποίηση δεικτών (indices) για σταθερά κατηγορήματα και κατηγορήματα ένωσης. Η αναλυτική παρουσίαση αυτών των δεικτών ξεφεύγει από τα όρια τις συγκεκριμένης εργασίας. Θα παρατεθούν απλά οι κυριότεροι εκπρόσωποί τους και θα αναφερθούν τα γενικά τους χαρακτηριστικά.

Δομές δεδομένων που έχουν χρησιμοποιηθεί για τη δεικτοδότηση σταθερών κατηγορημάτων είναι [Tat01]: Cluster Index, Interval Index, Hybrid Index που έχουν βασιστεί στις γεωμετρικές δομές δεδομένων Segment Tree [Bentley, 1977], Interval Tree [Edelsbrunner, 1980]. Τα κυριότερα τους χαρακτηριστικά είναι:

Ένα δέντρο δεικτοδότησης για κάθε πεδίο (attribute)

- Χρήση κατακερματισμού (hash) στα ονόματα των πεδίων
- Τα δέντρα δεικτοδότησης είναι μία παραλλαγή των δυαδικών δέντρων αναζήτησης
- Δύο διαφορετικές μέθοδοι για την κατασκευή των δέντρων δεικτοδότησης:
    - Με βάση τις συνοριακές τιμές των διαστημάτων
    - Με βάση τα στοιχειώδη διαστήματα μεταξύ διαδοχικών συνοριακών τιμών
- Ανύσματα δυαδικών ψηφίων (bit vectors) που υποδεικνύουν:
    - Ποια προφίλ ενδεχομένως ταιριάζουν (στον Cluster Index)
    - Ποια προφίλ σίγουρα ταιριάζουν (στους άλλους δείκτες)

Για τους δείκτες που μπορούν να χρησιμοποιηθούν στην περίπτωση των κατηγορημάτων ένωσης δεν μπορεί να γίνει τέτοια συνοπτική αναφορά. Στο Σχήμα 15 φαίνεται η αρχιτεκτονική όταν χρησιμοποιούνται οι παραπάνω δείκτες:



Σχήμα 15: Ταίριασμα με προφίλ με σταθερά κατηγορήματα (constant predicates)

Ενδιαφέρον σε αυτόν τον τρόπο ορισμού των προφίλ παρουσιάζουν και οι εργασίες που έχουν γίνει πάνω σε συνεχή ερωτήματα σε βάσεις δεομένων (*continuous queries*).

2.2.3.2 Χρήση της XPath ως γλώσσας ορισμού προφίλ

Στη δεύτερη περίπτωση έχουμε τη χρήση εγγράφων *XML* για τη δήλωση των γνωρισμάτων των χρηστών. Επειδή στα XML έγγραφα η δήλωση των γνωρισμάτων των χρηστών γίνεται με δομημένο τρόπο, ο ορισμός ενός προφίλ μπορεί εκμεταλλεύεται αυτήν την μετα-πληροφορία. Συνεπώς για τον ορισμό ενός προφίλ μπορούν να υποβληθούν πιο σύνθετα ερωτήματα.

Μια γλώσσα ερωτημάτων για XML έγγραφα είναι η *XPath* (http://www.w3.org/TR/xpath20/). Η XPath παρέχει έναν ευέλικτο τρόπο για τον καθορισμό εκφράσεων μονοπατιών. Με την XPath ένα XML έγγραφο αντιμετωπίζεται σαν ένα δέντρο κόμβων. Μέσα σε αυτό το δέντρο δεν αναζητούνται απλά τιμές γνωρισμάτων στους κόμβους, αλλά μπορεί να καθοριστούν χαρακτηριστικά των μονοπατιών που οδηγούν στους κόμβους αυτούς. Τελικά οι εκφράσεις-ερωτήματα της XPath αναζητούν μοτίβα(*patterns*) που εμφανίζονται σε ένα XML έγγραφο και επιστρέφουν είτε ένα σύνολο κόμβων είτε τιμές των κόμβων.

Τα μονοπάτια προς τους ζητούμενους κόμβους μπορούν να οριστούν είτε με απόλυτο τρόπο είτε σε σχέση με άλλους γνωστούς κόμβους. Κάθε ερώτημα της XPath ξεκινά με περιγραφή της θέσης του ζητούμενου χαρακτηριστικού. Οι ιεραρχικές σχέσεις μεταξύ των κόμβων εκφράζονται στο ερώτημα χρησιμοποιώντας είτε τον τελεστή *πατέρα-παιδιού* (*'/'*) είτε τον τελεστή *προ-*

γόνου-απογόνου(*'//'*). Π.χ. με το παρακάτω ερώτημα αναζητούμε όλα αγαπημένα συγκροτήματα ενός χρήστη:

/profile//music/bands

Τα συγκροτήματα αυτά είναι παιδιά του κόμβου music που βρίσκεται στο μονοπάτι με του κόμβου profile. Σε κάθε κόμβο του μονοπατιού που ορίζεται στο XPath ερώτημα μπορούν να τεθούν επιπλέον κριτήρια («*κατηγορήματα*») φιλτραρίσματος. Τα κριτήρια αυτά εσωκλείονται σε αγκύλες ('[', ']') και εφαρμόζονται κάθε φορά στο τρέχον στοιχείο υπό εξέταση. Σε κάθε θέση ενός εγγράφου πρέπει όλα τα κατηγορήματα που υπάρχουν στην έκφραση XPath να αποτιμώνται σε TRUE για να έχουμε ταίριασμα με το εξεταζόμενο έγγραφο.

Στην περίπτωση που η XPath χρησιμοποιείται για το ταίριασμα προφίλ, κάθε ερώτημα της θεωρείται ένα σύνθετο κατηγόρημα για ένα έγγραφο XML. Αν υπάρχει έστω ένα στοιχείο στο έγγραφο που ικανοποιεί το ερώτημα XPath τότε η αναζήτηση ολοκληρώνεται πρόωρα. Το έγγραφο – δήλωση γνωρισμάτων – θεωρείται ότι ανήκει στο καθορισμένο προφίλ.

Για το αποδοτικό ταίριασμα προφίλ με χρήση XPath έχουν αναπτυχθεί διάφορες τεχνικές, οι πιο σημαντικές από τις οποίες περιγράφονται σε επόμενη παράγραφο.

2.2.3.3 Ορισμός προφίλ χρήση λέξεων-κλειδιών

Αυτό το μοντέλο δεν έχει εφαρμογή στο είδος του συστήματος που εξετάζουμε. Ωστόσο, παρατίθεται χωρίς λεπτομέρειες για λόγους πληρότητας. Χρησιμοποιείται όταν οι χρήστες καθορίζουν σε ελεύθερο κείμενο τα γνωρίσματα τους. Τότε ο καθορισμός προφίλ έχει χαρακτηριστικά μηχανής αναζήτησης. Ένα προφίλ καθορίζεται με χρήση λέξεων-κλειδιών που αναζητούνται στα κείμενα δήλωσης γνωρισμάτων. Με τη χρήση διάφορων κριτηρίων (π.χ. σχετική θέση λέξεων-κλειδιών στο έγγραφο) αξιολογούνται οι διάφοροι χρήστες ως προς το βαθμό ικανοποίησης των περιορισμών του προφίλ. Τελικά, βάσει κυρίως ευριστικών μεθόδων, γίνεται τελικά το ταίριασμα προφίλ.

2.2.4 Τεχνικές για το αποδοτικό Ταίριασμα Προφίλ

Αφού έχουν εξεταστεί οι γλώσσες που χρησιμοποιούνται για τον ορισμό των γνωρισμάτων και προφίλ των χρηστών μπορούμε τώρα να δούμε τεχνικές που χρησιμοποιούν τα παραπάνω σε συστήματα για Υπηρεσίες με Βάση το Προφίλ κα τη Θέση σε μεγάλη κλίμακα. Σε αυτά τα συστήματα είναι χαρακτηριστική η συνεχής ροή πληροφοριών από τους χρήστες προς το σύστημα που πρέπει να συνδυαστεί με πληροφορίες που έχουν μεγαλύτερη περίοδο αλλαγών (γνωρίσματα χρήστη εκτός της τρέχουσας θέσης).

Σε πραγματικές συνθήκες και χωρίς τη χρήστη βελτιστοποιήσεων θα είχαμε ροή πληροφοριών από τους χρήστες προς τους παρόχους των υπηρεσιών μέσω ενός δικτύου διασύνδεσης. Οι πά-

ροχοι χρησιμοποιώντας τεχνικές ταιριάσματος προφίλ επιλέγουν σε ποιους χρήστες επιθυμούν να παρέχουν τις υπηρεσίες τους. Μια τέτοια στρατηγική έχει όμως αδυναμίες που καθιστούν ανέφικτη την υλοποίησή της σε μεγάλη κλίμακα. Αφ' ενός θα είχαμε πλημμύρα στο δίκτυο διασύνδεσης και αφ' ετέρου κάθε πάροχος θα ήταν υποχρεωμένος να έχει την επεξεργαστική ισχύ για το φιλτράρισμα σε πραγματικό χρόνο των πληροφοριών όλων των χρηστών.

Επειδή και τα δύο είναι ανέφικτα, έχουν προταθεί τεχνικές για κατανεμημένο ταίριασμα των προφίλ που ξεπερνά τις δυσκολίες που αναφέρονται παραπάνω. Θα παρουσιαστούν στη συνέχεια – μόνο το σκέλος του που αφορά στο ταίριασμα προφίλ – οι τεχνικές που χρησιμοποιούν τα συστήματα Wide-Area Event Notification Service [CRW01] και SemCast [PC05].

2.2.4.1 *Wide-Area Event Notification Service*

Πρόκειται για ένα σύστημα ευρείας κλίμακας που χρησιμοποιείται για τη μετάδοση ειδοποιήσεων με σημασιολογικά κριτήρια. Στο σύστημα δε λαμβάνεται υπόψη η μετακίνηση των χρηστών και συνεπώς τα χαρακτηριστικά τους έχουν μεγαλύτερο βαθμό στατικότητας. Σε αυτήν την περίπτωση οι ρόλοι είναι αντιστραμμένοι σε σχέση με το σύστημά μας. Οι χρήστες είναι αυτοί που υποβάλλουν συνεχή ερωτήματα (*persistent queries*) προς πηγές πληροφοριών που μεταβάλλονται σε σχέση με το χρόνο. Παρά τις διαφορετικές παραδοχές, οι τεχνικές που χρησιμοποιούνται για ταίριασμα προφίλ μπορούν να μεταφερθούν στα συστήματα που εξετάζουμε.

Θεμέλιο για τη βελτιστοποίηση που επιτυγχάνεται στο ταίριασμα προφίλ είναι ο ορισμός μίας διμερούς σχέσης μεταξύ των προφίλ. Αυτή η σχέση μεταξύ των προφίλ p, q που στο εξής θα αναφέρεται ως *covers(p, q)* σημαίνει πως το σύνολο των χρηστών που ικανοποιούν τα κριτήρια του προφίλ p είναι υπερσύνολο των χρηστών που ικανοποιούν τα κριτήρια του προφίλ του q. Στην περίπτωση που χρησιμοποιούνται σταθερά κατηγορήματα για τον ορισμό προφίλ ένα παράδειγμα είναι το εξής:

$$p: age > 21 \text{ AND } brothers > 2$$

$$q_1: age > 30 \text{ AND } brothers = 1$$

$$q_2: age = 39$$

Για τα παραπάνω προφίλ ισχύουν οι σχέσεις:

$$\text{covers}(p, q_1)$$

$$\text{covers}(p, q_2)$$

$$\text{covers}(q_1, q_2)$$

Η φιλοσοφία του συστήματος είναι η δημιουργία δέντρων ελάχιστης κάλυψης μεταξύ ενός χρήστη και των παρόχων υπηρεσιών με προφίλ χρηστών που συμπεριλαμβάνει το συγκεκριμένο χρήστη. Στη συνέχεια έχω πολυμετάδοση (*multicasting*) των πληροφοριών των χρηστών μέσω των παραπάνω δέντρων. Ο πάροχος της υπηρεσίας μπορεί να εντοπίσει το χρήστη από τις

πληροφορίες που λαμβάνει και του καταστήσει διαθέσιμες τις υπηρεσίες του. Με αυτόν τον τρόπο μειώνεται το απαιτούμενο εύρος ζώνης από το δίκτυο διασύνδεσης μεταξύ χρηστών και παρόχων των υπηρεσιών. Επίσης το ταίριασμα των προφίλ με τα γνωρίσματα των χρηστών γίνεται πλέον κατανεμημένα από όλους τους κόμβους του δικτύου διασύνδεσης και έτσι επιτυγχάνεται η επεκτασιμότητα του συστήματος.

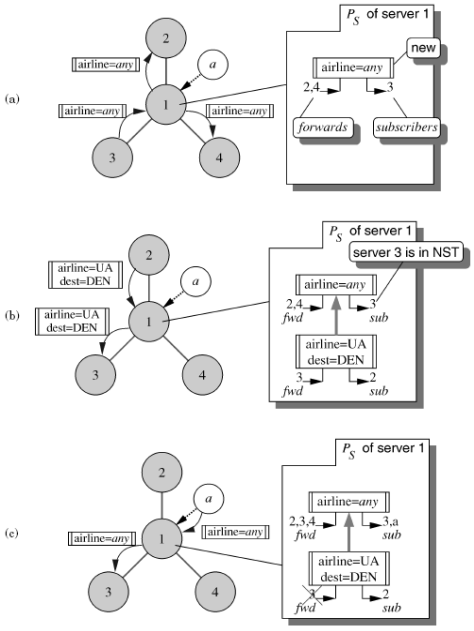
Μένει όμως να δούμε τον αλγόριθμο που χρησιμοποιείται για το σχηματισμό των δέντρων ελάχιστης κάλυψης. Ο αλγόριθμος είναι δυναμικός. Εφαρμόζεται σε κάθε δημοσίευση ενός προφίλ από παρόχους υπηρεσιών και έχει ως αποτέλεσμα την αποθήκευση πληροφοριών δρομολόγησης των εισερχόμενων πληροφοριών σε όλους τους κόμβους του ενδιάμεσους δικτύου. Προϋπόθεση του αλγορίθμου είναι η τοπολογία του δικτύου να είναι ακυκλικός γράφος. Σε διαφορετική περίπτωση χρειάζονται επιπλέον τεχνικές (όπως πεδίων TTL στο IP κτλ.) για να αποφευχθούν ατέρμονες κυκλικές πληροφορίες δεδομένων στο δίκτυο.

Ένας πάροχος μία υπηρεσίας δημοσιεύει σε έναν κόμβο το προφίλ *f* των χρηστών στους οποίους είναι πρόθυμος να παρέχει την υπηρεσία του. Με αρχή τον κόμβο δημοσίευσης κάθε κόμβος προωθεί το προφίλ που του δημοσιεύεται στους γειτονικούς κόμβους σύμφωνα με τη σχέση:

$$forwards(f) = neighbours - NST(f) - \bigcup_{covers(f',f)} forwards(f')$$

Δηλαδή η δημοσίευση γίνεται σε όλους τους γείτονες εκτός αυτών που:

- Δεν ανήκουν σε ένα δέντρο ελάχιστης κάλυψης προς υποψήφιους χρήστες
- Είναι στο μονοπάτι προς παρόχους υπηρεσιών με προφίλ που καλύπτει το νεοδημοσιευθέν προφίλ.



Σχήμα 16: Δημοσίευση προφίλ σε γειτονικούς κόμβους

Η εφαρμογή του αλγορίθμου κατά τη δημοσίευση νέων προφίλ χρηστών φαίνεται στο Σχήμα 16. Στην εικόνα (a) φαίνεται η προώθηση μίας νέας δημοσίευσης προφίλ χρηστών σε όλους τους κόμβους εκτός αυτών που δεν βρίσκονται σε δέντρο ελάχιστης κάλυψης (3). Στις εικόνες (b) και (c) φαίνεται πως χρησιμοποιείται η σχέση covers για την προώθηση ή μη στους γειτονικούς κόμβους μίας νέας δημοσίευσης προφίλ για παροχή υπηρεσιών.

Στη συνέχεια η προώθηση των δεδομένων που φτάνουν από τους χρήστες γίνεται μόνο στους κόμβους που κρατούν προφίλ με τα οποία επιτυγχάνεται το ταίριασμα. Έτσι γίνεται δυνατή η επιλεκτική μετάδοση των πληροφοριών και η μερική υπέρβαση των προβλημάτων επεκτασιμότητας που αναφέρονται στην εισαγωγή της τρέχουσας παραγράφου.

2.2.4.2 SemCast

Το *SemCast* είναι ένα σύστημα που επιτρέπει τη σημασιολογική επιλεκτική μετάδοση ροών δεδομένων με βάση το περιεχόμενό τους (*Semantic Multicast for Content-based Stream Dissemination*). Η φιλοσοφία του συστήματος είναι η πλήρης αποκέντρωση των επιλογών δρομολόγησης των εισερχόμενων πληροφοριών με μείωση του υπολογιστικού φορτίου των ενδιάμεσων κόμβων σε σχέση με το σύστημα που παρουσιάστηκε στην προηγούμενη παράγραφο.

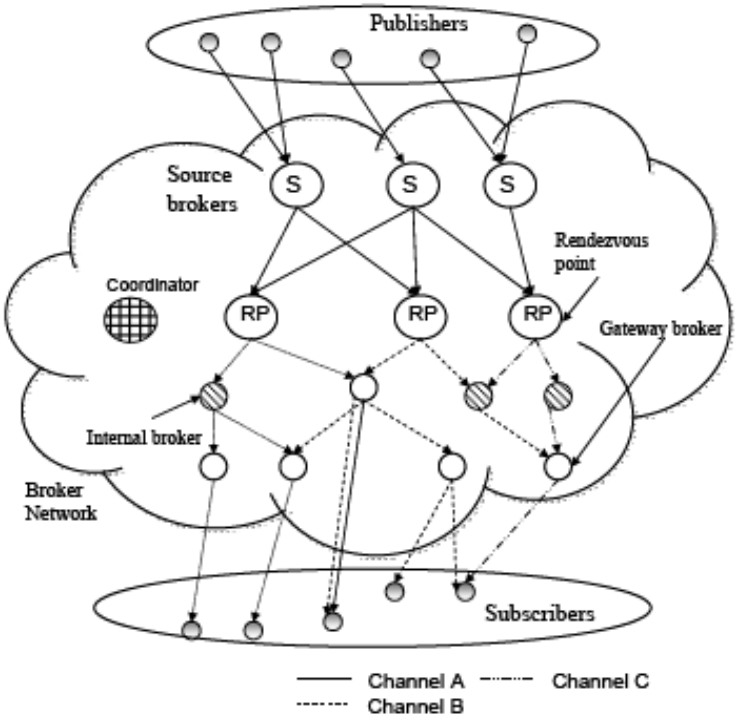
Όλα τα νέα δεδομένα που εισάγονται στο δίκτυο (στην περίπτωση του δικού μας συστήματος οι τρέχουσες θέσεις των χρηστών και οι δηλώσεις των γνωρισμάτων τους) αντιστοιχίζονται σε ένα συγκεκριμένο σημασιολογικό κανάλι πολλαπλής μετάδοσης (semantic multicast channel) και στη συνέχεια προωθούνται από το ενδιάμεσο δίκτυο προς τους παρόχους των υπηρεσιών με στοιχειώδεις επιλογές προώθησης, κοινές για όλα τα δεδομένα που μεταφέρονται μέσα από το ίδιο σημασιολογικό κανάλι. Έτσι κάθε ενδιάμεσος κόμβος του δικτύου, σε αντίθεση με ότι συνέβαινε στην προηγούμενη τεχνική, δεν επιβαρύνεται με σημασιολογικούς ελέγχους των μηνυμάτων που μεταφέρει κάτι που απαιτείται όταν έχω έλεγχο για ταίριασμα προφίλ σε κάθε κόμβο του ενδιάμεσου δικτύου.

Με τη σημασιολογική διάκριση των εισερχόμενων ρευμάτων δεδομένων σε κανάλια το SemCast έχει επίσης της δυνατότητα εφαρμογής πολιτικών δρομολόγησης που ενισχύουν την αξιοπιστία του συστήματος και βελτιστοποιούν την ποιότητα των παρεχόμενων υπηρεσιών χωρίς να εξαρτάται άμεσα από την τοπολογία του δικτύου.

Στο SemCast οι ρόλοι είναι αντίστροφη με αυτούς που έχουμε υποθέσει ως τώρα. Οι Publishers δημοσιεύουν XML έγγραφα. Από την άλλη οι subscribers υποβάλλουν ερωτήματα με τη χρήση της XPath και ύστερα από το κατάλληλο ταίριασμα προφίλ γίνονται αποδέκτες των XML εγγράφων που τους ταιριάζουν.

Το σύστημα αποτελείται από ένα σύνολο ενδιάμεσων κόμβων (*brokers*) που οργανώνονται σαν ένα δίκτυο P2P. Οι πάροχοι υπηρεσιών και οι πελάτες συνδέονται σε κόμβους – πύλες (*gateway brokers*). Το σύστημα προϋποθέτει ακόμα την ύπαρξη ενός κόμβου συντονιστή (*coordinator*).

Ο κόμβος αυτός είναι υπεύθυνος για τη διατήρηση των καναλιών της επιλεκτικής μετάδοσης πληροφοριών και τον καθορισμό του σημασιολογικού τους περιεχομένου. Επίσης, μερικοί κόμβοι καλούνται σημεία συνάντησης (*rendezvous points*). Καθένας από αυτούς είναι υπεύθυνος για τουλάχιστον ένα κανάλι μετάδοσης και λειτουργεί ως η ρίζα του δέντρου της πολλαπλής μετάδοσης. Το μοντέλο του συστήματος φαίνεται στο Σχήμα 17.



Σχήμα 17: Βασικό μοντέλο συστήματος SemCast

Οι κόμβοι-πύλες που δέχονται ένα XML έγγραφο καθορίζουν σε ποιο κανάλι πολλαπλής μετάδοσης πρέπει να μεταδοθεί. Το προωθεί έτσι στον κατάλληλο κόμβο – σημείο συνάντησης που είναι υπεύθυνος για την αποδοτική μετάδοση του εγγράφου. Η προώθηση από εκεί και εξής μέχρι τους subscribers αποτελεί ένα κλασικό πρόβλημα δρομολόγησης που βασίζεται στη χρήση επικεφαλίδων των προωθούμενων μηνυμάτων και πινάκων προώθησης.

Σημαντικό στη λειτουργία του συστήματος είναι η σημασιολογική κατάταξη των μηνυμάτων σε κανάλια. Για το σκοπό αυτό απαιτούνται ο προσδιορισμός του αριθμού των καναλιών, το περιεχόμενο του κάθε καναλιού, η απόφαση για τη χρήση ενός καναλιού με την άφιξη ενός νέου μηνύματος από έναν publisher και η σύνδεση κάθε subscriber με τα κατάλληλα κανάλια. Οι αλγόριθμοι που χρησιμοποιούνται για τα παραπάνω θέματα εξασφαλίζουν πως δεν υπάρχουν αποκλεισμοί, δηλαδή ότι κάθε XML μήνυμα φθάνει σε κάθε subscriber που ταιριάζει με το προφίλ του. Το SemCast εξασφαλίζει επίσης και τη χρήση μειωμένου εύρους ζώνης στο ενδιά-

μεσο δίκτυο, αλλά αυτό το κομμάτι των αλγορίθμων δε θα μας απασχολήσει στη συγκεκριμένη εργασία.

Για τη διάκριση των καναλιών χρησιμοποιούνται σχέσεις υπερκάλυψης μεταξύ των προφίλ ανάλογες με αυτές που χρησιμοποιούνται στο πρώτο σύστημα. Μία έκφραση XPath P_1 *περιέχει* μία άλλη έκφραση P_2 αν κάθε έγγραφο που ταιριάζει στην P_2 ταιριάζει και στην P_1.

Με την άφιξη ενός νέου μηνύματος οι κόμβοι πύλες ελέγχουν αν μπορούν να το προωθήσουν σε ένα από τα υπάρχοντα κανάλια, αν δηλαδή προωθούν ήδη μηνύματα με το ίδιο προφίλ. Διαφορετικά στέλνουν το σύστημα στον κόμβο συντονιστή. Ο συντονιστής αναζητά αν υπάρχει κάποιο κανάλι που το σημασιολογικό του περιεχόμενο υπερκαλύπτει το νέο μήνυμα. Αν δεν υπάρχει τότε δημιουργείται ένα νέο κανάλι για τα μηνύματα του αυτού προφίλ. Αν υπάρχει ακριβώς ένα τότε το μήνυμα ανατίθεται αυτό. Αν υπάρχουν περισσότερα τότε το μήνυμα ανατίθεται στο κανάλι με τη μικρότερη χρήση εύρους ζώνης. Ο κόμβος συντονιστής ενημερώνει τους κόμβους πύλες για την ανάθεση του νέου προφίλ.

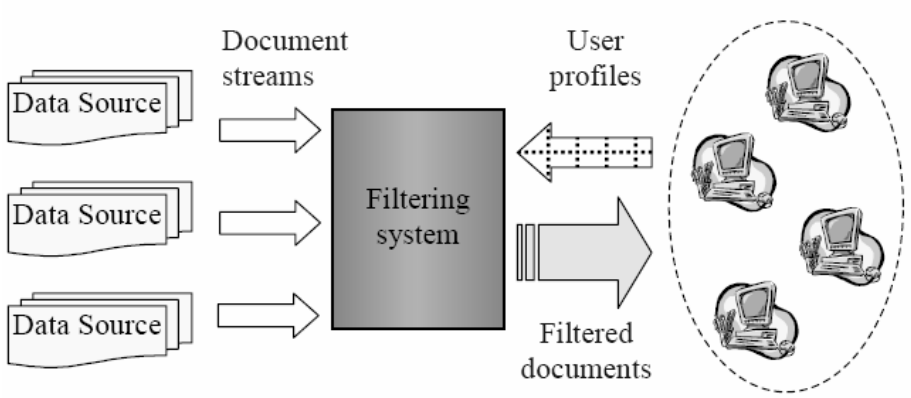
Τελικά μέσα σε κάθε κανάλι μεταδίδονται XML έγγραφα που έχουν μεταξύ τους σημασιολογική ιεραρχία (όπως καθορίζεται από την μεταξύ τους κάλυψη). Ο αλγόριθμος ταιριάσματος προφίλ επιβαρύνει μόνο τον κόμβο συντονιστή. Επειδή όμως το ταίριασμα αυτό γίνεται μία φορά για κάθε προφίλ το υπολογιστικό φορτίο δεν είναι μεγάλο και το σύστημα εξασφαλίζει την επεκτασιμότητά του σε μεγάλη κλίμακα.

2.2.5 Ταίριασμα Προφίλ με Φιλτράρισμα Εγγράφων XML

Γενικά, ένα σύστημα φιλτραρίσματος εγγράφων, όπως τα προφίλ των χρηστών και των παρόχων, φιλτράρει τα έγγραφα (και τις πληροφορίες που περιέχουν) με βάση τις προτιμήσεις που έχουν εκδηλώσει οι χρήστες. Υποθέτουμε ότι οι χρήστες εκδηλώνουν τις καταγράφουν τις προτιμήσεις τους σε ένα έγγραφο είτε χρησιμοποιώντας μια Γραφική Διασύνδεση Χρήστη (Graphical User Interface) και επιλέγοντας κάποια στοιχεία ή κατηγορίες είτε το σύστημα τις καταγράφει αυτόματα γι' αυτούς χρησιμοποιώντας τεχνικές μηχανικής μάθησης. Στη συνέχεια τα προφίλ των χρηστών μετατρέπονται σε μορφή τέτοια ώστε να μπορούν να αποθηκευτούν και στη συνέχεια να αποτιμηθούν με πολύ αποδοτικό τρόπο. Αυτά τα προφίλ μπορούν να θεωρηθούν και ως συνεχή ερωτήματα που τίθενται στο σύστημα και πρέπει συνεχώς το σύστημα να βρίσκει με ποια έγγραφα παρόχων ταιριάζουν. Γι' αυτό το λόγο σε πολλά σημεία θα αποκαλούμε τα προφίλ των χρηστών απλά ως ερωτήματα.

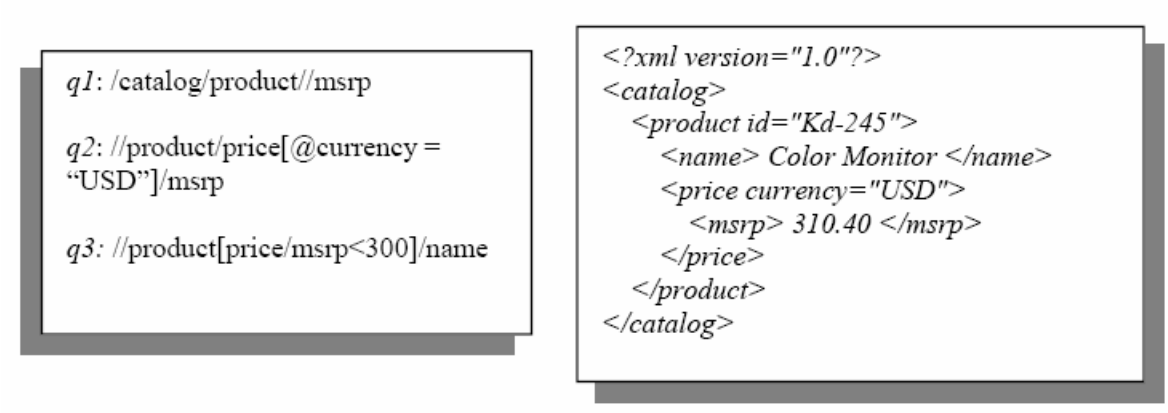
Εδώ υποθέτουμε ότι εκτός από τα ερωτήματα-προφίλ των χρηστών, έχουμε έγγραφα τα οποία πρέπει να φιλτραριστούν και να διανεμηθούν στους χρήστες ανάλογα με τις προτιμήσεις τους και φυσικά εγκαίρως. Το φιλτράρισμα γίνεται επιχειρώντας να ταιριάξουμε κάθε έγγραφο που εισέρχεται στο σύστημα με όλα τα προφίλ των χρηστών, ώστε να καταλήξουμε στο σύνολο των χρηστών που το προφίλ τους ταιριάζει με το νέο έγγραφο. Τότε το έγγραφο αυτό μπορεί να δια-

νεμηθεί σε καθέναν από τους χρήστες του συνόλου αυτού. Παρακάτω υποθέτουμε ότι τα έγγραφα καθώς εισέρχονται στο σύστημα τοποθετούνται σε μια ουρά αναμονής, με αποτέλεσμα να επεξεργαζόμαστε ένα έγγραφο κάθε φορά. Κάθε φορά βρίσκουμε *όλους* τους χρήστες που ενδιαφέρονται για το εκάστοτε έγγραφο και έτσι το έγγραφο αυτό μπορεί μετά να διαγραφεί από την ουρά. Ένα τέτοιο σύστημα φαίνεται καλύτερα στο Σχήμα 18.



Σχήμα 18: Ταίριασμα Προφίλ σε σύστημα Φιλτραρίσματος Προφίλ

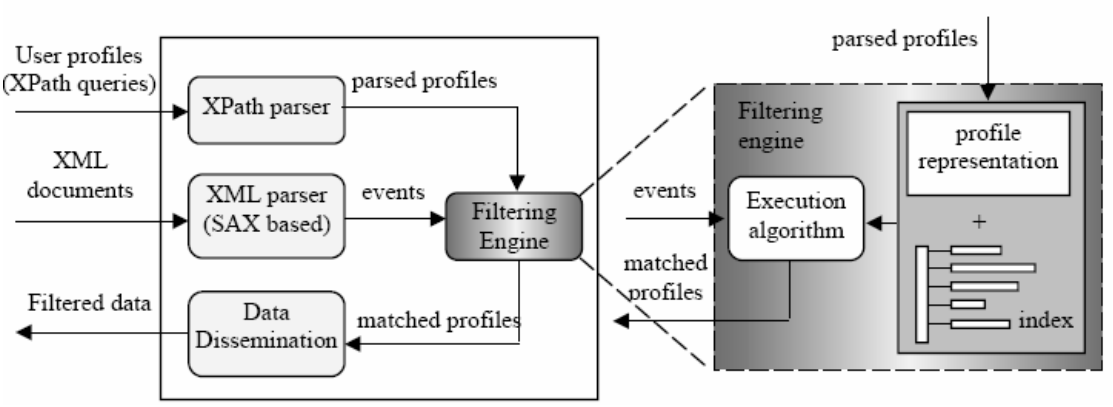
Για ένα σύστημα σαν το παραπάνω, που προορίζεται να χρησιμοποιηθεί σε μεγάλη κλίμακα (π.χ. internet), ο αριθμός των χρηστών μπορεί πολύ εύκολα να είναι τεράστιος. Για κάθε έγγραφο λοιπόν που καταφθάνει στο σύστημα πρέπει να εξεταστεί ένα τεράστιο σύνολο από χρήστες για να βρεθούν αυτοί που τους ενδιαφέρει το συγκεκριμένο έγγραφο. Στη συνέχεια περιγράφουμε μεθόδους που στοχεύουν στην επίλυση ακριβώς αυτού του προβλήματος.



Σχήμα 19: Παραδείγματα ερωτημάτων σε XPath και εγγράφου XML

Υποθέτουμε ότι τα έγγραφα που προσφέρουν οι πάροχοι του περιεχομένου στο σύστημα είναι έγγραφα XML. Όπως αναφέρεται και παραπάνω αυτό έχει το πλεονέκτημα ότι τα έγγραφα είναι ημι-δομημένα και ορισμένα ιεραρχικά(υπάρχει ένα στοιχείο-ρίζα το οποίο έχει υποστοιχεία

κτλ.), ενώ περιέχει και meta-πληροφορία σχετικά με τα δεδομένα που περιέχει. Σε αντιστοιχία με τα όσα ειπώθηκαν στην ενότητα «Γλώσσες για τον Ορισμό Προφίλ», θεωρούμε ότι τα προφίλ-ερωτήματα των χρηστών είναι εκφράσεις XPath. Επίσης θεωρούμε ότι κάθε έκφραση XPath επιλέγει ολόκληρα έγγραφα XML, δηλαδή αν μια έκφραση XPath αποτιμάται σε true για ένα στοιχείο (*element*) ενός XML εγγράφου, τότε θεωρούμε ότι όλο το έγγραφο ικανοποιεί την έκφραση. Σχηματικά, η αρχιτεκτονική του συστήματος που θα μελετήσουμε φαίνεται Σχήμα 20.



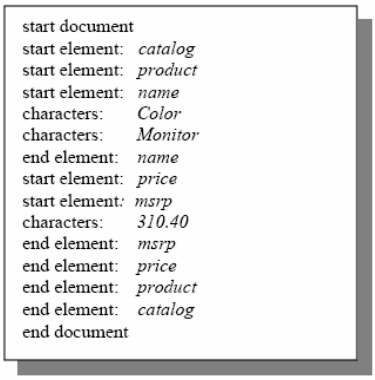
Σχήμα 20: Αρχιτεκτονική συστήματος Φιλτραρίσματος Προφίλ

Τα βασικά δομικά στοιχεία του συστήματος(που φαίνονται και στο προηγούμενο σχήμα) είναι τα ακόλουθα:

XPath parser: Ο συντακτικός αναλυτής (parser) για την XPath δέχεται ως είσοδο τις εκφράσεις XPath, τις αναλύει και στη συνέχεια μεταβιβάζει τα επεξεργασμένα αυτά προφίλ στη μηχανή φιλτραρίσματος (*filtering engine*) σε μια μορφή που να της είναι κατανοητή και κατάλληλη για να γίνει αποδοτικά το ταίριασμα με τα εισερχόμενα έγγραφα. Ένα καινούριο ερώτημα XPath μπορεί να προστεθεί στη μηχανή φιλτραρίσματος μόνο όταν αυτή δεν επεξεργάζεται κάποιο XML έγγραφο.

Event-based XML Parser (Συντακτικός Αναλυτής της XML Οδηγούμενος από Συμβάντα): Όταν ένα νέο XML έγγραφο καταφθάνει στο σύστημα, το σύστημα κάνει τη συντακτική του ανάλυση χρησιμοποιώντας ένα συντακτικό αναλυτή τύπου SAX, που είναι οδηγούμενος από συμβάντα (event-based). Το τελευταίο σημαίνει ότι καθώς ο συντακτικός αναλυτής κάνει τη συντακτική ανάλυση του εγγράφου, ειδοποιεί το χρήστη για κάθε γλωσσική συντακτική οντότητα που συναντά και εγείρει συμβάντα τα οποία χειρίζονται κάποιες συγκεκριμένες συναρτήσεις των οποίων τον κώδικα ο χρήστης μπορεί να τροποποιήσει ώστε να λάβει την επιθυμητή λειτουργικότητα. Για παράδειγμα όταν ο αναλυτής συναντήσει την αρχή και το τέλος του εγγράφου αντίστοιχα, εγείρει τα συμβάντα “startDocument” και “endDocument” αντίστοιχα. Παρόμοια, για κάθε στοιχείο (element) της XML ο αναλυτής εγείρει τα συμβάντα “startElement”

και “endElement”, ενώ το κείμενο που τυχόν εμπεριέχεται σε ένα στοιχείο εγείρει ένα συμβάν “characters”, όπως φαίνεται στο Σχήμα 21.



Σχήμα 21: Έγερση συμβάντων από συντακτικό αναλυτή

Οι συναρτήσεις χειρισμού τις οποίες ο χρήστης αναθέτει σε καθένα από τα συμβάντα που εγείρονται είναι αυτές που στο συγκεκριμένο σύστημα καθοδηγούν την εξέλιξη της διαδικασίας ταιριάσματος του εγγράφου με τα ερωτήματα – προφίλ χρηστών.

Μηχανή Φιλτραρίσματος(Filtering Engine): Η μηχανή φιλτραρίσματος αποτελεί τον πυρήνα του συστήματος, καθώς αφ’ ενός λαμβάνει τα συντακτικά αναλυμένα XPath προφίλ και τα μετατρέπει σε μια εσωτερική αναπαράσταση, αφ’ ετέρου λαμβάνει τα συμβάντα που εγείρονται από το συντακτικό αναλυτή του XML εγγράφου και αντιδρά αναλόγως, καλώντας κατάλληλες συναρτήσεις χειρισμού των συμβάντων τις οποίες η ίδια η μηχανή υλοποιεί, ώστε να κάνουν το ταίριασμα των προφίλ, όπως περιγράφεται παρακάτω.

Σε ένα σύστημα μεγάλης κλίμακας όπως το Internet ο αριθμός των χρηστών, και άρα ο αριθμός των προφίλ, είναι τόσο μεγάλος που η τυφλή αναζήτηση (brute force) στο σύνολο των προφίλ για κάθε νέο έγγραφο είναι ανέφικτη. Γι’ αυτό το λόγο πρέπει να υιοθετηθεί μια πιο έξυπνη προσέγγιση του προβλήματος, η οποία κάνει ακριβώς το αντίστροφο από αυτό που κάνουν τα παραδοσιακά συστήματα βάσεων δεδομένων για να ευρετηριάζουν τις αποθηκευμένες εγγραφές: σε μια παραδοσιακή βάση δεδομένων ένα μεγάλο σύνολο δεδομένων βρίσκεται αποθηκευμένο μόνιμα στο δίσκο. Τα ερωτήματα που έρχονται στη βάση από τους χρήστες, ένα κάθε φορά, αναζητούν αποτελέσματα στο μεγάλο αυτό σύνολο δεδομένων. Αντίθετα, στη μηχανή φιλτραρίσματος που περιγράφουμε εδώ, τα ερωτήματα είναι αυτά που αποθηκεύουμε μόνιμα και είναι τα εισερχόμενα έγγραφα αυτά που επιλέγουν τα ερωτήματα που τους ταιριάζουν. Για να αυξηθεί επομένως η αποδοτικότητα του συστήματος, το σύστημα υιοθετεί τεχνικές ευρετηρίασης (*indexing*) για τα ερωτήματα, για να πετύχει αυτό που οι παραδοσιακές βάσεις πετυχαίνουν στα δεδομένα, δηλαδή να ευρετηριάζουν τα δεδομένα χωρίς να απαιτείται η σειριακή τους αναζήτηση.

Συνθέτημα Διάδοσης των Πληροφοριών (Dissemination Component): Όταν για ένα έγγραφο βρεθεί το σύνολο των προφίλ με τα οποία ταιριάζει, τότε το έγγραφο αυτό πρέπει να αποσταλεί μέσω του δικτύου προς τους χρήστες αυτούς. Ο πιο εύκολος τρόπος να γίνει αυτό είναι η αποστολή με unicast σε όλους του χρήστες που ταιριάζουν. Για περιπτώσεις όμως που μας ενδιαφέρει η ελαχιστοποίηση του τηλεπικοινωνιακού κόστους, πρέπει να χρησιμοποιηθούν πιο εξελιγμένες τεχνικές, όπως για παράδειγμα να μη στέλνονται στους χρήστες ολόκληρα τα έγγραφα, αλλά μόνο τα κομμάτια (στοιχεία) από το κάθε έγγραφο που ταιριάζουν στο προφίλ του κάθε χρήστη.

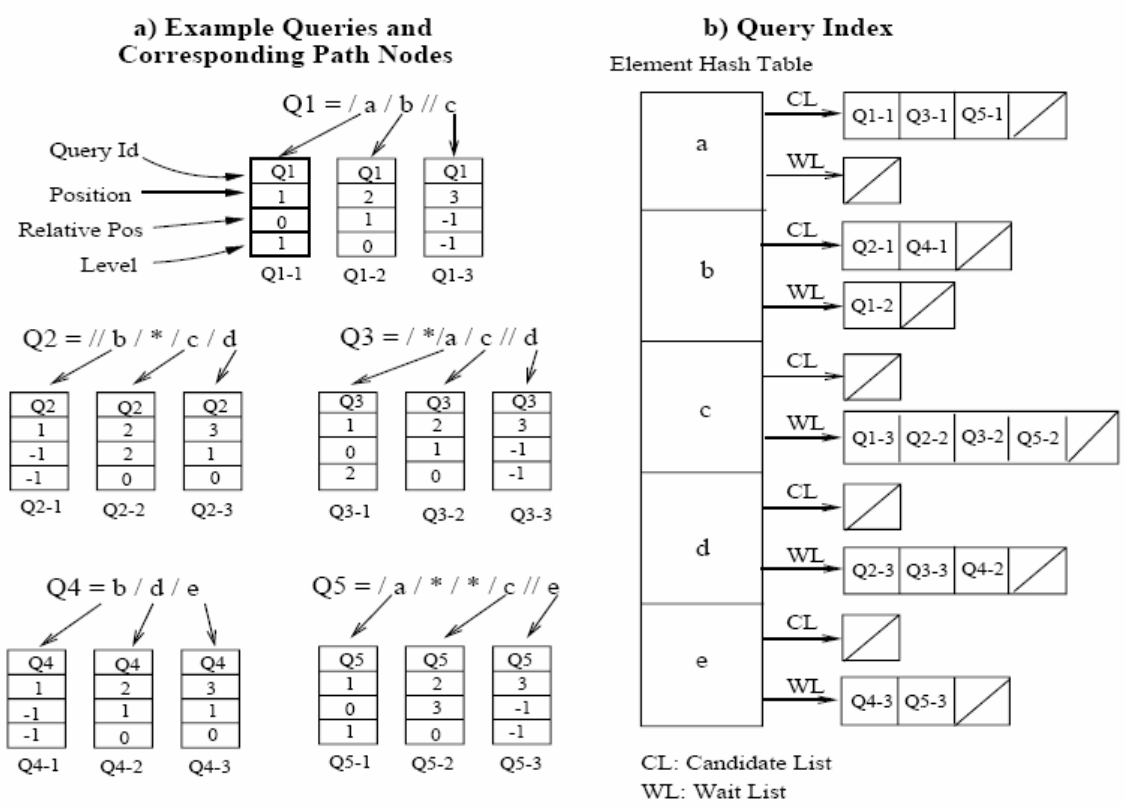
Με βάση το σενάριο για ταίριασμα προφίλ που έχει περιγραφεί παραπάνω, παρουσιάζουμε στη συνέχεια τεχνικές που χρησιμοποιούν Μηχανές Πεπερασμένων Καταστάσεων (Finite State Machines – **FSM**'s) για αποδοτικό ταίριασμα ενός XML εγγράφου με μεγάλο αριθμό XPath ερωτημάτων.

2.2.6 XFilter

Κάθε έκφραση μονοπατιού, όπως περιγράφηκε παραπάνω, αποτελείται από ονόματα στοιχείων (element names) και τελεστές «*» (wildcard operators) και από τους άξονες «/» και «//». Κάθε τέτοιο μονοπάτι μπορεί να περιγραφεί από μια κανονική έκφραση(*regular expression*). Ως εκ τούτου κάθε έκφραση μονοπατιού του παραπάνω τύπου μπορεί να παρασταθεί από μια Μηχανή Πεπερασμένων Καταστάσεων (*FSM*). Το XFilter ([DAF+03]) είναι μια τεχνική για το αποδοτικό φιλτράρισμα XML εγγράφων που βασίζεται στην παρατήρηση αυτή, δηλαδή μετατρέπει το κάθε ερώτημα XPath σε μια μηχανή πεπερασμένων καταστάσεων, η εκτέλεση της οποίας καθοδηγείται από α συμβάντα που εγείρει ο συντακτικός αναλυτής. Έτσι, θεωρούμε ότι ένα έγγραφο ταιριάζει σε ένα ερώτημα, δηλαδή στο προφίλ ενός χρήστη, όταν το FSM φτάσει σε μια τελική κατάσταση.

2.2.6.1 Αναπαράσταση των προφίλ

Κάθε ερώτημα XPath αποσυντίθεται σε ένα σύνολο *κόμβων μονοπατιού* (*path nodes)*, καθένας από τους οποίους αναπαριστά ένα κόμβο στοιχείου (element node) του ερωτήματος και αποτελεί μια κατάσταση της μηχανής πεπερασμένων καταστάσεων για το συγκεκριμένο ερώτημα(με εξαίρεση τα στοιχεία τύπου *, για τα οποία δε δημιουργείται κόμβος). Κάθε κόμβος μονοπατιού περιέχει τις πληροφορίες που φαίνονται στο Σχήμα 22 και εξηγούνται κατόπιν:



Σχήμα 22: Κόμβοι XFilter (α)

QueryID: Ένα μοναδικό αναγνωριστικό για την έκφραση μονοπατιού(path expression), στην οποία ανήκει ο συγκεκριμένος κόμβος.

Position: Ο αριθμός (όρος μιας ακολουθίας) που καθορίζει τη θέση του συγκεκριμένου κόμβου στην σειρά των κόμβων μονοπατιού που αντιστοιχούν στο συγκεκριμένο ερώτημα, δηλαδή την απόσταση σε κόμβους από τη ρίζα.

RelativePos: Ένας ακέραιος που περιγράφει την απόσταση σε επίπεδα(levels του εγγράφου) μεταξύ του συγκεκριμένου κόμβου και του προηγούμενου(ως προς τη θέση). Η τιμή αυτή είναι 0 για τον πρώτο κόμβο, αν αυτός δεν περιέχει τον τελεστή απογόνου(«//»). Κάθε κόμβος που χωρίζεται από τον προηγούμενό του από ένα τέτοιο τελεστή, έχει στο γνώρισμα αυτό την τιμή -1. Σε κάθε άλλη περίπτωση το γνώρισμα αυτό έχει την τιμή 1 αυξημένη κατά τον αριθμό των κόμβων τύπου «*» που υπάρχουν μεταξύ του κόμβου αυτού και του προγόνου του.

Level: Ένας ακέραιος που αναπαριστά το βάθος του XML εγγράφου στο οποίο θα πρέπει ο συγκεκριμένος κόμβος μονοπατιού να εξεταστεί. Επειδή σε ένα έγγραφο XML είναι δυνατό ένας συγκεκριμένος τύπος στοιχείου να εμφανίζεται σε διάφορα επίπεδα ενός εγγράφου και επειδή τα XPath ερωτήματα μπορεί να γίνονται χρησιμοποιώντας το σχετικό βάθος ενός στοιχείου ως προς συμφραζόμενα στοιχεία, η τιμή του γνωρίσματος αυτού δεν είναι πάντοτε δυνατό να καθοριστεί κατά τη συντακτική ανάλυση του εγγράφου. Γι' αυτό το λόγο η τιμή αυτή μπορεί να

αλλάξει κατά την *αποτίμηση* του ερωτήματος. Η τιμή του γνωρίσματος αυτού για ένα κόμβο που είναι το πρώτο στοιχείο ενός ερωτήματος και η απόστασή του από τη ρίζα είναι κατά απόλυτο τρόπο καθορισμένη τότε το γνώρισμα level για τον κόμβο αυτό έχει τιμή 1 + την απόστασή του από τη ρίζα. Αν το γνώρισμα RelativePos του κόμβου έχει τιμή -1, τότε επίσης και το level έχει τιμή -1, σε όλες τις υπόλοιπες περιπτώσεις το γνώρισμα αυτό έχει την τιμή 0.

NextPathNodeSet: Η τιμή του γνωρίσματος αυτού είναι ένας δείκτης προς τον επόμενο κόμβο μονοπατιού του ερωτήματος που πρέπει να αποτιμηθεί.

2.2.6.2 Δημιουργία του ευρετηρίου

Για να επιτύχουμε μεγαλύτερη αποδοτικότητα κατά το φιλτράρισμα των εισερχόμενων εγγράφων, το XFilter συντηρεί και ένα ανεστραμμένο ευρετήριο πάνω στα ερωτήματα, το οποίο ονομάζουμε Ευρετήριο Ερωτημάτων (*Query Index*) και το οποίο χρησιμοποιείται για να γίνεται αποδοτικά το ταίριασμα ενός εγγράφου με μεμονωμένα ερωτήματα. Όπως φαίνεται και στο δεξί μισό του προηγούμενου σχήματος, το query index είναι δομημένο ως ένας πίνακας κατακερματισμού (hash table) με κλειδί τα ονόματα στοιχείων (*element names*) που εμφανίζονται σε ερωτήματα XPath. Κάθε μοναδικό τέτοιο όνομα στοιχείο περιέχει δύο λίστες: τη *λίστα υποψηφίων* (*candidate list*) και τη *λίστα αναμονής* (*wait list*).

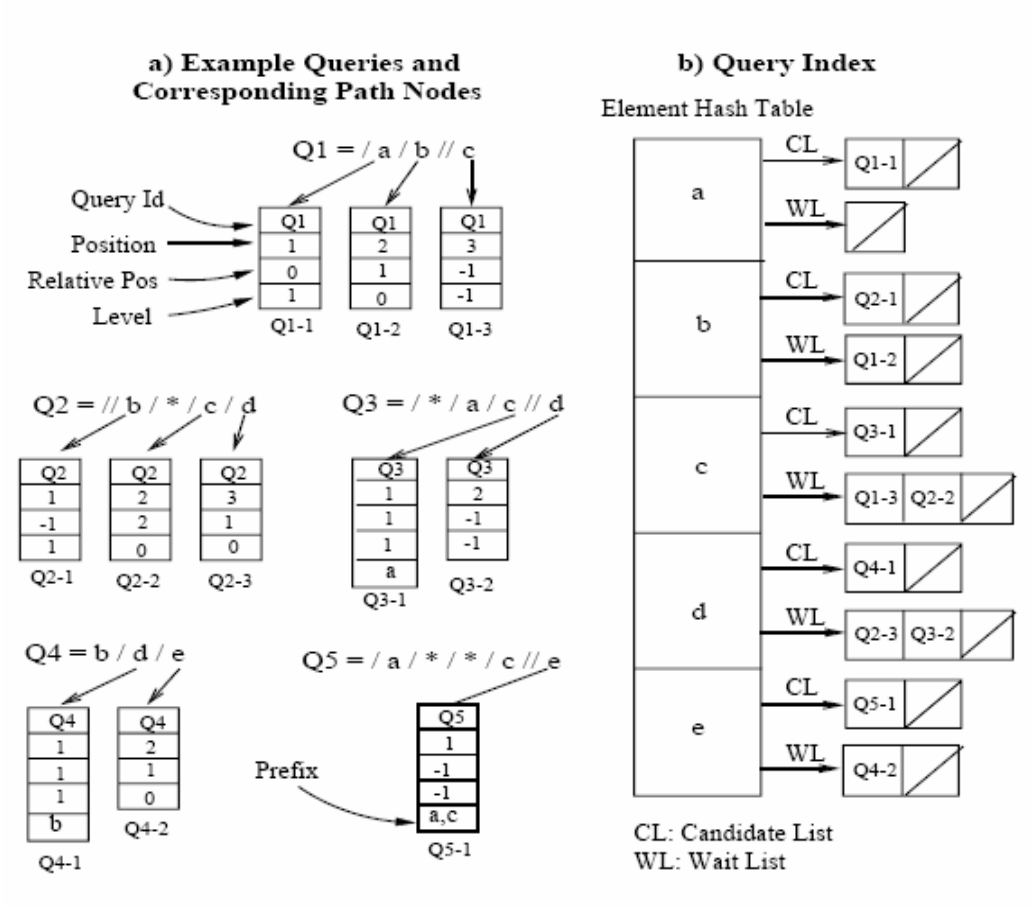
Οι λίστες υποψηφίων κρατάνε τους κόμβους μονοπατιού που αντιστοιχούν σε καταστάσεις, τις οποίες η μηχανή πεπερασμένων καταστάσεων προσπαθεί να ταιριάξει με το τρέχον έγγραφο σε μια δεδομένη χρονική στιγμή. Η λίστα αναμονής περιέχει κόμβους μονοπατιού που ακολουθούν τους κόμβους που βρίσκονται σε λίστες υποψηφίων. Το περιεχόμενο της κάθε λίστας υποψηφίων μεταβάλλεται συνεχώς καθώς τα συμβάντα που εγείρονται από το συντακτικό αναλυτή οδηγούν την εκτέλεση της μηχανής πεπερασμένων καταστάσεων.

Ένας σημαντικός παράγοντας που καθορίζει το πόσο αποδοτικό είναι το XFilter είναι η αρχική κατανομή των κόμβων στις παραπάνω λίστες, για παράδειγμα ποιος κόμβος ενός XPath ερωτήματος θα τοποθετηθεί αρχικά σε μια λίστα υποψηφίων. Ο πιο απλός τρόπος να γίνει αυτή η κατανομή είναι στις λίστες υποψηφίων να τοποθετήσουμε τους κόμβους μονοπατιού για τις αρχικές καταστάσεις. Για τις περισσότερες περιπτώσεις όμως αυτή η προσέγγιση δεν είναι καθόλου αποδοτική, καθώς τα πρώτα στοιχεία σε κάθε ερώτημα είναι το πιο πιθανό να έχουν τη μικρότερη επιλεκτικότητα (δηλ. να επιλέγουν *πολλά* στοιχεία), καθώς τα πρώτα στοιχεία των ερωτημάτων συνήθως αντιστοιχούν σε elements του εγγράφου που είναι σε πολύ μικρό βάθος στο έγγραφο (κοντά στη ρίζα), όπου τα πιθανά ονόματα για ένα στοιχείο είναι πολύ πιο περιορισμένα απ' ότι σε μεγαλύτερο βάθος. Έτσι, στο query index που θα προκύψει οι λίστες υποψηφίων θα είναι ανισόρροπα κατανεμημένες, καθώς θα υπάρχουν λίγες πολύ μεγάλες λίστες υποψηφίων. Το τελευταίο όμως έχει αρνητική επίπτωση στην απόδοση του συστήματος, καθώς η επεξεργασία τόσο μεγάλων λιστών (που έχουν πολύ μικρή επιλεκτικότητα) θα περιορίζει το σύνο-

λο των ερωτημάτων που πρέπει να εξεταστούν και σε επόμενη κατάσταση με πολύ αργούς ρυθμούς. Αντίθετα αν είχαμε μικρές λίστες με μεγάλη επιλεκτικότητα, σε κάθε κατάσταση της μηχανής θα περιοριζόταν πολύ το σύνολο των υποψήφιων προφίλ που πρέπει να συνεχίσουν να εξετάζονται.

Για τους λόγους αυτούς, το XFilter υιοθετεί την τεχνική *List Balance* για την επιλογή του κόμβου μονοπατιού που θα τοποθετηθεί αρχικά σε μια λίστα υποψηφίων για κάθε ερώτημα. Στόχος της τεχνικής αυτής είναι το αρχικό μήκος των λιστών υποψηφίων να είναι όσο το δυνατό πιο ισορροπημένο. Όταν προστίθεται ένα νέο ερώτημα στο Ευρετήριο Ερωτημάτων, βρίσκουμε το στοιχείο-κόμβο (*element node*) του ερωτήματος, για το οποίο η εγγραφή που του αντιστοιχεί στο Ευρετήριο Ερωτημάτων έχει τη μικρότερη σε μήκος λίστα υποψηφίων, και το ονομάζουμε ως κόμβο-άξονα (*pivot node*) του συγκεκριμένου ερωτήματος. Τότε ο κόμβος pivot τοποθετείται στη λίστα υποψηφίων που του αντιστοιχεί και έτσι καθίσταται ο πρώτος κόμβος από το συγκεκριμένο ερώτημα που θα εξεταστεί απέναντι σε ένα έγγραφο.

Ως εκ τούτου, η μηχανή πεπερασμένων καταστάσεων για το συγκεκριμένο ερώτημα πρέπει να τροποποιείται ώστε η αρχική της κατάσταση να είναι ο κόμβος pivot. Αυτό επιτυγχάνεται αναπαριστώντας το τμήμα του FSM που προηγείται του pivot κόμβου ως ένα *πρόθεμα*, το οποίο επισυνάπτεται στο συγκεκριμένο κόμβο. Όταν ο κόμβος αυτός ενεργοποιηθεί, το πρόθεμα ελέγχεται ως μια προϋπόθεση που πρέπει να ικανοποιείται. Αν δε συμβαίνει αυτό, τότε δεν έχει νόημα να συνεχιστεί η εκτέλεση για τον κόμβο αυτό και έτσι διακόπτεται. Για την αποτίμηση του προθέματος, χρησιμοποιούμε μια στοίβα στην οποία κρατάμε όλα τα στοιχεία-κόμβους του εγγράφου που έχουμε διασχίσει μέχρι να φτάσουμε εκεί. Με βάση τη στοίβα αυτή, μπορούμε να κάνουμε μια πολύ γρήγορη εκτέλεση του τμήματος του FSM που αντιστοιχεί στο συγκεκριμένο πρόθεμα.



Σχήμα 23: Κόμβοι XFilter (β)

Στο Σχήμα 23 φαίνονται παραδείγματα κόμβων μονοπατιού και ένα τροποποιημένο query index για τη μέθοδο List Balance. Με τη μέθοδο αυτή βλέπουμε ότι οι λίστες υποψηφίων είναι ίδιες για κάθε εγγραφή του ευρετηρίου. Το tradeoff για τη βελτίωση αυτή είναι η επιπλέον εργασία που χρειάζεται για την αποτίμηση των προθεμάτων όταν ενεργοποιείται ένας κόμβος pivot. Πειραματικές δοκιμές ωστόσο αποδεικνύουν ότι η επιβάρυνση από την επιπλέον αυτή εργασία είναι αμελητέα σε σχέση με τα οφέλη που αποκομίζουμε σε αποδοτικότητα από τις ισοζυγισμένες λίστες.

2.2.6.3 Αλγόριθμος Εκτέλεσης

Όπως έχει ήδη περιγραφεί, όταν ένα νέο XML έγγραφο εισέρχεται στο σύστημα, αρχίζει η συντακτική του ανάλυση, κατά την οποία εγείρονται συμβάντα και καλούνται συναρτήσεις χειρισμού τους. Στο XFilter τα μόνα συμβάντα που έχουν σημασία είναι το startElement και το endElement και γι' αυτά υλοποιούμε συναρτήσεις χειρισμού. Και στις δύο περιπτώσεις στις συναρτήσεις χειρισμού περνούν ως ορίσματα το όνομα του element για το οποίο ανέκυψε το συμβάν, καθώς και το βάθος μέσα στο εξεταζόμενο έγγραφο στο οποίο βρίσκεται ο συντακτικός αναλυτής.

`startElement()` : Εκτός από το όνομα και το βάθος του στοιχείου, στον handler αυτόν περνούν ως παράμετροι τα *γνωρίσματα* (*attributes*) του element και οι τιμές τους. Τότε αναζητείται το όνομα του στοιχείου στο Ευρετήριο Ερωτημάτων και αν βρεθεί εξετάζονται όλοι οι κόμβοι στη λίστα υποψηφίων της εγγραφής αυτής. Για κάθε κόμβο γίνεται έλεγχος βάθους. Ο έλεγχος αυτός έχει ως στόχο να εξακριβώσει ότι το συγκεκριμένο element που βρέθηκε στο έγγραφο βρίσκεται στο ίδιο βάθος του εγγράφου με αυτό που ανέμενε το ερώτημα. Αν ο κόμβος μονοπατιού περιέχει μη αρνητική τιμή στην ιδιότητα level, τότε ο έλεγχος βάθους επιτυγχάνει αν και μόνο αν οι δύο τιμές για το βάθος ταυτίζονται. Διαφορετικά, το βάθος στο οποίο πρέπει να βρίσκεται το element του εγγράφου μπορεί να είναι οποιοδήποτε και ως εκ τούτου ο έλεγχος επιτυγχάνει.

Αν ο παραπάνω έλεγχος επιτύχει, τότε ο κόμβος περνάει τον έλεγχο, και αν είναι ο τελευταίος κόμβος για το ερώτημα (δηλ. η τελική του κατάσταση), τότε το έγγραφο ταιριάζει στο συγκεκριμένο προφίλ. Αν δεν είναι ο τελευταίος κόμβος, τότε το FSM για το συγκεκριμένο ερώτημα περνάει στην επόμενη κατάστασή του. Αυτό γίνεται με το να *αντιγραφεί* ο επόμενος κόμβος μονοπατιού του ερωτήματος από τη λίστα αναμονής στη λίστα υποψηφίων που του αντιστοιχεί(ένα αντίγραφο παραμένει στη λίστα αναμονής). Αν η τιμή RelativePos του αντιγραμμένου κόμβου δεν είναι -1, ενημερώνεται η τιμή του level του κόμβου κατάλληλα.

`endElement()` : Όταν βρεθεί το τέλος ενός στοιχείου (element), διαγράφονται από τις λίστες υποψηφίων όλοι οι κόμβοι μονοπατιού που εισήχθησαν από τη startElement() για το συγκεκριμένο element, ώστε οι λίστες αυτές να βρεθούν στην κατάσταση που ήταν πριν αρχίσει η επεξεργασία του στοιχείου αυτού. Αυτό είναι απαραίτητο να γίνει, καθώς μέσα στο έγγραφο μπορεί να υπάρχουν elements με το ίδιο όνομα σε διαφορετικά επίπεδά του.

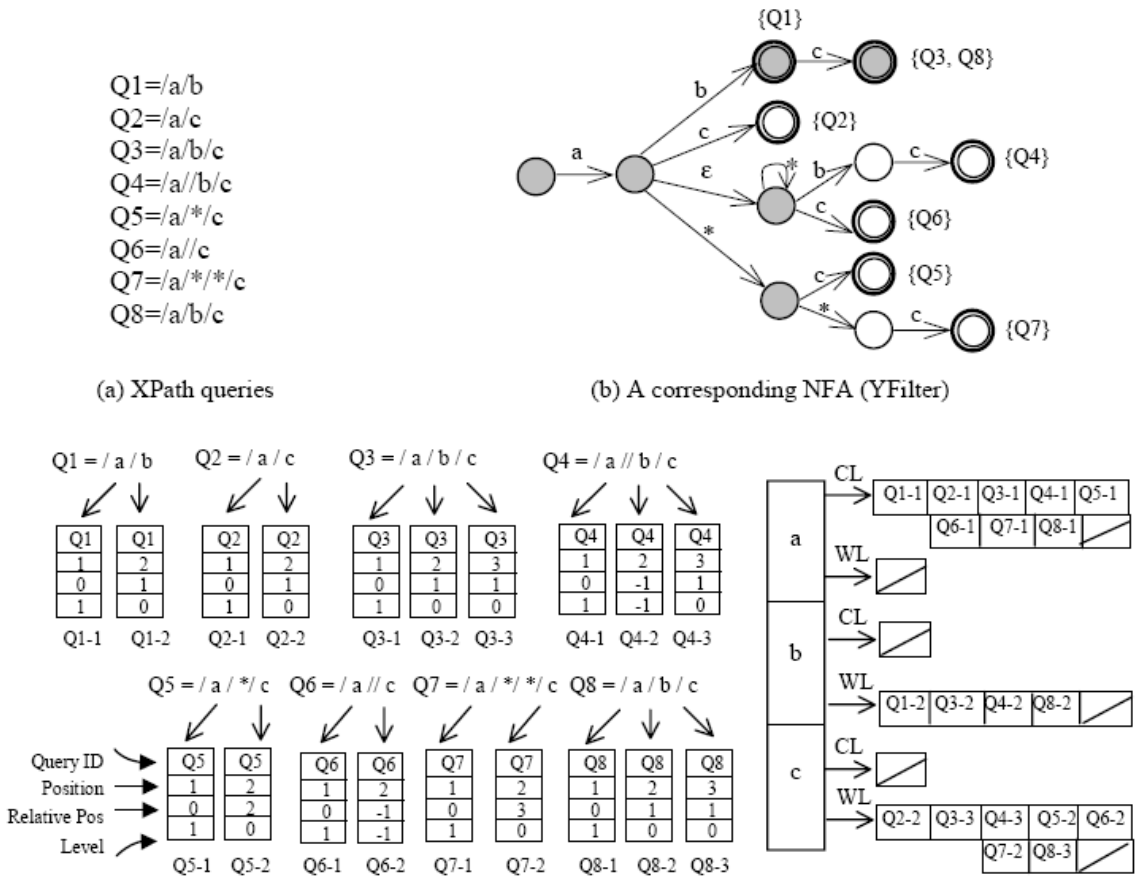
Το XFilter δημιουργεί ένα αποδοτικό ευρετήριο πάνω στα ερωτήματα με τρόπο τέτοιο ώστε ένα εισερχόμενο έγγραφο να μπορεί να ελεγχθεί ως προς το ταίριασμα με ένα μεγάλο πλήθος XPath ερωτημάτων *ταυτόχρονα*. Ωστόσο, το XFilter δεν εκμεταλλεύεται καθόλου το γεγονός ότι τα περισσότερα XPath ερωτήματα έχουν κάποιο κοινό μέρος, το οποίο μπορεί να ελεγχθεί μόνο για ένα από αυτά (αφού στα υπόλοιπα υπάρχει ακριβώς ίδιο). Στην συνέχεια παρουσιάζεται το YFilter, το οποίο ακολουθεί τις βασικές αρχές του XFilter, αλλά επιπλέον χρησιμοποιεί τεχνικές για ποιο μαζικό έλεγχο των ερωτημάτων σε σχέση με ένα έγγραφο, και άρα αυξημένη αποδοτικότητα.

2.2.7 YFilter

Στο YFilter ([DAF+03]) αντί να αναπαρίστανται κάθε ερώτημα σαν μια ξεχωριστή μηχανή πεπερασμένων καταστάσεων (FSM), συγκεντρώνονται όλες ο επιμέρους μηχανές πεπερασμένων καταστάσεων σε ένα *Μη-Ντετερμινιστικό Πεπερασμένο Αυτόματο* (*Nondeterministic Finite Automaton – NFA*). Στη δομή αυτή οι ετικέτες των μεταβάσεων από κατάσταση σε κατάσταση

αντιστοιχούν στα βήματα του μονοπατιού μιας έκφρασης XPath. Με τον τρόπο αυτό, το κοινό πρόθεμα των μονοπατιών αναπαρίσταται μόνο μια φορά στη δομή.

Στο Σχήμα 24 που ακολουθεί φαίνονται μερικά ερωτήματα XPath (στο a), το NFA που τα αναπαριστά στο YFilter (στο b) καθώς και οι μηχανές πεπερασμένων καταστάσεων που αντιστοιχούν στα ερωτήματα σύμφωνα με το XFilter:



Σχήμα 24: Παράδειγμα NFA του YFilter

Στο παραπάνω NFA κάθε κύκλος αναπαριστά μια κατάσταση, ενώ κάθε κύκλος με διπλή γραμμή δηλώνει τελική κατάσταση. Κάθε κατευθυνόμενη ακμή δηλώνει μια μετάβαση, ενώ η ετικέτα πάνω από κάθε ακμή δηλώνει το σύμβολο εισόδου που οδηγεί την εκτέλεση στην κατάσταση-στόχο. Κατά τα γνωστά, το σύμβολο «*» αναπαριστά οποιοδήποτε σύμβολο εισόδου, ενώ το «ε» δηλώνει μετάβαση που μπορεί να γίνει χωρίς να διαβαστεί κάτι από την είσοδο. Στο σχήμα οι γραμμοσκιασμένοι κύκλοι αναπαριστούν καταστάσεις που είναι κοινές σε διάφορα ερωτήματα. Το NFA μπορεί να έχει πολλές τελικές καταστάσεις(όπως στο σχήμα). Για κάθε ερώτημα υπάρχει μία και μόνο τελική κατάσταση, η οποία δηλώνει ότι το τρέχον έγγραφο ταιριάζει με το ερώτημα. Ταυτόσημα ερωτήματα, δομικά και σημασιολογικά, έχουν την ίδια τελική κατάσταση.

Η έξοδος της παραπάνω μηχανής είναι απεικόνιση από το σύνολο των τελικών καταστάσεων σε μια διαμέριση των αναγνωριστικών όλων των ερωτημάτων που υπάρχουν στο σύστημα.

2.2.7.1 Αποδοτικότητα

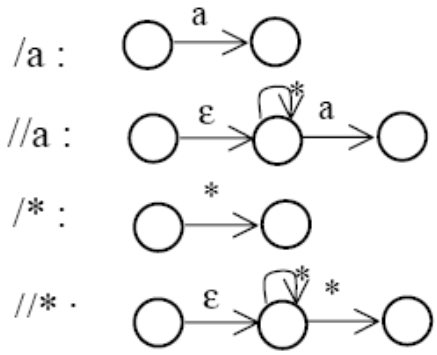
Η σημαντικότερη προσφορά της υιοθέτησης του NFA στην αποδοτικότητα του συστήματος, σε σχέση με τις FSM του XFilter, είναι η μεγάλη μείωση του μεγέθους της μηχανής που απαιτείται, καθώς απαιτεί εξαιρετικά μικρότερο αριθμό καταστάσεων. Αυτό βέβαια γίνεται με το κόστος ότι πρέπει να υποστηρίζονται πολλαπλές μεταβάσεις για κάθε κατάσταση. Για να αποφευχθεί αυτό το κόστος, μια τεχνική είναι η μετατροπή του NFA στο ισοδύναμο DFA. Αυτό όμως θα οδηγούσε σε απότομη αύξηση του αριθμού των καταστάσεων, γεγονός που θα καθιστούσε το σύστημα ακατάλληλο για χρήση σε μεγάλη κλίμακα. Η μεγάλη αυτή αύξηση του αριθμού των καταστάσεων θα μπορούσε βέβαια να μετριαστεί αποτελεσματικά αν επιβληθούν περιορισμοί ή πρότυπα στα XML έγγραφα (π.χ. DTD's ή XML Schema) και η κατασκευή του DFA να γίνεται σε στάδια. Παρόλα αυτά, πειραματικά αποτελέσματα δείχνουν ότι η αποτίμηση των μονοπατιών γίνεται ικανοποιητικά γρήγορα και έτσι το κόστος από τις πολλαπλές μεταβάσεις δεν έχει ουσιαστική επίδραση στην αποδοτικότητα του YFilter, καθώς δεν είναι ο κυρίαρχος παράγοντας κόστους. Σε πολλές μάλιστα περιπτώσεις, η συντακτική ανάλυση του XML εγγράφου είναι πιο επίπονη διαδικασία από πλευράς απόδοσης, ειδικά όταν τα ερωτήματα για τα οποία κατασκευάζεται το NFA μοιάζουν σε μεγάλο βαθμό μεταξύ τους.

2.2.7.2 Κατασκευή του NFA

Η κατασκευή του NFA γίνεται αυξητικά. Τα τέσσερα βασικά βήματα εύρεσης ενός στοιχείου σε μια έκφραση XPath είναι τα:

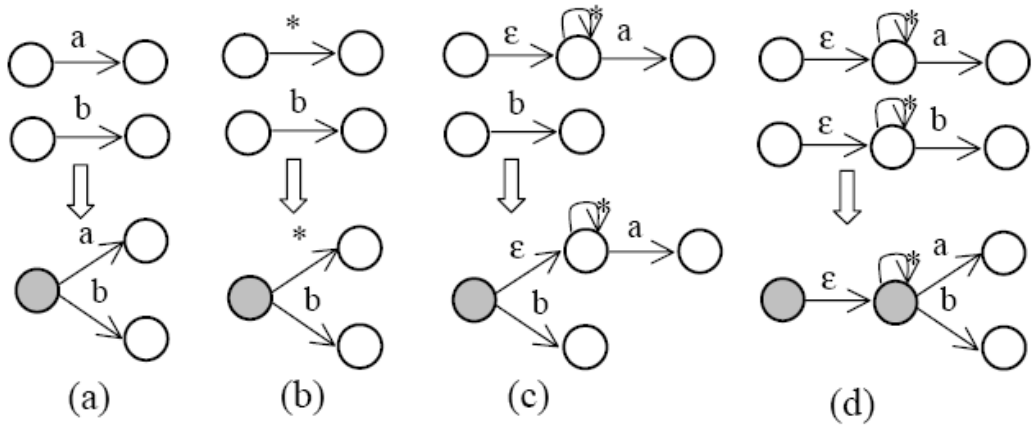
```
/a          //a          /*          //*
```

όπου το a είναι ένα οποιοδήποτε σύμβολο από το αλφάβητο από το οποίο προέρχονται τα ονόματα των elements. Για καθένα από αυτά τα βήματα που μπορεί να βρούμε σε μια έκφραση XPath, κατασκευάζουμε και ένα κομμάτι (*fragment)* του NFA που αντιστοιχεί σε αυτό, όπως φαίνεται στο Σχήμα 25.



Σχήμα 25: Κατασκευή NFA του YFilter (α)

Όπως φαίνεται και στο σχήμα, για κάθε κόμβο του μονοπατιού που έχουμε τελεστή «//», εισάγουμε στο NFA μια *ε* μετάβαση προς μια κατάσταση με βρόχο προς την ίδια. Αυτή η ε-μετάβαση είναι απαραίτητη ώστε όταν συνδυάζονται κομμάτια του NFA που αποτελούνται από «//» αλλά και από «/» να διατηρείται επακριβώς η διαφορετική σημασιολογία των δύο αυτών τελεστών. Το τελικό NFA για μια έκφραση XPath (το οποίο συμβολίζουμε ως NFA_p) μπορεί να προκύψει με απλή παράθεση των επιμέρους κομματιών NFA που αντιστοιχούν στα βήματα της έκφρασης XPath, όπως φαίνεται στο σχήμα που ακολουθεί:



Σχήμα 26: Κατασκευή NFA του YFilter (α)

Τα NFA_p συνδυάζονται ως εξής: υπάρχει μια απλή αρχική κατάσταση, την οποία μοιράζονται όλα τα NFA_p. Για να εισάγουμε ένα νέο NFA_p, διασχίζουμε το συνολικό NFA που έχει προκύψει μέχρι τώρα, μέχρι είτε να συναντήσουμε την τελική κατάσταση του νέου NFA_p, είτε να φτάσουμε σε μια κατάσταση του συνολικού NFA στην οποία να μην υπάρχει μετάβαση ισοδύναμη με τη μετάβαση του NFA_p. Στην πρώτη περίπτωση μαρκάρουμε την κατάσταση στην οποία φτάσαμε ως τελική και για το συνολικό NFA (αν δεν είναι ήδη) και προσθέτουμε το αναγνωριστικό του ερωτήματος(query id) στο σύνολο των ερωτημάτων το οποίο κρατάμε για κάθε τελική κατάσταση(ώστε να γνωρίζουμε σε ποια ερωτήματα αντιστοιχεί η τελική αυτή κατάσταση). Στην δεύτερη περίπτωση δημιουργούμε ένα νέο «κλαδί» στο NFA που ξεκινάει από την τελευταία κατάσταση του συνολικού NFA στην οποία φτάσαμε. Το νέο αυτό κομμάτι αποτελείται από την κατάσταση αυτή και το υπόλοιπο του NFA_p.

Τα παραπάνω φαίνονται στο προηγούμενο σχήμα. Στο (a) συνενώνονται δύο κομμάτια που αντιστοιχούν στα XPath ερωτήματα */a* και */b*. Είναι εμφανές ότι δε συνενώνουμε την ακμή a με την ακμή b σε μία που να ονομάζεται a, b διότι οι καταστάσεις στις οποίες φτάνουμε από την a και την b είναι διακριτές σημασιολογικά. Για τον ίδιο λόγο στο σχήμα (b) το σύμβολο * αντιμετωπίζεται σαν ένα οποιοδήποτε άλλο σύμβολο. Στο (c) φαίνεται η διαδικασία σύμφωνα με την οποία συγχωνεύουμε το κομμάτι που αντιστοιχεί στο *//a* με το κομμάτι που αντιστοιχεί στο */b*, όπου φαίνεται και ο λόγος για τον οποίο στο κομμάτι του //a εισάγουμε μια επιπλέον κατάσταση και μια *ε*-μετάβαση. Χωρίς αυτή, ο συνδυασμός των δύο αυτών κομματιών θα ήταν σημασι-

ολογικά ισοδύναμος με το συνδυασμό των *//a* και *//b*. Επίσης, το κομμάτι που αντιστοιχεί στο //* συνενώνεται με άλλα κομμάτια κατά πλήρη αντιστοιχία με το *//a*.

Παρατηρούμε ότι οι τελεστές «*» και «//» είναι αυτοί που εισάγουν τη μη-ντετερμινιστικότητα στο σύστημά μας, καθώς ο «*» απαιτεί να ακολουθηθούν δύο ακμές, μία που να έχει το σύμβολο της εισόδου και μία που να έχει το *. Ο τελεστής απογόνου «//» σημαίνει ότι ο σχετικός έλεγχος για τον κόμβο μπορεί να ικανοποιείται σε οποιοδήποτε επίπεδο χαμηλότερο ή ίσο από το τρέχον στο έγγραφο. Έτσι στο NFA αν διαβαστεί ένα κατάλληλο σύμβολο, τότε η εκτέλεση της μηχανής θα πρέπει και να προχωρήσει στην επόμενη κατάσταση και να παραμείνει στην ίδια.

Σημειώνεται επίσης ότι επειδή η κατασκευή του NFA γίνεται *αυξητικά*, νέα ερωτήματα μπορούν να προστίθενται δυναμικά στη μηχανή φιλτραρίσματος, χαρακτηριστικό που είναι ιδιαίτερα επιθυμητό καθώς νέοι χρήστες(δηλ. προφίλ) μπορούν να μπαίνουν στο σύστημα, αλλά και οι παλιοί μπορούν να τροποποιούν τα προφίλ τους.

2.2.7.3 Υλοποίηση του NFA

Για να επιτευχθεί αποδοτική εκτέλεση του NFA, η υλοποίηση της δομής βασίζεται σε πίνακα κατακερματισμού (hash-table), που έχει αποδειχθεί ότι έχει εξαιρετικά μικρή πολυπλοκότητα όταν εισάγουμε/διαγράφουμε καταστάσεις και μεταβάσεις, αλλά και κατά την εκτέλεση των μεταβάσεων. Δημιουργείται έτσι μια δομή, στην οποία για κάθε κατάσταση κρατάμε:

α) το αναγνωριστικό της κατάστασης (*stateID*),

β) πληροφορίες για τον τύπο της κατάστασης (π.χ. αν είναι τελική ή έπεται ενός «//» τελεστή),

γ) ένα μικρό πίνακα κατακερματισμού που περιέχει όλες τις έγκυρες μεταβάσεις από την κατάσταση αυτή προς άλλες και

δ) αν πρόκειται για τελική κατάσταση, κρατάμε μια λίστα με τα αναγνωριστικά όλων των ερωτημάτων στα οποία η τελική κατάσταση αντιστοιχεί.

Ο πίνακας μεταβάσεων για κάθε κατάσταση περιέχει ζεύγη [*σύμβολο*, *stateID*], όπου *σύμβολο* είναι το κλειδί και συμβολίζει την ετικέτα της μετάβασης προς κάποιον άλλο κόμβο, δηλαδή είναι είτε το όνομα ενός element, το «*» ή το «ε». Το stateID είναι το αναγνωριστικό της κατάστασης προς την οποία οδηγεί η παραπάνω μετάβαση. Σημειώνεται ότι αφού οι καταστάσεις στις οποίες φτάνουμε με ε-μετάβαση έχουν ένα βρόχο από τον εαυτό του προς τον εαυτό τους με ετικέτα «*», δεν καταχωρούμε το βρόχο στο ευρετήριο, αλλά ο χειρισμός των μεταβάσεων αυτών γίνεται με ξεχωριστό τρόπο, που περιγράφεται αργότερα.

2.2.7.4 Εκτέλεση του NFA

Η εκτέλεση του NFA γίνεται με τρόπο παρόμοιο με αυτόν που ακολουθεί το XFilter για την εκτέλεση της μηχανής πεπερασμένων καταστάσεων, δηλαδή οδηγούμενη από συμβάντα. Καθώς εισέρχεται στο σύστημα ένα νέο XML έγγραφο και γίνεται η συντακτική του ανάλυση, εγείρονται συμβάντα από το συντακτικό αναλυτή, οι συναρτήσεις χειρισμού των οποίων είναι αυτές που οδηγούν την εκτέλεση του NFA. Η δυνατότητα φωλιάσματος των elements της XML απαιτεί όταν φτάνουμε σε ένα συμβάν endElement(), η εκτέλεση της μηχανής πρέπει να οπισθοχωρεί στην κατάσταση που ήταν ακριβώς πριν το startElement() συμβάν για το συγκεκριμένο στοιχείο. Για το λόγο αυτό κατά την εκτέλεση χρησιμοποιείται μια στοίβα, η οποία μάλιστα έχει δυνατότητα να καταγράφει πολλά μονοπάτια εκτέλεσης, αφού το NFA μπορεί σε μια δεδομένη χρονική στιγμή να βρίσκεται σε πολλές καταστάσεις.

Παρακάτω περιγράφονται οι συναρτήσεις χειρισμού για τα συμβάντα που εγείρει ο συντακτικός αναλυτής:

`startDocument()` : Όταν ξεκινάει η συντακτική ανάλυση ενός νέου εγγράφου, η εκτέλεση του NFA ξεκινάει από την αρχική κατάσταση. Έτσι, η αρχική κατάσταση εισάγεται στη στοίβα εκτέλεσης.

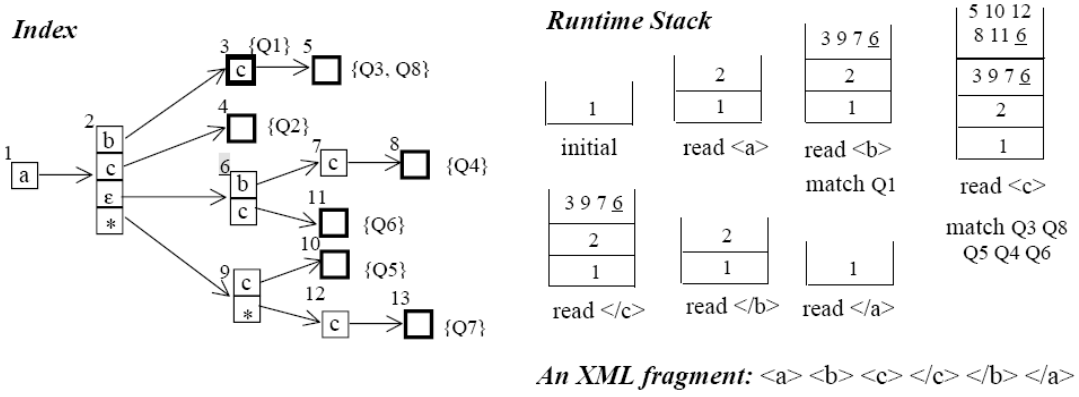
`startElement()` : Όταν διαβάζεται ένα νέο element από το έγγραφο XML, το NFA όλες τις μεταβάσεις που ταιριάζουν στο όνομα του element που διαβάστηκε, από όλες τις καταστάσεις στις οποίες βρίσκεται το NFA. Αυτό γίνεται ως εξής:

1. Αρχικά το όνομα του element που διαβάστηκε αναζητείται στον πίνακα κατακερματισμού της τρέχουσας κατάστασης. Αν υπάρχει δηλ. υπάρχει ακμή με ετικέτα το όνομα του element, προστίθεται το stateID της κατάστασης στην οποία δείχνει η ακμή στο σύνολο των καταστάσεων-στόχων.
2. Αναζητείται στον πίνακα κατακερματισμού το σύμβολο «*». Αν υπάρχει, προστίθεται το κατάλληλο stateID στο σύνολο των καταστάσεων στόχων. Αφού το σύμβολο αυτό συμβολίζει οποιοδήποτε όνομα element, οι μεταβάσεις που έχουν ως ετικέτα το «*» γίνονται πάντα.
3. Στη συνέχεια ελέγχεται ο τύπος της κατάστασης. Αν η κατάσταση αντιστοιχεί σε βήμα μονοπατιού που έπεται ενός τελεστή «//», τότε το stateID της ίδιας της κατάστασης προστίθεται στο σύνολο των καταστάσεων στόχων. Έτσι τελικά υλοποιείται μετάβαση που συμβολίζεται με ένα βρόχο με «*» από μια κατάσταση προς τον εαυτό της.
4. Για να εκτελεστεί μια ε-μετάβαση, ο πίνακας κατακερματισμού ελέγχεται για το σύμβολο «ε» και αν αυτό υπάρχει η κατάσταση που καθορίζεται από το αντίστοιχο stateID(και που έπεται ενός τελεστή «//») επεξεργάζεται αναδρομικά, σύμφωνα με τους τρεις παραπάνω κανόνες.

Όταν με αυτό τον τρόπο τελειώσει ο έλεγχος όλων των ενεργών καταστάσεων, το σύνολο των καταστάσεων-στόχων που έχει δημιουργηθεί εισάγεται στην κορυφή της στοίβας εκτέλεσης. Οι καταστάσεις του συνόλου αυτού γίνονται οι ενεργές καταστάσεις στο επόμενο συμβάν. Αν μια κατάσταση στο σύνολο των καταστάσεων στόχων είναι τελική κατάσταση, τα αναγνωριστικά όλων των ερωτημάτων που σχετίζονται με την κατάσταση καταγράφονται και προστίθενται στην έξοδο.

`startElement()` : Όταν βρεθεί το τέλος ενός element του εγγράφου, η οπισθοδρόμηση υλοποιείται αφαιρώντας από τη στοίβα το σύνολο των καταστάσεων που βρίσκονται στην κορυφή της στοίβας.

Επίσης, σε αντίθεση με τα παραδοσιακά NFA, όπου ο στόχος της εκτέλεσης είναι να φτάσει σε μια τελική κατάσταση, στην περίπτωσή μας η εκτέλεση πρέπει να βρει όλα τα ερωτήματα που ταιριάζουν στο έγγραφο. Γι' αυτό το λόγο, ακόμα και αν βρεθεί μια τελική κατάσταση για ένα συγκεκριμένο έγγραφο, πρέπει η εκτέλεση του NFA να συνεχιστεί μέχρι να ολοκληρωθεί η επεξεργασία του εγγράφου. Στο επόμενο σχήμα φαίνεται ένα παράδειγμα εκτέλεσης για το NFA που παρουσιάσαμε παραπάνω. Σε κάθε πίνακα κατακερματισμού αναγράφεται πάνω αριστερά ένα stateID, ενώ τα τετράγωνα με έντονη γραφή είναι τελικές καταστάσεις. Στο δεξί μέρος του σχήματος απεικονίζονται τα περιεχόμενα της στοίβας εκτέλεσης κατά τη διάρκεια της επεξεργασίας ενός τμήματος εγγράφου XML. Στη στοίβα κάθε κατάσταση αναπαρίσταται από το αναγνωριστικό της, ενώ τα υπογραμμισμένα αναγνωριστικά υποδηλώνουν καταστάσεις που έπονται ενός τελεστή «//»:



Σχήμα 27. Ετέλεση NFA του YFilter

2.2.7.5 Αποτίμηση Κατηγορημάτων σε ερωτήματα XPath

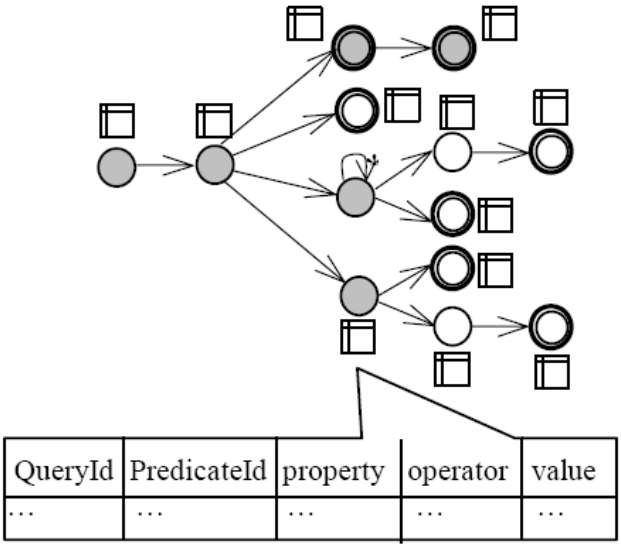
Σε ένα ερώτημα XPath μπορεί να υπάρχουν και κάποιες συνθήκες(κατηγορήματα), τα οποία ελέγχουν ως προς το ταίριασμα τα elements του XML εγγράφου σε ότι αφορά τη θέση τους, το περιεχόμενό τους ή τα γνωρίσματά τους(attributes) με βάση κάποια τιμή. Για παράδειγμα:

- Το *//product/price[@currency = "USD"]* ελέγχει την τιμή ενός γνωρίσματος (currency) του element price.
- Το *//product/price[text() <= 300]* είναι ένα κατηγόρημα σχετικά με το κείμενο που περιέχει το XML element
- Το */catalog/product[position() =2]* ταιριάζει μόνο στο 2ο element product που είναι παιδί του catalog

Θα επικεντρωθούμε κυρίως σε ερωτήματα XPath που περιέχουν κατηγορήματα σχετικά με κάποιο γνώρισμα ή τη θέση ενός στοιχείου και όχι τόσο με το περιεχόμενο του στοιχείου (όπως το 2ο από τα παραπάνω παραδείγματα). Το YFilter μπορεί να επεκταθεί με δύο διαφορετικές τεχνικές ώστε να χειρίζεται τα κατηγορήματα σε XPath ερωτήματα. Οι τεχνικές αυτές είναι:

2.2.7.6 Τεχνική Inline

Σύμφωνα με αυτή την προσέγγιση, επεκτείνουμε τις πληροφορίες που αποθηκεύονται σε κάθε κατάσταση του NFA ώστε να συμπεριλάβουμε και όλα τα κατηγορήματα που σχετίζονται με τη συγκεκριμένη κατάσταση. Τα κατηγορήματα αυτά αποθηκεύονται σε έναν πίνακα, όπως αυτός που φαίνεται στο Σχήμα 28. Επειδή πολλές εκφράσεις μπορεί να έχουν κάποιες καταστάσεις κοινές, ο πίνακας αυτός μπορεί να περιέχει κατηγορήματα που ανήκουν σε διαφορετικά ερωτήματα και ως εκ τούτου αντί για το αναγνωριστικό του κατηγορήματος (predicateID) μόνο, στον πίνακα αυτό αποθηκεύουμε ένα ζεύγος *(queryID, predicateID)*.



Σχήμα 28: Αποθήκευση κατηγορημάτων στο NFA του YFilter (α)

Σύμφωνα με την προσέγγιση αυτή, όταν εγείρεται ένα startElement() συμβάν το NFA μεταβαίνει στις επόμενες καταστάσεις που περιγράφονται παραπάνω και για κάθε νέα κατάσταση στην οποία φτάνει, τα κατηγορήματα που είναι αποθηκευμένα εκεί ελέγχονται. Επίσης, για κάθε ερώτημα κρα-

τάμε κάπου ποια κατηγορήματα του συγκεκριμένου ερωτήματος έχουν ικανοποιηθεί. Όταν η εκτέλεση του NFA φτάσει σε μια τελική κατάσταση, για καθένα από τα ερωτήματα που αντιστοιχούν στην κατάσταση αυτή, ελέγχεται ποια από τα κατηγορήματα του ερωτήματος έχουν ικανοποιηθεί. Όσα ερωτήματα έχουν όλα τα κατηγορήματά τους ικανοποιημένα, επιστρέφονται ως έξοδος, ότι δηλαδή ταιριάζουν στο υπό εξέταση έγγραφο.

Ένα θέμα το οποίο προκύπτει από την προσέγγιση αυτή είναι το πιθανό όφελος από πρώιμο έλεγχο των κατηγορημάτων. Η μη ικανοποίηση ενός κατηγορήματος σε κάποια κατάσταση σε καμία περίπτωση δε σημαίνει ότι πρέπει να σταματήσει η επεξεργασία κατά μήκος του συγκεκριμένου μονοπατιού εκτέλεσης, καθώς είναι πολύ πιθανό να υπάρχουν άλλα ερωτήματα που μοιράζονται την κατάσταση αυτή και τα οποία η κατάσταση αυτή ικανοποιεί. Επίσης, αν ένα ερώτημα περιέχει τελεστή «//» πριν από ένα κατηγόρημα, τότε ακόμα και αν το κατηγόρημα αποτιμηθεί σε false, το ερώτημα συνεχίζει να ισχύει και να είναι ενεργό, εξαιτίας της μη-ντετερμινιστικότητας που εισάγεται από τον τελεστή. Για το λόγο αυτό δεν είναι δυνατό να βελτιστοποιήσουμε το σύστημα εκτελώντας τις επιλογές (selects) στην αρχή του μονοπατιού (για να έχουμε μεγαλύτερο κλάδεμα).

Ένα δεύτερο ζήτημα που ανακύπτει είναι ότι εξαιτίας της εμφωλευμένης δομής των XML εγγράφων, είναι δυνατό να έχουμε οπισθοδρομήσεις (backtracking) κατά την εκτέλεση του NFA (όπως έχει περιγραφεί παραπάνω), με αποτέλεσμα να δημιουργούνται επιπλέον δυσκολίες στη διαδικασία καταγραφής των κατηγορημάτων που έχουν ήδη ικανοποιηθεί. Έστω για παράδειγμα το ερώτημα q = "//a[@a_1 = v_1] [@a_2 = v_2]", που περιέχει ένα τελεστή απογόνου και κατηγορήματα σχετικά με τα γνωρίσματα του element a, a_1 και a_2. Έστω επίσης το ακόλουθο κομμάτι ενός XML εγγράφου:

<a a_1 = v_1>

<a a_2 = v_2>

Αν η οπισθοδρόμηση (backtracking) δε γίνει με κατάλληλο τρόπο, τότε μπορεί να θεωρηθεί ότι το κομμάτι αυτό του XML εγγράφου ταιριάζει στο ερώτημα q, παρόλο που τα ικανοποιούμενα κατηγορήματα ανήκουν σε διαφορετικά στοιχεία a. Το πρόβλημα αυτό μπορεί να λυθεί, αν κάθε φορά που γίνεται οπισθοδρόμηση από μία κατάσταση, αναιρεθούν όλες οι αλλαγές τις οποίες εισήγαγε η κατάσταση αυτή στις πληροφορίες που κρατάμε για τα κατηγορήματα.

Ακόμα όμως και μετά από αναίρεση των αλλαγών αυτών, ένα παρόμοιο πρόβλημα με το παραπάνω εξακολουθεί να υπάρχει για *αναδρομικά φωλιασμένα* στοιχεία. Για παράδειγμα, έστω ότι έχουμε το παραπάνω ερώτημα q και το ακόλουθο κομμάτι XML εγγράφου:

<a a_1 = v_1>

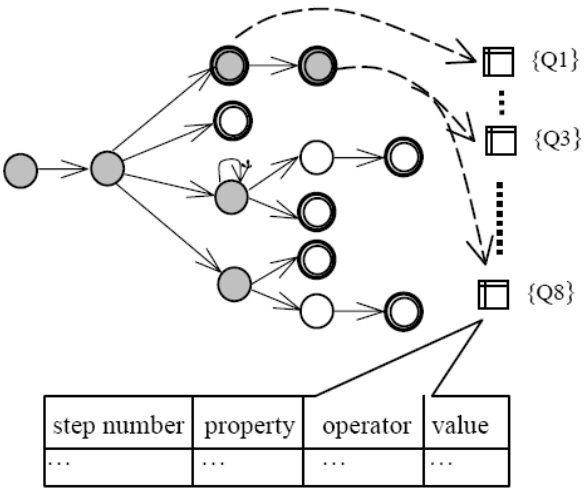
<a a_2 = v_2>

όπου τα elements a είναι φωλιασμένα. Σε αυτή την περίπτωση για να διακριθούν τα δύο στοιχεία απαιτείται να κρατάμε επιπλέον πληροφορία, σχετικά με το ποιο συγκεκριμένο στοιχείο ικανοποίησε το εκάστοτε κατηγόρημα. Όταν η εκτέλεση φτάσει στην τελική κατάσταση ενός ερωτήματος, το ερώτημα ικανοποιείται μόνο αν όλα τα κατηγορήματα που βρίσκονται στο ίδιο επίπεδο του ερωτήματος έχουν ικανοποιηθεί από το ίδιο element.

2.2.7.7 Τεχνική Selection Postponed

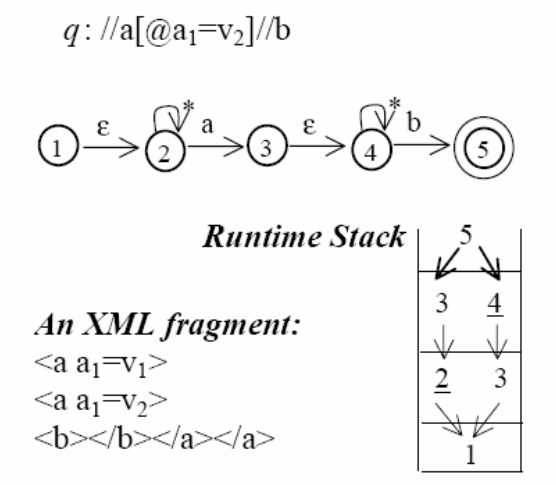
Στην τεχνική Inline που εξετάσαμε παραπάνω, ο χρόνος και η επεξεργαστική ισχύς που ξοδεύεται για την αποτίμηση των κατηγορημάτων πάει χαμένος τελικά αν το ερώτημα δεν ταιριάζει καν δομικά με το υπό εξέταση XML έγγραφο. Η τεχνική Selection Postponed αποφεύγει αυτή τη σπατάλη με το να καθυστερεί την αποτίμηση των κατηγορημάτων έως ότου έχει γίνει το δομικό ταίριασμα του ερωτήματος και του εγγράφου (δηλαδή έχουμε βεβαιωθεί ότι αν το ερώτημα δεν περιελάμβανε κατηγορήματα, θα ταίριαζε με το έγγραφο). Η τεχνική αυτή έχει και άλλα πλεονεκτήματα, αν αναλογιστούμε ότι τα κατηγορήματα σε διαφορετικά elements ενός ερωτήματος ισοδυναμούν με μια σύζευξη. Δηλαδή αν κάποιο από τα κατηγορήματα αυτά δεν ικανοποιείται, τότε το ερώτημα δεν ικανοποιείται και ως εκ τούτου δεν έχει νόημα η αποτίμηση των υπολοίπων κατηγορημάτων. Επίσης, όπως θα φανεί και στη συνέχεια, με τη μέθοδο αυτή δεν απαιτείται επέκταση της λογικής της οπισθοδρόμησης στο NFA, όπως με την Inline.

Όπως φαίνεται στο Σχήμα 29, τα κατηγορήματα εδώ αποθηκεύονται σε κάθε ερώτημα και ευρετηριάζονται από το πεδίο *αριθμός βήματος* (*step number*). Όταν η εκτέλεση φτάσει σε μια κατάσταση που είναι τελική, οι επιλογές (δηλ. η αποτίμηση των κατηγορημάτων) γίνονται μαζικά για κάθε ερώτημα, και αν όλα τα κατηγορήματα ενός ερωτήματος ικανοποιούνται, τότε το ερώτημα ικανοποιείται.



Σχήμα 29: Αποθήκευση κατηγορημάτων στο NFA του YFilter (β)

Για να μπορεί βέβαια να καθυστερηθεί η αποτίμηση των κατηγορημάτων, το NFA πρέπει να καταγράφει κάπου ένα ιστορικό με τις καταστάσεις που έχει επισκεφθεί μέχρι να αποφασίσει ότι το XML έγγραφο ταιριάζει με ένα ερώτημα. Έστω ότι έχουμε το ερώτημα *q* που φαίνεται στο Σχήμα 30.



Σχήμα 30: Εκτέλεση NFA για τμήμα XML εγγράφου

Όταν η συντακτική ανάλυση φτάσει στο στοιχείο b, το NFA φτάνει σε μια τελική κατάσταση για το συγκεκριμένο ερώτημα. Έτσι, όταν γίνεται η αποτίμηση των κατηγορημάτων για το q, πρέπει να αποφασιστεί σε ποιο από τα δύο στοιχεία με όνομα a θα εφαρμοστεί το κατηγόρημα. Φυσικά, το πιο απλό θα ήταν να εφαρμοστεί για όλα τα στοιχεία "a" που έχουν βρεθεί. Σε περιπτώσεις όμως με πολλούς τελεστές "//" στο ερώτημα ή πολλά στοιχεία αναδρομικά φωλιασμένα στο έγγραφο, αυτό θα είχε ως αποτέλεσμα μεγάλη επιβάρυνση στην απόδοση, καθώς στη χειρότερη περίπτωση είναι εξίσου ακριβό υπολογιστικά, όσο το να εκτελέσουμε από την αρχή το NFA για το συγκεκριμένο ερώτημα. Για την αποφυγή αυτού, επεκτείνουμε το NFA ώστε στην έξοδο να μη βγάζει μόνο αναγνωριστικά ερωτημάτων (query Id's), αλλά μια λίστα από μονοπάτια εκτέλεσης που δίνουν ταίριασμα για το ερώτημα. Έτσι, καθένα από αυτά τα μονοπάτια δίνει την πληροφορία για το με βάση ποια στοιχεία πρέπει να γίνει η αποτίμηση των κατηγορημάτων.

Στο προηγούμενο παράδειγμα, στην τελική κατάσταση για το ερώτημα q, το NFA θα ανέφερε δύο μονοπάτια με τα οποία προκύπτει ταίριασμα, τα "a1_b" και "a2_b" (όπου το a1 συμβολίζει το πρώτο element "a" που συναντήθηκε και το a2 δεύτερο, κτλ). Καθώς τα κατηγορήματα για κάθε ερώτημα ευρετηριάζονται με βάση τον «αριθμό βήματος», είναι εύκολο να αποφασιστεί ποια elements είναι αυτά που πρέπει να ελεγχθούν. Επιστρέφοντας στο προηγούμενο παράδειγμα, το πρώτο μονοπάτι δεν ικανοποιεί το ερώτημα, γιατί το a1 δεν ικανοποιεί το κατηγόρημα, ενώ το δεύτερο το ικανοποιεί.

Για να δίνει ως έξοδο αυτά τα μονοπάτια, το NFA επεκτείνεται ώστε να συνδέει τις καταστάσεις που βρίσκονται στη στοίβα εκτέλεσης και προς τα πίσω, προς την αρχική. Έτσι για κάθε

κατάσταση-στόχο στην οποία φτάνουμε από την ενεργή κατάσταση, εισάγουμε ένα δείκτη που δείχνει από την κατάσταση στόχο προς τα *πίσω*, δηλαδή προς την τρέχουσα ενεργή κατάσταση, και στη συνέχεια η κατάσταση-στόχος εισάγεται στη στοίβα (βλ. και προηγούμενο σχήμα).

Με τον τρόπο αυτό, για κάθε τελική κατάσταση μπορούμε να διασχίσουμε τους δείκτες προς τα πίσω και να βρούμε την ακολουθία των καταστάσεων που προηγήθηκε μέχρι να φτάσουμε στη συγκεκριμένη τελική κατάσταση. Τα στοιχεία (elements) που ενεργοποιούν μεταβάσεις προς καταστάσεις που έπονται ενός τελεστή «//» (με βρόχους που ξεκινάνε από ένα στοιχείο και καταλήγουν στον εαυτό του), μπορούν σε αυτό τα στάδιο να αγνοηθούν, καθώς δε συμμετέχουν στην αποτίμηση των κατηγορημάτων. Στο προηγούμενο παράδειγμα, υπάρχουν δύο ακολουθίες επισκέψεων σε καταστάσεις, οι "2, 3, 5" και "2, 4, 5" όταν διαβάστηκαν τα στοιχεία a1 και a2. Αν απαλειφθούν τα στοιχεία(elements) που ενεργοποιούν μεταβάσεις προς καταστάσεις που έπονται τελεστών «//» για καθεμία από τις παραπάνω ακολουθίες, τότε παίρνουμε τις ακολουθίες των στοιχείων που προκαλούν ταίριασμα, δηλ. "a1_b" και "a2_b και μπορούμε στη συνέχεια να αποτιμήσουμε τα αντίστοιχα κατηγορήματα.

Όπως αναφέρθηκε και στην αρχή της παραγράφου, κατηγορήματα που αφορούν το περιεχόμενο των elements δεν είναι δυνατό να αποτιμηθούν με τους παραπάνω τρόπους, καθώς το περιεχόμενο (text data) ενός στοιχείου δεν είναι διαθέσιμο κατά την έγερση του συμβάντος startElement(). Με τη μέθοδο Selection Postponed, η αποτίμηση και ο έλεγχος των κατηγορημάτων διαχωρίζεται από την οδηγούμενη από συμβάντα διαδικασία επεξεργασίας του εισερχόμενου εγγράφου και έτσι κατηγορήματα που αφορούν το κείμενο που περιέχεται σε ένα element μπορούν να αποτιμηθούν ως εξής: όταν εγερθεί το αντίστοιχο συμβάν (*characters()*) καταγράφουμε το κείμενο στου στοιχείου. Έτσι μαζί με κάθε διαφορετικό μονοπάτι που δίνει ως έξοδο το αυτόματο, μπορεί να δίνει και το περιεχόμενο των elements που αποτελούν το μονοπάτι, ώστε να μπορεί να γίνει η αποτίμηση των κατηγορημάτων.

Συνοψίζοντας, μπορούμε να εστιάσουμε στις βασικές διαφορές μεταξύ των δύο τεχνικών για αποτίμηση κατηγορημάτων:

1. Η Inline αποτιμά τα κατηγορήματα κάθε ερωτήματος πρώιμα, χωρίς να γνωρίζει αν υπάρχει δομικό ταίριασμα του ερωτήματος με το έγγραφο και ως εκ τούτου ελέγχονται όλα τα κατηγορήματα μέχρι να αποφασισθεί αν υπάρχει δομικό ταίριασμα. Από τη άλλη, στη μέθοδο Selection Postponed, πρώτα γίνεται το δομικό ταίριασμα και κατόπιν η αποτίμηση των κατηγορημάτων, έτσι ώστε μη γίνεται χωρίς λόγο αποτίμηση κατηγορημάτων που ανήκουν σε ερωτήματα που δεν ταιριάζουν δομικά με το έγγραφο.
2. Στη μέθοδο Inline η αποτίμηση των ερωτημάτων για ένα συγκεκριμένο ερώτημα γίνεται ανεξάρτητα σε διάφορες καταστάσεις της μηχανής, ενώ στη Selection Postponed η αποτυχία ικανοποίησης ενός κατηγορήματος σταματά την περαιτέρω αποτίμηση των κατηγορημάτων του ερωτήματος.

3. Η Inline (σε αντίθεση με τη SelectionPostponed) απαιτεί την καταγραφή στοιχείων για ενδιάμεσα αποτελέσματα κατά την αποτίμηση ενός κατηγορήματος για την τελική αποτίμηση ενός ερωτήματος. Επιπλέον εκτός από την αποθήκευση της πληροφορίας αυτής απαιτείται και η αναίρεσή της κατά τη διαδικασία της οπισθοδρόμησης(backtracking). Αυτό δημιουργεί μεγαλύτερες απαιτήσεις σε μνήμη.

2.2.7.8 Φωλιασμένες Εκφράσεις

Όπως είναι γνωστό, κάθε ερώτημα XPath εκτός από την «εξωτερική» έκφραση μονοπατιού μπορεί να περιλαμβάνει και φωλιασμένες εκφράσεις μονοπατιού(π.χ. μέσα σε κάποιο κατηγόρημα). Στην παράγραφο αυτή περιγράφουμε το πώς υλοποιούνται οι φωλιασμένες εκφράσεις μονοπατιού στο YFilter. Αρχικά, περιγράφουμε τη λύση θεωρώντας ερωτήματα με μόνο ένα επίπεδο φωλιάσματος, δηλαδή θεωρούμε ότι μια φωλιασμένη έκφραση μονοπατιού (path expression) δεν μπορεί να περιέχει φωλιασμένη κάποια άλλη έκφραση μονοπατιού. Για τέτοια ερωτήματα ορίζουμε τους ακόλουθους όρους:

α) *Κύριο Μονοπάτι* (*Main Path*) είναι η δομή του ερωτήματος που απομένει αν αφαιρεθούν από αυτό όλες οι φωλιασμένες εκφράσεις μονοπατιού.

β) *Βήμα Σύναψης* (*Anchor Step*) ενός φωλιασμένου μονοπατιού είναι το βήμα εκείνο στο οποίο το φωλιασμένο μονοπάτι επισυνάπτεται στο κύριο μονοπάτι

γ) *Εκτεταμένο Φωλιασμένο Μονοπάτι* (*Extended Nested Path*) είναι ένα φωλιασμένο μονοπάτι

Όταν ένα ερώτημα που περιέχει φωλιασμένες εκφράσεις αναλύεται συντακτικά, αποσυντίθεται σε μια λίστα απολύτων μονοπατιών: το κύριο μονοπάτι και όποια εκτεταμένα φωλιασμένα μονοπάτια υπάρχουν. Για παράδειγμα, έστω το ερώτημα q = "/a[d]//b[e/f]/c", που περιέχει τα δύο φωλιασμένα μονοπάτια "d" και "e/f". Η αποσύνθεση του ερωτήματος θα έχει ως αποτέλεσμα το κύριο μονοπάτι "/a//b/c" και δύο εκτεταμένα (extended) μονοπάτια, τα "/a/d" και "/a//b/e/f". Για καθένα από αυτά τα μονοπάτια κρατάμε ένα αναγνωριστικό της μορφής (*queryID*, *PathID*), όπου το κύριο μονοπάτι του ερωτήματος έχει πάντοτε *pathID = 0* και τα φωλιασμένα μονοπάτια έχουν αυξανόμενο pathID (το $1^{ο}$ έχει το 1, το $2^{ο}$ το 2 κτλ.). Στη συνέχεια όλα αυτά τα μονοπάτια εισάγονται στη μηχανή ταιριάσματος μονοπατιών, καθένα με το αναγνωριστικό του. Θεωρούμε ότι η μηχανή επιστρέφει ταιριάσματα μονοπατιών σε ερωτήματα χρησιμοποιώντας όχι μόνο το queryID, αλλά και το pathID για τον εντοπισμό του μονοπατιού και εσωτερικά στο ερώτημα.

Μετά το ταίριασμα των μονοπατιών ακολουθεί το στάδιο της μεταεπεξεργασίας χρησιμοποιούνται τελεστές τους οποίους θα αποκαλούμε *Φίλτρα Φωλιασμένων Μονοπατιών* (*Nested Path Filters – NP-Filters)*, καθένα από τα οποία αντιστοιχεί σε κάποιο ερώτημα. Θεωρώντας ότι υπάρχει μόνο ένα επίπεδο φωλιάσματος, μόνο ένα τέτοιο φίλτρο απαιτείται ανά ερώτημα. Το

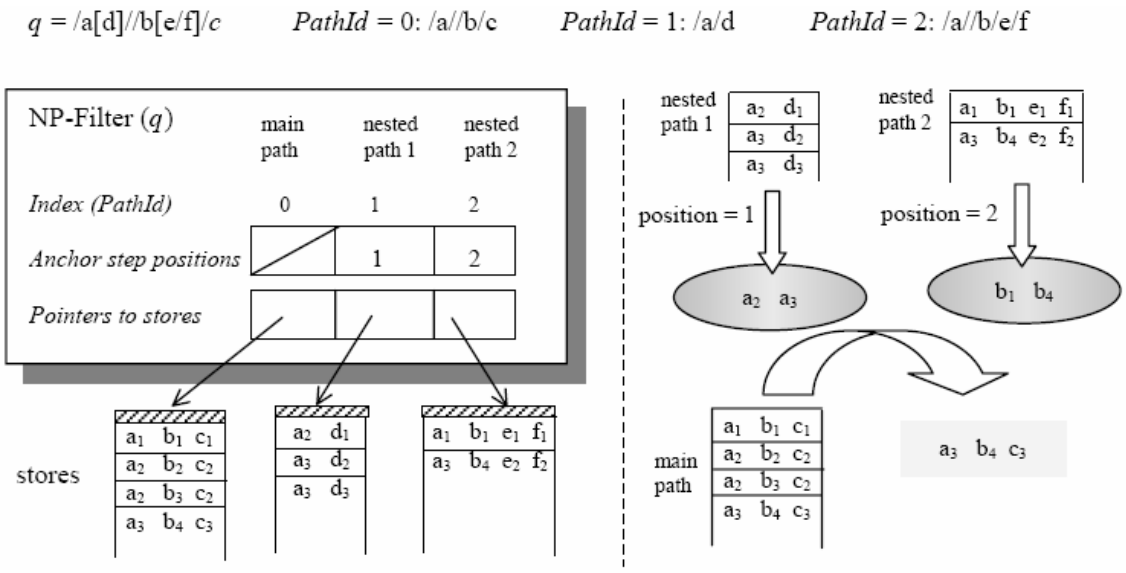
NP-Filter περιέχει πληροφορίες για κάθε μονοπάτι του ερωτήματος στο οποίο αντιστοιχεί. Συγκεκριμένα, αποθηκεύει τη θέση του anchor step σχετικά με το κύριο μονοπάτι του ερωτήματος, με βάση την οποία μπορεί να βρεθεί το τελευταίο κοινό element του εκτεταμένου μονοπατιού με το κύριο. Επίσης, το φίλτρο κρατάει για κάθε μονοπάτι ένα *πεδίο* στο οποίο αποθηκεύει τα ταιριάσματα για το συγκεκριμένο μονοπάτι.

Με βάση τα όσα περιγράψαμε σε προηγούμενη ενότητα για τη μηχανή ταιριάσματος μονοπατιών και έχοντας υπόψη ότι τόσο τα κύρια όσο και τα εκτεταμένα μονοπάτια εισάγονται στη μηχανή αυτή, είναι εμφανές ότι η επεξεργασία κοινών προθεμάτων μοιράζεται ανάμεσα στα κύρια και τα εκτεταμένα μονοπάτια που βρίσκονται στη μηχανή. Όταν βρεθεί ένα ταίριασμα μονοπατιού, η μηχανή ενημερώνει τα ερωτήματα που περιέχουν το μονοπάτι αυτό και τους λέει και το pathID του συγκεκριμένου μονοπατιού σε καθένα από τα ερωτήματα αυτά. Καθένα από τα ερωτήματα-παραλήπτες αποθηκεύει το ταίριασμα αυτό στο κατάλληλο πεδίο, ανάλογα με το pathID του μονοπατιού.

Η μεταεπεξεργασία γίνεται εντός του κάθε NP-Filter, στο τέλος της επεξεργασίας κάθε XML εγγράφου. Περιλαμβάνει τα ακόλουθα βήματα:

1. *Έλεγχος πεδίων* : Αν κάποιο από τα πεδία που αντιστοιχούν στα μονοπάτια που συνθέτουν το ερώτημα είναι κενό, τότε επιστρέφει False, καθώς για ένα τουλάχιστο μονοπάτι δεν υπάρχει ταίριασμα.
2. *Κατασκευή Φίλτρου* : Διαφορετικά, κατασκευάζεται ένα φίλτρο για κάθε φωλιασμένο μονοπάτι από το πεδίο που του αντιστοιχεί. Αυτό γίνεται παίρνοντας το σύνολο των element id's (χωρίς διπλότυπα) που εμφανίζονται στη θέση του anchor step του φωλιασμένου μονοπατιού.
3. *Φιλτράρισμα Ταιριασμάτων* : Οι δομές ταιριάσματος του κυρίου μονοπατιού περνούν τότε από τα φίλτρα όλων των φωλιασμένων μονοπατιών. Για κάθε ταίριασμα του κυρίου μονοπατιού, το φίλτρο ενός φωλιασμένου μονοπατιού εφαρμόζεται στο element id στην αντίστοιχη θέση σύναψης (anchor step). Αν το φίλτρο δεν περιέχει το συγκεκριμένο element id, το κύριο μονοπάτι δε φεύγει ποτέ από το φίλτρο. Αν το ταίριασμα ενός κυρίου μονοπατιού περάσει από όλα τα φίλτρα, το ερώτημα αποτιμάται σε True και το NP-Filter σταματά.

Στο Σχήμα 31 φαίνονται τα τρία μονοπάτια που συνθέτουν το ερώτημα q και έναν NP-Filter τελεστή γι' αυτό και περιγράφεται η διαδικασία της μεταεπεξεργασίας για το συγκεκριμένο ερώτημα.



Σχήμα 31: Μεταεπεξεργασία ερωτήματος στο YFilter

Στην αριστερή πλευρά φαίνονται εντός του πλαισίου οι δομές δεδομένων τις οποίες κρατάει το NP-Filter. Στη λίστα με της θέσεις βήματος σύναψης (anchor step positions) το στοιχείο της λίστας στη θέση 1 αντιστοιχεί στο πρώτο φωλιασμένο μονοπάτι (δηλ. με pathID = 1), πράγμα που σημαίνει ότι το anchor step γι' αυτό το φωλιασμένο μονοπάτι βρίσκεται στη θέση 1 του κυρίου μονοπατιού. Το στοιχείο της λίστας στη θέση 2 κρατάει την πληροφορία για το δεύτερο φωλιασμένο μονοπάτι. Στην λίστα πεδίων, τρεις δείκτες δείχνουν προς τα πεδία(*stores*) που περιέχουν ταιριάσματα μονοπατιών για τα τρία μονοπάτια που συνθέτουν το ερώτημα.

Στο δεξί μέρος του σχήματος φαίνεται η εκτέλεση του NP-Filter. Τα βέλη με φορά προς τα κάτω δείχνουν την κατασκευή του φίλτρου για τα δύο φωλιασμένα μονοπάτια. Οι θέσεις των βημάτων σύναψης(anchor steps) χρησιμοποιούνται για να βρεθούν τα element id's για κάθε φίλτρο. Το βέλος κάτω από τα φίλτρα δείχνει το πέρασμα των ταιριασμάτων του κυρίου μονοπατιού μέσα από τα δύο φίλτρα. Το πρώτο ταίριασμα του κυρίου μονοπατιού απαλείφεται από το πρώτο φίλτρο, καθώς το αναγνωριστικό του στοιχείου "a" σε αυτό το ταίριασμα δεν περιέχεται στο φίλτρο. Τα επόμενα δύο ταιριάσματα απαλείφονται με τον ίδιο τρόπο από το δεύτερο φίλτρο. Τελικά, το τελευταίο ταίριασμα του κυρίου μονοπατιού περνά και τα δύο φίλτρα και το ερώτημα αποτιμάται έτσι σε *True*.

2.2.7.9 Πολλαπλά επίπεδα φωλιάσματος

Η πρώτη επέκταση που πρέπει να γίνει για υποστηριχθούν πολλαπλά επίπεδα φωλιάσματος από το σύστημα είναι ότι πρέπει να τροποποιηθούν οι τελεστές NP-Filter, ώστε να δίνουν σαν έξοδο ένα ή και όλα τα ταιριάσματα που προέρχονται από τα φίλτρα των φωλιασμένων ερωτημάτων. Στη συνέχεια, για κάθε ερώτημα που περιλαμβάνει πολλαπλά επίπεδα φωλιάσματος, ένα NP-Filter ανατίθεται σε κάθε έκφραση μονοπατιού (απόλυτη ή φωλιασμένη) που περιέχει φωλιασμένα μονοπάτια στα κατηγορήματά της. Αν στα φωλιασμένα ερωτήματα αυτής της έκφρα-

σης μονοπατιού ανατεθούν επιπρόσθετα NP-Filters, τότε το NP-Filter τα χειρίζεται ως τελεστές-παιδιά του. Με αυτό τον τρόπο σχηματίζεται μια ιεραρχία από NP φίλτρα, αντίστοιχη με την ιεραρχία φωλιάσματος των μονοπατιών.

Κατά τη διάρκεια της μεταεπεξεργασίας, η ιεραρχία των NP φίλτρων εκτελείται από κάτω προς τα πάνω. Τα φίλτρα στο κατώτερο επίπεδο της ιεραρχίας προσπελαύνουν τα πεδία που περιέχουν ταιριάσματα μονοπατιών και εκτελούν το φιλτράρισμα που περιγράψαμε παραπάνω και κατόπιν δίνουν ως έξοδο όλα τα ταιριάσματα κυρίων μονοπατιών που προέρχονται από φίλτρα μονοπατιών που είναι φωλιασμένα μέσα σε αυτά. Όταν τα NP φίλτρα του αμέσως ανώτερου επιπέδου λάβουν τα ταιριάσματα μονοπατιών από τους τελεστές – παιδιά τους, αρχίζουν τη δική τους εκτέλεση και βγάζουν με τη σειρά τους την αντίστοιχη έξοδο. Αυτή η διαδικασία συνεχίζεται προς τα πάνω στην ιεραρχία των φίλτρων μέχρις ότου κάποιο από τα φίλτρα που βρίσκονται στο ανώτατο επίπεδο βρει ταίριασμα για κάποιο κύριο μονοπάτι ή εξαντλήσει όλα τα ταιριάσματα που έχει πάρει ως είσοδο. Στην πρώτη περίπτωση το ερώτημα αποτιμάται ως *True*, ενώ στη δεύτερη ως *False*.

2.3 Χωροχρονική Ευρετηρίαση (Spatiotemporal Indexing)

Προκειμένου να υποστηριχθούν αυτές οι υπηρεσίες με βάση τη θέση σε μεγάλη κλίμακα, όπως έχει εξηγηθεί και παραπάνω, το σύστημα θα πρέπει να υλοποιεί:

- Αποδοτικούς αλγόριθμους για το indexing ακίνητων αλλά και κινούμενων αντικειμένων και των τροχιών που αυτά ακολουθούν
- Ερωτήματα εύρεσης κοντινότερων γειτόνων (*nearest neighbor queries*), που είναι ένας από τους βασικότερους τύπους ερωτημάτων για ένα τέτοιο σύστημα

2.3.1 Χωρικές Βάσεις και Indexing Χωρικών και Πολυδιάστατων Δεδομένων

Οι *χωρικές βάσεις δεδομένων* περιέχουν πολυδιάστατα δεδομένα σχετικά με ένα χώρο για τον οποίο έχουμε ρητή γνώση και δεδομένα για τα οποία γνωρίζουμε τις χωρικές τους ιδιότητες, όπως είναι η έκτασή τους και η θέση τους στο χώρο (όπως π.χ. περιοχές και σημεία ενδιαφέροντος). Τα αντικείμενα αυτά συνήθως αναπαρίστανται σε διανυσματική μορφή. Για παράδειγμα, μια χωρική βάση δεδομένων μπορεί να περιλαμβάνει χωρικές πληροφορίες για την περιοχή της Αττικής, δηλαδή το οδικό δίκτυο της περιοχής και σημεία ενδιαφέροντος, όπως είναι βενζινάδικα, φαρμακεία, νοσοκομεία, μηχανές αυτόματης ανάληψης μετρητών κτλ. Η κύρια πρόκληση για ένα σχεδιαστή μιας χωρικής βάσης δεδομένων δεν είναι τόσο η υλοποίηση μιας συλλογής από δομές δεδομένων ειδικού σκοπού για τη συγκεκριμένη εφαρμογή, όσο το να ευρεθούν η αρχιτεκτονική και οι αφαιρέσεις ώστε να υλοποιηθεί ένα σύστημα γενικής χρησιμότητας, με δυνατότητες διαχείρισης όσο το δυνατόν γενικότερης μορφής χωρικών δεδομένων, το οποίο να

μπορεί να παραμετροποιηθεί κατά περίπτωση ώστε να έρθει στα μέτρα ενός συγκεκριμένου πεδίου εφαρμογής. Τα πιο σημαντικά ζητήματα που πρέπει να ληφθούν υπόψη είναι ο χειρισμός μοντέλων χωρικών δεδομένων, βελτιστοποιημένες μέθοδοι πρόσβασης σε πολυδιάστατα δεδομένα, καθώς και γλώσσες ερωτήσεων πάνω σε χωρικά δεδομένα (βλ. [GG97]).

Τα χωρικά δεδομένα στη γενική περίπτωση χαρακτηρίζονται κάποιες ιδιότητες, που τα διαφοροποιούν από τα υπόλοιπα δεδομένα σχετικά με τον τρόπο διαχείρισης τους.

1. Η πιο χαρακτηριστική ιδιότητα των χωρικών δεδομένων είναι ότι είναι *σύνθετα* δεδομένα. Ένα χωρικό αντικείμενο μπορεί να αποτελείται από ένα μόνο σημείο, είτε από πολλές χιλιάδες σύνολα σημείων, αυθαίρετα κατανεμημένα στο χώρο. Ως εκ τούτου είναι γενικώς αδύνατο να χρησιμοποιηθούν παραδοσιακά συστήματα βάσεων δεδομένων, όπου κάθε πλειάδα έχει συγκεκριμένο μέγεθος, για να αποθηκευτούν σύνολα από τέτοια αντικείμενα.
2. Τα χωρικά δεδομένα είναι *δυναμικά*. Οι εισαγωγές και οι διαγραφές είναι αναμεμειγμένες με ενημερώσεις και γι' αυτό το λόγο πρέπει οι δομές δεδομένων που θα χρησιμοποιηθούν να υποστηρίζουν αυτή τη δυναμική συμπεριφορά.
3. Οι χωρικές βάσεις δεδομένων τείνουν να είναι *μεγάλες* σε μέγεθος. Το πιο χαρακτηριστικό παράδειγμα, οι γεωγραφικοί χάρτες είναι συνήθως πολλά gigabytes σε μέγεθος. Γι' αυτό απαιτείται η διάφανη προς το χρήστη χρησιμοποίηση δευτερεύουσας μνήμης.
4. Δεν υπάρχει πρότυπη και τυποποιημένη *άλγεβρα* ορισμένη επί χωρικών δεδομένων. Αυτό σημαίνει ότι δεν υπάρχει ένα καθορισμένο σύνολο τελεστών, αν και ορισμένοι από τους τελεστές χρησιμοποιούνται πιο συχνά από άλλους(όπως η τομή). Οι τελεστές που χρησιμοποιούνται εξαρτώνται άμεσα από το θεματικό πεδίο της κάθε εφαρμογής.
5. Πολλοί χωρικοί τελεστές δεν είναι *κλειστοί*. Για παράδειγμα, η τομή μεταξύ δύο πολυγώνων στο επίπεδο μπορεί να επιστρέψει οποιοδήποτε αριθμό από άσχετα μεταξύ τους σημεία, γωνιές και ξένα μεταξύ τους πολύγωνα. Αυτό δυσκολεύει περισσότερα τα πράγματα, όταν τέτοιοι τελεστές χρησιμοποιούνται επαναληπτικά.
6. Αν και το υπολογιστικό κόστος διαφέρει για κάθε τελεστή, οι χωρικοί τελεστές είναι εν γένει *πιο ακριβοί* σε επεξεργαστική ισχύ απ' ότι οι κοινοί σχεσιακοί τελεστές.

Μια σημαντική κλάση γεωμετρικών τελεστών που απαιτεί ειδική υποστήριξη στο φυσικό επίπεδο είναι η κλάση των *χωρικών τελεστών αναζήτησης*. Η ανάκτηση και η ενημέρωση χωρικών δεδομένων συνήθως εξαρτάται όχι μόνο από την τιμή κάποιων αλφαριθμητικών χαρακτηριστικών, αλλά και από τη χωρική θέση ενός αντικειμένου. Μια ερώτηση ανάκτησης σε μια χωρική βάση συχνά απαιτεί τη γρήγορη εκτέλεση μιας ερώτησης σημείου ή περιοχής. Δοθέντος ενός συνόλου γεωμετρικών αντικειμένων στο κ-διάστατο Ευκλείδιο χώρο E^{κ}, αποθηκευμένου σε μια χωρική βάση δεδομένων, μια *ερώτηση περιοχής (region query)* υπολογίζει τα αντικείμενα της βάσης που επικαλύπτουν ένα δεδομένο διάστημα αναζήτησης $S \subset E^{\kappa}$. Η *ερώτηση σημείου (point query)* μπορεί να θεωρηθεί ως μια εκφυλισμένη ερώτηση περιοχής, που υπολογίζει τα

αντικείμενα που περιέχουν ένα σημείο $p \in E^{\kappa}$. Οι δύο αυτές λειτουργίες απαιτούν πολύ γρήγορη πρόσβαση στα αντικείμενα της βάσης που χωρικά βρίσκονται μέσα με μια περιοχή.

Για την υποστήριξη τέτοιου είδους λειτουργιών χρειάζονται μέθοδοι προσπέλασης πολυδιάστατων δεδομένων (*multidimensional access methods*). Το κύριο πρόβλημα κατά το σχεδιασμό τέτοιων μεθόδων είναι ότι δεν υπάρχει μια *ολική διάταξη* των χωρικών δεδομένων που να διατηρεί τη χωρική τους εγγύτητα. Δηλαδή, δεν υπάρχει αντιστοίχηση από το διδιάστατο (ή πολυδιάστατο) χώρο προς το μονοδιάστατο, έτσι ώστε δύο αντικείμενα που βρίσκονται «κοντά» στον πολυδιάστατο χώρο, να βρίσκονται κοντά και στη μονοδιάστατη αντιστοίχησή του. Αυτό καθιστά το σχεδιασμό αποδοτικών μεθόδων προσπέλασης στα χωρικά δεδομένα πολύ πιο δύσκολο απ' ότι στις παραδοσιακές βάσεις δεδομένων, για τις οποίες υπάρχει μεγάλος αριθμός αποδοτικών μεθόδων πρόσβασης στα δεδομένα, όπως είναι τα B-δέντρα και ο επεκτατός κατακερματισμός. Γενικά, μια μέθοδος για προσπέλαση πολυδιάστατων αντικειμένων θα πρέπει να ικανοποιεί τις ακόλουθες απαιτήσεις:

1. *Δυναμικότητα.* Καθώς τα δεδομένα της βάσης εισάγονται/διαγράφονται από τη βάση με τυχαία σειρά, το σύστημα θα πρέπει να καταγράφει συνεχώς τις όποιες αλλαγές.
2. *Διαχείριση δευτερεύουσας μνήμης.* Παρόλο το μεγάλο πλέον μέγεθος της κύριας μνήμης των υπολογιστών, είναι αρκετά συχνό φαινόμενο η βάση δεδομένων να μη χωράει στην κύρια μνήμη. Ως εκ τούτου, το σύστημα θα πρέπει να χρησιμοποιεί διαφανώς και δευτερεύουσα μνήμη.
3. *Μεγάλο εύρος υποστηριζόμενων λειτουργιών.* Οι μέθοδοι προσπέλασης δε θα πρέπει να υποστηρίζουν πιο αποδοτικά μια λειτουργία σε βάρος της αποδοτικότητας των υπολοίπων
4. *Ανεξαρτησία από τα δεδομένα.* Οι μέθοδοι προσπέλασης θα πρέπει να διατηρούν την αποδοτικότητά τους ακόμα και αν τα δεδομένα εισόδου είναι εξαιρετικά ασύμμετρα. Αυτό έχει ιδιαίτερη σημασία για δεδομένα που κατανέμονται διαφορετικά στις διάφορες διαστάσεις.
5. *Απλότητα.* Περίπλοκες μέθοδοι που λαμβάνουν υπόψη ειδικές περιπτώσεις, μπορεί να είναι δύσκολο να υλοποιηθούν σωστά.
6. *Διαβαθμισιμότητα (scalability).* Οι μέθοδοι προσπέλασης θα πρέπει να προσαρμόζονται στο μέγεθος της βάσης δεδομένων.
7. *Αποδοτικότητα.* Επειδή η αναζήτηση σε χωρικά δεδομένα απαιτεί συνήθως πολλή επεξεργαστική ισχύ και πολλές I/O λειτουργίες, η μέθοδος προσπέλασης θα πρέπει να είναι αρκετά αποδοτική ως προς τη χρήση της CPU και του I/O.

Οι «κλασσικοί» αλγόριθμοι indexing για μονοδιάστατα δεδομένα, όπως τα B και B+ δέντρα, δεν είναι κατάλληλα για indexing χωρικών και πολυδιάστατων δεδομένων. Γι' αυτό το λόγο έχουν προταθεί πολλές άλλες δομές για την ευρετηρίαση πολυδιάστατων δεδομένων. Μια από τις σημαντικότερες από αυτές είναι το R-Tree. Στη συνέχεια προτάθηκαν πολλές παραλλαγές

της δομής αυτής, με αποτέλεσμα να δημιουργηθεί μια «οικογένεια» από δομές που βασίζονται σε R δέντρα, από τις οποίες καμία δεν υπερτερεί των άλλων για όλα τα δυνατά σύνολα δεδομένων, αλλά κάθε τέτοια δομή απευθύνεται κυρίως σε συγκεκριμένες περιοχές εφαρμογών. Μια άλλη αρκετά σημαντική δομή είναι το *quad tree*, για το οποίο έχουν επίσης προταθεί πολλές παραλλαγές, καθεμία από τις οποίες στοχεύει στην επίλυση συγκεκριμένων προβλημάτων και είναι ως εκ τούτου κατάλληλη για συγκεκριμένα πεδία εφαρμογών.

2.3.2 R Trees

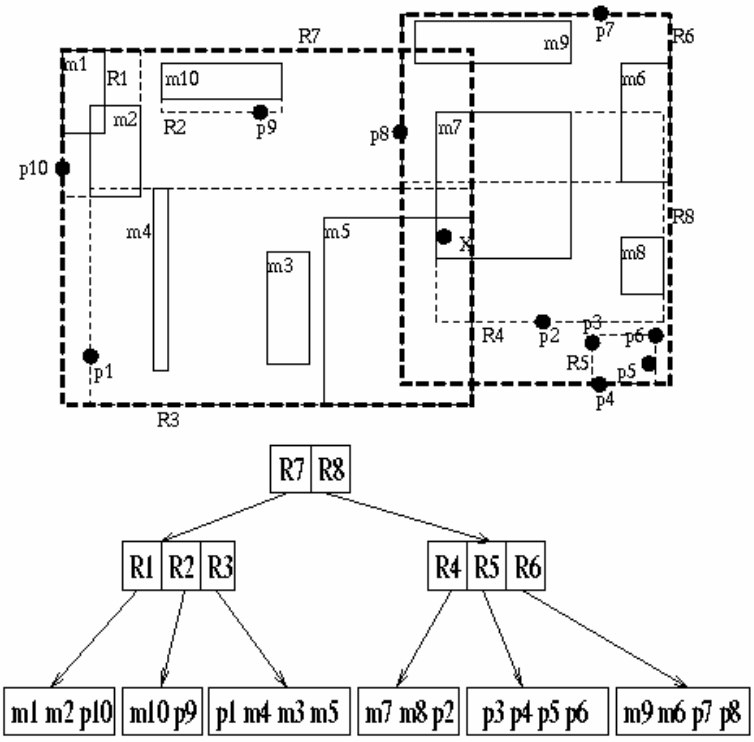
Τα R-Trees (βλ. [Gut84]) είναι μια επέκταση των B+ δέντρων στον πολυδιάστατο χώρο, που αναπαριστούν τα δεδομένα ως διαστήματα(*intervals*) στο χώρο των πολλών διαστάσεων. Δηλαδή ένα R δέντρο είναι μια ιεραρχία από εμφωλευμένα n-διάστατα διαστήματα. Οι κόμβοι των δέντρων αυτών είναι πλειάδες της μορφής *(I, tuple_id)*, όπου $I = (I_0, I_1, I_2, \ldots I_{n-1})$ και το I_i αναπαριστά την έκταση του περιβάλλοντος κουτιού *(bounding box)* του αντικειμένου I στην i-οστή διάσταση. Αν το *v* είναι ένα φύλλο του δέντρου, τότε το $I_n(v)$ είναι το n-διάστατο *ελάχιστο* περιβάλλον κουτί (*Minimum Bounding Box, MBB)* των αντικειμένων που είναι αποθηκευμένα στο *v*. Μερικά από τα πιο εξέχοντα χαρακτηριστικά της δομής αυτής είναι:

- Κάθε κόμβος περιέχει μεταξύ *m* και *M* εγγραφές (εκτός αν είναι η ρίζα).Το κάτω όριο αποτρέπει τον εκφυλισμό του δέντρου και εξασφαλίζει αποδοτική χρησιμοποίηση του αποθηκευτικού χώρου. Όποτε ο αριθμός των απογόνων ενός κόμβου πέσει κάτω από το m, ο κόμβος διαγράφεται και οι απόγονοί του κατανέμονται σε γειτονικούς κόμβους (*condensation)*. Το άνω όριο M προκύπτει από την επιθυμητή ιδιότητα κάθε κόμβος του δέντρου να αντιστοιχεί σε μια ακριβώς σελίδα του δίσκου.
- Τα R-trees έχουν ισοζυγισμένο ύψος, δηλαδή όλα τα φύλλα βρίσκονται στο ίδιο βάθος
- Το ύψος ενός R-Tree είναι $\log_m N$, όπου N είναι ο αριθμός των εγγραφών του ευρετηρίου

Όταν δοθεί μια ερώτηση για μια περιοχή (πολυδιάστατο διάστημα) E η αναζήτηση γίνεται περίπου όπως και στα B δέντρα, δηλαδή ξεκινάει από τη ρίζα του δέντρου και βρίσκει όλες τις εγγραφές με τις οποίες το διάστημα E έχει κάποια επικάλυψη. Στη συνέχεια επισκεπτόμαστε όλους του κόμβους-παιδιά για τους οποίους $I^n(v_i) \cap E \neq \varnothing$ μέχρι να φτάσουμε σε κόμβους-φύλλα οπότε μπορούμε να προσδιορίσουμε αν έχουμε ταίριασμα ή όχι (επειδή στο R-tree χειριζόμαστε μόνο bounding boxes των αντικειμένων, όταν ένα αντικείμενο δεν έχει το σχήμα ενός τέτοιου box(που είναι και η συνήθης περίπτωση), για να λυθεί πλήρως το πρόβλημα της αναζήτησης πρέπει να ελεγχθεί αυτό το σύνολο από υποψήφια αντικείμενα στα οποία δείχνουν τα φύλλα, για να διαπιστωθεί αν το δοθέν σημείο ανήκει στη γεωμετρία του αντικειμένου).

Φυσικά, κατά την πορεία από τη ρίζα προς τα φύλλα μπορεί να ακολουθηθούν πολλά δυνατά μονοπάτια, πράγμα που συνεπάγεται διάβασμα περισσότερων σελίδων από το δίσκο και άρα

μεγαλύτερο κόστος. Στο παρακάτω παράδειγμα και στην περίπτωση του τελεστή αναζήτησης σημείου με σημείο προς αναζήτηση το x του σχήματος (Σχήμα 32), η αναζήτηση θα γίνει κατά μήκος των εξής δύο μονοπατιών: R8 → R4 → m7 και R7 → R3 → m5:



Σχήμα 32: Αναζήτηση σε R tree

Για να εισάγουμε ένα αντικείμενο *o* στο δέντρο, εισάγουμε το minimum bounding box $I^n(o)$ και ένα δείκτη προς το ίδιο το αντικείμενο. Σε αντίθεση με την αναζήτηση, ακολουθούμε μόνο ένα μονοπάτι από τη ρίζα προς τα φύλλα και σε κάθε επίπεδο του δέντρου επιλέγουμε τον κόμβο-παιδί *v* του οποίου το αντίστοιχο διάστημα $I^n(v)$ χρειάζεται τη μικρότερη επέκταση για να περιλάβει το νέο αντικείμενο. Αν πολλά τέτοια διαστήματα ικανοποιούν το κριτήριο αυτό, τότε επιλέγουμε ένα τυχαία και με αυτό τον τρόπο εξασφαλίζουμε ότι κάθε αντικείμενο εισάγεται μόνο σε μια μεριά και δε διασκορπίζεται. Μόλις διασχίζουμε ολόκληρο το μονοπάτι και φθάσουμε σε φύλλο, τότε αν απαιτείται επέκταση της περιοχής στην οποία αντιστοιχεί το φύλλο την προσαρμόζουμε και διαδίδουμε την αλλαγή αυτή προς τα πάνω στο δέντρο. Διαφορετικά, αν το αντικείμενο δε χωράει, χωρίσουμε το φύλλο σε δύο φύλλα και κατανέμουμε τις εγγραφές σε αυτά(δηλ. στην παλαιά και τη νέα σελίδα), ενώ στη συνέχεια διαδίδουμε αυτή τη διάσπαση προς τα πάνω στο δέντρο.

Η εισαγωγή μιας νέας εγγραφής σε ένα R-tree μπορεί να γίνει με βάση τον ακόλουθο αλγόριθμο:

```
Insert (Record E) {
  Node L = ChooseLeaf(E);
  if ( L έχει αρκετό χώρο )
```

```
   insert E into L
   else {
      LL = SplitNode(L);
      insert E into L or LL
   }
   // Διάδοση των αλλαγών προς τα πάνω στο δέντρο
   AdjustTree(L,[LL]);
   if (η ρίζα του δέντρου υποστεί split) {
      // «μεγάλωμα» του δέντρου
      Πρόσθεσε ένα νέο κόμβο ως ρίζα
      Βάλε τη νέα ρίζα να «δείχνει» τα δύο υποδέντρα
   }
}

//Βοηθητικές Συναρτήσεις:
Node ChooseLeaf(Record E) {
   Node N = root;
   while (N δεν είναι φύλλο){
      Επέλεξε μια εγγραφή F τέτοια ώστε η επέκταση του παραλληλογράμου
      να είναι η ελάχιστη δυνατή
      N = F.node;
   }
   return N;
}

AdjustTree(Node first, Node second) {
   Node N = first;
   Node NN = second;
   while (το N δεν είναι η ρίζα) {
      Έστω P ο γονέας του N
      Προσάρμοσε την εγγραφή του N στο P ώστε
      να περιέχει όλα τα παραλληλόγραμμα του N
      Αν το NN υπάρχει(οπότε είχε γίνει διαχωρισμός πριν)
      Τότε βάλε μια εγγραφή για το NN στον P. Αν δε χωράει
      Κάλεσε και πάλι τη SplitNode()
   }
}
```

Για τη διαγραφή ενός στοιχείου, αρχικά κάνουμε μια αναζήτηση στο δέντρο για να δούμε αν το στοιχείο αυτό υπάρχει στο δέντρο. Αν το βρούμε, το διαγράφουμε και έχουμε δύο περιπτώσεις: αν η διαγραφή αυτή δεν προκαλεί συνένωση του κόμβου με κάποιον άλλο κόμβο, τότε απλά ελέγχουμε αν το περιβάλλον διάστημα που του αντιστοιχεί μπορεί να περιοριστεί. Διαφορετικά, αν η διαγραφή αφήσει τον κόμβο με λιγότερα από τα μισά στοιχεία, τότε ο κόμβος αυτός αφαιρείται από το δέντρο και κρατάμε κάπου τα περιεχόμενά του, τα οποία στη συνέχεια επανεισάγουμε στο δέντρο. Μια άλλη προσέγγιση είναι να προσθέσουμε τις εγγραφές αυτές σε αδερφικούς κόμβους, διαδίδοντας τις απαραίτητες αλλαγές προς τα πάνω στο δέντρο.

Ο αλγόριθμος για τη διαγραφή μπορεί να παρασταθεί και ως εξής:

```
delete(Node E){
   // R είναι η ρίζα του δέντρου
   Node L = FindLeaf(R,E);
   Remove E from L;
   CondenseTree();
   Αν η ρίζα έχει ένα μόνο παιδί τότε
   ο κόμβος παιδί γίνεται η νέα ρίζα
```

```
}

Node FindLeaf(Node T, Record E) {
   if (ο Τ είναι φύλλο){
     Έλεγξε κάθε εγγραφή του Τ για να
     δεις αν ταιριάζει με το Ε
     Αν βρεθεί τέτοια return
   }
   else {
     Για κάθε εγγραφή στο Τα δες αν
     η περιοχή που αναπαριστά το Ε
     έχει επικάλυψη με αυτή και αν υπάρχει
   τέτοια επέστρεψε το FindLeaf(T.entry,E)
   }
}
```

Το update της δομής μπορεί να γίνει με συνδυασμό των λειτουργιών του insert και του delete.

Όπως παρατηρήσαμε και παραπάνω, κατόπιν της δομής αυτής παρουσιάστηκαν και μια σειρά από δομές που ήταν παραλλαγές του R-tree και προσπαθούσαν να επιλύσουν κάποια προβλήματά του. Για παράδειγμα, το R^+-tree (βλ. [SRF87]) χρησιμοποιεί την τεχνική της αποκοπής, δηλαδή απαιτεί να μην υπάρχει επικάλυψη μεταξύ των διαστημάτων I^n που βρίσκονται στο ίδιο επίπεδο του δέντρου. Με αυτή την τεχνική οι ερωτήσεις σημείου απαιτούν την αναζήτηση κατά μήκους ενός μόνο μονοπατιού του δέντρου, γεγονός που αυξάνει κατά πολύ την απόδοση της δομής δεδομένων.

Μια άλλη παραλλαγή, το R^*-tree βασίζεται στην παρατήρηση ότι η εισαγωγή των αντικειμένων στο δέντρο έχει πολύ μεγάλο αντίκτυπο στην ολική του απόδοση και γι' αυτό το λόγο χρησιμοποιεί την τεχνική της *επιβεβλημένης επανεισαγωγής (forced-reinsert)*. Σύμφωνα με αυτή όταν ένας κόμβος υπερχειλίζει δεν πρέπει να σπάσει αμέσως, αλλά είναι καλύτερα να αφαιρέσουν έναν αριθμό από εγγραφές του κόμβο και να τον επανεισάγουν στο δέντρο. Μια επέκταση του R^* δέντρου είναι και το TPR tree που είναι μια από τις πιο αποδοτικές λύσεις για το indexing κινούμενων αντικειμένων με δυνατότητες επεξεργασίας ερωτημάτων πρόβλεψης (*predictive queries*). Για το λόγο αυτό θα παρουσιάσουμε κάποιες λεπτομέρειες του R^* δέντρου και στη συνέχεια θα τις επεκτείνουμε για το TPR tree.

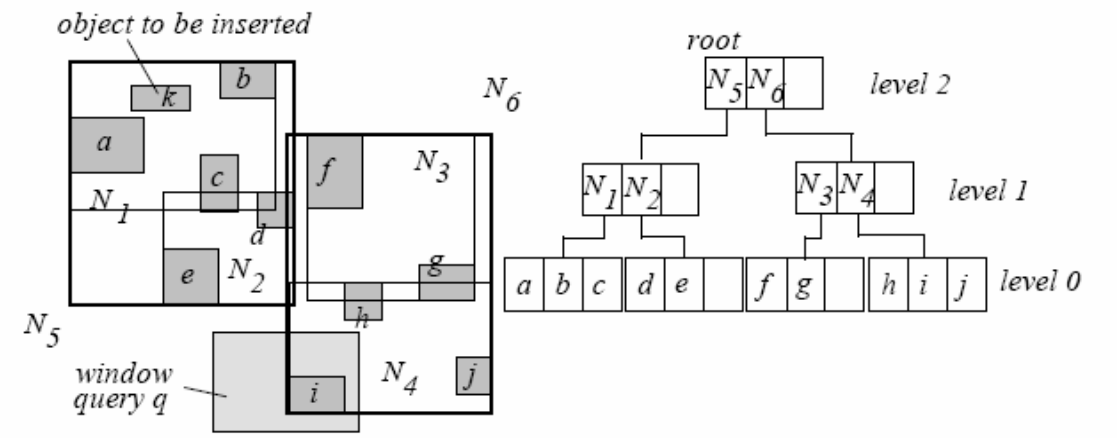
2.3.2.1 R^* Tree

Ο αλγόριθμος κατασκευής ενός R^* δέντρου (βλ.[BKSS90], [Sel00]) έχει ως στόχο την ελαχιστοποίηση των ακόλουθων τεσσάρων τιμών, καθώς αποδεικνύεται ότι όσο μικρότερες είναι αυτές οι τιμές, τόσο μικραίνει η πιθανότητα ένα περιβάλλον κουτί κάποιου αντικειμένου να τέμνει την περιοχή ενός ερωτήματος:

- Το εμβαδόν του κάθε minimum bounding rectangle(*MBR*)
- Την περίμετρό του

- Την επικάλυψη μεταξύ των MBRs δύο αντικειμένων στον ίδιο κόμβο
- Την απόσταση μεταξύ του κέντρου ενός MBR του αντικειμένου και του κέντρου του MBR που αντιστοιχεί στον κόμβο που περιέχει το αντικείμενο, την οποία θα ονομάζουμε *κεντρική* απόσταση

Σχετικά με την εισαγωγή, ο αλγόριθμος αποφασίζει σε κάθε επίπεδο του δέντρου ποιο μονοπάτι θα ακολουθήσει με άπληστο τρόπο. Για παράδειγμα, στο σχήμα που ακολουθεί (Σχήμα 33), έστω ότι εισάγουμε το αντικείμενο k. Στο επίπεδο της ρίζας ο αλγόριθμος επιλέγει την εγγραφή της οποίας το MBR χρειάζεται τη μικρότερη επέκταση για να καλύψει και το k. Επιλέγεται το N_5 που δε χρειάζεται να διευρυνθεί καθόλου(ενώ το N_6 χρειάζεται). Τότε, στο επόμενο επίπεδο, δηλαδή σε κάποιο παιδί του N_5 ο αλγόριθμος επιλέγει την εγγραφή της οποίας το MBR όταν επεκταθεί θα οδηγήσει στην μικρότερη δυνατή επικάλυψη σε σχέση με τους υπόλοιπους «αδερφικούς» κόμβους.

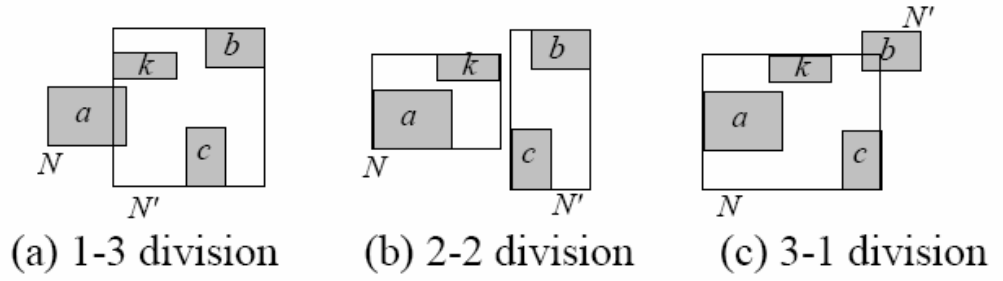


Σχήμα 33: Εισαγωγή σε R* tree

Μια *υπερχείλιση* συμβαίνει όταν το φύλλο στο οποίο θα φτάσουμε (το N_1 εδώ) είναι γεμάτο, περιέχει το μέγιστο αριθμό εγγραφών. Σε αυτήν την περίπτωση ο αλγόριθμος, όπως ειπώθηκε και παραπάνω, προσπαθεί να αφαιρέσει και να επανεισάγει ένα μέρος από τις εγγραφές του κόμβου με σκοπό να αποφύγει το διαχωρισμό του κόμβου, αν κάποια εγγραφή μπορεί να μεταφερθεί σα άλλον κόμβο. Έχει διαπιστωθεί ότι καλύτερη απόδοση υπάρχει όταν το σύνολο των εγγραφών που θα επανεισαχθούν είναι το 30% των εγγραφών με τη μεγαλύτερη κεντρική απόσταση. Στο παραπάνω σχήμα, το b επιλέγεται, που η κεντρική του απόσταση είναι η μεγαλύτερη.

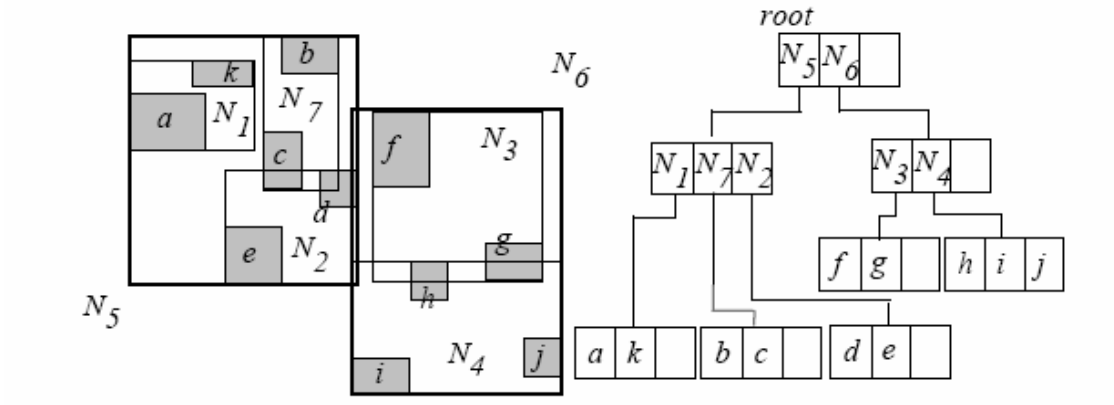
Το σπάσιμο κόμβων τελικά γίνεται αν μετά από τις επανεισαγωγές συνεχίζεται η υπερχείλιση. Ο αλγόριθμος για το σπάσιμο των κόμβων αποτελείται από δύο βήματα. Στο πρώτο βήμα αποφασίζεται ο άξονας στον οποίο θα γίνει ο χωρισμός, και είναι αυτός στον οποίο έχουμε τη μικρότερη *συνολική περίμετρο* που υπολογίζεται ως εξής: για το x-άξονα για παράδειγμα, ο αλγόριθμος ταξινομεί όλες τις εγγραφές με βάση τις συντεταγμένες της αριστερής πλευράς τους(στο

παραπάνω σχήμα η λίστα αυτή θα είναι a, k, b, c) και στη συνέχεια θεωρεί κάθε δυνατή διαίρεση στα δύο της ταξινομημένης λίστας από την οποία εξασφαλίζεται ότι κάθε κόμβος θα είναι τουλάχιστον κατά 40% πλήρης. Αυτός ο περιορισμός παραβλέπεται στο ακόλουθο σχήμα (Σχήμα 34), το οποίο συνεχίζει το προηγούμενο παράδειγμα. Η πρώτη διαίρεση για παράδειγμα τοποθετεί την πρώτη εγγραφή στον N και τις υπόλοιπες 3 στον κόμβο N'.



Σχήμα 34: Διαίρεση Κόμβου R* tree

Ο αλγόριθμος υπολογίζει τις περιμέτρους για το N και το N' για καθεμιά από αυτές τις περιπτώσεις. Κατόπιν, ένα δεύτερο πέρασμα επαναλαμβάνει τη διαδικασία αυτή με βάση τη δεξιά πλευρά των MBRs και τελικά η συνολική περίμετρος στο x-άξονα ισούται με το άθροισμα των περιμέτρων που ελήφθησαν από τα δύο περάσματα. Μετά την επιλογή του άξονα(που είναι αυτός με τη μικρότερη συνολική περίμετρο)ο αλγόριθμος ταξινομεί τις εγγραφές με βάση την πάνω ή την κάτω πλευρά τους στην επιλεγμένη διάσταση και θεωρεί και πάλι όλες τις δυνατές διαιρέσεις. Η τελική διαίρεση είναι αυτή που έχει τη μικρότερη επικάλυψη μεταξύ των MBRs των κόμβων που προκύπτουν. Αναφερόμενοι στο παραπάνω παράδειγμα, έστω ότι επιλέγεται ως άξονας χωρισμού ο x, τότε από τις διάφορες δυνατές διαιρέσεις, η 2-2 διαίρεση εγγυάται μηδενική επικάλυψη και γι' αυτό το λόγο επιλέγεται για το σπάσιμο του κόμβου. Με βάση τα παραπάνω, το δέντρο μετά την εισαγωγή του νέου κόμβου θα μοιάζει ως εξής (Σχήμα 35):



Σχήμα 35: Το R* tree μετά την εισαγωγή

Η διαγραφή στα R* δέντρα είναι σχετικά απλή και γίνεται με τον ίδιο τρόπο που περιγράψαμε για τα R-trees.

2.3.3 Time Parametrized Tree (TPR-Tree)

Τα *TPR δέντρα* (βλ. [SJL00], [TPS03]) αποτελούν μια επέκταση του R^* δέντρου η οποία στοχεύει στο να υποστηρίζονται αποδοτικά ερωτήματα σχετικά με την τρέχουσα αλλά και την προβλεπόμενη θέση ενός αντικειμένου που κινείται σε χώρους 1, 2 ή 3 διαστάσεων καθώς και σύνολα δεδομένων (datasets) τα οποία είναι δυναμικά, δηλαδή τα αντικείμενα μπορεί να εμφανίζονται και να εξαφανίζονται και γενικότερα να αλλάζουν οι αναμενόμενες θέσεις των αντικειμένων.

Σε σχέση με το παραπάνω πρόβλημα μπορούν να ακολουθηθούν πολλές προσεγγίσεις. Θεωρώντας ότι τα αντικείμενα κινούνται σε ένα d-διάστατο χώρο (d=1,2,3), οι μελλοντικές τους τροχιές στο χρόνο μπορούν να ευρετηριαστούν ως γραμμές στον *(d+1)*-διάστατο χώρο. Εναλλακτικά, κανείς θα μπορούσε να απεικονίσει τις τροχιές αυτές σε *σημεία* κάποιου χώρου περισσοτέρων διαστάσεων και στη συνέχεια να δημιουργήσει ένα index για αυτά. Η προσέγγιση που ακολουθείται στα TPR δέντρα είναι ότι δημιουργείται ένας index στον αρχικό d-διαστατο χώρο των αντικειμένων, παραμετροποιώντας τη δομή δεδομένων με διανύσματα ταχύτητας, οπότε μπορούμε να «δούμε» τη δομή σε μια μελλοντική στιγμή.

Επίσης πολλές προτεινόμενες προσεγγίσεις απαιτούν περιοδική ανανέωση και χτίσιμο από την αρχή του ευρετηρίου, δηλαδή χρησιμοποιούν διαφορετικά ευρετήρια για διαφορετικές χρονικές περιόδους και για το λόγο αυτό όταν η χρονική περίοδος ισχύος ενός ευρετηρίου εκπνεύσει, πρέπει να δημιουργηθεί ένα νέο ευρετήριο. Στα TPR δέντρα το ευρετήριο είναι το ίδιο για όλες τις χρονικές περιόδους, αλλά είναι βελτιστοποιημένο για ένα συγκεκριμένο ορίζοντα χρόνου με αποτέλεσμα από ένα χρονικό σημείο και μετά η απόδοσή του να φθίνει.

Στο TPR tree τα περιβάλλοντα ορθογώνια που αποθηκεύονται στο δέντρο είναι συναρτήσεις του χρόνου, όπως και οι θέσεις των κινουμένων αντικειμένων οι οποίες ευρετηριάζονται. Με αυτή την προσέγγιση, τα περιβάλλοντα ορθογώνια(bounding rectangles) έχουν την ικανότητα να «ακολουθούν» συνεχώς τα σημεία που περιέχουν(ή άλλα ορθογώνια) καθώς αυτά κινούνται.

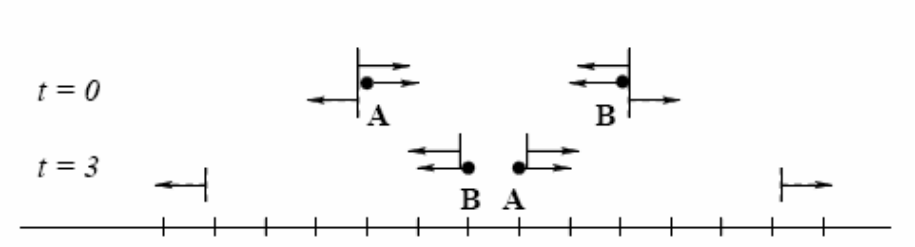
2.3.3.1 Δομή

Όπως έχει γίνει φανερό και από τα παραπάνω, το TPR tree είναι ένα ευρετήριο που έχει την ίδια δομή με ένα R^* δέντρο. Οι εγγραφές στα φύλλα περιέχουν τη θέση ενός κινούμενου αντικειμένου καθώς και ένα δείκτη προς το ίδιο το αντικείμενο. Κατ' αντιστοιχία οι κόμβοι που δεν είναι φύλλα περιέχουν εγγραφές που είναι ζεύγη ενός δείκτη προς ένα υποδέντρο που ξεκινάει από τον κόμβο και ένα ορθογώνιο που φράσσει τις θέσεις όλων των κινούμενων αντικειμένων(ή άλλων ορθογωνίων) που βρίσκονται στο υποδέντρο αυτό.

Ένα κινούμενο αντικείμενο αναπαρίσταται από μια θέση αναφοράς και ένα διάνυσμα ταχύτητας, δηλ. ένα ζεύγος *(x, v)*, όπου $x = x(t_{ref})$, όπου t_{ref} είναι είτε η χρονική στιγμή της δημιουρ-

γίας του ευρετηρίου είτε κάποιος άλλος χρόνος αναφοράς. Για το φράξιμο ενός συνόλου από d-διάστατα αντικείμενα χρησιμοποιούνται d-διάστατα ορθογώνια που είναι επίσης χρονικά παραμετροποιημένα, δηλαδή οι συντεταγμένες τους είναι συναρτήσεις του χρόνου. Ένα χρονικά παραμετροποιημένο φράσσον ορθογώνιο φράσσει όλα τα αντικείμενα ή ορθογώνια στο παρόν και στο μέλλον, δηλαδή σε όλες τις χρονικές στιγμές εκτός από τις παρελθούσες.

Ο αποθηκευτικός χώρος που απαιτείται για να κρατάμε το περιβάλλον ορθογώνιο είναι τόσο μεγαλύτερος, όσο πιο «στενά» το ορθογώνιο αυτό φράσει τα περιεχόμενα σε αυτό αντικείμενα, τόσο στη χωρική όσο και στη χρονική τους διάσταση. Επομένως η ιδανική περίπτωση θα ήταν να χρησιμοποιούμε χρονικά παραμετροποιημένα περιβάλλοντα ορθογώνια (MBRs) τα οποία να είναι πάντοτε *ελάχιστα*. Η απαίτηση αυτή όμως θα είχε απαγορευτικές απαιτήσεις σε αποθηκευτικό χώρο (θα ήταν στην ουσία σαν να βλέπουμε ξεχωριστά καθένα από τα αντικείμενα που θα περιέχονταν στο ορθογώνιο). Γι΄ αυτό το λόγο το TPR tree υιοθετεί μια πιο *συντηρητική* προσέγγιση και χρησιμοποιεί περιβάλλοντα ορθογώνια τα οποία είναι μεν ελάχιστα σε κάποια δεδομένη χρονική στιγμή, αλλά όχι απαραίτητα στις επόμενες χρονικές στιγμές. Για τέτοιου είδους συντηρητικά περιβάλλοντα διαστήματα σε μια διάσταση για παράδειγμα, το κάτω όριο του διαστήματος πρέπει να κινείται με την ελάχιστη ταχύτητα των αντικειμένων που περικλείει, ενώ το πάνω όριο με τη μέγιστη από τις ταχύτητες των περικλειόμενων αντικειμένων. Με αυτόν τον τρόπο είμαστε σίγουροι ότι το περιβάλλον διάστημα θα φράσσει όλα τα περικλειόμενα αντικείμενα και σε οποιαδήποτε μελλοντική χρονική στιγμή. Για μία διάσταση αυτό φαίνεται παρακάτω (Σχήμα 36) :



Σχήμα 36: Το περιβάλλον διάστημα σε TPR tree

Όπως φαίνεται στο σχήμα, το αριστερό άκρο του διαστήματος ξεκινά από τη θέση του αντικειμένου Α τη χρονική στιγμή Ο και κινείται προς τα αριστερά με την ταχύτητα του αντικειμένου Β, ενώ το δεξί άκρο του διαστήματος ξεκινά από το αντικείμενο Β τη χρονική στιγμή 0 και κινείται προς τα δεξιά με την ταχύτητα του αντικειμένου Α. Όπως είναι εμφανές, τα περιβάλλοντα διαστήματα που ορίζονται με τον παραπάνω συντηρητικό τρόπο δε συρρικνώνονται ποτέ. Στην καλύτερη περίπτωση διατηρούν σταθερό μέγεθος, όταν όλα τα περικλειόμενα σημεία έχουν το ίδιο διάνυσμα ταχύτητας.

Ένα d-διάστατο ορθογώνιο μπορούμε να το αναπαριστούμε με τις προβολές του στους d άξονες $[a_1^{\prec}, a_1^{\succ}],...,[a_d^{\prec}, a_d^{\succ}]$ με $a_1^{\prec} < a_1^{\succ}$ (για όλο το υπόλοιπο κείμενο το σύμβολο «$^{\prec}$» θα συμβολίζει

το κάτω άκρο ενός διαστήματος, ενώ το «$\succ$» το πάνω άκρο του διαστήματος. Θεωρώντας τώρα ότι η t_{ref} είναι η χρονική στιγμή της δημιουργίας του ευρετηρίου ($t_{ref} = t_l$), για ένα μονοδιάστατο χρονικά παραμετροποιημένο περικλείον διάστημα θα έχουμε αντίστοιχα:

$$[x^{\prec}(t), x^{\succ}(t)] = [x^{\prec}(t_l) + v^{\prec}(t - t_l), x^{\succ}(t_l) + v^{\succ}(t - t_l)]$$

Και έτσι το διάστημα αυτό μπορεί να παρασταθεί ως $(x^{\prec}, x^{\succ}, v^{\prec}, v^{\succ})$, όπου:

(το o_i είναι το i-οστό περικλειόμενο αντικείμενο)

$$x^{\prec} = x^{\prec}(t_l) = \min_i \{o_i.x^{\prec}(t_l)\}$$

$$x^{\succ} = x^{\succ}(t_l) = \max_i \{o_i.x^{\succ}(t_l)\}$$

$$v^{\prec} = \min_i \{o_i.v^{\prec}\}$$

$$v^{\succ} = \max_i \{o_i.v^{\succ}\}$$

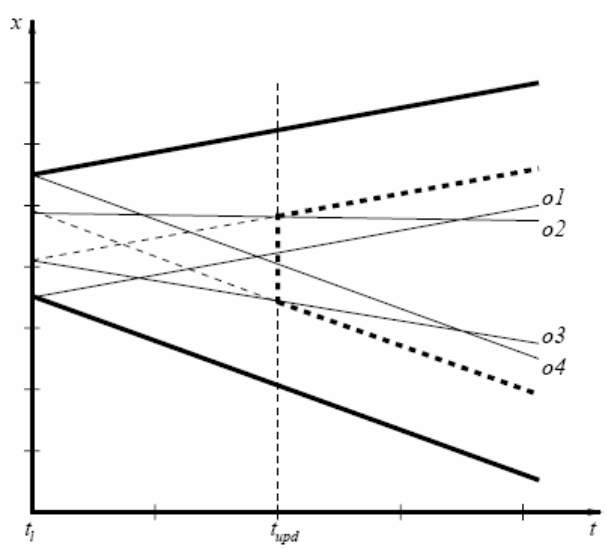
Φυσικά, αν αντί από περικλειόμενο διάστημα έχουμε περικλειόμενο σημείο, τότε τα $o_i.x^{\prec}(t_l)$ και $o_i.x^{\succ}(t_l)$ αντικαθίστανται από το $o_i.x\ (t_l)$ και το αντίστοιχο γίνεται και για τις ταχύτητες.

Τα περιβάλλοντα ορθογώνια που περιγράφηκαν παραπάνω καλούνται *load-time* περιβάλλοντα ορθογώνια, επειδή δημιουργούνται όταν φορτώνεται το ευρετήριο για πρώτη φορά. Επειδή τα ορθογώνια αυτά ποτέ δε συρρικνώνονται, αλλά αντιθέτως τείνουν να διευρύνονται πάρα πολύ, είναι επιθυμητό να μπορούμε περιστασιακά να προσαρμόζουμε το μέγεθός τους. Συγκεκριμένα, με δεδομένο ότι τα ερωτήματα που γίνονται στο ευρετήριο αφορούν πάντοτε την παρούσα χρονική στιγμή ή μελλοντικές χρονικές στιγμές, τα περιβάλλοντα ορθογώνια μπορούν να προσαρμόζονται όταν γίνει ενημέρωση σε κάποιο από τα κινούμενα αντικείμενα ή διαστήματα που περιβάλλουν. Αν t_{upd} είναι η χρονική στιγμή στην οποία γίνεται η ενημέρωση, οι παραπάνω προσαρμογές μπορούν να γίνουν με βάση τους τύπους:

$$x^{\prec} = \min_i \{o_i.x^{\prec}(t_{upd})\} - v^{\prec}(t_{upd} - t_l) \text{ και}$$

$$x^{\succ} = \max_i \{o_i.x^{\succ}(t_{upd})\} - v^{\succ}(t_{upd} - t_l)$$

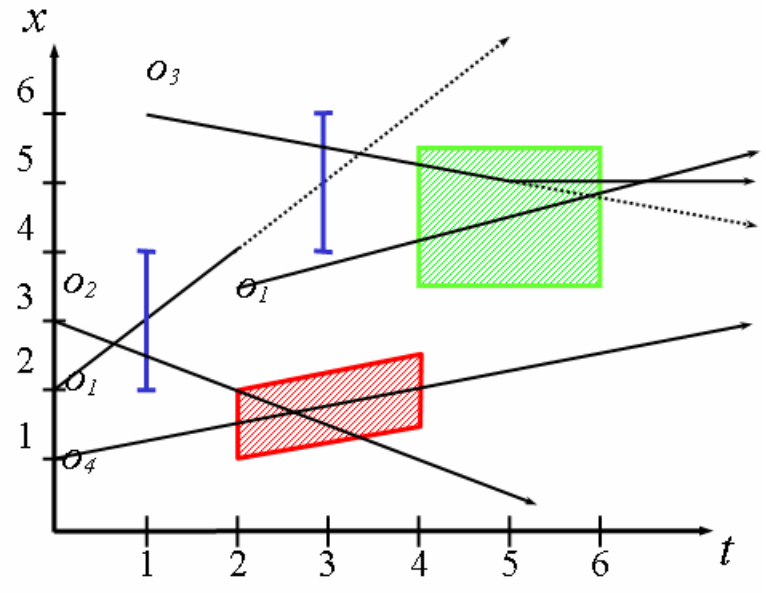
Τα νέα περιβάλλοντα ορθογώνια που προκύπτουν από τις παραπάνω ενημερώσεις ονομάζονται γι' αυτό *update-time* περιβάλλοντα ορθογώνια. Στο σχήμα που ακολουθεί φαίνονται για τέσσερα κινούμενα αντικείμενα τόσο τα load-time περιβάλλοντα διαστήματα τους (με τις έντονες γραμμές), όσο και το update-time περιβάλλον διάστημα (με διακεκομμένη γραμμή) που δημιουργείται κατά την ενημέρωση τη χρονική στιγμή t_{upd}. Όπως φαίνεται και από το σχήμα (Σχήμα 37), το νέο περιβάλλον διάστημα είναι αρκετά πιο στενό από το αρχικό, και επομένως καλύτερο:



Σχήμα 37: Load time και update-time περιβάλλοντα διαστήματα

2.3.3.2 Επεξεργασία Ερωτημάτων

Ο πρώτος τύπος ερωτημάτων είναι ερωτήματα που ορίζονται για ένα υπερ-ορθογώνιο R σε κάποιο χρονικό σημείο t, δηλαδή ερωτήματα του τύπου «να βρεθούν όλα τα αντικείμενα που βρίσκονται εντός του ορθογωνίου R τη χρονική στιγμή t». Τα ερωτήματα του τύπου αυτού μπορούν να απαντηθούν όπως και σε ένα απλό R-tree, με μόνη διαφορά ότι όλα τα περιβάλλοντα ορθογώνια υπολογίζονται για τη χρονική στιγμή που ορίζει το ερώτημα, δηλαδή ένα περιβάλλον διάστημα $(x^{\prec}, x^{\succ}, v^{\prec}, v^{\succ})$ ικανοποιεί ένα ερώτημα $([a^{\prec}(t), a^{\succ}(t)], t^q)$ αν και μόνο αν $a^{\prec} \le x^{\prec} + v^{\prec}(t^q - t_l) \wedge a^{\succ} \ge x^{\succ} + v^{\succ}(t^q - t_l)$. Ο τύπος αυτός των ερωτημάτων παριστάνεται στο σχήμα (Σχήμα 38) που ακολουθεί με μπλε χρώμα:

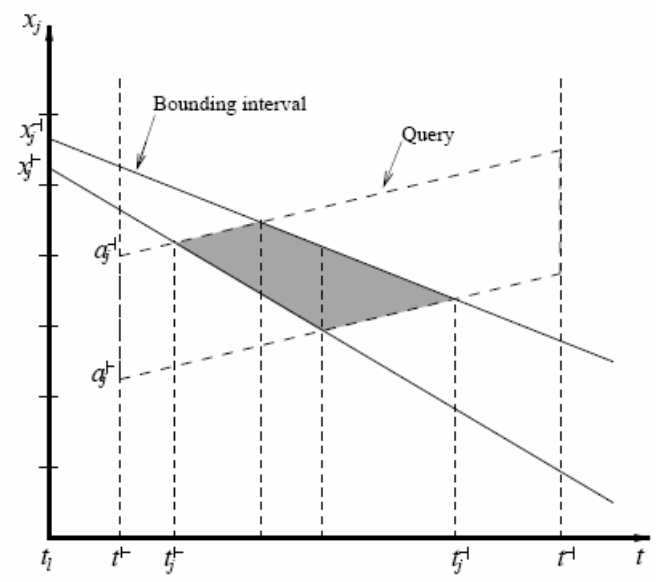


Σχήμα 38: Τύποι ερωτημάτων σε TPR tree

Ο δεύτερος τύπος ερωτημάτων αποτελείται από ερωτήματα παραθύρου, δηλαδή της μορφής $(R, t^{\prec}, t^{\succ})$, που επιστρέφουν αντικείμενα που τέμνουν το ορθογώνιο R σε κάποια χρονική στιγμή από $t^{\prec}$ έως $t^{\succ}$. Πιο φορμαλιστικά, ένα τέτοιο ερώτημα μας δίνει αντικείμενα με τροχιές στο χώρο $(\overline{x}, t)$ που τέμνουν το υπερ-ορθογώνιο (hyper-rectangle) $\left([a_1^{\prec}, a_1^{\succ}], [a_2^{\prec}, a_2^{\succ}], ..., [a_d^{\prec}, a_d^{\succ}], [t^{\prec}, t^{\succ}]\right)$. Στο παραπάνω σχήμα ένα τέτοιο ερώτημα φαίνεται με πράσινο χρώμα.

Ο τρίτος τύπος ερωτημάτων, τα κινούμενα ερωτήματα, επιστρέφουν αντικείμενα τα οποία τέμνουν ένα *κινούμενο* ορθογώνιο σε κάποια χρονική στιγμή από $t^{\prec}$ έως $t^{\succ}$. Στο προηγούμενο σχήμα φαίνεται με κόκκινο χρώμα ένα ερώτημα του τύπου αυτού.

Για να απαντηθούν ερωτήματα του δεύτερου και τρίτου τύπου πρέπει να είμαστε σε θέση να ελέγξουμε αν το ορθογώνιο (τραπεζοειδές για ερωτήματα του 3ου τύπου) του ερωτήματος τέμνεται με το τραπεζοειδές που σχηματίζεται από το κομμάτι της τροχιάς κάποιου περιβάλλοντος ορθογωνίου μεταξύ των χρονικών στιγμών που καθορίζει το ερώτημα (βλ. Σχήμα 39). Αυτό είναι απλό να γίνει όταν ο χώρος είναι μονοδιάστατος, αλλά διαφορετικά είναι ένα μη τετριμμένο πρόβλημα. Ακολουθεί ένας αλγόριθμος με βάση τον οποίο μπορούμε να κάνουμε τον παραπάνω έλεγχο.



Σχήμα 39: Απάντηση ερωτημάτων 3ου τύπου

Ο αλγόριθμος αυτός μας απαντάει δηλαδή κατά πόσο ένα d-διάστατο χρονικά παραμετροποιημένο περιβάλλον ορθογώνιο R, που ορίζεται ως $(x_1^{\prec}, x_1^{\succ}, x_2^{\prec}, x_2^{\succ}, ...x_d^{\prec}, x_d^{\succ}, v_1^{\prec}, v_1^{\succ}, v_2^{\prec}, v_2^{\succ}, ...v_d^{\prec}, v_d^{\succ})$ τέμνει ένα κινούμενο ερώτημα , το $Q = (([a_1^{\prec}, a_1^{\succ}], [a_2^{\prec}, a_2^{\succ}], ..., [a_d^{\prec}, a_d^{\succ}], [w_1^{\prec}, w_1^{\succ}], [w_2^{\prec}, w_2^{\succ}], ..., [w_d^{\prec}, w_d^{\succ}]), t^{\prec}, t^{\succ})$.

Ο συγκεκριμένος αλγόριθμος βασίζεται στην παρατήρηση ότι για να τέμνονται δύο κινούμενα ορθογώνια πρέπει να υπάρχει τουλάχιστο μια χρονική στιγμή κατά την οποία οι εκτάσεις τους σε κάθε διάσταση του χώρου να τέμνονται επίσης. Έτσι, για κάθε διάσταση j $(1 \le j \le d)$, ο αλγόριθμος υπολογίζει το χρονικό διάστημα $I_j = [t_j^{\prec}, t_j^{\succ}] \subset [t^{\prec}, t^{\succ}]$ κατά το οποίο τα δύο ορθογώνια τέμνονται στη συγκεκριμένη διάσταση. Αν συνολικά $I = \cap_{j=1}^{d} I_j = \varnothing$, τότε τα κινούμενα αντικείμενα δεν τέμνονται και έτσι επιστρέφεται ως αποτέλεσμα το κενό σύνολο. Διαφορετικά, ο αλγόριθμος γνωρίζει το διάστημα *I* κατά το οποίο τα ορθογώνια τέμνονται.

Τα διαστήματα για κάθε διάσταση υπολογίζονται ως:

$$\mathrm{I}_j = \begin{cases} \varnothing & ,\alpha\nu \quad a_j^{\prec} > x_j^{\succ}(t^{\prec}) \wedge a_j^{\prec}(t^{\succ}) > x_j^{\succ}(t^{\succ}) \vee a_j^{\succ} < x_j^{\prec}(t^{\prec}) \wedge a_j^{\succ}(t^{\succ}) < x_j^{\prec}(t^{\succ}) \\ [t_j^{\prec}, t_j^{\succ}] & ,\textit{διαφορετικα} \end{cases}$$

Η πρώτη από τις συζεύξεις στη συνθήκη σημαίνει ότι το Q είναι πάνω από το R, ενώ η δεύτερη το αντίθετο. Για να υπολογίσουμε τα $t_j^{\prec}$ και $t_j^{\succ}$ σκεφτόμαστε ως εξής. Θεωρούμε την περίπτωση που το Q είναι κάτω από το R τη χρονική στιγμή $t_j^{\succ}$. Τότε το Q δε μπορεί να είναι κάτω από το R τη χρονική στιγμή $t_j^{\prec}$, γιατί αυτό θα σήμαινε ότι είναι συνεχώς κάτω από το R και επομένως δεν υπάρχει τομή, γεγονός που έχει αποκλειστεί όμως σε προηγούμενο βήμα του αλγορίθμου. Κατά συνέπεια η ευθεία $a_j^{\succ} + w_j^{\succ}(t - t^{\prec})$ τέμνει την ευθεία $x_j^{\prec}(t^{\prec}) + v_j^{\prec}(t - t^{\prec})$ εντός του διαστήματος $[t^{\prec}, t^{\succ}]$. Η λύση του συστήματος των δύο αυτών εξισώσεων μας δίνει το επιθυμητό $t_j^{\succ}$. Για την εύρεση και του άλλου άκρου του διαστήματος σκεπτόμαστε με τον ίδιο τρόπο. Οι τελικοί τύποι για τον υπολογισμό των λύσεων φαίνονται παρακάτω:

$$t_j^{\prec} = \begin{cases} t^{\prec} + \dfrac{x_j^{\succ}(t^{\prec}) - a_j^{\prec}}{w_j^{\prec} - v_j^{\succ}} & ,\alpha\nu \quad a_j^{\prec} > x_j^{\succ}(t^{\prec}) \\ t^{\prec} + \dfrac{x_j^{\prec}(t^{\prec}) - a_j^{\succ}}{w_j^{\prec} - v_j^{\succ}} & ,\alpha\nu \quad a_j^{\succ} > x_j^{\succ}(t^{\prec}) \\ t^{\prec}, & \textit{διαφορετικα} \end{cases}$$

Για το $t_j^{\succ}$:

$$t_j^{\succ} = \begin{cases} t^{\prec} + \dfrac{x_j^{\succ}(t^{\prec}) - a_j^{\prec}}{w_j^{\prec} - v_j^{\succ}} & ,\alpha\nu \quad a_j^{\prec}(t^{\succ}) > x_j^{\succ}(t^{\prec}) \\ t^{\prec} + \dfrac{x_j^{\prec}(t^{\prec}) - a_j^{\succ}}{w_j^{\succ} - v_j^{\prec}} & ,\alpha\nu \quad a_j^{\succ}(t^{\succ}) > x_j^{\prec}(t^{\prec}) \\ t^{\succ}, & \textit{διαφορετικα} \end{cases}$$

2.3.3.3 Εισαγωγή και Διαγραφή

Ο αλγόριθμος σύμφωνα με τον οποίο γίνονται οι εισαγωγές στο δέντρο είναι παρόμοιος με αυτόν του R* δέντρου. Ο αλγόριθμος κατασκευής του δέντρου έχει ως στόχο την ελαχιστοποίηση των ιδίων τιμών με αυτές που προσπαθεί να ελαχιστοποιήσει ο αλγόριθμος εισαγωγής του R* δέντρου που περιγράψαμε παραπάνω, αν καθεμιά από αυτές τις τιμές αντικατασταθεί από το χρονικό της ολοκλήρωμα, ήτοι:

- $\int_{T_C}^{T_C+H} A(N,t)dt$ για το εμβαδόν, αν *A(N,t)* το εμβαδόν τη χρονική στιγμή t
- $\int_{T_C}^{T_C+H} P(N,t)dt$ για την περίμετρο, αν *P(N,t)* η περίμετρος τη στιγμή t
- $\int_{T_C}^{T_C+H} OVR(N_1,N_2,t)dt$ για την επικάλυψη των περιβαλλόντων ορθογωνίων των N_1 και N_2, αν η *OVR* επιστρέφει την επικάλυψή τους τη χρονική στιγμή t
- $\int_{T_C}^{T_C+H} CDist(N_1,N_2,t)dt$ για την κεντρική τους απόσταση, αν η *CDist* επιστρέφει την κεντρική τους απόσταση τη στιγμή t.

Ο υπολογισμός των ολοκληρωμάτων για το εμβαδό, την περίμετρο και την κεντρική απόσταση μπορεί να γίνει απευθείας. Για τον υπολογισμό του ολοκληρώματος της τομής δύο χρονικά παραμετροποιημένων MBRs μπορούμε να χρησιμοποιήσουμε τον αλγόριθμο που περιγράφηκε προηγουμένως για την εύρεση της τομής δύο ορθογωνίων. Συγκεκριμένα ο αλγόριθμος διαιρεί το χρονικό διάστημα που επιστρέφεται από τον αλγόριθμο εύρεσης επικαλύψεων σε συνεχόμενα διαστήματα τέτοια ώστε κατά τη διάρκεια καθενός από αυτά η τομή να εκφράζεται ως ένα χρονικά παραμετροποιημένο ορθογώνιο. Τότε το ολοκλήρωμα μπορεί να υπολογιστεί ως το άθροισμα των εμβαδών των ορθογωνίων αυτών. Αυτό μπορεί να φανεί και στο σχήμα της σελίδας 14 όπου έχουμε διαίρεση του χρονικού διαστήματος σε τρία μικρότερα συνεχόμενα διαστήματα.

Μια άλλη διαφορά που εισάγουν τα παραπάνω ολοκληρώματα προκύπτει αν θυμηθούμε πως ο R* επιλέγει πως θα γίνει ο χωρισμός των κόμβων που υπερχειλίζουν. Ο R* επιλέγει μία από όλες τις δυνατές κατανομές των εγγραφών σε δύο κόμβους από ένα σύνολο δυνατών κατανομών που παράγονται με βάση την ταξινόμηση των σημείων ανάλογα με τις προβολές τους στους άξονες. Στο TPR δέντρο για την ταξινόμηση αυτή χρησιμοποιούνται οι θέσεις των κινούμενων αντικειμένων σε διάφορες χρονικές στιγμές. Για τα load-time bounding rectangles χρησιμοποιείται η θέση τους τη χρονική στιγμή της δημιουργίας του ευρετηρίου t_l, ενώ στην περίπτωση των up-

date-time BRs χρησιμοποιείται η θέση την τρέχουσα χρονική στιγμή. Επιπλέον, επιχειρείται να ομαδοποιηθούν τα διάφορα ορθογώνια σε κόμβους με βάση το ρυθμό με τον οποίο αυξάνεται η έκτασή τους και γι' αυτό το λόγο ταξινομούνται και με βάση τις προβολές των ταχυτήτων τους στους άξονες.

Η διαγραφή εγγραφών γίνεται με τον ίδιο τρόπο που γίνεται στο R^* δέντρο. Αν ένας κόμβος μείνει με λιγότερα από τα ελάχιστα επιτρεπτά στοιχεία, τότε διαγράφεται και τα στοιχεία αυτά επανεισάγονται στη δομή.

2.3.3.4 Κόστος

Θα υπολογίσουμε το κόστος στις δύο διαστάσεις. Έστω ο και q δύο κινούμενα αντικείμενα. Το *μετασχηματισμένο ορθογώνιο o' του o σε σχέση με το q* είναι ένα ορθογώνιο για το οποίο:

(i) η έκταση στον i-οστό άξονα ισούται με:

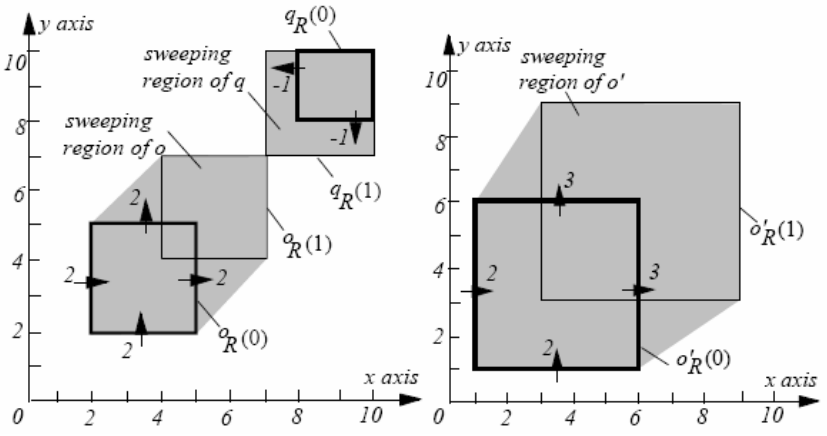
$$\{x_{o,i}^{\prec} - \frac{|x_{q,i}|}{2}, x_{o,i}^{\succ} + \frac{|x_{q,i}|}{2}\}$$

(ii) το διάνυσμα των ταχυτήτων στον i-οστό άξονα είναι:

$$\{x_{o,i}^{\prec} - |x_{q,i}^{\succ}|, x_{o,i}^{\succ} - |x_{q,i}^{\prec}|\}$$

Παρατηρούμε ότι το περιβάλλον ορθογώνιο του o' είναι το περιβάλλον ορθογώνιο του o διευρυμένο κατά αυτό του q.

Ορίζουμε επίσης ως *περιοχή σάρωσης (sweeping region) SR(o,T)* για ένα αντικείμενο o και ένα χρονικό διάστημα T, την περιοχή που σαρώνεται από το o κατά τη διάρκεια της χρονικής αυτής περιόδου, και με A_{SR} συμβολίζουμε το εμβαδό της περιοχής αυτής. Στο σχήμα που ακολουθεί (Σχήμα 40) φαίνονται σκιασμένες οι SR(o,[0,1]), SR(q,[0,1]) και SR(o',[0,1]) με A_{SR}(o,[0,1]) = 21, A_{SR}(q,[0,1]) = 9, A_{SR}(o',[0,1]) = 58:



Σχήμα 40: Περιοχές Σάρωσης του TPR tree

Με βάση τα παραπάνω αποδεικνύεται ότι για ένα ερώτημα q για το οποίο:

(i) το MBR κατανέμεται ομοιόμορφα στο χώρο των δεδομένων και έχει έκταση $\left|x_{q,i}\right|$ στην i-οστή διάσταση

(ii) έχει διάνυσμα ταχύτητας q_V

(iii) γίνεται για το χρονικό διάστημα $[q_T^{\prec}, q_t^{\succ}]$

Τότε ο μέσος αριθμός των κόμβων που πρέπει να επισκεφτούμε είναι :

$$Cost(q) = \sum_{\forall o} A_{SR}(o', q_T) \quad (*)$$

Όπου το άθροισμα υπολογίζεται για όλους τους κόμβους ο.

Σημειώνεται ότι για όλα τα ερωτήματα με τις ίδιες παραμέτρους $\left|x_{q,i}\right|$, q_V, q_T η τιμή $A_{SR}(o', q_T)$ είναι η ίδια για ένα συγκεκριμένο κόμβο ο. Επίσης, η το $A_{SR}(o', q_T)$ είναι διαφορετικό γενικά από την τιμή του $\int_{T_C}^{T_C+H} A(o', t)dt$. Αυτό φαίνεται καθαρά αν θεωρήσουμε δύο αντικείμενα o_1 και o_2 τα οποία έχουν περιβάλλοντα ορθογώνια με την ίδια έκταση σε κάθε διάσταση, και το o_1 κινείται στο διάστημα T ενώ το o_2 παραμένει ακίνητο. Τότε, παρόλο που θα έχουμε $\int_{T_C}^{T_C+T} A(o_1, t)dt = \int_{T_C}^{T_C+T} A(o_2, t)dt$, θα έχουμε σίγουρα ότι $A_{SR}(o_1', T) > A_{SR}(o_2', T)$, γεγονός που φανερώνει και τη μεγαλύτερη πιθανότητα που έχει το κινούμενο o_1 να προσπελαστεί.

2.3.3.5 TPR Tree*

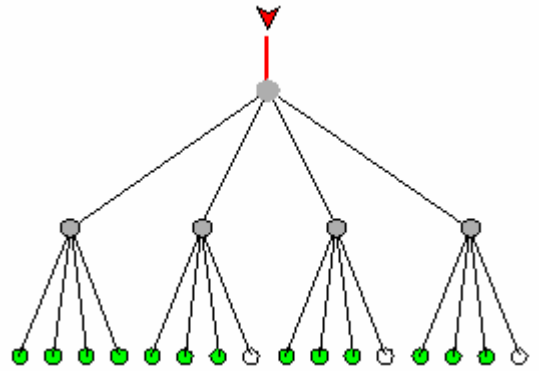
Το TPR* Tree (βλ. [TPS03]) είναι μια διαφοροποίηση του TPR δέντρου, η οποία έχει ως κύριο στόχο την ελαχιστοποίηση του κόστους που περιγράφει η (*). Φυσικά αυτό δεν μπορεί να γίνει για όλους τους τύπους ερωτημάτων. Έτσι το TPR* δέντρο βελτιστοποιεί την απόδοσή του για ερωτήματα που αφορούν σημειακά και στατικά (το διάνυσμα της ταχύτητας είναι 0 σε όλες τις διαστάσεις)αντικείμενα και ενδιαφερόμαστε για μια χρονική διάρκεια μήκους T. Η βελτιστοποίηση αυτή οφείλεται στην υιοθέτηση πιο εξελιγμένων τεχνικών εισαγωγής και διαγραφής στη δομή, οι οποίες όμως δε θα αναφερθούν εδώ. Για το συγκεκριμένο τύπο ερωτημάτων, η δομή αυτή έχει σημαντικά καλύτερη απόδοση από το TPR tree και επειδή τέτοιου τύπου ερωτήματα είναι πολύ συχνά και σημαντικά σε εφαρμογές Location Based Services, το TPR* tree ευρετήριο είναι μια από τις πλέον κατάλληλες δομές για τέτοιες εφαρμογές.

2.3.4 *Quad Trees*

Οι παραλλαγές των R-trees που περιγράφηκαν στις παραπάνω παραγράφους είναι οι πλέον κατάλληλες και ευρέως χρησιμοποιούμενες για τη χωρική ευρετηρίαση μη-σημειακών αντικειμένων, όπως είναι πολύγωνα, γραμμές κτλ. Επίσης, οι παραλλαγές του TPR tree είναι οι πιο αποδοτικές δομές για την ευρετηρίαση κινούμενων αντικειμένων με στόχο την απάντηση ερωτημάτων πρόβλεψης (predictive queries). Στην παρούσα εργασία, ωστόσο, μας ενδιαφέρει ιδιαίτερα η απόδοση της χωρικής δομής στην ευρετηρίαση *σημειακών* αντικειμένων, χωρίς να δίνεται έμφαση στην επεξεργασία και απάντηση ερωτημάτων πρόβλεψης. Ως εκ τούτου, οι προηγούμενες δομές αποδεικνύονται αργές για την προσφορά των υπηρεσιών με βάση τη θέση και το προφίλ σε σημειακούς χρήστες σε μεγάλη κλίμακα, σε σχέση με άλλες δομές, όπως η οικογένεια των quad trees που περιγράφεται παρακάτω.

Τα Quad Trees (βλ. [Sam89]) είναι μια οικογένεια δέντρων με πολλές παραλλαγές. Το κοινό τους στοιχείο είναι ότι αποτελούν ιεραρχικές δομές δεδομένων, που βασίζουν τη λειτουργία τους στην αναδρομική αποσύνθεση (*decomposition*) του χώρου. Οι διάφορες παραλλαγές quadtrees διαφοροποιούνται με βάση το είδος των δεδομένων που αναπαριστούν, τη μέθοδο (αρχή) με την οποία γίνεται η αποσύνθεση του χώρου (από μεγαλύτερες σε μικρότερες περιοχές) και το αν η ευκρίνεια (*resolution*) της δομής καθορίζεται στατικά ή μπορεί να μεταβάλλεται.

Εδώ θα ασχοληθούμε κυρίως με quad trees για την ευρετηρίαση σημειακών αντικειμένων στο διδιάστατο χώρο, καθώς όταν έχουμε να κάνουμε με αντικείμενα που έχουν έκταση (περιοχές) ή μήκος (γραμμές) τυχαίου σχήματος, τα R-Trees εμφανίζουν πολύ καλύτερη απόδοση σε σχέση με τα quad trees. Ένα quad tree φαίνεται στο σχήμα που ακολουθεί (Σχήμα 41):

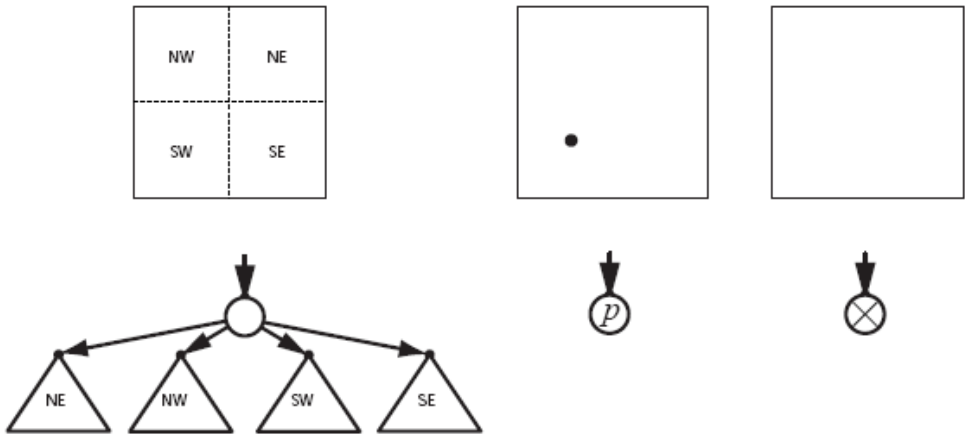


Σχήμα 41: Quad tree

Ένα quad tree είναι, όπως λέει και το όνομά του, ένα τετραδικό δέντρο. Ως εκ τούτου κάθε εσωτερικός κόμβος του έχει 4 κόμβους - παιδιά. Κάθε κόμβος αντιστοιχεί σε μια περιοχή του χώρου και κάθε παιδί του αντιστοιχεί σε ένα υποσύνολο του πατρικού χώρου. Εδώ ασχολούμαστε με περιπτώσεις στις οποίες ο χωρισμός του χώρου γίνεται *κανονικά* δηλαδή κάθε κόμβος είναι τετραγωνικού σχήματος και τα παιδιά του είναι ισομεγέθη. Έτσι κάθε κόμβος αντιστοιχεί σε μια τετραγωνική περιοχή του χώρου και κάθε παιδί του αντιστοιχεί στο ¼ του χώρου του.

Συγκεκριμένα ο πατέρας διαμερίζεται στα τέσσερα τεταρτημόριά(*quadrants*) του, δηλαδή το κάθε παιδί του αντιστοιχεί σε ένα τεταρτημόριό του και για το λόγο αυτό μπορούμε να συμβολίσουμε τα παιδιά του με βάση τη θέση τους ως προς το κέντρο του πατρικού τετραγώνου δηλ. *NW* (*north-west*), *NE* (*north-east*), *SW* (*south-west*), *SE* (*south-east*). Φύλλα του δέντρου είναι οι κόμβοι εκείνοι για τους οποίους δεν απαιτείται περεταίρω διαμέριση. Τα δέντρα του τύπου αυτού ονομάζονται *region quadtrees*.

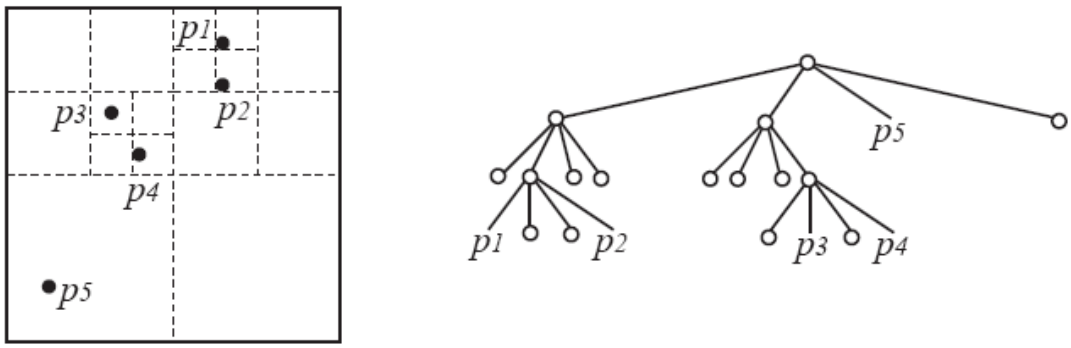
Σε περιπτώσεις που θέλουμε να κάνουμε index μια εικόνα(που είναι μια πολύ συχνή χρήση των quadtrees) κάθε φύλλο απλά κρατά την τιμή του χρώματος για την περιοχή αυτή(ο χώρος αποσυντίθεται σε περιοχές με το ίδιο χρώμα). Αντίθετα στην παρούσα εργασία μας ενδιαφέρει για κάθε περιοχή του χώρου να γνωρίζουμε ποια σημειακά αντικείμενα οριοθετούνται από αυτή. Γι' αυτό το λόγο σε κάθε φύλλο αποθηκεύουμε τα σημειακά αντικείμενα που περιέχει. Αν θεωρήσουμε ότι κάθε φύλλο του δέντρου χωράει μόνο ένα σημειακό αντικείμενο (για απλότητα), τότε μπορεί να έχουμε τρία είδη κόμβων (Σχήμα 42):



Σχήμα 42: Είδη κόμβων Quad tree

- Εσωτερικούς κόμβους (δηλαδή κόμβοι που δεν είναι φύλλα)
- Κόμβους – φύλλα που περιέχουν ένα σημείο
- Κόμβους – φύλλα που είναι άδειοι.

Το είδος αυτό του δέντρου, που συσχετίζει τεταρτημόρια με σημεία ονομάζεται *PR Quadtree* (*Point – Region Quadtree*). Παρακάτω (Σχήμα 43) φαίνεται ο αναδρομικός χωρισμός του αρχικού χώρου μετά την προσθήκη πέντε σημείων, καθώς και το αντίστοιχο δέντρο που προκύπτει.



Σχήμα 43: Παράδειγμα αναδομικού χωρισμού του χώρου

Σε ένα PR quadtree κάθε κόμβος είναι μια εγγραφή που περιέχει τις ακόλουθες καταχωρήσεις:

- Τέσσερις δείκτες προς τα παιδιά του κόμβου, καθένας από τους οποίους δηλαδή αντιστοιχεί σε ένα τεταρτημόριο. Αν ο *P* είναι δείκτης προς ένα κόμβο και το *I* είναι ένα τεταρτημόριο, οι καταχωρήσεις αυτές αναφέρονται ως *SON(P, I)*.
- Το πέμπτο πεδίο, *NODETYPE*, έχει την τιμή *BLACK* αν πρόκειται για φύλλο που περιέχει σημείο, *WHITE* αν πρόκειται για κενό φύλλο και *GRAY* αν πρόκειται για κόμβο που δεν είναι φύλλο.
- Το πεδίο *NAME*, δηλαδή το όνομα του σημειακού αντικειμένου που είναι αποθηκευμένο στον κόμβο
- Τα πεδία *XCOORD* και *YCOORD*, δηλαδή τις συντεταγμένες του σημειακού αντικειμένου. Θεωρούμε ότι κάθε αντικείμενο είναι μοναδικό.

Η δημιουργία της εγγραφής γίνεται με την ακόλουθη συνάρτηση:

```
pointer node procedure CREATE_PNODE(C)
/* Δημιουργία κόμβου με χρώμα C και επιστροφή ενός δείκτη σε αυτόν */

begin
  value color C;
  pointer node P;
  quadrant I;
  P := create(node);
  NODETYPE(P) := C;
  for I in {'NW','NE','SW','SE'} do
    SON(P, I) := NIL;
  return P;
end;
```

Επίσης θεωρούμε ότι για κάθε τεταρτημόριο το κάτω και το αριστερό σύνορο είναι κλειστά, ενώ το άνω και το δεξί είναι ανοικτά. Έτσι για να βρούμε το τεταρτημόριο στο οποίο βρίσκεται ένα σημείο, χρησιμοποιούμε την ακόλουθη συνάρτηση:

```
quadrant procedure PR_COMPARE(P, X, Y)
/* Επιστρέφει το τεταρτημόριο του δέντρου με ρίζα στο (x, y) στο οποίο
ανήκει ο κόμβος P */

begin
   value pointer node P;
   value real X, Y;
   return (if XCOORD(P) < X then
          if YCOORD(P) <Y then 'SW'
          else 'NW'
         else if YCOORD(P) < Y then 'SE'
          else 'NE');
end;
```

Για την *εισαγωγή* ενός νέου σημειακού αντικειμένου στο δέντρο αρχικά δημιουργείται η κατάλληλη εγγραφή (έστω A) για το συγκεκριμένο σημείο και στη συνέχεια κάνουμε αναζήτηση στο δέντρο για να βρεθεί η θέση στην οποία θα μπει η νέα εγγραφή. Δηλαδή αναζητούμε το τεταρτημόριο(quadrant) στο οποίο πρέπει να τοποθετήσουμε τη νέα εγγραφή χρησιμοποιώντας την παραπάνω συνάρτηση PR_COMPARE και τις συντεταγμένες x και y της νέας εγγραφής. Αν το τεταρτημόριο αυτό είναι κατειλημμένο ήδη από μια άλλη εγγραφή με διαφορετικές συντεταγμένες x και y (έστω B), τότε πρέπει το τεταρτημόριο αυτό να διαιρεθεί αναδρομικά σε μικρότερα έως ότου οι δύο εγγραφές να πέφτουν σε διαφορετικό τεταρτημόριο(δηλ. φύλλο του δέντρου). Είναι εμφανές ότι έτσι μπορεί να προκύψει ένας μεγάλος αριθμός διαδοχικών διαιρέσεων, ειδικά αν τα δύο σημεία περιέχονται σε ένα πολύ μικρό κομμάτι του τεταρτημορίου. Για να προκύψει η κατάσταση αυτή πρέπει η ευκλείδια απόσταση μεταξύ των δύο αυτών σημείων να είναι πολύ μικρή.

Από τα παραπάνω γίνεται αντιληπτό ότι η σειρά με την οποία εισάγονται τα σημεία ενός συνόλου δεδομένων(data set) στο δέντρο δεν επηρεάζει την τελική μορφή του δέντρου. Επηρεάζεται φυσικά η μορφή του δέντρου κατά τα ενδιάμεσα βήματα. Η διαδικασία της εισαγωγής φαίνεται καλύτερα στον ψευδοκώδικα που ακολουθεί:

```
procedure PR_INSERT(P, R, X, Y, LX, LY)

/* Εισαγωγή του κόμβου P στο δέντρο με ρίζα R, που είναι το κέντρο ε-
νός LX x LY παραλληλογράμμου στη θέση (X, Y). Αν το δέντρο είναι κενό,
τότε στο R ανατίθεται η ρίζα του νέου δέντρου. */

begin
```

```
value pointer node P;
reference pointer node R;
value real X, Y, LX, LY;
pointer node T, U;
quadrant Q, UQ;

real array XF['NW','NE','SW','SE'] = {-0.25,0.25,-0.25,0.25}
real array YF['NW','NE','SW','SE'] = {0.25,0.25,-0.25,-0.25}

if null(R) then    /* Κενό δέντρο */
  begin
    R := P;
    return;
  end;
else if not(GRAY(R)) then  /* Το δέντρο έχει 1 μόνο κόμβο */
  if XCOORD(P) = XCOORD(R) and YCOORD(P) = YCOORD(R)
    then return;
  else
    begin
      U := R;
      R := CREATE_PNODE('GRAY');
      Q := PR_COMPARE(U, X, Y);
      SON(R, Q) := U;
    end;
T := R;
Q := PR_COMPARE(P, X, Y);

while not(null(SON(T, Q))) and GRAY(SON(T, Q)) do
/* Εντοπισμός του quadrant που ανήκει το (X, Y) */
    begin
      T  := SON(T, Q);
      X  := X + XF[Q] * LX;
      LX := LX / 2.0;
      Y  := Y + YF[Q] *LY;
      LY := LY / 2.0;
      Q  := PR_COMPARE(P, X, Y);
    end;

   if null(SON(T, Q)) then
        SON(T, Q) := P;
else if XCOORD(P) = XCOORD(SON(T, Q)) and
     YCOORD(P) = YCOORD(SON(T, Q)) then
    return;  /* ο P υπάρχει ήδη στο δέντρο */

else  /* Το τεταρτημόριο στο οποίο πρέπει να μπει ο P είναι
     ήδη κατειλημμένο από το U */
  begin
    U := SON(T, Q);
    do/* Αναδρομική διαμέριση του Q έως ότου το P
                      και το U να ανήκουν σε διαφορετικό  τεταρ-
                      τημόριο  */
      begin
        SON(T, Q) := CREATE_PNODE('GRAY');
        T  := SON(T, Q);
        X  := X + XF[Q] * LX;
        LX := LX / 2.0;
                          Y  := Y + YF[Q] *LY;
          LY := LY / 2.0;
          Q  := PR_COMPARE(P, X, Y);
          UQ := PR_COMPARE(U, X, Y);
        end
```

```
          until Q;
          SON(T, Q)  := P;  /* Εισαγωγή του P */
        SON(T, UQ) := U;  /* Εισαγωγή του U */
      end;
end;
```

Η *διαγραφή* ενός αντικειμένου από το PR Quadtree είναι σχετικά απλή, καθώς όλα τα σημεία βρίσκονται αποθηκευμένα στα φύλλα του δέντρου. Απαιτείται αρχικά η εύρεση του φύλλου στο οποίο βρίσκεται η ζητούμενη εγγραφή και στη συνέχεια η διαγραφή του από το φύλλο. Αν μετά από κάποια διαγραφή σε ένα φύλλο, τα υπόλοιπα αδερφικά φύλλα έχουν συνολικά μόνο ένα σημείο αποθηκευμένο (δηλ. από τα τέσσερα φύλλα μόνο το ένα είναι κατειλημμένο) τότε γίνεται *σύμπτυξη* (*collapsing*) του δέντρου. Δηλαδή, αφού το υποδέντρο που έχει ρίζα τον πατέρα του φύλλου στο οποίο έγινε διαγραφή περιέχει μόνο ένα σημείο, ο πατέρας αυτός μετατρέπεται ο ίδιος σε φύλλο και τα παιδιά του διαγράφονται. Η σύμπτυξη είναι η ακριβώς αντίθετη διαδικασία της υποδιαίρεσης ενός κόμβου. Η διαδικασία αυτή διαδίδεται προς τα πάνω στο δέντρο, μέχρι να βρεθεί ένας πρόγονος που έχει περισσότερα από 1 αντικείμενα αποθηκευμένα στο υποδέντρο που ξεκινάει από αυτόν.

```
procedure PR_DELETE(P, R, X, Y, LX, LY)

/* Διαγραφή του P από το δέντρο που έχει ρίζα τον κόμβο R και είναι το
κέντρο ενός LX x LY παραλληλογράμμου στη θέση (X, Y). Αν το P είναι ο
μοναδικός κόμβος του δέντρου, τότε η ρίζα τίθεται στο NIL. */

begin
  value pointer node P;
  reference pointer node R;
  value real X, Y, LX, LY;
  pointer node F, FT, T, TEMP;
  quadrant Q, QF;
  integer S;

     real array XF['NW','NE','SW','SE'] = {-0.25,0.25,-0.25,0.25}
  real array YF['NW','NE','SW','SE'] = {0.25,0.25,-0.25,-0.25}

  if null(R) then    /* Κενό δέντρο */
    return;
  else if not(GRAY(R)) then  /* Το δέντρο έχει 1 μόνο κόμβο */
    if XCOORD(P) = XCOORD(R) and YCOORD(P) = YCOORD(R)
      then return;
    else
      begin
        free(R);
        R := NIL;
        return;
      end;
  T := R;
  F := NIL;

  do
    begin  /* Εντοπισμός του quadrant που ανήκει το P */
```

```
      Q := PR_COMPARE(P, X, Y);
      if GRAY(SON(T, Q) and
                  not(null(SON(T, CQUAD(Q))) and
                  not(null(SON(T, OPQUAD(Q))) and
                  not(null(SON(T, CCQUAD(Q))) then
                 begin
                 /* Καθώς κατεβαίνουμε το δέντρο τα F και QF
                    κρατάνε τον πλησιέστερο πρόγονο του (X, Y)
                    που έχει περισσότερα από 1 κατοικημένα
                         παιδιά */
                    F  := T;
                    QF := Q;
                 end;
                FT := T;
                T  := SON(T, Q);
                X  := X + XF[Q] * LX;
                LX := LX / 2.0;
                Y  := Y + YF[Q] * LY;
                LY := LY / 2.0;
                end;

  until null(T) or not(GRAY(T));

  if null(T) or
        not(XCOORD(P) = XCOORD(T) and YCOORD(P) = YCOORD(T) then
      return;  /* Το P δεν υπάρχει στο δέντρο */
     else
      begin /* Εξετάζουμε αν χρειάζεται σύμπτυξη */
        free(T);
        SON(FT, Q) := NIL;
        S := 0;
        for Q in {'NW','NE','SW','SE'} do
          begin
            if GRAY(SON(FT, Q)) then
            /* Δεν μπορεί να γίνει σύμπτυξη */
              return;
            else if not(null(SON(FT, Q))) then
              S := S + 1;
          end;
        if (S > 1) then  /* Δε μπορεί να γίνει σύμπτυξη */
          return;
        else
          begin
            if null(F) then
              T := R;
            else
              T := SON(F, QF);
            Q := 'NW';

            while GRAY(T) do
              begin
              /* Σύμπτυξη ενός επιπέδου
                                                        τη φορά */
              while null(SON(T, Q)) do
                   Q := CQUAD(Q);
                TEMP := SON(T, Q);
                SON(T, Q) := NIL;
                free(T);
                T := TEMP;
              end;
            if null(F) then
```

```
                                        R := T
          else
             SON(F, QF) := T;
        end;
    end;
end;
```

Το κόστος των λειτουργιών εισαγωγής και διαγραφής στο δέντρο εξαρτάται από τα σημεία που έχουν ήδη εισαχθεί στο δέντρο και συγκεκριμένα είναι ανάλογο προς το μέγιστο ύψος του δέντρου. Για παράδειγμα, δεδομένης μιας τετραγωνικής περιοχής πλευράς s και θεωρώντας ότι η ελάχιστη απόσταση μεταξύ δύο σημειακών αντικειμένων είναι d, τότε το μέγιστο ύψος του δέντρου μπορεί να είναι μέχρι και $\lceil \log_2((\frac{s}{d})\cdot\sqrt{2}) \rceil$ (βλ. [HS02]).

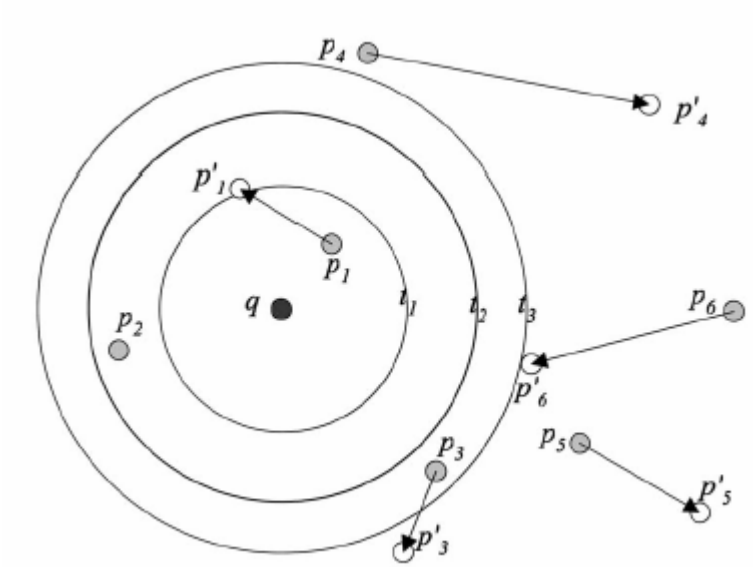
2.3.5 Continuous Nearest Neighbor Monitoring

Όπως, αναφέρθηκε και παραπάνω, ένας από τους πιο συχνούς και σημαντικούς τύπους ερωτημάτων σε ένα σύστημα Location Based Services και μια άμεση εφαρμογή των ευρετηρίων που περιγράφηκαν προηγουμένως είναι η εύρεση των k κοντινότερων γειτόνων συγκεκριμένου τύπου ενός χρήστη, πρόβλημα που αναφέρεται ως k Nearest Neighbor Problem(k-NN). Επέκταση του προβλήματος αυτού είναι συνεχής παρακολούθηση των k κοντινότερων γειτόνων ενός χρήστη καθώς αυτός διαγράφει μια τροχιά κινούμενος στο χώρο. Το πρόβλημα αυτό αναφέρεται ως k Continuous Nearest Neighbor Monitoring Problem(k-CNN).

Έστω ένα σύνολο από κινούμενα αντικείμενα και ένας κεντρικός εξυπηρετητής που παρακολουθεί την κίνησή τους στο χρόνο με σκοπό το να δίνει απαντήσεις σε ερωτήματα k continuous nearest neighbors. Ένα ρεαλιστικό σενάριο είναι το ακόλουθο: μια εταιρία ταξί που θέλει να παρακολουθεί (από διάφορα σημεία-πελάτες) το στόλο των οχημάτων της καθώς κινείται σε μια γεωγραφική περιοχή και θέλει ανά πάσα στιγμή να γνωρίζει τους k κοντινότερους γείτονες από καθένα από τα οχήματά της. Τόσο τα κινούμενα αντικείμενα όσο και οι πελάτες μπορεί να εμφανίζονται και να εξαφανίζονται (αν για παράδειγμα κάποια ταξί βγουν εκτός υπηρεσίας). Οι πελάτες και τα κινούμενα αντικείμενα δεν επικοινωνούν άμεσα, αλλά με τη μεσολάβηση ενός κεντρικού εξυπηρετητή, στον οποίο πηγαίνουν τόσο τα ερωτήματα των πελατών όσο και οι πληροφορίες για τη γεωγραφική θέση των κινούμενων αντικειμένων. Βλέποντας το σύστημα από την τηλεπικοινωνιακή του πλευρά θεωρούμε ότι ο εξυπηρετητής μπορεί να κάνει broadcasting (εκτός από unicasting) προς τα κινούμενα αντικείμενα, καθένα από τα οποία μπορεί να του απαντάει με unicast καθώς και ότι το κόστος του broadcasting είναι πολύ μικρότερο από αυτό πολλών unicast μηνυμάτων με το ίδιο περιεχόμενο. Επίσης, κάθε κινούμενο αντικείμενο, εκτός από το να βλέπει τη θέση του με τη βοήθεια ενός GPS δέκτη, μπορεί να κάνει και στοιχειώδεις υπολογισμούς, όπως να αποθηκεύει τα τρέχοντα ερωτήματα και να υπολογίζει την απόστασή του από αυτά. Ο χρόνος για την κίνηση των οχημάτων είναι συνεχής, ωστόσο οι ενη-

μερώσεις προς τον κεντρικό εξυπηρετητή γίνονται σε διακριτά χρονικά διαστήματα, δηλαδή υπάρχει ένα ελάχιστο χρονικό διάστημα που πρέπει να μεσολαβήσει μεταξύ δύο ενημερώσεων. Διαφορετικά, θα μπορούσαμε να πούμε ότι δύο διαδοχικές ενημερώσεις διαφέρουν τουλάχιστο κατά ένα διάστημα dt και το round trip time ενός μηνύματος είναι αμελητέος συγκρινόμενος με το dt.

Σκοπός της μεθόδου που περιγράφεται στη συνέχεια (βλ. [MPBT05]) είναι να μειωθεί το τηλεπικοινωνιακό κόστος του συστήματος, δηλαδή ο αριθμός των μηνυμάτων που ανταλλάσσονται μεταξύ του εξυπηρετητή και των κινούμενων αντικειμένων. Διαισθητικά είναι προφανές ότι παρόλο που μπορεί να υπάρχουν χιλιάδες κινητά αντικείμενα που κινούνται προς όλες τις δυνατές διαφορετικές κατευθύνσεις και με πολύ διαφορετικές ταχύτητες, οι μόνες ενημερώσεις που ενδιαφέρουν το server είναι όταν η αλλαγή θέσης ενός αντικειμένου μπορεί να επηρεάσει το αποτέλεσμα ενός ερωτήματος (δηλαδή μετά την αλλαγή αυτή να μεταβληθεί το σύνολο των κοντινότερων γειτόνων κάποιου αντικειμένου). Για τα υπόλοιπα αντικείμενα ο εξυπηρετητής δε χρειάζεται να γνωρίζει τίποτα σχετικά με τη θέση τους. Για το λόγο αυτό ορίζουμε κάποιες τιμές *κατωφλίου* και κάθε αντικείμενο είναι υποχρεωμένο να αναφέρει τη θέση του μόνο αν υπερβεί κάποια από αυτές τις τιμές. Στο παρακάτω σχήμα (Σχήμα 44):

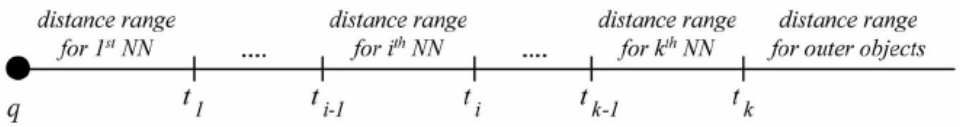


Σχήμα 44: Παράδειγμα τιμών κατωφλίου

έστω ότι θέλουμε να παρακολουθούμε τους 3-NN για κάποιο στατικό ερώτημα q. Ονομάζουμε *εσωτερικά* τα αντικείμενα εκείνα τα οποία συμπεριλαμβάνονται στο αποτέλεσμα του ερωτήματος(δηλ. εδώ είναι 3-NN) ενώ *εξωτερικά* τα υπόλοιπα αντικείμενα. Στη συγκεκριμένη περίπτωση τα εσωτερικά αντικείμενα είναι τα p_1, p_2, p_3, ενώ τα εξωτερικά είναι τα p_4, p_5, p_6. Σαν τιμές κατωφλίου ορίζουμε το ήμισυ της διαφοράς των αποστάσεων δύο διαδοχικών NN από το ερώτημα q. Στο παραπάνω σχήμα οι τιμές κατωφλίου είναι οι t_1, t_2, t_3. Οι τιμές αυτές ορίζουν μια περιοχή για κάθε αντικείμενο τέτοια που αν το αντικείμενο παραμείνει εντός των ορίων της πε-

ριοχής αυτής, τότε το αποτέλεσμα του ερωτήματος σίγουρα θα παραμείνει αναλλοίωτο. Εδώ για παράδειγμα αν το p_1 παραμείνει εντός του t_1, τότε θα είναι σίγουρα ο 1-NN. Παρόμοια αν το p_2 παραμείνει εντός των t_1 και t_2 τότε σίγουρα θα είναι εντός των 2-NN κτλ.

Κάθε αντικείμενο γνωρίζει ποια είναι η περιοχή που ορίζουν γι' αυτό οι τιμές κατωφλίου και επομένως όταν παραβιάσει κάποια από αυτές τις τιμές, μπορεί να ενημερώσει τον εξυπηρετητή για το γεγονός αυτό. Αν η απόσταση του p_1 από το q γίνει μεγαλύτερη από t_1(αν το p_1 κινηθεί προς την p_1') τότε η διάταξη μεταξύ των δύο πρώτων NN μπορεί να μεταβληθεί, δηλ. ο p_2 να γίνει ο NN του q αν το p_2 κινηθεί προς το q, διαφορετικά το αποτέλεσμα παραμένει το ίδιο. Γι' αυτό το λόγο ο server πρέπει να μάθει την τρέχουσα θέση του p_2. Σε κάθε περίπτωση η τιμή του t_1 ενημερώνεται για να συμφωνεί με τις νέες θέσεις των p_1, p_2. Είναι εμφανές ότι οι θέσεις των υπολοίπων αντικειμένων δεν έχουν καμία σημασία καθώς δε μπορούν με κανένα τρόπο να επηρεάσουν το αποτέλεσμα του ερωτήματος χωρίς πρώτα να έχουν αναφέρει την παραβίαση κάποιου κατωφλίου. Συνεχίζοντας τη μελέτη του παραπάνω παραδείγματος, αν το p_3 κινηθεί σε ακτίνα μεγαλύτερη του t_3, τότε θα ενημερώσει τον εξυπηρετητή για την παραβίαση του κατωφλίου και τότε καθένα από τα εξωτερικά αντικείμενα είναι ένας πιθανός νέος τρίτος NN. Τώρα ο server πρέπει να μάθει τις θέσεις των εξωτερικών αντικειμένων και για να μειώσει τον αριθμό των πιθανών υποψηφίων για τη θέση του 3ου NN στέλνει με broadcasting ένα μήνυμα προς όλα τα εξωτερικά αντικείμενα που βρίσκονται σε απόσταση από το q μικρότερη από ότι το p_3' (η νέα θέση του p_3). Με βάση τις απαντήσεις των αντικειμένων μπορεί τώρα να βρεθεί ο νέος NN και να καθοριστεί η νέα τιμή του εξωτερικού κατωφλίου η οποία πρέπει στη συνέχεια να γίνει broadcast σε όλα τα αντικείμενα.



Σχήμα 45: Αποστάσεις εγγύτερων γειτόνων

Στη γενική περίπτωση ο εξυπηρετητής διατηρεί σε μια λίστα τα id's των αντικειμένων που έχουν πιο πρόσφατα αναφερθεί ως k-NN καθώς και τις k τιμές κατωφλίου που ορίζουν για κάθε αντικείμενο την περιοχή που μπορεί να κινείται χωρίς να προκαλεί κάποια αλλαγή στο αποτέλεσμα του ερωτήματος. Έστω $p_i \ \ (i = 1,...,k)$ ο i-οστός NN και d_i η απόστασή του από το σημείο q, όπου γίνεται το ερώτημα. Όπως φαίνεται και στο παραπάνω σχήμα, η επιτρεπτή περιοχή $[t_{i-1}, t_i)$ του p_i ορίζεται από τις τιμές κατωφλίου t_{i-1} και t_i. Στην ιδανική περίπτωση, η τιμή κατωφλίου t_i πρέπει να είναι τέτοια ώστε να καθυστερείται όσο το δυνατό περισσότερο η παραβίασή του από κάποιο αντικείμενο. Επειδή όμως τα διάφορα αντικείμενα κινούνται με τρόπο τυχαίο στο χώρο, δεν μπορούμε παρά να θέσουμε $t_i = \frac{d_i + d_{i+1}}{2}$. Με βάση αυτά, κάθε κινούμενο

αντικείμενο πρέπει να αποθηκεύει τοπικά τη θέση του ερωτήματος και τις σχετικές με το αντικείμενο αυτό τιμές κατωφλίου: για τα εξωτερικά αντικείμενα την τιμή t_k, ενώ για ο i-οστός NN πρέπει να γνωρίζει τις τιμές t_{i-1} και t_i, ώστε όταν κινούμενο το αντικείμενο παραβιάσει κάποια από τις τιμές αυτές να μπορεί να ειδοποιήσει τον κεντρικό server για την παραβίαση αυτή.

Όταν ένα νέο ερώτημα q φτάσει στο server, χρειάζεται πρώτα απ' όλα να βρεθεί το αρχικό σύνολο k-NN για το ερώτημα αυτό. Για την επίτευξη αυτού, ο εξυπηρετητής κάνει broadcast ένα μήνυμα με το οποίο την τρέχουσα θέση όλων των αντικειμένων που βρίσκονται μέσα σε μια ακτίνα r_1 από το ερώτημα q. Η ακριβής τιμή για το r_1 μπορεί να αποφασιστεί με βάση τη σχέση κόστους μεταξύ των broadcast kai των uplink (από τα αντικείμενα προς το server) μηνυμάτων. Όσο μεγαλύτερη είναι αυτή η ακτίνα, τόσο μεγαλύτερη είναι και η πιθανότητα εντός της ακτίνας αυτής να βρίσκονται περισσότερα από k αντικείμενα που χρειαζόμαστε. Αλλά τότε θα έχουμε και περισσότερα uplink μηνύματα, πράγμα που σημαίνει μεγαλύτερο τηλεπικοινωνιακό κόστος. Από την άλλη, μια μικρή τιμή για το r_1 μπορεί να έχει ως αποτέλεσμα να μη βρεθούν k αντικείμενα που χρειαζόμαστε και ως εκ τούτου να απαιτηθεί η επανάληψη της διαδικασίας με αυξημένη ακτίνα r_1, οπότε και πάλι έχουμε αυξημένο τηλεπικοινωνιακό κόστος. Αν ο εξυπηρετητής γνωρίζει μόνο τον πληθάριθμο $|N|$ του συνόλου των κινούμενων αντικειμένων τότε ορίζει:

$$r_1 = \sqrt{\frac{k}{\pi} \cdot |N|}$$

με τη λογική ότι αν έχουμε ομοιόμορφη κατανομή των αντικειμένων, τότε στον κύκλο (q, r_1) αναμένεται να περικλείονται k αντικείμενα. Έστω k_1 ο αριθμός των αντικειμένων που απαντούν στο server. Αν $k_1 > k$ τότε ο server έχει αρκετές πληροφορίες στη διάθεσή του για να υπολογίσει το σύνολο των k-NN. Διαφορετικά, πρέπει να εκπεμφθεί και πάλι ένα παρόμοιο μήνυμα που να ζητά τις θέσεις των εξωτερικών αντικειμένων που βρίσκονται σε απόσταση r_2 από το q, όπου r_2 είναι μια νέα ακτίνα που εκτιμούμε ότι περιέχει τα k ζητούμενα αντικείμενα. Οι απαντήσεις που λαμβάνει ο server σε αυτό του το μήνυμα είναι αφ' ενός από αντικείμενα με απόσταση $(r_1, r_2]$ από το q, αφ' ετέρου δε αντικείμενα των οποίων η θέση έχει αλλάξει από την προηγούμενη αναφορά τους. Θεωρώντας ότι ο χρόνος για το round-trip ενός μηνύματος είναι αμελητέος, μόνο αντικείμενα στο διάστημα $(r_{i-1}, r_i]$ θα αποκρίνονται στο i-οστό ερώτημα, δηλαδή οι απαντήσεις σε προηγούμενα ερωτήματα των αντικειμένων που βρίσκονται στο διάστημα $(0, r_{i-1}]$ θα ισχύουν ακόμη. Η διαδικασία αυτή συνεχίζεται μέχρι ο εξυπηρετητής να συγκεντρώσει απαντήσεις από τουλάχιστον k αντικείμενα. Κατά την αναζήτηση των αντικειμένων αυτών η ακτίνα r_i του i-οστού μηνύματος υπολογίζεται ως εξής:

$$r_i = r_{i-1} \cdot \sqrt{\frac{k}{k_{i-1}}}$$

Μετά από τον υπολογισμό του αρχικού συνόλου των nearest neighbors, ο server πρέπει συνεχώς να παρακολουθεί το σύνολο αυτό και να χειρίζεται αλλαγές σε αυτό. Για κάποια ενημέρωση, έστω I το σύνολο των αντικειμένων που εισέρχονται σε ακτίνα μικρότερη από t_k (*incoming*) και O το σύνολο των αντικειμένων που ξεφεύγουν σε μεγαλύτερη απόσταση από αυτή(*outgoing*). Διακρίνουμε τις ακόλουθες περιπτώσεις:

(i) $|I| \geq |O|$. Αυτό σημαίνει ότι τα αντικείμενα που εισήλθαν στο εσωτερικό του t_k είναι περισσότερα από αυτά που έφυγαν προς το εξωτερικό του και επομένως οι k NNs θα βρίσκονται σε απόσταση μικρότερη από t_k (ή ίση). Ως εκ τούτου το νέο t_k' θα έχει μικρότερη τιμή. Οι νέες τιμές κατωφλίων πρέπει ως εκ τούτου να μεταδοθούν μόνο στα άμεσα επηρεαζόμενα αντικείμενα και όχι στα εξωτερικά, πράγμα που μπορεί να γίνει με unicast και να αποφευχθεί το broadcasting. Τα εξωτερικά αντικείμενα μπορούν να συνεχίζουν να νομίζουν ότι η τιμή του t_k είναι ίδια όπως παλιά, αφού αν αναφέρουν παραβίασή της θα ενημερωθούν από το server για την πλάνη τους(η νέα τιμή σίγουρα δε θα έχει παραβιαστεί, αφού είναι μικρότερη).

(ii) $|I| < |O|$. Σε αυτή την περίπτωση τα αντικείμενα που έχουν βγει προς την εξωτερική περιοχή είναι περισσότερα από αυτά που έχουν μπει στην εσωτερική και έτσι σε ακτίνα t_k περιλαμβάνονται τώρα λιγότερα από k αντικείμενα. Άρα κάποια από τα αντικείμενα της εξωτερικής περιοχής πρέπει να ενημερώσουν το server με τις νέες τους θέσεις, αφού μπορεί τώρα να ανήκουν στο σύνολο των NNs. Αυτό μπορεί να γίνει με μια διαδικασία παρόμοια με αυτή που ανακαλύπτει τους k-NN στην αρχή. Στη συνέχεια μπορούν να υπολογιστούν όλες οι νέες τιμές κατωφλίου. Η τιμή t_k θα πρέπει με broadcasting να αποσταλεί προς όλα τα εξωτερικά αντικείμενα.

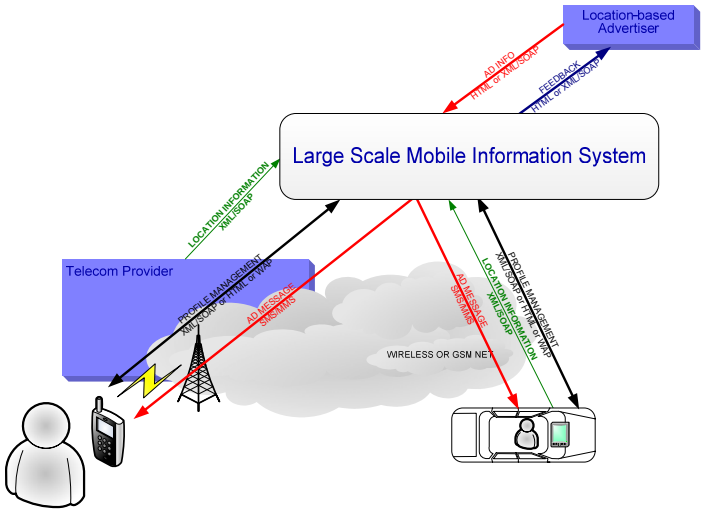
2.3.5.1 Road Networks

Οι τεχνικές της παραπάνω μεθόδου έχουν το πλεονέκτημα ότι μπορούν να χρησιμοποιηθούν αυτούσιες και για τον υπολογισμό κοντινότερων γειτόνων όχι με βάση τον ευκλείδειο χώρο και την ευκλείδεια απόσταση, αλλά και αποστάσεις σε γραφήματα και δίκτυα. Για παράδειγμα όταν έχουμε κίνηση ενός αυτοκινήτου σε οδικό δίκτυο συνήθως μας ενδιαφέρει περισσότερο να υπολογίζουμε τις αποστάσεις όχι ως ευκλείδειες (αφού το αυτοκίνητο μπορεί να κινείται μόνο εντός των δρόμων και όχι προς όλες τις κατευθύνσεις) αλλά με βάση την απόσταση που πρέπει να διανύσει το όχημα κινούμενο *πάνω* στο οδικό δίκτυο για να φτάσει στον προορισμό του (για περισσότερες πληροφορίες (βλ. [BPW98], [CC05]).

3

Ανάλυση Απαιτήσεων – Σχεδιασμός Συστήματος

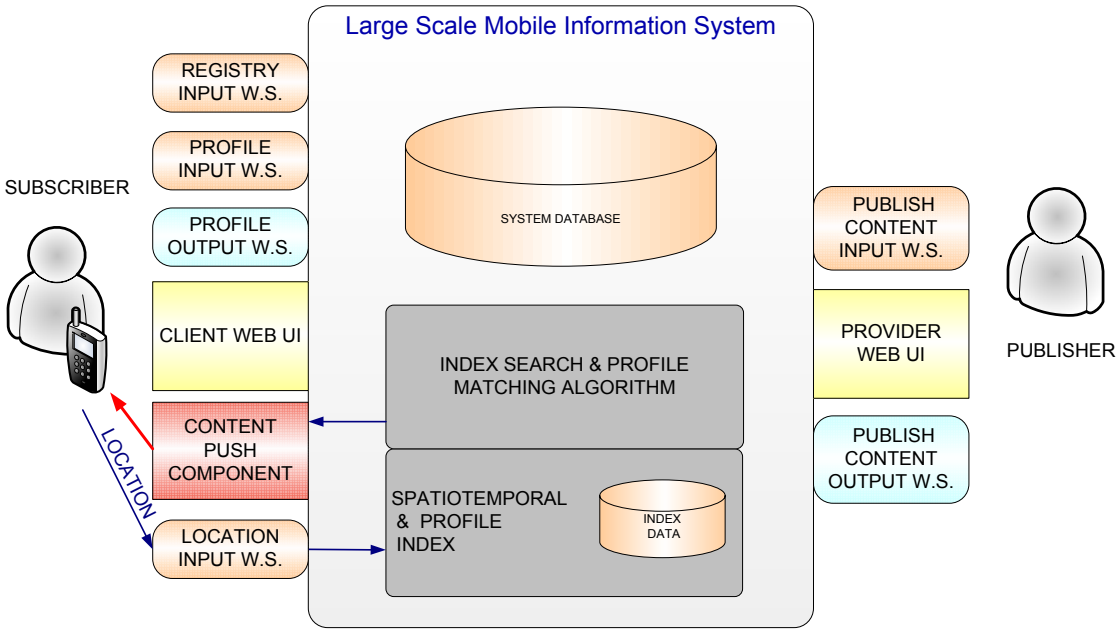
Βασικός στόχος της εργασίας είναι να αναπτυχθεί μια πρότυπη πλατφόρμα για την υποστήριξη Profile & Location Based Services σε πραγματικές εφαρμογές μεγάλης κλίμακας. Αναλυτικότερα, απαιτείται ο σχεδιασμός και η υλοποίηση των συστατικών μερών ενός συστήματος το οποίο θα είναι σε θέση να παρέχει πληροφορίες σε κινητούς χρήστες με βάση τη γεωγραφική τους θέση σε πραγματικό χρόνο και με βάση κάποια κριτήρια που καθορίζονται από τους χρήστες. Το περιεχόμενο δίνεται στο σύστημα από παρόχους πληροφοριών και διαχέεται στους κατάλληλους χρήστες από το σύστημα. Οι χρήστες είναι κινούμενοι στο χώρο, ενώ καθένας από αυτούς δηλώνει το προφίλ των ενδιαφερόντων του καθορίζοντας έτσι το είδος και τη θεματική υφή των πληροφοριών που θέλει να λαμβάνει. Η γενική ιδέα του συστήματος φαίνεται στο Σχήμα 46. Στη συνέχεια αναλύονται οι απαιτήσεις για κάθε συνιστώσα του συστήματος.



Σχήμα 46: Γενική άποψη χρήσης Συστήματος LSMIS

3.1 Αρχιτεκτονική Συστήματος

Η γενική αρχιτεκτονική του συστήματος φαίνεται στο Σχήμα 47:



Σχήμα 47: Αρχιτεκτονική συστήματος LSMIS

Η αρχιτεκτονική του συστήματος είναι *κεντρικοποιημένη* (centralized). Ο κεντρικός εξυπηρετητής αναλαμβάνει την ευρετηρίαση (indexing) των θέσεων και των προφίλ των εγγεγραμμένων χρηστών. Παράλληλα αναλαμβάνει την μετάδοση των πληροφοριών των παρόχων περιεχομένου στους ενδιαφερόμενους χρήστες που βρίσκονται εντός μία περιοχής ενδιαφέροντος ή στον εγγύτερο χρήστη από ένα σημείο ενδιαφέροντος. Η επικοινωνία των δραστών του συστήματος (εγγεγραμμένοι χρήστες, πάροχοι περιεχομένου), οι ρόλοι των οποίων αποσαφηνίζονται σε επόμενη παράγραφο, γίνεται μέσω *web services*. Η επιλογή αυτή ελαχιστοποιεί το υπολογιστικό φορτίο στις συσκευές που χρησιμοποιούν οι εγγεγραμμένοι χρήστες και επιτρέπει τη χρήση φορητών συσκευών (mobile devices) με μικρούς διαθέσιμους πόρους (low resources).

3.2 Δράστες (Actors) του Συστήματος

Στο σύστημα διακρίνονται δύο κατηγορίες δραστών: οι εγγεγραμμένοι χρήστες και οι πάροχοι περιεχομένου. Η διάκριση αυτή αφορά στις λειτουργικές δυνατότητες που είναι διαθέσιμες στην κάθε κατηγορία και δεν είναι αποκλειστικά διαζευκτική αφού ένας χρήστης του συστήματος μπορεί ταυτόχρονα έχει και τους δύο ρόλους. Παρακάτω αναλύονται οι δύο ρόλοι και μοντελοποιούνται οι περιπτώσεις χρήσης του συστήματος από τις αντίστοιχες οντότητες.

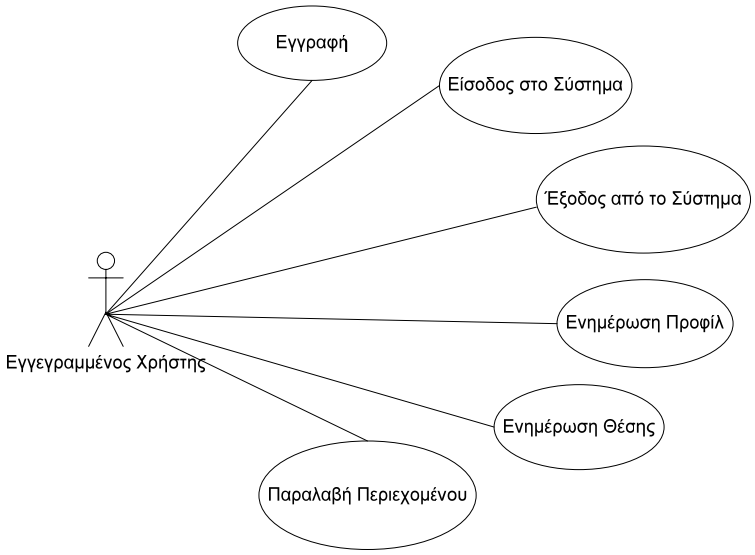
3.2.1 Εγγεγραμμένοι χρήστες

Οι χρήστες που ενδιαφέρονται να λαμβάνουν περιεχόμενο από τους παρόχους μπορούν να εγγραφούν (*register*) στο σύστημα. Στη συνέχεια μπορούν να εισέρχονται (*move in*) και να εξέρχονται (*move out*) από αυτό.

Ο χρήστης έχει τη δυνατότητα να καθορίσει το προφίλ του και να το μεταβάλλει (*set profile*) οποιαδήποτε στιγμή μετά την εγγραφή του στο σύστημα. Τρέχον προφίλ για κάθε χρήστη θεωρείται αυτό το οποίο υπέβαλε τελευταίο χρονικά. Η πληροφορία του προεπιλεγμένου προφίλ (*default profile*) διατηρείται στη ΒΔ και μπορεί να αλλάξει με τροποποίηση σε αυτήν. Το προεπιλεγμένο προφίλ αποδίδεται στο χρήστη κάθε φορά που εισέρχεται στο σύστημα και το διατηρεί μέχρι την πρώτη ανανέωση είτε του προεπιλεγμένου είτε του τρέχοντος προφίλ.

Οι εγγεγραμμένοι χρήστες του συστήματος είναι κινούμενοι χρήστες. Η κίνηση τους στο χώρο είναι γίνεται προς οποιαδήποτε κατεύθυνση και με οποιαδήποτε ταχύτητα – δεν χρησιμοποιούνται πληροφορίες σχετικά με το υπάρχον οδικό δίκτυο, ύπαρξη εμποδίων ή άλλες. Κάθε χρήστης είναι υπεύθυνος για τον εντοπισμό της θέσης του με κάποιες από τις διαθέσιμες τεχνικές εντοπισμού θέσης (βλ. 2.1.8) και τη δημοσιοποίηση της μετακίνησης του (*move*) στο κεντρικό σύστημα. Ο χρήστης αναφέρει περιοδικά τη θέση του στο κεντρικό σύστημα, με συχνότητα τέτοια ώστε το σύστημα να έχει όσο το δυνατό ακριβέστερη εικόνα για τη γεωγραφική του θέση.

Οι περιπτώσεις χρήσης του συστήματος από τους εγγεγραμμένους χρήστες φαίνονται στο UML διάγραμμα στο Σχήμα 48.



Σχήμα 48: Περιπτώσεις χρήσης του συστήματος από τους Εγγεγραμμένους Χρήστες

Για όλες τις περιπτώσεις χρήσης θα πρέπει να υλοποιηθούν τα αντίστοιχα Web Services. Η περιγραφή τους με χρήση της γλώσσας WSDL παρατίθεται στο αρχείο PLBS.wsdl που βρίσκεται στο συνοδευτικό υλικό.

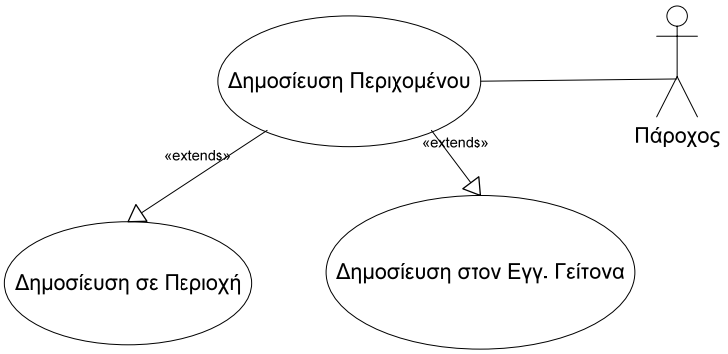
3.2.2 Πάροχοι Περιεχομένου

Οι πάροχοι δημοσιεύουν περιεχόμενο στο σύστημα με στόχο τους εγγεγραμμένους χρήστες. Οι δημοσιεύσεις μπορούν να έχουν ετερογενείς σκοπούς, όπως διαφημίσεις (π.χ. παρουσίαση ενός προϊόντος ή υπηρεσίας που διατίθεται στην περιοχή), ενημέρωση (π.χ. ενημέρωση για την κυκλοφορία ή δελτίο καιρού στην περιοχή) κτλ.

Η δημοσίευση του περιεχομένου γίνεται υπό μορφή εγγράφων XML τα οποία στέλνουν οι πάροχοι στον Κεντρικό Εξυπηρετητή, ο οποίος στη συνέχεια αναλαμβάνει να διεκπεραιώσει τη μεταβίβασή τους στους κατάλληλους χρήστες με βάση το προφίλ τους. Οι πάροχοι μπορούν να δημοσιεύουν τη θέση τους στο σύστημα (μέσω των Web Services που χρησιμοποιούν και οι εγγεγραμμένοι χρήστες) είτε να μην τη δημοσιεύουν. Και στις δύο περιπτώσεις μπορούν να δημοσιεύσουν περιεχόμενο σε μία συγκεκριμένη περιοχή (*send Document To Range*). Σε κάθε περίπτωση η πληροφορία που με αυτόν τον τρόπο δημοσιεύει ο πάροχος φθάνει μόνο στους τελικούς χρήστες τους οποίους ενδιαφέρει θεματικά με βάση το προφίλ τους.

Σε περίπτωση που ο χρήστης δημοσιεύει τη θέση του – είναι και εγγεγραμμένος χρήστης – μπορεί να αποστείλει ένα έγγραφο στο εγγύτερο ενδιαφερόμενο γείτονα (*send Document To NN*).

Οι περιπτώσεις χρήσης του συστήματος από τους παρόχους περιεχομένου φαίνονται στο UML διάγραμμα στο Σχήμα 49.



Σχήμα 49: Περιπτώσεις χρήσης συστήματος από τους Παρόχους Περιεχομένου

Για όλες τις περιπτώσεις χρήσης θα πρέπει να υλοποιηθούν τα αντίστοιχα Web Services. Η περιγραφή τους με χρήση της γλώσσας WSDL παρατίθεται στο αρχείο PLBS.wsdl που βρίσκεται στο συνοδευτικό υλικό.

3.3 Κεντρικός Εξυπηρετητής Συστήματος

Ο κεντρικός εξυπηρετητής του συστήματος φέρει σχεδόν όλο το υπολογιστικό και επικοινωνιακό φορτίο του συστήματος. Η απαίτηση για λειτουργία του συστήματος σε μεγάλη κλίμακα απαιτεί την ανάπτυξη αποδοτικών δομών και χρήση αλγορίθμων που θα επιτρέπουν την λει-

τουργία ενός τέτοιου συστήματος χωρίς τη χρήση κατανεμημένων αρχιτεκτονικών. Όρια στην επεκτασιμότητα του συστήματος θέτει και το διαθέσιμο εύρος ζώνης που θα είναι διαθέσιμο στον εξυπηρετητή για την επικοινωνία με τους χρήστες του συστήματος. Καρδιά του συστήματος θα είναι οι δομές χωρικής ευρετηρίασης (*spatial indexing*) και ευρετηρίασης των χρηστών με βάση το προφίλ τους (*profile indexing*).

3.3.1 Χωρική ευρετηρίαση (spatial indexing)

Το σκέλος που αφορά τον χωρικό δείκτη θα πρέπει να είναι αποδοτικό στις εξής λειτουργίες:

- Ευρετηρίαση σημείων
- Γρήγορες ενημερώσεις των θέσεων των σημείων (fast updates)
- Ερωτήματα περιοχής (range queries)
- Ερωτήματα εγγύτερης γειτονίας (nearest neighbor queries)

Η φύση των πληροφοριών που πρέπει να αποθηκεύονται και οι ενημερώσεις που απαιτούνται διαφοροποιούν το σύστημα από ένα παραδοσιακό ΣΔΒΔ ακόμα και αν αυτό ενσωματώνει δυνατότητες για διαχείριση γεωγραφικών πληροφοριών. Το σύστημα διαχειρίζεται τις τρέχουσες θέσεις, τα προφίλ των χρηστών και τις δημοσιεύσεις των παρόχων. Με εξαίρεση τα προφίλ των χρηστών, η σημασία της γνώσης των υπόλοιπων πληροφοριών δεν έχει παρά προσωρινό χαρακτήρα. Ενώ π.χ. είναι κρίσιμη για τη συνολική απόδοση του συστήματος η ταχύτητα με την οποία γίνεται η ενημέρωση της θέσης των χρηστών, δεν είναι εξίσου η σημαντική η ανάκτηση των τελευταίων θέσεων των χρηστών ύστερα από κατάρρευση του συστήματος για μία ώρα. Επίσης, οι δημοσιεύσεις των παρόχων δεν έχουν μόνιμο χαρακτήρα, αλλά αφορούν χρήστες σε συγκεκριμένη περιοχή με δεδομένα χαρακτηριστικά τη στιγμή της δημοσίευσης τους.

Ένα επιπλέον χαρακτηριστικό που μεταβάλλει τις απαιτήσεις για το σχεδιαζόμενο σύστημα είναι η συνεχής ενημέρωση όλης της ΒΔ που αφορά τις θέσεις και τα προφίλ των χρηστών. Ενώ σε ένα παραδοσιακό ΣΔΒΔ οι δοσοληψίες των εφαρμογών που τρέχουν διαχειρίζονται συνήθως ένα μικρό κομμάτι της ΒΔ, το οποίο και φέρνουν στη μνήμη, κάθε καταχώρηση στο σύστημα μας θα πρέπει να ενημερώνεται με το ρυθμό που θα επιλεγεί να γίνεται ενημέρωση από τις μικροσυσκευές (< 30 δευτερόλεπτα).

Οι παραπάνω παρατηρήσεις καθιστούν απαραίτητη την ανάπτυξη του χωρικού ευρετηρίου και του ευρετηρίου των προφίλ ως *δομές κύριας μνήμης*. Σε διαφορετική περίπτωση θα υπήρχαν απώλειες για τις ΙΟ προσβάσεις στο δίσκο χωρίς σημαντικό όφελος για την διατήρηση των δεδομένων σε περίπτωση αστοχίας του συστήματος.

3.3.2 Ταίριασμα προφίλ (profile matching)

Με βάση τη μελέτη που έγινε και παρατέθηκε παραπάνω η γλώσσα για τα έγγραφα των παρόχων θα είναι στο σύστημα μας η XML, ενώ τα προφίλ των χρηστών θα διατυπώνονται σε XPath. Η παραπάνω επιλογή επιτρέπει και την οργανωμένη δήλωση γνωρισμάτων και προφίλ με τη χρήση κατάλληλων XML σχημάτων (XML schemas) ενώ παράλληλα δίνει μεγαλύτερη ευελιξία για τον ορισμό προφίλ σε σχέση με συνεχή ερωτήματα σε σχεσιακές βάσεις δεδομένων (βλ. Παρ. 2.2).

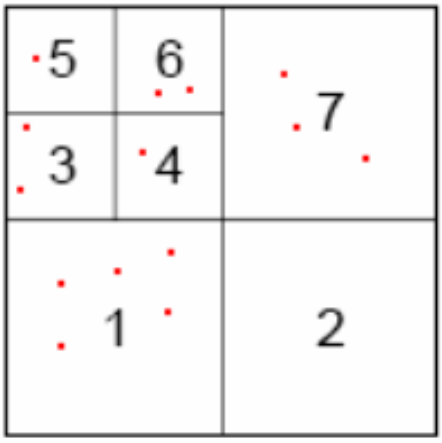
3.3.3 Βάση Δεδομένων

Η ΒΔ που φαίνεται στο σχέδιο της αρχιτεκτονικής του συστήματος θα χρησιμοποιείται για την αποθήκευση δεδομένων που δεν υπόκεινται στους περιορισμούς των παραπάνω παρατηρήσεων. Αυτά είναι δεδομένα που αναφέρονται στους εγγεγραμμένους χρήστες και παρόχους και θα χρησιμοποιούνται κατά τη φάση επαλήθευσης των χρηστών που εισέρχονται στο σύστημα. Στη ΒΔ θα οργανώνονται πληροφορίες χρήσιμες για την εξαγωγή στατιστικών στοιχείων.

3.4 SPI tree

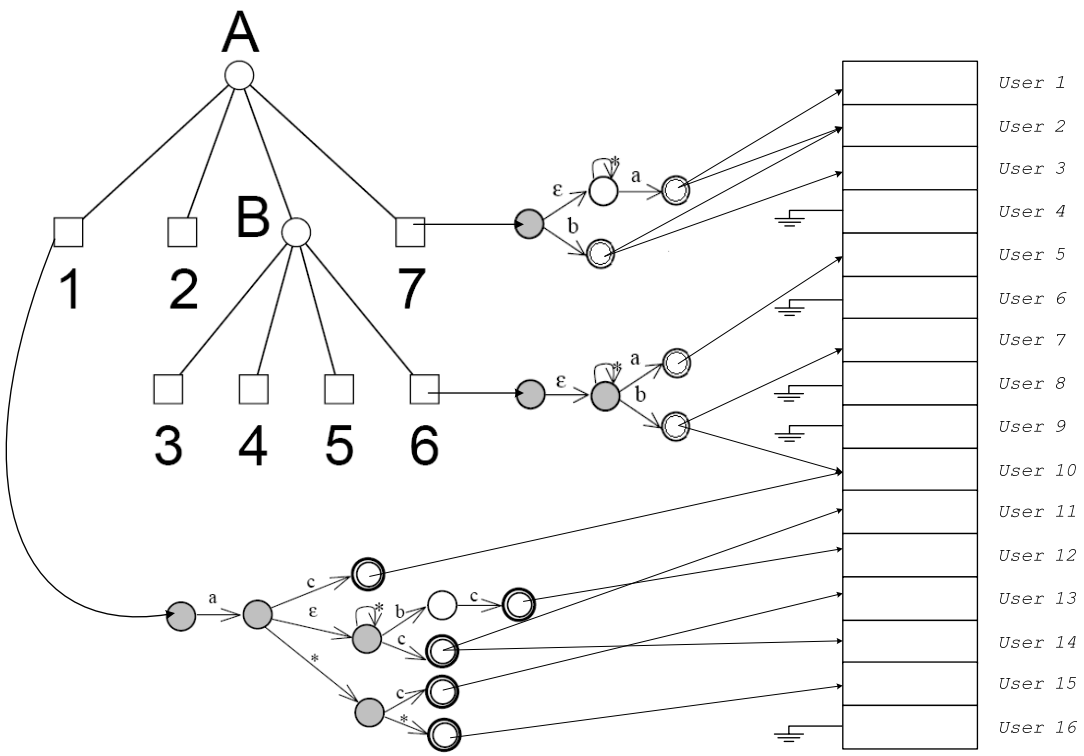
Όπως έχει ειπωθεί και παραπάνω, η «καρδιά» του κεντρικού εξυπηρετητή, αλλά και ολόκληρου του συστήματος γενικότερα, είναι οι δομές και οι αλγόριθμοι με τους οποίους ευρετηριάζονται οι χρήστες χωρικά και γίνεται το ταίριασμα των προφίλ μεταξύ των δραστών του συστήματος. Η πιο απλοϊκή λύση στο πρόβλημα που προσπαθούμε να λύσουμε είναι η ευρετηρίαση που θέλουμε να γίνεται σε δύο στάδια: στο πρώτο στάδιο να γίνεται η χωρική ευρετηρίαση όλων των χρηστών που μας ενδιαφέρουν χωρικά (δηλ. βρίσκονται εντός της εκάστοτε περιοχής ενδιαφέροντος του ερωτήματος) και στη συνέχεια το ταίριασμα του προφίλ του παρόχου που υποβάλλει το ερώτημα, με καθέναν από τους υποψήφιους χωρικά χρήστες, για να βρεθεί τελικά η απάντηση του ερωτήματος, δηλαδή το σύνολο των χρηστών που ικανοποιούν το ερώτημα τόσο ως προς τα χωρικά του κριτήρια, όσο και ως προς το περιεχόμενο (ή με την αντίστροφη σειρά των δύο σταδίων). Ωστόσο, μια τέτοια λύση είναι πολύ μακριά από τη βέλτιστη καθώς τα δύο στάδια εκτελούνται εντελώς ανεξάρτητα το ένα από το άλλο, με αποτέλεσμα να εξετάζεται πολύ μεγάλος αριθμός υποψήφιων χρηστών σε κάθε στάδιο, που θα μπορούσε να περιοριστεί. Για την αποδοτικότερη λειτουργία του κεντρικού εξυπηρετητή, στη λύση που παρουσιάζουμε παρακάτω, η χωρική ευρετηρίαση των χρηστών και το ταίριασμα των προφίλ αποτελούν μια ενιαία διαδικασία, η οποία διεκπεραιώνεται από μια δενδρική δομή δεδομένων, που συνδυάζει μεθόδους χωρικής ευρετηρίασης στους ενδιάμεσους κόμβους του δέντρου με τεχνικές ταιριάσματος προφίλ στα φύλλα του. Τη δομή αυτή ονομάζουμε Spatial and Profile Indexing tree (SPI tree) και περιγράφουμε στην συνέχεια.

Σε γενικές γραμμές, το SPI tree είναι μια δομή που προκύπτει από το «πάντρεμα» των PR-Quadtrees και των NFA που χρησιμοποιεί το YFilter (βλ. και κεφάλαιο 2). Πρόκειται για ένα τετραδικό δέντρο, το οποίο ευρετηριάζει το χώρο χωρίζοντας τον αναδρομικά σε τετράγωνα, ακριβώς όπως και το PR-Quadtree. Αντίθετα όμως με το PR-Quadtree, το SPI tree δεν αποθηκεύει στα φύλλα του ένα δείκτη προς κάποιον χρήστη (ή ένα κουβά με δείκτες προς περισσότερους χρήστες), αλλά ένα Nondeterministic Finite Automaton τύπου YFilter, που χρησιμοποιείται ως ευρετήριο των χρηστών που αντιστοιχούν στο φύλλο αυτό με βάση το προφίλ τους. Πιο συγκεκριμένα, κάθε εσωτερικός κόμβος του έχει 4 κόμβους - παιδιά. Κάθε κόμβος αντιστοιχεί σε μια τετραγωνική περιοχή του χώρου και κάθε παιδί του αντιστοιχεί στο ¼ της περιοχής του πατρικού του κόμβου, δηλαδή κάθε παιδί ενός κόμβου καλύπτει το ένα τεταρτημόριό του, ενώ ο κόμβος-ρίζα του δέντρου αντιστοιχεί σε όλο το χώρο. Από την άλλη, κάθε φύλλο του δέντρου, που επίσης αντιστοιχεί σε μια περιοχή-τεταρτημόριο της περιοχής του πατέρα του, περιέχει και ένα YFilter NFA, δηλαδή μιας δομής στην οποία ευρετηριάζονται τα XPath προφίλ των χρηστών του φύλλου. Επίσης, είναι εμφανές ότι σε κάθε φύλλο του δέντρου δεν αποθηκεύουμε τα χωρικά αντικείμενα (χρήστες) με τις συντεταγμένες τους, όπως στις παραδοσιακές δομές χωρικής ευρετηρίασης, αλλά μόνο το NFA το οποίο είναι ένα ευρετήριο για τους χρήστες με βάση το προφίλ τους – XPath query. Έτσι, η χωρητικότητα ενός φύλλου ορίζεται ως η χωρητικότητα του NFA του φύλλου σε XPath queries και ένας κόμβος φύλλο μετατρέπεται σε εσωτερικό κόμβο του δέντρου (και χάνει το NFA του) όταν ο αριθμός των XPath queries που αντιστοιχούν σε χρήστες που βρίσκονται στην περιοχή που καλύπτει το φύλλο, ξεπεράσει τη μέγιστη χωρητικότητα του NFA. Στο παρακάτω παράδειγμα (Σχήμα 50), υποθέτουμε ότι κάθε NFA, και άρα φύλλο, χωράει το πολύ 5 XPath ερωτήματα.



Σχήμα 50: Χωρική Ευρετηρίαση του SPI tree για μέγεθος κουβά 5

Ακολουθεί το σχήμα του αντίστοιχου SPI δέντρου (Σχήμα 51). Οι εσωτερικοί κόμβοι ονομάζονται με γράμματα, ενώ οι κόμβοι φύλλα με αριθμούς (για την οικονομία της παρουσίασης φαίνονται μόνο τα NFA που αντιστοιχούν στα φύλλα 1, 6, 7):



Σχήμα 51: Μορφή SPI tree

Συγκεκριμένα, στο SPI tree κάθε κόμβος του δέντρου είναι μια εγγραφή που περιέχει τα ακόλουθα πεδία:

- Ένα αντικείμενο *Quadrant* που είναι τύπου *Rectangle*, δηλαδή έχει τα υπο-πεδία *minLongitude, minLatitude, width, height* και αναπαριστά την ορθογωνική χωρική περιοχή στην οποία αντιστοιχεί ο κόμβος.
- Ένα πεδίο *nodeLocation* που μπορεί να πάρει τις ακόλουθες τιμές: *"UP_RIGHT"*, *"UP_LEFT"*, *"BOTTOM_RIGHT"*, *"BOTTOM_LEFT"*. Στο πεδίο αυτό αποθηκεύεται για κάθε κόμβο, σε ποιο από τα τέσσερα τεταρτημόρια του πατέρα του αντιστοιχεί (το πάνω βόρειο-ανατολικό έχει UP_RIGHT κτλ.).
- Ένα δείκτη *fatherNode* που δείχνει τον πατρικό του κόμβο (γεγονός που σημαίνει ότι μπορούμε να ανεβαίνουμε από τα φύλλα προς τη ρίζα).
- Αν ο κόμβος είναι εσωτερικός (τύπου *IndexInternalNode*), τότε περιέχει ένα πίνακα δεικτών 4 θέσεων *childNodes[]*, στον οποίο αποθηκεύονται οι δείκτες προς τα τέσσερα παιδιά του κόμβου(στην πρώτη θέση το UP_RIGHT παιδί, στη δεύτερη το UP_LEFT, στην τρίτη το BOTTOM_RIGHT και στην τέταρτη το BOTTOM_LEFT)
- Επίσης, στην περίπτωση εσωτερικού κόμβου, υπάρχει ένα πεδίο ακεραίου *noOfOccupiedChildren* στο οποίο κρατάμε τον αριθμό των παιδιών του κόμβου που δεν είναι άδεια, δηλ. περιέχουν τα προφίλ κάποιων χρηστών.

- Αν αντίθετα πρόκειται για κόμβο φύλλο(τύπου *IndexLeaf*), υπάρχει ένας δείκτης *nodeFilter* που δεικτοδοτεί το NFA που αντιστοιχεί στο φύλλο. Επίσης, υπάρχει ένας ακέραιος *nodeUsersNo* που κρατάει τον αριθμό των χρηστών, των οποίων το προφίλ υπάρχει αποθηκευμένο στο δέντρο(στη συνέχεια θεωρούμε για απλότητα ότι το προφίλ κάθε χρήστη αποτελείται από ένα XPath ερώτημα).

Οι χρήστες, τους οποίους το SPI tree ευρετηριάζει, έχουν ο καθένας από ένα μοναδικό id, που είναι ακέραιος αριθμός, και τους προσδιορίζει μονοσήμαντα. Επίσης κάθε αντικείμενο χρήστη έχει ένα XPath query που προσδιορίζει το προφίλ του και φυσικά τα πεδία longitude και latitude που αντιστοιχούν στις γεωγραφικές του συντεταγμένες(η πρώτη στο γεωγραφικό μήκος και η δεύτερη στο γεωγραφικό του πλάτος). Κάθε χρήστης γνωρίζει το φύλλο του δέντρου στο οποίο βρίσκεται εισηγμένος ανά πάσα στιγμή. Αυτό γίνεται διατηρώντας ένα δείκτη *fatherLeaf* στο αντικείμενο του χρήστη που δεικτοδοτεί το φύλλο του SPI tree στο οποίο βρίσκεται ο χρήστης.

Σημειώνεται επίσης ότι εκτός από το καθαυτό SPI tree, διατηρείται και μια επικουρική δομή δεδομένων για την ευρετηρίαση των χρηστών με βάση το αναγνωριστικό (*id*) τους, δηλαδή για να έχουμε γρήγορη πρόσβαση στις ιδιότητες ενός χρήστη με βάση το id του και να βρίσκουμε άμεσα το φύλλο του δέντρου στου οποίου το NFA βρίσκεται εισηγμένος ο χρήστης. Η δομή αυτή είναι ένα Red-Black tree, το οποίο είναι ένα ισοζυγισμένο δυαδικό δέντρο αναζήτησης για την αποθήκευση αντικειμένων για τα οποία ορίζεται μια σχέση ολικής διάταξης. Πρόκειται για μια από τις δομές που εξασφαλίζουν βέλτιστο χρόνο εισαγωγής, διαγραφής, και κυρίως αναζήτησης καθώς εξασφαλίζει για τις λειτουργίες αυτές χρόνο *log* n. Η δομή αυτή περιέχει όλους τους δηλωμένους χρήστες.

Παρακάτω περιγράφονται οι αλγόριθμοι που χρησιμοποιούνται για τις διάφορες λειτουργίες στο SPI tree.

3.4.1 Εισαγωγή (Insertion)

Η εισαγωγή ενός νέου χρήστη στο δέντρο είναι μια διαδικασία που γίνεται από τη ρίζα του δέντρου προς τα φύλλα του. Αρχικά πρέπει να βρεθεί σε ποιο φύλλο του δέντρου ανήκει χωρικά ο χρήστης, δηλαδή να βρεθεί το μοναδικό φύλλο του δέντρου του οποίου η περιοχή περιέχει τις συντεταγμένες του χρήστη (στο εξής αναφερόμαστε στην περιοχή του χώρου στην οποία αντιστοιχεί ένας εσωτερικός κόμβος ή φύλλο ως «περιοχή του κόμβου» ή «περιοχή του φύλλου αντίστοιχα»). Αυτό γίνεται ξεκινώντας από τη ρίζα του δέντρου και ακολουθώντας το μονοπάτι που μας πηγαίνει κάθε φορά στο παιδί του τρέχοντος κόμβου που αντιστοιχεί στο τεταρτημόριο που περιέχει τις συντεταγμένες του χρήστη προς εισαγωγή(και το οποίο προφανώς είναι μοναδικό). Πιο απλά, όταν είμαστε σε ένα κόμβο του δέντρου που δεν είναι φύλλο και θέλουμε να βρούμε το φύλλο στο οποίο πρέπει να μπει ο χρήστης, επισκεπτόμαστε το παιδί του τρέχοντος

κόμβου του οποίου η περιοχή περικλείει το σημείο στο οποίο βρίσκεται ο χρήστης. Αφού κατεβούμε το μονοπάτι από τη ρίζα του δέντρου προς το κατάλληλο φύλλο υπάρχουν δύο περιπτώσεις: είτε ο νέος χρήστης να χωράει στο φύλλο (δηλ. το προφίλ του να χωράει στο NFA) είτε το φύλλο να είναι γεμάτο, δηλαδή να περιέχει στο NFA του το μέγιστο αριθμό χρηστών. Στην πρώτη περίπτωση απλά προσθέτουμε το χρήστη στο φύλλο, με το να εισάγουμε το XPath query που του αντιστοιχεί στο NFA του φύλλου, δηλαδή δημιουργώντας μέσα στο φύλλο ένα ευρετήριο προς το χρήστη με βάση το προφίλ του. Αν στο φύλλο δεν υπήρχε κανένας χρήστης, τότε πρέπει να δημιουργηθεί και το NFA του φύλλου πρώτα, καθώς σε κάθε άδειο φύλλο που κατασκευάζεται δε δημιουργούμε από την αρχή τη δομή NFA, καθώς το φύλλο αυτό μπορεί να μην αποκτήσει ποτέ χρήστες. Στην περίπτωση που ο νέος χρήστης δε χωράει στο φύλλο, τότε το φύλλο «σπάει» σε τέσσερα νέα φύλλα αναδρομικά. Δηλαδή το φύλλο που έχει υπερχειλίσει μετατρέπεται σε εσωτερικό κόμβο του δέντρου και δημιουργούνται τέσσερα νέα φύλλα, τα οποία ορίζονται ως παιδιά του. Οι χρήστες που περιέχονταν στο αρχικό φύλλο κατανέμονται στα παιδιά του με το γνωστό τρόπο (στο κατάλληλο τεταρτημόριο με βάση τις συντεταγμένες τους). Σημειώνεται ότι ορίζουμε και το μέγιστο ύψος του δέντρου (MAX_HEIGHT) και θεωρούμε ότι ένα φύλλο που βρίσκεται στο μέγιστο βάθος έχει άπειρη χωρητικότητα(δε διασπάται όταν αποκτήσει περισσότερους χρήστες από το όριο). Παρακάτω φαίνεται σε ψευδογλώσσα ο αλγόριθμος που περιγράψαμε:

```
// Insert a user into an internal node.
InternalNode.Insert(User user) {

   // user is out of indexing bounds of node
   if (! this.contains(user.longitude, user.latitude) )
      return;

   Node current, child, zombie := null;
   current := this;

   // find the appropriate leaf
   repeat{
      child = current.getChildContaining(user.longitude, user.latitude);
      current := child;
   }
   until (child is Leaf);

   child.Insert(user);
}



// Insert directly into a leaf node
Leaf.Insert(User user) {

   if (Leaf is not full || depth > MAX_HEIGHT) {
      // Leaf is safe. No need to split
```

```
    if(Leaf is empty) {
      // Initializations needed for the first record of the leaf
      fatherNode.incrOccupiedChildren();
      nodeFilter := new YfilterNFA();
    }

    user.fatherLeaf := leafNode;
    nodeFilter.addProfile(user.profile);
  }
  else {
    // Unsafe Leaf. Leaf splitting required

    // Init replacing node. It has the same father and range
    // with the leaf that it going to replace. 4 children leaf
    // modes are initialized too.
    newNode := new IndexInteranlNode(fatherNode, range;);
    newNode.noOfOccupiedChildren := 0;

    fatherNode.setChild(replacingNode);
    // Redistribute user records of zombie leaf
    for (all existing users in leafNode)
      newNode.Insert(currentUser);   // newNode is already locked.
                                     // InternalNode.Insert needs some
                                     // trivial modifications to avoid
                                     // deadlock

    // Insert initial user
    newNode.Insert(user);
  }
}
```

3.4.2 Διαγραφή (Deletion)

Για τη διαγραφή ενός χρήστη από το SPI tree με βάση το id του, αρχικά εντοπίζεται το φύλλο του δέντρου στο οποίο βρίσκεται εισηγμένος ο χρήστης. Αυτό γίνεται εύκολα ακολουθώντας το δείκτη fatherLeaf του αντικειμένου που αντιστοιχεί στο χρήστη (το οποίο βρίσκεται εύκολα μέσω του Red-Black tree). Στη συνέχεια διαγράφεται ο χρήστης από το φύλλο του δέντρου, δηλαδή διαγράφεται από το NFA του φύλλου το XPath ερώτημα που αποτελεί το προφίλ του χρήστη. Φυσικά ενημερώνεται και ο μετρητής nodeUsersNo που καρτάει τον αριθμό των χρηστών που βρίσκονται εντός του δέντρου. Αν μετά τη διαγραφή αυτή το φύλλο είναι άδειο, δηλαδή δεν περιέχει κανέναν άλλο χρήστη, ενημερώνεται ο πατρικός κόμβος του και συγκεκριμένα ο μετρητής του noOfOccupiedChilden ελαττώνεται κατά 1. Αν ο μετρητής αυτός φτάσει στο 0, τότε έχουμε *σύμπτυξη (collapsing)* του δέντρου, που είναι η αντίστροφη διαδικασία του χωρισμού ενός κόμβου-φύλλου σε τέσσερα νέα φύλλα. Η σύμπτυξη («μάζεμα») δηλαδή του δέντρου λαμβάνει χώρα αδειάσουν και τα τέσσερα φύλλα-παιδιά ενός εσωτερικού κόμβου, δηλαδή όταν ο μετρητής noOfOccupiedChilden ενός κόμβου που έχει ως παιδιά 4 φύλλα γίνει 0. Τότε τα τέσσερα αυτά φύλλα διαγράφονται και ο πατέρας τους από εσωτερικός κόμβος μετατρέπεται σε φύλλο (που δεν έχει βέβαια κανένα χρήστη) και έτσι το ύψος του συγκεκριμένου υποδέντρου ελαττώνεται κατά 1. Το μάζεμα αυτό διαδίδεται και προς τα πάνω στο δέντρο μέ-

χρι να βρεθεί κάποιος κόμβος ο οποίος να έχει τουλάχιστον ένα φύλλο μη αδειανό. Με αυτό τον τρόπο το δέντρο προσαρμόζεται στις μετακινήσεις των χρηστών και διαγράφει τα περιττά φύλλα και τις περιττές δομές NFA συμβάλλοντας έτσι στην καλύτερη αξιοποίηση της μνήμης. Πιο συγκεκριμένα η διαγραφή ενός χρήστη γίνεται με βάση τον ακόλουθο αλγόριθμο:

```
Delete (User user) {
   // Find leaf that contains the user
   leaf := user.fatherLeaf;

   // delete from leaf
   leaf.nodeFilter.deleteProfile(user.profile);
   user.fatherLeaf := null;

   // If Leaf is empty, inform father that this Leaf is no
   // more occupied
   if (Leaf is empty now) {
      leaf.nodeFilter := null;
      leaf.fatherNode.DecrOccupiedChildren();
   }
}

InternalNode.DecrOccupiedChildren() {
   noOfOccupiedChildren--;
   boolean fatherNeedsUpdate := false;
   if (noOfOccupiedChildren == 0 and depth > 0) {

      // Init replacing leaf. It has the same father and range
      // with the leaf that it is going to replace
      newNode := new Leaf(father, range);

      // Set father's new child
      fatherNode.setChild(replacingNode);
      fatherNeedsUpdate := true;
   }

   // Perform backtracking if needed
   if (fatherNeedsUpdate)
      fatherNode.DecrOccupiedChildren();
}
```

3.4.3 Ενημέρωση (Update)

Ενημέρωση της θέσης ενός χρήστη έχουμε προφανώς όταν ο χρήστης μεταβαίνει από μια θέση σε μια νέα. Πριν τη μετακίνηση ο χρήστης είναι εισηγμένος στο NFA του φύλλου που αντιστοιχεί σε μια περιοχή που περιέχει τη θέση του. Η νέα θέση στην οποία μεταβαίνει ο χρήστης μπορεί να είναι ή να μην είναι εντός της περιοχής αυτής, δηλαδή η νέα θέση του χρήστη μπορεί να περικλείεται από το φύλλο στο οποίο ήδη βρισκόταν ή όχι. Αν αυτό συμβαίνει, δηλαδή η νέα θέση του χρήστη βρίσκεται εντός του φύλλου στο οποίο ήταν ήδη εισηγμένος, τότε η ενημέρωση είναι πολύ απλή καθώς ο χρήστης παραμένει στο ίδιο φύλλο και ως εκ τούτου στο ίδιο NFA. Το μόνο που απαιτείται είναι φυσικά η ενημέρωση των συντεταγμένων του χρήστη. Σε αντίθετη περίπτωση, η νέα θέση του χρήστη είναι σε κάποιο άλλο φύλλο του δέντρου. Η διαδικασία που ακολουθείται η εξής: αρχικά ξεκινάμε από το φύλλο στο οποίο ήταν εισηγμένος ο

χρήστης και προχωράμε προς τα πάνω στο δέντρο έως ότου να βρούμε έναν εσωτερικό κόμβο(έστω Α) ο οποίος αντιστοιχεί σε μια περιοχή που περικλείει τη θέση-προορισμό της μετακίνησης του χρήστη. Τότε γνωρίζουμε ότι το φύλλο που περικλείει την νέα θέση του χρήστη είναι απόγονος του Α. Έτσι, διαγράφουμε το χρήστη από το φύλλο που βρισκόταν πριν τη μετακίνηση και στη συνέχεια τον εισάγουμε στον κόμβο Α (και φυσικά η εισαγωγή προχωρά αναδρομικά μέχρι τα φύλλα όπως έχει περιγραφεί παραπάνω). Παρακάτω φαίνεται ο αλγόριθμος της ενημέρωσης σε ψευδοκώδικα:

```
Update (User user, real newLongitude, real newLatitude) {
   // Find the common node of the lowest level of the following paths:
   // 1. from source leaf to root
   // 2. from destination leaf to root
   commonNode = user.fatherLeaf.getCommonNode(newLongitude, newLatitude);

   if (commonNode == null){
      Error("User moved out of indexing bounds");
      return;
   }

   if (commonNode == user.fatherLeaf){
      // User moved within the bounds of the containing leaf.
      // Just update user's coordinates
      user.longitude := newLongitude;
      user.latitude := newLatitude;
      return;
   }

   Delete(user);

   user.longitude := newLongitude;
   user.latitude := newLatitude;
   commonNode.Insert(user);
}

Node.getCommonNode(real longitude, real latitude) {
   if (Node.contains(longitude, latitude))
      // Node found
      return Node;

   if (Node is root)
      // Coordinates are out of indexing bounds
      return null;

   return Node.fatherNode.getCommonNode(longitude, latitude);
}
```

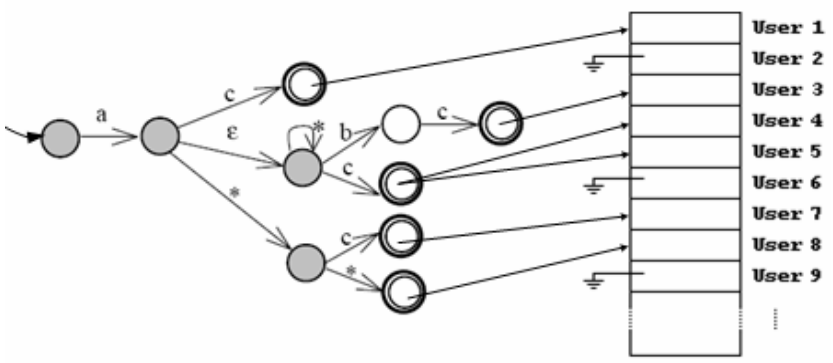
3.4.4 Λειτουργία NFA

Οι δομές Nondeterministic Finite Automata που χρησιμοποιεί το σύστημα YFilter (βλ. κεφάλαιο 2ο) τροποποιήθηκαν ώστε να επιτύχουμε την επιθυμητή τους λειτουργία στους κόμβους φύλλα του SPI tree. Συγκεκριμένα, για την πιο αποδοτική λειτουργία του κεντρικού εξυπηρετητή, η συντακτική ανάλυση κάθε XPath ερωτήματος (δηλαδή του προφίλ ενός χρήστη) γίνεται τοπικά στο τερματικό του χρήστη και αποστέλλεται στο σύστημα σε μορφή που να είναι πολύ

πιο εύκολη η εισαγωγή του στο εκάστοτε NFA. Στη μορφή αυτή αποθηκεύεται και στο αντικείμενο που αντιστοιχεί σε κάθε χρήστη.

Καθώς η εκτέλεση ενός αυτόματου οδηγείται από ένα XML έγγραφο, το αυτόματο φτάνει σε μια τελική κατάσταση αν το έγγραφο αυτό ικανοποιεί ένα ή περισσότερα XPath ερωτήματα (που βρίσκονται αποθηκευμένα στο αυτόματο). Τα ερωτήματα που ικανοποιούνται είναι αυτά που μοιράζονται την τελική κατάσταση στην οποία έφτασε η εκτέλεση του αυτόματου. Για το λόγο αυτό, σε κάθε τελική κατάσταση του αυτόματου εισαγάγαμε δείκτες προς τους χρήστες οι οποίοι έχουν υποβάλλει κάποιο XPath ερώτημα, η αναπαράσταση του οποίου στο αυτόματο έχει τη συγκεκριμένη κατάσταση ως τελική. Με αυτό τον τρόπο όποτε ένα XPath ερώτημα βρίσκεται να ταιριάζει με ένα έγγραφο, το σύστημα μπορεί άμεσα να βρει τους χρήστες στους οποίους πρέπει να αποσταλεί το έγγραφο.

Σημειώνεται επίσης ότι η προσθήκη ή αφαίρεση χρηστών από το NFA γίνεται συγχρονισμένα, δηλαδή μόνο όταν το NFA δεν εκτελείται (μεταξύ της εκτέλεσης δύο εγγράφων).



Σχήμα 52: Ευρετηρίαση Προφίλ του SPI tree

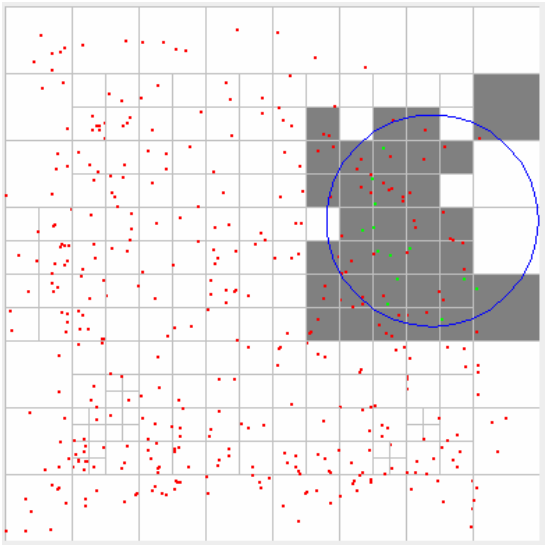
3.4.5 Εκτέλεση Ερωτημάτων

Το περιεχόμενο των υπηρεσιών που παρέχονται στους χρήστες μέσω του συστήματος καθορίζεται από τους παρόχους της πληροφορίας στο σύστημα και συγκεκριμένα από τα έγγραφα XML τα οποία οι πάροχοι περιεχομένου υποβάλλουν προς το σύστημα. Τα έγγραφα αυτά εκτός του ότι χρησιμοποιούνται για το ταίριασμα προφίλ μεταξύ παρόχου και χρήστη, είναι αυτά που περιέχουν και την καθαυτό πληροφορία που ο πάροχος επιθυμεί να μεταβιβάσει στο χρήστη. Συγκεκριμένα, για κάθε έγγραφο XML που υποβάλλεται προς ένα πληθυσμό χρηστών (με τους τρόπους που παρουσιάζονται παρακάτω), το σύστημα αρχικά εντοπίζει τους χρήστες των οποίων το προφίλ ταιριάζει με το περιεχόμενο του εγγράφου και στη συνέχεια αποστέλλει το XML έγγραφο, που περιέχει και την ουσιαστική για τους χρήστες πληροφορία, προς τους χρήστες. Η υποβολή ενός εγγράφου στο σύστημα αποτελεί ένα *ερώτημα* προς το σύστημα. Κάθε πάροχος περιεχομένου/χρήστης του συστήματος μπορεί να υποβάλει δύο ειδών ερωτήματα: ερωτήματα με βάση κάποιο χωρικό εύρος (*range queries*) και ερωτήματα εύρεσης εγγύτερου

γείτονα (*nearest neighbor queries*). Η σημασία και ο τρόπος αποτίμησης των ερωτημάτων αυτών εξηγείται στη συνέχεια.

3.4.5.1 Ερωτήματα Περιοχής (Range Queries)

Σε ένα ερώτημα περιοχής ένας πάροχος περιεχομένου υποβάλλει στο σύστημα ένα XML έγγραφο και μια κλειστή γεωγραφική περιοχή (π.χ. μια κυκλική περιοχή) και παίρνει ως απάντηση το σύνολο των χρηστών που α) βρίσκονται τη στιγμή της αποτίμησης του ερωτήματος εντός της γεωγραφικής αυτής περιοχής και β) το έγγραφο ταιριάζει θεματικά με τις προτιμήσεις των χρηστών. Για παράδειγμα, μια κινηματογραφική εταιρία στέλνει μια διαφήμιση για μια ταινία που προβάλλεται σε μια κινηματογραφική αίθουσα σε όλους του χρήστες που βρίσκονται σε ακτίνα 2 χιλιομέτρων από την αίθουσα και έχουν κινηματογραφικά ενδιαφέροντα. Σχηματικά, ένα ερώτημα περιοχής φαίνεται παρακάτω (Σχήμα 53):



Σχήμα 53: Ερώτημα Περιοχής στο SPI tree

Στο παραπάνω σχήμα απεικονίζεται ολόκληρη η περιοχή στην οποία παρέχονται οι υπηρεσίες και φαίνεται ο αναδρομικός χωρισμός του χώρου από το τετραδικό δέντρο (τα τετράφωνα με τις γκρι πλευρές). Με κόκκινες κουκίδες απεικονίζονται οι χρήστες του συστήματος. Στο παραπάνω στιγμιότυπο, έχουμε υποβάλλει ένα ερώτημα περιοχής στην περιοχή που οροθετείται από το μπλε κύκλο. Τα σκιασμένα γκρι τετράγωνα αντιστοιχούν στα φύλλα του δέντρου στα οποία έφτασε τελικά το XML έγγραφο (με τη μέθοδο που περιγράφεται παρακάτω), ενώ με πράσινες κουκίδες φαίνονται οι χρήστες που αποτελούν την απάντηση στο ερώτημα, δηλαδή βρίσκονται εντός της κυκλικής περιοχής και το προφίλ τους ταιριάζει με το XML έγγραφο του ερωτήματος.

Η αποτίμηση ενός ερωτήματος περιοχής περιλαμβάνει αφενός την εύρεση των φύλλων του δέντρου στα οποία πρέπει να «σταλεί» το XML έγγραφο, δηλαδή των φύλλων του δέντρου που αντιστοιχούν στη γεωγραφική περιοχή-στόχο του ερωτήματος και αφ' ετέρου το ταίριασμα του

εγγράφου με τα προφίλ των χρηστών που βρίσκονται στα συγκεκριμένα φύλλα, δηλαδή την οδήγηση της εκτέλεση των NFA των φύλλων αυτών από το συγκεκριμένο XML έγγραφο. Αρχικά, υπολογίζεται το περιγεγραμμένο ορθογώνιο της περιοχής-στόχου του ερωτήματος και στην συνέχεια κατεβαίνοντας το δέντρο από τη ρίζα προς τα φύλλα, κάθε κόμβος προωθεί το ερώτημα (δηλ. το ορθογώνιο και το XML έγγραφο) σε κάθε παιδί του, το οποίο αντιστοιχεί σε γεωγραφική περιοχή που επικαλύπτεται (σε ένα ή περισσότερα σημεία) από το ορθογώνιο του ερωτήματος. Με τον τρόπο αυτό το έγγραφο φτάνει τελικά στα φύλλα του δέντρου που επικαλύπτονται από κάποιο κομμάτι του ορθογωνίου του ερωτήματος. Στην συνέχεια εκτελούνται τα NFA των φύλλων αυτών με βάση το έγγραφο του ερωτήματος και από τις τελικές καταστάσεις στις οποίες φτάνει η εκτέλεση τους, προκύπτουν οι χρήστες που το προφίλ τους ταιριάζει με το έγγραφο. Στη συνέχεια βρίσκουμε ποιοι από τους χρήστες αυτούς βρίσκονται στην αρχική περιοχή του ερωτήματος (και όχι μόνο στο περιγεγραμμένο της ορθογώνιο) και οι χρήστες αυτοί αποτελούν την απάντηση του ερωτήματος, καθώς το ικανοποιούν και χωρικά και θεματικά. Σημειώνεται ότι η προώθηση του ερωτήματος στα φύλλα του δέντρου γίνεται με βάση το περιγεγραμμένο ορθογώνιο της περιοχής του ερωτήματος, λόγω του ότι αυτό απλοποιεί κατά πολύ υπολογιστικά την εύρεση της επικάλυψης με τους ορθογωνικούς κόμβους του δέντρου. Σε αυτό οφείλεται και το γεγονός ότι στο παραπάνω σχήμα το έγγραφο έχει φτάσει σε φύλλα του δέντρου που βρίσκονται εκτός της κυκλικής περιοχής του ερωτήματος (αλλά φυσικά εντός του περιγεγραμμένου της τετραγώνου). Ο αλγόριθμος φαίνεται σε ψευδογλώσσα παρακάτω:

```
InternalNode.sendDocument(Shape region, XMLDocument doc, List result) {
  // result is the list to store the result set of users
  for (all children of Node) {
    if (overlaps(child.range, region.boundingRectangle))
      child.sendDocument(region, doc, result);
  }
}

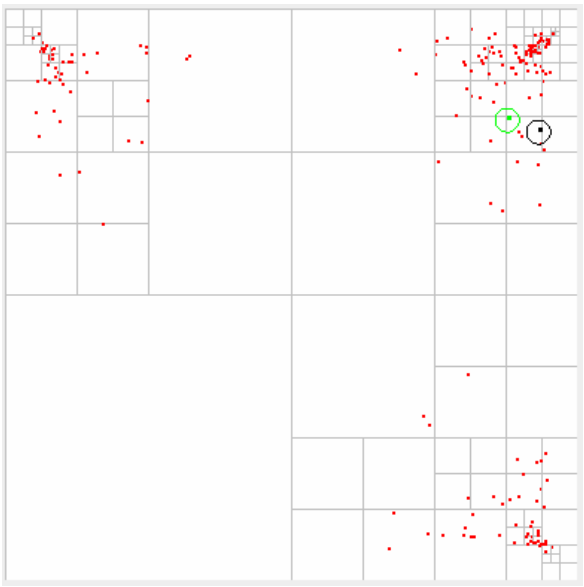
Leaf.sendDocument(Shape region, XMLDocument doc, List result) {
  if (Leaf.nodeFilter != null)
    // parse doc and return matching user list
    newUsers := leaf.nodeFilter.parse(doc);

  // check user position
  for (all users in newUsers list) {
    if (contains(region, user.longitude, user.latitude))
      result.append(user);
  }
}
```

3.4.5.2 Ερωτήματα Εγγύτερου Γείτονα (Nearest Neighbor Queries)

Σε ένα ερώτημα εγγύτερου γείτονα ένας πάροχος περιεχομένου υποβάλλει στο σύστημα ένα XML έγγραφο και το αναγνωριστικό (*id*) ενός χρήστη (συνήθως το αναγνωριστικό του ιδίου) και παίρνει ως απάντηση το αναγνωριστικό του χρήστη που α) το προφίλ του ταιριάζει με το

XML έγγραφο και β) έχει την ελάχιστη ευκλείδεια απόσταση από το χρήστη-αποστολέα, δηλαδή με ένα τέτοιο ερώτημα ο αποστολέας βρίσκει τον κοντινότερο χρήστη του οποίου το προφίλ ταιριάζει με το έγγραφο. Για παράδειγμα, ένας χρήστης μπορεί με το ερώτημα αυτό να εντοπίσει ποιος από τους φίλους του είναι πιο κοντά του σε μια δεδομένη χρονική στιγμή. Για να γίνει αυτό, αρκεί στο XML έγγραφο να έχει το id του και καθένας από τους φίλους του έχει στο προφίλ του να προσδιορίζει τους φίλους του(π.χ. με ένα κατηγόρημα XPath). Στο παρακάτω σχήμα (Σχήμα 54) φαίνεται το στιγμιότυπο ενός τέτοιου ερωτήματος, όπου με μαύρη κουκίδα φαίνεται ο χρήστης με βάση τον οποίο γίνεται το ερώτημα, ενώ με πράσινη κουκίδα φαίνεται η απάντηση στο ερώτημα, δηλαδή ο κοντινότερος στον «μαύρο» χρήστης το προφίλ του οποίου ταιριάζει με το XML έγγραφο.



Σχήμα 54: Ερώτημα Εγγύτερου γείτονα στο SPI tree

Η αποτίμηση ενός ερωτήματος εγγύτερου γείτονα βασίζεται στο γεγονός ότι αν βρεθεί ένας χρήστης του οποίου το προφίλ ταιριάζει στο XML έγγραφο του ερωτήματος, τότε είτε αυτός ο χρήστης είναι η απάντηση στο ερώτημα είτε ο χρήστης-απάντηση του ερωτήματος βρίσκεται σε απόσταση μικρότερη από αυτή του πρώτου χρήστη που βρέθηκε. Αρχικά η αναζήτηση γίνεται στο φύλλο όπου υπάρχει ο χρήστης-αποστολέας του ερωτήματος, δηλαδή το έγγραφο αποτιμάται στο NFA του φύλλου. Αν δε βρεθεί κάποιος χρήστης που το προφίλ του ταιριάζει με το έγγραφο, τότε το έγγραφο στέλνεται στον πατρικό κόμβο του φύλλου και αυτός το στέλνει στα παιδιά του(στων οποίων τα NFA αποτιμάται) και η διαδικασία συνεχίζεται αναδρομικά προς τα πάνω στο δέντρο μέχρι να βρεθεί κάποιος χρήστης που το προφίλ του ταιριάζει με το έγγραφο. Αν δε βρεθεί τέτοιος χρήστης, η διαδικασία τερματίζεται και δεν υπάρχει ο ζητούμενος κοντινότερος γείτονας. Αν ένας τέτοιος χρήστης βρεθεί (έστω Β), είτε στο φύλλο που βρίσκεται ο αποστολέας (έστω Α) είτε σε κάποιο άλλο φύλλο, τότε απομένει να διαπιστωθεί αν αυτός ο χρήστης είναι όντως ο κοντινότερος ή όχι. Για το σκοπό αυτό υπολογίζεται η απόσταση του

χρήστη A από το B και στη συνέχεια εκτελείται ένα ερώτημα εύρους (range query) με κέντρο το χρήστη A και ακτίνα AB. Αν το ερώτημα αυτό επιστρέψει το κενό αποτέλεσμα, αυτό σημαίνει ότι ο A είναι όντως ο ζητούμενος χρήστης, διαφορετικά ανάμεσα στους χρήστες που επέστρεψε το range query αναζητείται αυτός με την ελάχιστη απόσταση από τον A, ο οποίος τώρα γνωρίζουμε ότι είναι ο κοντινότερος γείτονας του A που το προφίλ του ταιριάζει με το έγγραφο. Ακολουθεί ο αλγόριθμος σε ψευδογλώσσα:

```
sendNNDocument(User sender, XMLDocument doc) {
  // find initial matching user
  node := sender.fatherLeaf;
  matchingUser := null;
  while (matchingUser == null && node != null) {
    // recurse to upper tree level
    tmpNode := node.fatherNode;

    // find a matching user in node's descendants
    matchingUser := node.findNeighbor(doc);
    node := tmpNode;
  }

  if (matchingUser == null)
    print("No matching user found");
  else {
    // initial user found
    dx := sender.longitude – matchingUser.longitude;
    dy := sender.latitude – matchingUser.latitude;
    radius := sqrt(dx^2 + dy^2);
    c := new Circle(sender.longitude, sender.latitude, radius);
    root.sendDocument(c, doc, resultUsers = new List());
    nearestNeighbor := matchingUser;
    d := dx^2 + dy^2;

    for (all candidates in resultUsers) {
    // find matchingUser with minimum distance
      cand_dx := sender.longitude – candidate.longitude;
      cand_dy := sender.longitude – candidate.longitude;
      if (cand_dx^2 + cand_dy^2 < d){
        nearestNeighbor := candidate;
        minDistance := cand_dx^2 + cand_dy^2;
      }
    }
  }
  return nearestNeighbor;
}

InternalNode.findNeighbor(XMLDocument doc) {
  for (all children of Node) {
    resultUser := child.findNeighbor(doc);

    // Stop at the first success
    if (returnUser != null)
      return resultUser;
  }
  return null;
}
```

```
Leaf.findNeighbor(XMLDocument doc) {
   if (leaf.nodeFilter != null) {
     resultUsers := leaf.nodeFilter.parse(doc);

     // pick a random matching user
     if (!isEmptyList(resultUsers))
       return resultUsers.getFirst();
   }
   return null;
}
```

3.5 Πρωτόκολλο κλειδωμάτων του SPI tree

Οι αλγόριθμοι που παρατέθηκαν παραπάνω εξασφαλίζουν τη συνέπεια του δέντρου του SPI ύστερα από κάθε μεμονωμένη προσπέλαση σε αυτό. Δεν έχει ληφθεί όμως καμία πρόνοια για τη διατήρηση του δέντρου σε συνεπή μορφή και το διάβασμα έγκυρων τιμών όταν έχουμε ταυτόχρονες προσπελάσεις σε αυτό για ενημερώσεις ή αναγνώσεις. Η προφανής λύση της σειριακής εκτέλεσης των προσπελάσεων στο δέντρο, παρά την απλότητα της, επιφέρει ανεπίτρεπτη καθυστέρηση στην εκτέλεση τους και δεν επιτρέπει την εκμετάλλευση των πλεονεκτημάτων της εκτέλεσης σε ένα πολυνηματικό (multithreading) περιβάλλον. Σε πιο σύνθετους αλγόριθμους κλειδωμάτων υπεισέρχονται τα κλασικά προβλήματα του ταυτοχρονισμού όπως τα αδιέξοδα (deadlocks) και η λιμοκτονία (starvation).

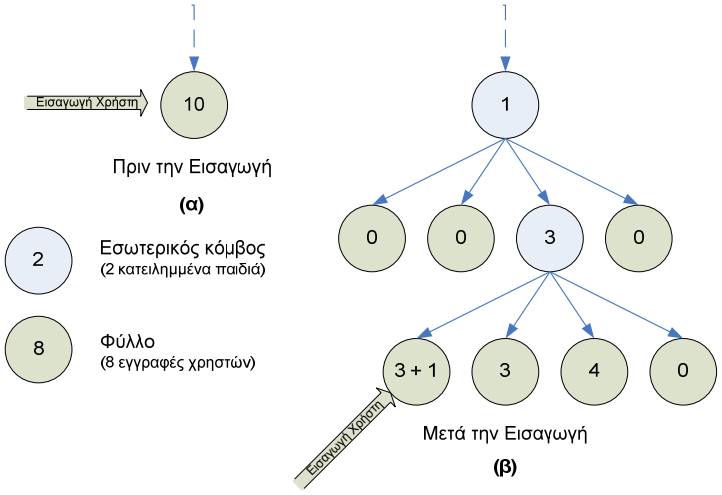
Η διαφοροποίηση του SPI σε σχέση με τα παραδοσιακά δέντρα ευρετηρίασης ως προς τον τρόπο προσπέλασης στους κόμβους του - εισαγωγές (insertions) από πάνω προς τα κάτω (top – down), ενημερώσεις (updates) και αφαιρέσεις (deletions) από κάτω προς τα πάνω (bottom – up) - καθιστά αναγκαία την προσαρμογή υπαρχουσών πρωτοκόλλων κλειδωμάτων. Στη συνέχεια παρατίθεται το πρωτόκολλο κλειδωμάτων που χρησιμοποιείται με τις αποδείξεις για την ορθή λειτουργία του και ότι δεν οδηγεί σε αδιέξοδα (deadlock – free). Το πρωτόκολλο κατατάσσεται στα αισιόδοξα πρωτόκολλα (optimistic protocols) και έχει κάποιες θεωρητικές αδυναμίες (δεν εξασφαλίζει την αποφυγή λιμοκτονίας) αλλά αποδεικνύεται ιδιαίτερα λειτουργικό στην πράξη.

Σημειώνουμε πως όλοι οι αλγόριθμοι κλειδωμάτων που ακολουθούν αφορούν μόνο το Quad Tree από το δέντρο του SPI. Οι προσπελάσεις στο NFA του YFilter είναι σε κάθε περίπτωση μεμονωμένες κλειδώνοντας όλο το NFA. Διαφορετική προσέγγιση θα απαιτούσε την υλοποίηση του YFilter εξ αρχής χωρίς έμπρακτο αποτέλεσμα λόγω της πολυπλοκότητας της δομής του.

3.5.1 Ασφαλείς (safe) και μη ασφαλείς (unsafe) κόμβοι

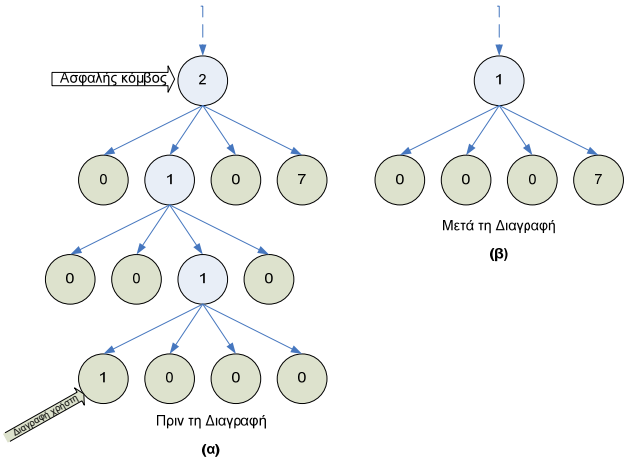
Πριν την παρουσίαση του πρωτοκόλλων εστιάζουμε στις προσπελάσεις στο δέντρο που είναι κρίσιμες για το δομικό του μετασχηματισμό. Κατά την εισαγωγή ενός νέου χρήστη μπορεί να διασπαστεί ένα φύλλο σε τέσσερα νέα και να αποκτήσει τη θέση ενός ενδιάμεσου κόμβου αν το πλήθος των χρηστών σε αυτό είναι ίσο με το μέγεθος του κάδου εναπόθεσης (bucket). Παρατη-

ρούμε ότι η εισαγωγή δεν μπορεί σε καμία περίπτωση να μεταβάλλει τη μορφή του δέντρου παρά μόνο από το φύλλο που γίνεται η εισαγωγή και κάτω (βλ. Σχήμα 55)



Σχήμα 55: Εισαγωγή χρήστη στο SPI tree με διάσπαση φύλλου

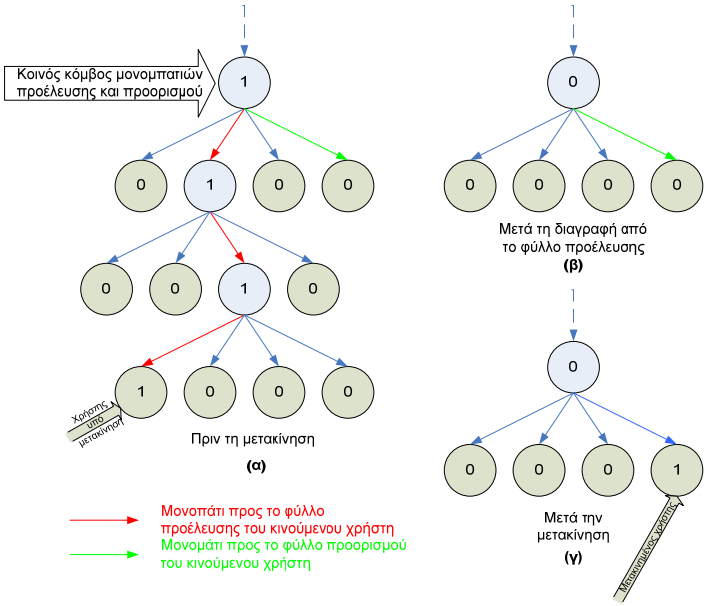
Στην αφαίρεση ενός στοιχείου αντίθετα η σύμπτυξη των κόμβων μπορεί γίνει σε αρκετά επίπεδα πιο πάνω από το επίπεδο του φύλλου που βρίσκεται ο υπό διαγραφή χρήστες. Σε αυτήν την περίπτωση ασφαλής κόμβος, κόμβος πέρα από τον οποίο δεν μπορεί να συνεχιστεί η σύμπτυξη, είναι ένα ενδιάμεσος κόμβος που έχει περισσότερα του ενός κατειλημμένα παιδιά (βλ. Σχήμα 56). Η αφαίρεση προφανώς μπορεί να γίνει επίσης χωρίς να προκαλέσει καμία αλλαγή στη δομή του δέντρου. Αυτό συμβαίνει στις περιπτώσεις που το φύλλο από το οποίο γίνεται η αφαίρεση έχει περισσότερες της μίας εγγραφές ή ο «πατέρας» του φύλλου από το οποίο γίνεται η διαγραφή είναι ασφαλής (έχει περισσότερα του ενός κατειλημμένα παιδιά – φύλλα).



Σχήμα 56: Ασφαλής κόμβος κατά τη διαγραφή χρήστη

Παρόλο που όλες οι τροποποιήσεις στο δέντρο ευρετηρίασης μπορούν να αναχθούν σε προσθέσεις και αφαιρέσεις θα κάνουμε σε αυτό το σημείο ειδική μνεία για την πράξη της μετακίνησης μίας εγγραφής στο δέντρο, καθώς αυτή εμφανίζεται κατά κόρον στις εφαρμογές για τις οποίες

προορίζεται το ευρετήριο μας. Στις περιπτώσεις λοιπόν που η αλλαγή της θέσης ενός χρήστη συνεπάγεται αλλαγή της θέσης του στο δέντρο του ευρετηρίου, η γνώση του κόμβου προορισμού μας επιτρέπει να θεωρήσουμε ως ασφαλή κόμβο το κοινό κόμβο που βρίσκεται στο χαμηλότερο επίπεδο μεταξύ του μονοπατιού από τον κόμβο που τώρα βρίσκεται ο χρήστης προς τη ρίζα και του μονοπατιού από τον κόμβο στον οποίο θα μετακινηθεί ο χρήστης μέχρι τη ρίζα (Σχήμα 57). Αν και τα κατειλημμένα παιδιά αυτού κόμβου μπορούν προσωρινά να μηδενιστούν (Σχήμα 57β), κάτι που σε διαφορετική περίπτωση θα σήμαινε σύμπτυξη του κόμβου, ο κόμβος διατηρείται αφού ξέρουμε με βεβαιότητα πως στο επόμενο στάδιο θα εισαχθεί μία εγγραφή σε ένα από τα παιδιά του (Σχήμα 57γ).



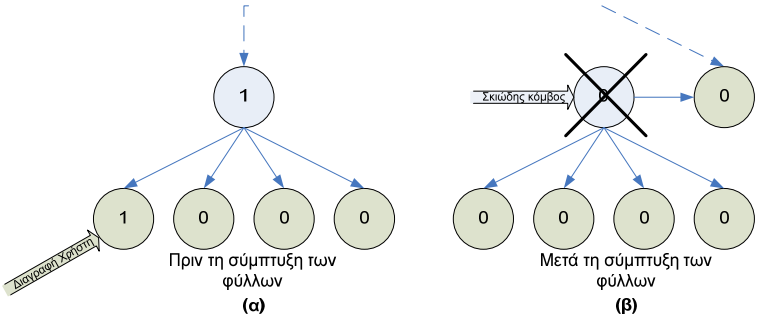
Σχήμα 57: Ασφαλής κόμβος κατά τη μετακίνηση – ενημέρωση ενός χρήστη

3.5.2 Πρωτόκολλο κλειδωμάτων

Το πρωτόκολλο που θα παρουσιαστεί για το Quad-Tree βασίζεται σε πρωτόκολλα κλειδωμάτων με οπισθοχώρηση (backtracking) για Β δέντρα που προσαρμόστηκαν κατάλληλα για τις δομές που χρησιμοποιούμε.

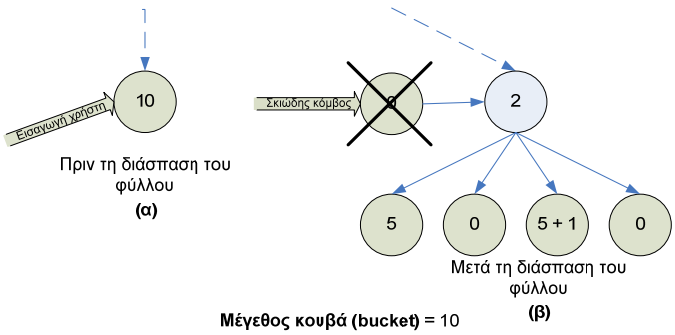
Χρησιμοποιούμε δύο τύπους κλειδωμάτων: το κλείδωμα αποκλειστικής πρόσβασης (*eXclusive lock* ή *x-lock* ή απλά *lock*) και το κλείδωμα μοιραζόμενης πρόσβασης (Shared lock ή s-lock). Στη θέση των κλειδωμάτων προθέσεων (*ix*, *is-lock*) το πρωτόκολλο μας δεν χρησιμοποιεί κλειδώματα. Η απαλοιφή αυτών των τύπων κλειδωμάτων δεν θα μπορούσε να επιτευχθεί χωρίς την εισαγωγή της έννοιας του σκιώδους κόμβου (*zombie node*). Οι σκιώδεις κόμβοι είναι αυτοί που προκύπτουν κατά την διάσπαση ενός φύλλου ή τη σύμπτυξη ενός εσωτερικού κόμβου. Στην πρώτη περίπτωση το υπό διάσπαση φύλλο μετατρέπεται σε σκιώδη κόμβο και αποκτά ένα δείκτη προς τον εσωτερικό κόμβο που το αντικαθιστά. Στη δεύτερη περίπτωση ο εσωτερικός κόμβος μετατρέπεται σε σκιώδη και δεικτοδοτεί το φύλλο που τον αντικατέστησε. Οι δείκτες χρη-

σιμοποιούνται για την ανακατεύθυνση της διάσχισης του δέντρου είτε αυτή γίνεται από πάνω προς τα κάτω είτε από κάτω προς τα πάνω σε περίπτωση που η διάσχιση – λόγω της έλλειψης κλειδωμάτων – έχει οδηγηθεί σε κάποιον σκιώδη κόμβο. Παραδείγματα σκιωδών κόμβων φαίνονται στο Σχήμα 58 και στο Σχήμα 59.



Σχήμα 58: Σκιώδης κόμβος κατά τη διαγραφή χρήστη

Παρακάτω δίνονται οι αλγόριθμοι για την εισαγωγή, αφαίρεση, μετακίνηση χρήστη και ανάγνωση του δέντρου. Με μπλε γράμματα σημειώνονται οι προσθήκες στους αλγορίθμους που δόθηκαν και παραπάνω λόγω της εφαρμογής του πρωτοκόλλου κλειδωμάτων.



Σχήμα 59: Σκιώδης κόμβος κατά την εισαγωγή χρήστη

3.5.2.1 Εισαγωγή χρήστη

Κατά την εισαγωγή ενός χρήστη ο αλγόριθμος κλειδωμάτων έχει σκοπό να εξασφαλίσει την διατήρηση του μονοπατιού που διασχίστηκε από τη ρίζα μέχρι τον κόμβο-φύλλο στον οποίο θα γίνει η εισαγωγή. Σε διαφορετική περίπτωση ο κόμβος που θα εντοπιζόταν για να γίνει η εισαγωγή θα μπορούσε να μείνει ορφανός λόγω σύμπτυξης του δέντρου σε ανώτερα επίπεδα και η εισαγωγή να είναι τελικά τυφλή. Για αυτό το λόγο το πρωτόκολλο προχωρά με διαδοχικά κλειδώματα (lock-coupling) αφήνοντας το κλείδωμα ενός κόμβου μόνο αν έχει πάρει το κλείδωμα του παιδιού του. Όταν η διάσχιση μας οδηγήσει στο φύλλο που μας ενδιαφέρει ο πατέρας διατηρείται κλειδωμένος και γίνεται η εισαγωγή.

Στην περίπτωση διάσπασης ενός κόμβου, διατηρείται κλειδωμένος τόσο ο υπό διάσπαση κόμβος (σκιώδης κόμβος) όσο και ο κόμβος που τον αντικαθιστά. Έτσι παρεμποδίζεται προσωρινά

και η ενημέρωση των εγγραφών-χρηστών που δεν έχουν μεταφερθεί οριστικά στη νέα τους έγκυρη θέση στο δέντρο.

Η διάσχιση του μονοπατιού μπορεί να οδηγήσει σε σκιώδη κόμβο . Αυτό μπορεί να γίνει μόνο σε περίπτωση σύμπτυξης του δέντρου λόγω διαγραφής, αφού σε περίπτωση διάσπασης κόμβου το κλείδωμα της διαδικασίας εισαγωγής που προκάλεσε τη διάσπαση στον πατέρα του φύλλου που διασπάται θα κρατούσε όλες τις άλλες διαδικασίες εισαγωγής ανεπηρέαστες από τη διάσπαση (είναι αναγκασμένες να περιμένουν να ολοκληρωθεί). Έτσι αν βρεθεί σκιώδης κόμβος διατηρείται το κλείδωμα στον πατέρα, στο σκιώδη κόμβο και κλειδώνεται και ο κόμβος-φύλλο που αντικατέστησε τον κόμβο και στον οποίο θα γίνει η εισαγωγή.

```
// Insert a user into an internal node. Internal node cannot be a
// zombie node
InternalNode.Insert(User user) {

   // user is out of indexing bounds of node
   if (! this.contains(user.longitude, user.latitude) )
     return;

   Node current, child, zombie := null;
   current := this;
   current.lock();

   // find the appropriate leaf
   repeat{
     child = current.getChildContaining(user.longitude, user.latitude);
     child.lock();
     if (child.isZombieNode){
       zombie := child;
       child := zombie.replacingNode;
       child.lock();
     }
     else if(child is Internal Node){
       current.unlock();
       current := child;
     }
   }
   until (child is Leaf);

   child.Insert(user);

   child.unlock();
   if (zombie is not null)
     zombie.unlock();
   current.unlock();
}

// Insert directly into a leaf node
Leaf.Insert(User user) {

   if (Leaf is not full || depth > MAX_HEIGHT) {
     // Leaf is safe. No need to split

     if(Leaf is empty) {
       // Initializations needed for the first record of the leaf
       fatherNode.incrOccupiedChildren();
```

```
        nodeFilter := new YfilterNFA();
      }

      user.fatherLeaf := leafNode;
      nodeFilter.addProfile(user.profile);
    }
    else {
      // Unsafe Leaf. Leaf splitting required

      // Init replacing node. It has the same father and range
      // with the leaf that it going to replace. 4 children leaf
      // modes are initialized too.
      newNode := new IndexInteranlNode(fatherNode, range;);
      newNode.noOfOccupiedChildren := 0;
      newNode.lock();

      replacingNode = newNode;
      isZombieNode = true;


      fatherNode.setChild(replacingNode);
      // Redistribute user records of zombie leaf
      for (all existing users in leafNode)
        newNode.Insert(currentUser);   // newNode is already locked.
                                       // InternalNode.Insert needs some
                                       // trivial modifications to avoid
                                       // deadlock

      // Insert initial user
      newNode.Insert(user);
    }
}
```

3.5.2.2 *Αφαίρεση χρήστη*

Κατά την αφαίρεση ενός χρήστη, η ίδια η ύπαρξη του χρήστη μας βεβαιώνει για την διατήρηση του μονοπατιού από τον κόμβο του χρήστη μέχρι τη ρίζα. Η ύπαρξη έστω και ενός χρήστη δεν επιτρέπει την σύμπτυξη του κόμβου αυτού. Συνεπώς αρκεί μόνο το κλείδωμα του φύλλου που περιέχει το χρήστη.

Πρόνοια λαμβάνεται μόνο για την περίπτωση που ο κόμβος-φύλλο που περιείχε το χρήστη διασπάστηκε μέχρι τη λήψη του κλειδιού του. Σε αυτήν την περίπτωση η διαγραφή του χρήστη καλείται εκ νέου, ώστε η διαδικασία διαγραφής να ανακατευθυνθεί στο νέο φύλλο που περιέχει το χρήστη.

Στην περίπτωση που ο κόμβος-φύλλο εκκενώνεται με τη διαγραφή του χρήστη ενημερώνεται ο πατέρας. Αν ο πατέρας μείνει χωρίς κατειλημμένα παιδιά συμπτύσσεται κ.ο.κ.. Για τη διαδικασία αυτή της οπισθοδρόμησης (backtracking) κλειδώνεται κάθε φορά μόνο ο κόμβος που πρόκειται να ενημερωθεί (σε διαφορετική περίπτωση το κλείδωμα από πάνω προς τα κάτω θα μπορούσε να οδηγήσει σε αδιέξοδα συνδυαζόμενο με τον αλγόριθμο εισαγωγής). Το γεγονός αυτό μπλοκάρει τις διαδικασίες εισαγωγής που επιδιώκουν να κλειδώσουν το κόμβο που συμπτύσσε-

ται ενώ θέτουν σε σειρά τις προσπελάσεις στους ενδιάμεσους κόμβους για ενημέρωση του αριθμού των κατειλημμένων παιδιών ώστε να διατηρείται η συνέπεια του δέντρου.

```
Delete (User user) {
  // Find leaf that contains the user
  leaf := user.fatherLeaf;

  leaf.lock();
  // Leaf may become zombie until this process locks it. User
  // will have moved to new Leaf when this zombie leaf is locked
  if (leaf.isZombieLeaf){
    leaf.unlock();
    Delete(user);
    return;
  }

  // delete from leaf
  leaf.nodeFilter.deleteProfile(user.profile);
  user.fatherLeaf := null;

  // If Leaf is empty, inform father that this Leaf is no
  // more occupied
  if (Leaf is empty now) {
    leaf.nodeFilter := null;
    leaf.unlock();
    leaf.fatherNode.DecrOccupiedChildren();
  }
  else
    leaf.unlock();
}

InternalNode.DecrOccupiedChildren() {
  lock();
  noOfOccupiedChildren--;
  boolean fatherNeedsUpdate := false;
  if (noOfOccupiedChildren == 0 and depth > 0) {

    // Init replacing leaf. It has the same father and range
    // with the leaf that it is going to replace
    newNode := new Leaf(father, range);

    replacingNode := newNode;
    isZombieNode := true;

    // Set father's new child
    fatherNode.setChild(replacingNode);
    fatherNeedsUpdate := true;
  }
  unlock();
```

```
    // Perform backtracking if needed
    if (fatherNeedsUpdate)
      fatherNode.DecrOccupiedChildren();
}
```

3.5.2.3 Ενημέρωση χρήστη

Η ενημέρωση χρήστη γίνεται με τη χρήση της αφαίρεσης και της πρόσθεσης. Όμως, για τη βελτίωση της αποδοτικότητας της ενημέρωσης

- η πρόσθεση του χρήστη δεν γίνεται ξεκινώντας από τη ρίζα και
- η ενδεχόμενη σύμπτυξη του δέντρου λόγω της αφαίρεσης ενός χρήστη δεν αφήνεται να προχωρήσει στον ασφαλή κόμβο της ενημέρωσης.

Λαμβάνοντας υπόψη την παρατήρηση που έγινε στην 3.5.1 κλειδώνεται ο ασφαλής κόμβος της πράξης της μετακίνησης. Με αυτόν τον τρόπο εμποδίζεται και η οπισθοδρόμηση (backtracking) στη σύμπτυξη αλλά εντοπίζεται και ο κοντινότερος ενδιάμεσος κόμβος στον τρέχων φύλλο του χρήστη από τον οποίο μπορεί να ξεκινήσει η εισαγωγή με κατάληξη το νέο φύλλο.

Ο κοινός ενδιάμεσος κόμβος που εντοπίζεται αρχικά μπορεί μέχρι το κλείδωμα του να έχει μετατραπεί σε σκιώδη λόγω της διάσπασης του (αποκλείεται η σύμπτυξη με ανάλογο σκεπτικό που αναφέρθηκε στην αφαίρεση). Σε αυτήν την περίπτωση κλειδώνεται και ο κόμβος που τον αντικαθιστά και η διαδικασία ενημέρωσης συνεχίζεται κανονικά.

```
Update (User user, real newLongitude, real newLatitude) {
    // Find the common node of the lowest level of the following paths:
    // 1. from source leaf to root
    // 2. from destination leaf to root
    commonNode = user.fatherLeaf.getCommonNode(newLongitude, newLatitude);

    if (commonNode == null){
      Error("User moved out of indexing bounds");
      return;
    }

    commonNode.lock();
    if (commonNode == user.fatherLeaf){
      // User moved within the bounds of the containing leaf.
      // Just update user's coordinates
      user.longitude := newLongitude;
      user.latitude := newLatitude;
      commonNode.unlock();
      return;
    }

    if (commonNode.isZombieNode){
```

```
      zombieNode := commonNode;
      commonNode := commonNode.replacingNode;
      commonNode.lock();
      zombieNode.unlock();

      commonNode = commonNode.getDescedantCommonNode(user.longitude,
                           user.latitude, newLongitude, newLatitude)
    }

    Couple lock nodes until source Node and keep Common Node locked;
    Delete(user);                  //and unlock source Node

    user.longitude := newLongitude;
    user.latitude := newLatitude;
    // Insert: Lock coupling until destination Node (probably unlock
    // common Node and after insertion release all lockings
    commonNode.Insert(user);
}

Node.getCommonNode(real longitude, real latitude) {
    if (Node.contains(longitude, latitude))
      // Node found
      return Node;

    if (Node.isZombieNode)
      return Node.replacingNode;

    if (Node is root)
      // Coordinates are out of indexing bounds
      return null;
    return Node.fatherNode.getCommonNode(longitude, latitude);
}
```

3.5.2.4 *Προσπελάσεις για ανάγνωση*

Ο αλγόριθμος κλειδωμάτων για αναγνώσεις από το ευρετήριο σχεδιάστηκε με γνώμονα την απόδοσή του ώστε να μεγιστοποιείται ο αριθμός των ερωτημάτων που μπορεί αν εξυπηρετήσει το σύστημα στη μονάδα του χρόνου. Η πράξη της ανάγνωσης είναι η μοναδική στην οποία χρησιμοποιείται το κλείδωμα μοιραζόμενης πρόσβασης (s-lock).

Η γενική ιδέα του αλγορίθμου είναι η εξής: η διάσχιση του δέντρου ξεκινά από τη ρίζα με κλειδώματα ανά ζεύγη και κατεβαίνει στα χαμηλότερα επίπεδα του δέντρου μέχρι τα φύλλα. Στις περιπτώσεις που κάποιος κόμβος κατά τη διαδρομή είναι σκιώδης η διάσχιση ανακατευθύνεται στον κόμβο που τον αντικαθιστά.

Όταν η διάσχιση του δέντρου φτάσει στα φύλλα τότε διαβάζονται τα δεδομένα που αναζητούσαμε και ελευθερώνεται το κλείδωμα του κόμβου.

```
// When entering this method Node is supposed to be s-locked
Node.Read (info about data to read) {

   if (current.isZombieNode){
        replacingNode.slock();
        current.unlock();
        replacingNode.Read(info about data to read);
        return;
   }

   if (Node is Internal Node){
      for (every child of Node matching info about data to read)
        child.slock();
      Node.unlock();
      for (every child of Node matching info about data to read)
        child.Read(info about data to read);
      return;
   }

   if (Node is Leaf)
      Read information from node;
      current.unlock();
   }
}
```

3.5.3 *Απόδειξη ορθότητας πρωτοκόλλου κλειδωμάτων*

Για την απόδειξη της ορθότητας του πρωτοκόλλου κλειδωμάτων θα πρέπει να αποδειχθεί ότι οι παρακάτω δύο προϋποθέσεις ισχύουν για κάθε διεργασία:

1. δεν οδηγείται σε αδιέξοδο (deadlock)
2. έχει εκτελέσει σωστά την επιθυμητή λειτουργία όταν τερματίζει. Ιδιαίτερα:
   a. ότι όλες οι πράξεις που τροποποιούν το δέντρο του ευρετηρίου το διατηρούν σε συνεπή μορφή
   b. όλες οι άλλες διεργασίες πλην αυτής που το τροποποιεί βλέπουν το δέντρο σε συνεπή μορφή

3.5.3.1 *Αποφυγή αδιεξόδων*

Για την απόδειξη πως το πρωτόκολλο κλειδωμάτων που ορίσαμε παραπάνω δεν οδηγεί σε αδιέξοδα θα πρέπει αρχικά να ορίσουμε μία σχέση διάταξης μεταξύ των κόμβων του δέντρου.

Θεωρούμε την εξής σχέση μερικής διάταξης (<) στο σύνολο των κόμβων του δέντρου:

(1) Για δύο κόμβους α, β του δέντρου λέμε «α < β» αν και μόνον αν ο κόμβος β είναι πρόγονος του α.

(2) Καταχρηστικά, κατά τη διάρκεια ύπαρξης ενός σκιώδους κόμβου, οπότε το δέντρο μετά την σύνδεση του πατέρα με το νέο κόμβο είναι κατ' ουσία γράφος, θα λέμε «α < α_{repl}», όπου α ο κόμβος και α_{repl} ο νέος κόμβος.

Γενικώς δεν ορίζεται διάταξη με μεταξύ δύο μη συγγενών κόμβων.

Για την απόδειξη πως το πρωτόκολλο κλειδωμάτων που χρησιμοποιούμε δεν οδηγεί σε αδιέξοδα θα πρέπει αφενός να δείξουμε πως οι κόμβοι του δέντρου παραμένουν σε κάθε περίπτωση καλώς διατεταγμένοι και πως οι πράξεις προσπέλασης στο δέντρο κλειδώνουν τους κόμβους του ακολουθώντας την διάταξη των κόμβων.

Λήμμα 1. *Αν τη χρονική στιγμή t_0 για δύο κόμβους α, β ισχύει α < β, τότε θα ισχύει α < β για κάθε χρονική στιγμή $t > t_0$, εφόσον α, β ∈ στο δέντρο τη στιγμή αυτή t.*

Τη χρονική στιγμή t_0 ισχύει $\alpha < \beta$. Οι πράξεις που μπορούν να έχουν μεταβάλει τη δομή του δέντρου το διάστημα $[t_0, t)$ είναι η διάσπαση ενός κόμβου (node splitting) ύστερα από την πρόσθεση μίας εγγραφής σε ένα γεμάτο φύλλο ή η σύμπτυξη κόμβων (node collapsing) ύστερα από την αφαίρεση της τελευταίας εγγραφής από τα παιδιά-φύλλα ενός εσωτερικού κόμβου. Κατά την διάσπαση ενός κόμβου α προκύπτουν οι κόμβοι $\alpha_{repl}, \alpha_1, \alpha_2, \alpha_3, \alpha_4$ για τους οποίους έχουμε:

$$a_1, \ldots, a_4 < a < a_{repl}$$

Από τα παραπάνω και τον τρόπο που γίνεται η διάσπαση θα ισχύει:

$$\forall_\beta \beta > \alpha \Leftrightarrow \beta > \alpha_{repl}, \alpha_1, \ldots, \alpha_4$$

Σημειωτέον πως δεν υπάρχει κόμβος β τέτοιος ώστε $\beta < \alpha$ αφού ο α πρέπει να είναι φύλλο για να διασπαστεί.

Κατά τη σύμπτυξη κόμβων αν ο α δεν είναι ένας από τους κόμβους που συμπτύσσεται ή ο πατέρας του τότε προφανώς δεν επηρεάζεται η σχέση διάταξης σε σχέση με οποιοδήποτε κόμβο β. Αν ο α είναι ένας από τους κόμβους που συμπτύσσεται τότε μετά τη σύμπτυξη δεν θα ανήκει στο δέντρο. Τέλος αν α είναι ο πατέρας των κόμβων που συμπτύσσεται τότε θα ισχύει σε σχέση με τον κόμβο α_{repl} που τον αντικαθιστά:

$$\alpha > \alpha_{repl}$$

και συνεπώς

$$\forall_\beta \beta > \alpha \Leftrightarrow \beta > \alpha_{repl}$$

Συνεπώς καμία προσπέλαση ή τροποποίηση στο δέντρο του ευρετηρίου δεν μεταβάλλει τη διάταξη των κόμβων του. ■

Λήμμα 2. *Μία διεργασία που κατέχει το κλειδί ενός κόμβου α δεν διεκδικεί ποτέ το κλείδωμα ενός κόμβου β με β >α (ή του α).*

Για την πράξη της αφαίρεσης ενός χρήστη από το δέντρο το παραπάνω είναι προφανές αφού σε καμία περίπτωση η διεργασία που εκτελεί τη διαγραφή δεν κατέχει περισσότερα του ενός κλειδιά κόμβων.

Οι πράξεις της εισαγωγής και ανάγνωσης κλειδώνουν τους κόμβους του δέντρου από πάνω προς τα κάτω και δεν αφήνουν το κλείδωμα ενός κόμβου αν δεν πάρουν το κλειδί του παιδιού του. Σε κάθε περίπτωση η σειρά με την οποία διεκδικούνται τα κλειδώματα από τους κόμβους ακολουθεί τη διάταξη των κόμβων.

Τέλος στην ενημέρωση της εγγραφή ενός χρήστη κλειδώνεται αρχικά ο ασφαλής κόμβος της ενημέρωσης. Στη συνέχεια ακολουθείται διαδικασία κλειδωμάτων ανάλογης της εισαγωγής μέχρι το κόμβο που θα γίνει η διαγραφή και η εισαγωγή του χρήστη. Σημειώνεται ότι τα δύο μονοπάτια είναι διακριτά οπότε δεν υπάρχει διάταξη μεταξύ των κόμβων τους. Η διαγραφή γίνεται χωρίς να εκτελεστεί οπισθοδρόμηση ώστε να μην επιχειρηθεί κλείδωμα του ασφαλούς κόμβου. Τέλος γίνεται η εισαγωγή, απελευθερώνονται όλα τα κλειδώματα και ξεκινά η οπισθοδρόμηση για τη σύμπτυξη κόμβων λόγω της διαγραφής. ■

Από τα παραπάνω λήμματα προκύπτει πως όλα τα κλειδώματα που παίρνει και κρατά μία διεργασία είναι καλώς διατεταγμένα σε ότι αφορά στους κόμβους που αντιστοιχούν. Έτσι καταλήγουμε τελικά στο θεώρημα:

Θεώρημα 1: *Το πρωτόκολλο κλειδωμάτων του SPI δέντρου δεν οδηγεί σε αδιέξοδα.*

3.5.3.2 Συνέπεια των πράξεων τροποποίησης του δέντρου

Για να αποδείξουμε πως η δομή του δέντρου διατηρείται πάντα σε συνεπή μορφή θα πρέπει αρχικά να διασφαλίσουμε πως όλες οι συναρτήσεις τροποποίησης του δέντρου, όταν αυτές εκτελούνται μεμονωμένα, και παίρνουν ως είσοδο το δέντρο σε συνεπή μορφή το επιστρέφουν πάλι σε συνεπή μορφή. Παρατίθεται το εξής θεώρημα χωρίς απόδειξη:

Θεώρημα 2: *Όλες οι συναρτήσεις τροποποίησης του δέντρου διατηρούν το δέντρο σε συνεπή μορφή.*

Η απόδειξη του θεωρήματος προκύπτει εύκολα από την περιγραφή της υλοποίησης των συναρτήσεων τροποποίησης του δέντρου. Ο σχεδιασμός τους είναι τέτοιος ώστε να διατηρείται ύστερα από την από την ολοκλήρωση της εκτέλεσης τους η συνέπεια του δέντρου που συνίσταται στα εξής: το ευρετήριο παραμένει δέντρο, κάθε χρήστης ανήκει στο σωστό φύλλο, οι εσωτερικοί κόμβοι και τα φύλλα διατηρούν τη σωστή πληροφορία για τον αριθμό των κατειλημμένων παιδιών και τον αριθμό των χρηστών αντίστοιχα.

3.5.3.3 Απομόνωση πράξεων προσπέλασης στο δέντρο

Για την πληρότητα της απόδειξης της ορθότητας του πρωτοκόλλου κλειδωμάτων θα πρέπει να αποδειχτεί πως κάθε πράξη προσπέλασης δεν «βλέπει» τις τροποποιήσεις που κάνουν άλλες πράξεις προσπέλασης στο δέντρο που εκτελούνται την ίδια χρονική στιγμή. Πλήρης απόδειξη της απομόνωσης των πράξεων απαιτεί μελέτη όλων των δυνατών ζευγαριών των πράξεων όταν εκτελούνται ταυτόχρονα σε κοινά ή μη κοινά δεδομένα – όταν δεν εκτελούνται ταυτόχρονα μας καλύπτει το Θεώρημα 2. Σε αυτήν την εργασία θα εστιάσουμε μόνο στην πιο σύνθετη απόδειξη η οποία είναι της απομόνωση μεταξύ μίας πράξης ανάγνωσης και ενημέρωσης.

Λήμμα 3. *Μία δοσοληψία που διαβάζει δεδομένα με χρήση του ευρετηρίου και εκτελείται ταυτόχρονα με μία πράξη ενημέρωσης δεν βλέπει τα ενδιάμεσα αποτελέσματα της ενημέρωσης.*

Θα διακρίνουμε τις εξής περιπτώσεις:

- Η διάσχιση για τα φύλλα στα οποία θα γίνει η ανάγνωση δεν περνά από τον ασφαλή κόμβο της ενημέρωσης. Σε αυτήν την περίπτωση δεν υπάρχει πρόβλημα καθώς οι δύο πράξεις ενεργούν σε ξένα σύνολα δεδομένων.
- Η ανάγνωση αφορά μόνο ένα από τα δύο φύλλα (φύλλο προέλευσης ή προορισμού) τα οποία τροποποιεί η ενημέρωση. Και σε αυτήν την περίπτωση μας είναι αδιάφορο αν θα γίνει πρώτα η ανάγνωση ή η τροποποίηση αφού η ανάγνωση θα μπορεί εύκολα να σειριοποιηθεί με την τροποποίηση με κριτήριο τη σειρά με την οποία προσπέλασαν το συγκεκριμένο φύλλο.
- Η ανάγνωση γίνεται και στα δύο φύλλα τα οποία τροποποιεί η ενημέρωση. Εδώ διακρίνουμε δύο υποπεριπτώσεις. Στην πρώτη η πράξη της ενημέρωσης παίρνει το αποκλειστικό κλείδωμα στον κοινό κόμβο πριν το κλείδωμα της ανάγνωσης. Έτσι αναστέλλεται προσωρινά η ανάγνωση, η τροποποίηση παίρνει το κλείδωμα στον κόμβο διαγραφής (προέλευσης) και στη συνέχεια αφήνει την ανάγνωση να προσπεράσει τον ενδιάμεσο κόμβο ενώ αυτή προπορευόμενη κλειδώνει το και φύλλο εισαγωγής (προορισμού). Η ανάγνωση προσπελαύνει τα δύο φύλλα αφού έχουν τροποποιηθεί. Στη δεύτερη περίπτωση η πράξη της ανάγνωσης παίρνει πρώτη το κλείδωμα του κοινού κόμβου. Το κρατά μέχρι να αποκτήσει το κλείδωμα σε όλα τα παιδιά του τα οποία θα διασχίσει. Έτσι η τροποποίηση και στους δύο κόμβους θα έπεται πάντα της ανάγνωσης.

Σε κάθε περίπτωση οι δύο πράξεις εκτελούνται απομονωμένες. ■

Με τη βοήθεια και άλλων ανάλογων λημμάτων αποδεικνύεται τελικά το θεώρημα:

Θεώρημα 3: *Με τη χρήση του πρωτοκόλλου κλειδωμάτων του δέντρου SPI, όλες οι πράξεις προσπέλασης σε αυτό εκτελούνται απομονωμένα.*

Με τα Θεωρήματα 1, 2, 3 ολοκληρώνεται η απόδειξη της ορθότητας του πρωτοκόλλου κλειδωμάτων του SPI.

4 *Υλοποίηση Συστήματος*

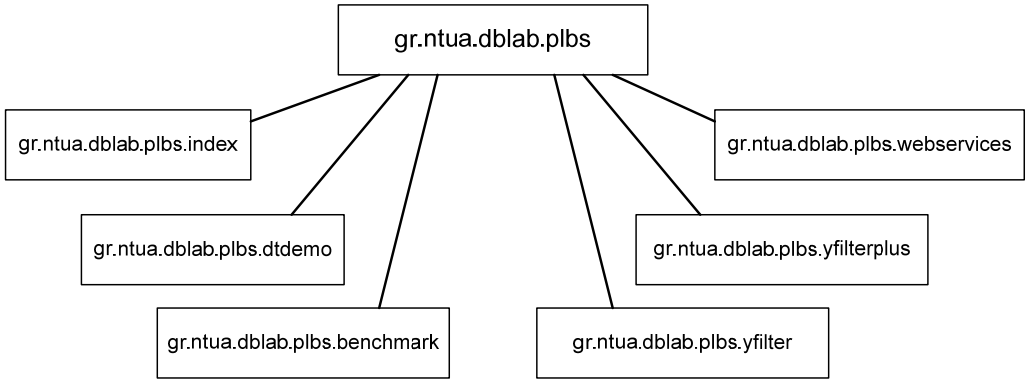
Το σύστημα υλοποιήθηκε ως αυτόνομη εφαρμογή σε γλώσσα προγραμματισμού JAVA Enterprise Edition v 5.0. Αναπτύχθηκαν τόσο το βασικό ευρετήριο του συστήματος και οι αλγόριθμοι ευρετηρίασης και αποτίμησης των ερωτημάτων, όσο και οι προβλεπόμενες από τη σχεδίαση Υπηρεσίες Ιστού (Web Services) για τη χρήση των υπηρεσιών του συστήματος και την κλήση των μεθόδων του. Σημειώνεται ότι για την υλοποίηση των NFA των φύλλων του SPI Tree χρησιμοποιήθηκε ο κώδικας για το YFilter του Πανεπιστημίου UC Berkeley, ο οποίος τροποποιήθηκε όμως εκτεταμένα για την αποδοτική ενσωμάτωσή του στην υπόλοιπη δομή και τη βέλτιστη λειτουργία του ως ευρετήριο προφίλ στα φύλλα του SPI tree.

Η χρήση της πλατφόρμας JAVA Enterprise Edition v 5.0 έγινε παρά τις εγγενείς της αδυναμίες στην άμεση διαχείριση μνήμης (έλλειψη δεικτών) που είναι σημαντική για την ανάπτυξη μιας δομής ευρετηρίου, αφού έτσι μπορούσε να χρησιμοποιηθεί μεγάλο μέρος του κώδικα του YFilter.

Ειδικά για την επίδειξη της λειτουργικότητας του SPI tree αναπτύχθηκε ένα μία εφαρμογή με γραφικό περιβάλλον στο οποίο ο χρήστης μπορεί να έχει άμεση εποπτεία της ευρετηρίασης του SPI tree και απεικόνιση των αποτελεσμάτων ερωτημάτων περιοχής και εγγύτερης γειτονίας. Επίσης, υλοποιήθηκαν κλάσεις για την πειραματική δοκιμή της νέας δομής ευρετηρίου, όπως περιγράφεται στο κεφάλαιο που ακολουθεί, και για τη μέτρηση των επιδόσεών του υπό διάφορες συνθήκες λειτουργίας.

4.1 Δομή Κώδικα

Οι κλάσεις που κατασκευάστηκαν περιέχονται στο πακέτο (*package*) *gr.ntua.dblab.plbs*. Τα περιεχόμενα του πακέτου αυτού φαίνονται στο Σχήμα 60.



Σχήμα 60: Απεικόνιση των πακέτων της υλοποίησης

Στα πακέτα index, webservices, yfilter, yfilterplus υλοποιείται η λειτουργικότητα του συστήματος. Στο πακέτο dtdemo περιέχονται όλες οι κλάσεις που αφορούν στην ανάπτυξης της εφαρμογής επίδειξης του SPI tree (Παράγραφος 4.3) και στο πακέτο benchmark περιέχονται οι κλάσεις για τον έλεγχο της λειτουργίας του συστήματος προσομοιώνοντας υποθετικές συνθήκες λειτουργίας. Η παρουσίαση αυτού του πακέτου αφήνεται για το Κεφάλαιο 5.

Τα πακέτα yfilterplus και yfilter έχουν προκύψει από την υλοποίηση του YFilter που είναι ανοιχτή στο κοινό (yfilter.cs.berkeley.edu). Δε θα δώσουμε περισσότερες λεπτομέρειες για την υλοποίηση τους καθώς για τεκμηρίωση μπορεί να ανατρέξει κανείς http://yfilter.cs.berkeley.edu/html/manual/YFilter_User_Manual.html. Στον κώδικα των κλάσεων συνοδεύουμε με σχόλια όλες τις αλλαγές που πραγματοποιήσαμε για να προσαρμόσουμε τη λειτουργικότητα του YFilter στο σύστημά μας.

Στο πακέτο index υλοποιείται το SPI tree που είναι η κύρια δομή στην οποία βασίζεται η λειτουργία του κεντρικού συστήματος. Για την περιγραφή αυτού του πακέτου αφιερώνεται η Παράγραφος 4.2.1.

Τέλος, στο πακέτο webservices υλοποιείται το κομμάτι του συστήματος που αφορά την εξαγωγή των Υπηρεσιών του συστήματος στον Ιστό και την επικοινωνία της δομής κύριας μνήμης με μία παραδοσιακό ΣΔΒΔ και αναλύεται στην Παράγραφο 4.2.2.

4.2 Υλοποίηση Συστήματος

Στην παράγραφο αυτή παρουσιάζονται τα κύρια σημεία του κώδικα της εφαρμογής καθώς και ο τρόπος με τον οποίο έχουν υλοποιηθεί τα κυριότερα σημεία του συστήματος.

4.2.1 Υλοποίση SPI tree

Το βασικότερο πακέτο της υλοποίησης της νέας δομής ευρετηρίου είναι το πακέτο (*package*) gr.ntua.dblab.plbs.index, καθώς περιλαμβάνει τις κλάσεις που αναπαριστούν το SPI tree και διαχειρίζονται τη λειτουργία του. Το πακέτο περιλαμβάνει τις ακόλουθες κλάσεις:

Πίνακας 1: Κλάσεις του πακέτου gr.ntua.dblab.plbs.index

Rectangle	Αναπαράσταση ενός ορθογωνίου στο χώρο
IndexTree	Κλάση περιτύλιξης (wrapper) για το δέντρο SPI
IndexNode	Γενική κλάση για την αναπαράσταση ενός κόμβου του SPI tree
IndexInternalNode	Αναπαράσταση ενός εσωτερικού κόμβου του δέντρου
IndexLeaf	Αναπαράσταση ενός κόμβου-φύλλου του δέντρου
EndUser	Αναπαράσταση ενός χρήστη
IndexImage	Κλάση για το χειρισμό της οπτικοποίησης του δέντρου
MoveOutOfBoundsException	Κλάση εξαίρεσης για το χειρισμό μετάβασης εκτός επιτρεπτής περιοχής

Παρακάτω περιγράφεται συνοπτικά η κάθε κλάση και τα κύρια μέλη και μέθοδοί της.

4.2.1.1 gr.ntua.dblab.plbs.index.rectangle

Η κλάση αυτή αποτελεί αναπαράσταση μιας ορθογωνικής περιοχής του χώρου. Περιέχει τα ακόλουθα πεδία (*members*):

```
private double minLongitude, minLatitude, height, width;
private double centerLongitude, centerLatitude;
```

Τα minLingitude και minLatitude αποθηκεύουν το γεωγραφικό μήκος και το γεωγραφικό πλάτος της νοτιοδυτικής κορυφής του ορθογωνίου, ενώ τα height και width το ύψος και το πλάτος του αντίστοιχα. Στα πεδία centerLongitude και centerLatitude κρατάμε τις συντεταγμένες (γεωγραφικό μήκος και γεωγραφικό πλάτος αντίστοιχα) της τομής των διαγωνίων του ορθογωνίου για την αποδοτική αποτίμηση πράξεων μεταξύ τέτοιων αντικειμένων. Οι μέθοδοι της κλάσης είναι οι:

- `public boolean` **`isPointIn`**`(double longitude, double latitude);`

  Η μέθοδος αυτή ελέγχει αν το σημείο του χώρου με συντεταγμένες (longitude, latitude) βρίσκεται εντός του καλούμενου αντικειμένου Rectangle. Ο έλεγχος αυτός γίνεται απλά συγκρίνοντας το (longitude, latitude) με τις συντεταγμένες των κορυφών του καλούμενου ορθογωνίου.

- `public boolean` **`isPointIn`**`(double longitude, double latitude, DocumentAreaShape shape);`

  Η μέθοδος αυτή ελέγχει αν το σημείο του χώρου με συντεταγμένες (longitude, latitude) βρίσκεται εντός του σχήματος shape που έχει ως περιγεγραμμένο ορθογώνιο το καλούμενο αντικείμενο Rectangle. Το shape μπορεί να έχει τιμές SQUARE και CIRCLE αλλά μπορεί πολύ εύκολα να επεκταθεί για να περιλαμβάνει και άλλα σχήματα, αν αυτό είναι επιθυμητό. Ο έλεγχος γίνεται με διαφορετικό τρόπο για κάθε τύπο σχήματος.

- `public boolean` **`overlaps`**`(Rectangle r);`

  Η παρούσα μέθοδος υπολογίζει αν το Rectangle r έχει κοινά σημεία με το καλούμενο αντικείμενο Rectangle. Ο υπολογισμός γίνεται συγκρίνοντας τις μέγιστες και τις ελάχιστες συστεταγμένες των δύο αντικειμένων σε κάθε διάσταση.

4.2.1.2 gr.ntua.dblab.plbs.index.EndUser

Η κλάση αυτή αποτελεί την αναπαράσταση ενός χρήστη στο σύστημα. Τα πεδία της κλάσης είναι τα ακόλουθα:

`private double` **`longitude, latitude;`**

Οι τρέχουσες συντεταγμένες του χρήστη, δηλαδή το γεωγραφικό μήκος και το γεωγραφικό πλάτος στο οποίο βρίσκεται ο χρήστης αντίστοιχα. Τα πεδία αυτά λαμβάνουν την τιμή της τελευταίας θέσης που έχει δηλώσει ο χρήστης στο σύστημα.

private int **id**;

Το μοναδικό αναγνωριστικό του χρήστη στο σύστημα.

private IndexLeaf **fatherLeaf**;

Δείκτης προς το φύλλο του SPI tree στο οποίο βρίσκεται εισηγμένος ο χρήστης την τρέχουσα χρονική στιγμή. Φυσικά, έχει την τιμή null όταν ο χρήστης έχει αποχωρήσει από το σύστημα.

private String **profileString**;

Η συμβολοσειρά που αποτελεί το προφίλ του χρήστη. Το πεδίο αυτό δηλαδή περιέχει σε μορφή συμβολοσειράς το XPath ερώτημα που ορίζει το προφίλ του χρήστη.

private String **name**;

Συμβολοσειρά για το πλήρες όνομα του χρήστη. Περισσότερες λεπτομέρειες για τα προσωπικά στοιχεία του χρήστη μπορούν να αποθηκεύονται σε βάση δεδομένων.

private Query **profile**;

Αντικείμενο της κλάσης Query που υπάρχει στο πακέτο gr.ntua.dblab.plbs.yfilterplus. Αποτελεί συντακτικά ανελυμμένη αναπαράσταση του XPath ερωτήματος, ώστε να γίνεται η εισαγωγή του στα NFA του YFilter χωρίς το κόστος της ανάλυσης αυτής.

Οι κυριότερες και χρησιμότερες μέθοδοι της κλάσης αυτής είναι οι ακόλουθες:

- `public` **`EndUser`**`(int id,String profileString, String name);`

  Δημιουργία ενός νέου χρήστη με id που του δίνεται από το σύστημα, Xpath προφίλ που περιγράφεται από τη συμβολοσειρά profileString και όνομα την τιμή της μεταβλητής name.

- `public void` **`move`**`(double longitude, double latitude)`
  `throws MoveOutOfBoundsException;`

  Η μέθοδος αυτή καλείται από ένα χρήστη για να δηλώσει τη μετακίνησή του από την προηγούμενη θέση που είχε δηλώσει στη νέα του θέση (double longitude, double latitude). Η μέθοδος αυτή εγείρει εξαίρεση MoveOutOfBoundsException σε περίπτωση που ο χρήστης μεταβεί εκτός ορίων του συνολικού χώρου όπου παρέχονται οι υπηρεσίες. Η μέθοδος αυτή αρχικά βρίσκει (ακολουθώντας το δείκτη fatherLeaf) το φύλλο στο οποίο είναι εισηγμένος ο χρήστης και στη συνέχεια εντοπίζει το φύλλο προορισμού και τον τελευταίο κοινό κόμβο στα μονοπάτια από τη ρίζα προς τον κόμβο προέλευσης και προορισμού και ζητά τα απαραίτητα κλειδώματα για τη διαγραφή από τον κόμβο προέλευσης και την εισαγωγή στον κόμβο προορισμού, τις οποίες στη συνέχεια εκτελεί καλώντας τις κατάλληλες μεθόδους του IndexTree (που περιγράφεται παρακάτω).

- `public void` **`sendDocument`**`(XMLTree doc);`

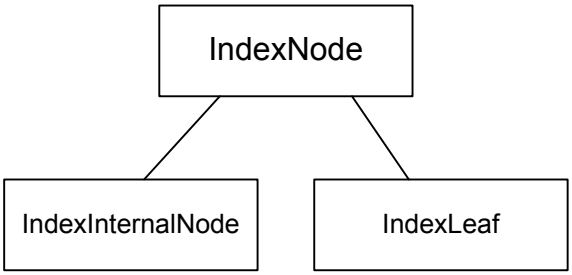
  Η μέθοδος αυτή, η οποία δεν είναι υλοποιημένη, αποστέλλει στο messaging component του συστήματος το XML έγγραφο που περνιέται ως παράμετρος. Αυτό είναι το τελευταίο στάδιο της διαδικασίας αποτίμησης ενός ερωτήματος.

- `public void` **`setProfile`**`(Query profile);`

  Η συνάρτηση αυτή ενημερώνει το προφίλ του χρήστη, θέτωντας ως νέο προφίλ το XPath ερώτημα που περιγράφεται από το αντικείμενο Query της παραμέτρου. Αν το παλαιό προφίλ ήταν εισηγμένο σε κάποιο NFA του SPI tree, διαγράφεται και προστίθεται το νέο.

4.2.1.3 gr.ntua.dblab.plbs.index.IndexNode

Η κλάση IndexNode είναι μια *abstract* κλάση που αναπαριστά ένα κόμβο του δέντρου και εξειδικεύεται στις κλάσεις IndexInternalNode και IndexLeaf, που κληρονομούν από αυτή όπως φαίνεται στο Σχήμα 61:



Σχήμα 61: Η abstract κλάση IndexNode

Η κλάση IndexNode έχει τα ακόλουθα πεδία:

```
protected Rectangle quadrant;
```

Αντικείμενο τύπου Rectangle που αναπαριστά την ορθογωνική γεωγραφική περιοχή που αντιστοιχεί στον κόμβο.

```
protected int depth;
```

Ακέραιο πεδίο που κρατάει το βάθος στο οποίο βρίσκεται ο κόμβος.

```
protected NodeLocation nodeLocation;
```

Το πεδίο αυτό κρατάει την πληροφορία σχετικά με το σε ποιο τεταρτημόριο του κόμβου-πατέρα του αντιστοιχεί ο κόμβος. Ο τύπος NodeLocation ορίζεται ως εξής:

```
public enum NodeLocation {UP_RIGHT, UP_LEFT, BOTTOM_LEFT, BOTTOM_RIGHT};
```

```
protected IndexInternalNode fatherNode;
```

Το πεδίο αυτό είναι ένας δείκτης προς τον πατρικό κόμβο. Δίνει τη δυνατότητα να διατρέχουμε τα κλαδιά του δέντρου όχι μόνον από τη ρίζα προς τα φύλλα, αλλά και αντιστρόφως.

```
protected Semaphore nodeSemaphore;
```

Πρόκειται για το σημαφόρο που ελέγχει την πρόσβαση στον κόμβο, όπως περιγράφεται στους αλγόριθμους κλειδωμάτων του δέντρου.

```
protected boolean isZombieNode;
```

Η μεταβλητή αυτή παίρνει κατά τη δημιουργία του κόμβου την τιμή false. Τίθεται όμως σε true αν ο κόμβος γίνει σκιώδης, αν δηλαδή ο IndexInternalNode αντικατασταθεί από ένα IndexLeaf ενόσω κάποιος περιμένει να αποκτήσει κλείδωμα στον κόμβο.

```
protected IndexNode replacingNode;
```

Ο κόμβος που αντικαθιστά τον τρέχοντα κόμβο στο δέντρο, όταν αυτός γίνει σκιώδης.

Οι μέθοδοι της κλάσης είναι οι ακόλουθες:

- `public` **`IndexNode`**`(NodeLocation nodeLocation, IndexInternalNode fatherNode);`

  Δημιουργία ενός νέου κόμβου με πατρικό τον fatherNode, που αντιστοιχεί το τεταρτημόριό του που περιγράφει η τιμή nodeLocation.

- `public void` **`lock`**`();`

  Μέθοδος που υλοποιεί την αίτηση για κλείδωμα του κόμβου.

- `public void` **`unlock`**`();`

  Μέθοδος που υλοποιεί την άρση του κλειδώματος από τον κόμβο.

- `public IndexNode` **`getImmediateAncestorNode`**`();`

  Μέθοδος που επιστρέφει τον πατέρα του κόμβου. Αν ο κόμβος έχει μετατραπεί σε σκιώδη κατά τη διάσπαση ενός φύλλου σε τέσσερα νέα, η μέθοδος αυτή επιστρέφει το fatherLeaf του replacingNode.

- `public IndexNode` **`getLowerCommonNode`**`(double longitude, double latitude);`

  Η μέθοδος αυτή επιστρέφει τον κατώτερο κόμβο του δέντρου που βρίσκεται στο μονοπάτι από τη ρίζα προς φύλλα που περιέχει το σημείο (longitude, latitude) χρησιμοποιείται για για την εύρεση του τελευταίου κοινού κόμβου στα μονοπάτια από τη ρίζα προς τον κόμβο προέλευσης και προορισμού κατά τη μετακίνηση ενός χρήστη.

Στην κλάση υπάρχουν ακόμη οι ακόλουθες γενικές (*abstract*) μέθοδοι, που εξειδικεύονται στις υποκλάσεις της IndexNode:

```
public abstract void insertUser(EndUser user);
public abstract IndexLeaf getContainingNode(EndUser user);
public abstract void sendDocument(XMLTree doc, Rectangle range,
                        DocumentAreaShape shape, Stack<EndUser> resultUsers);
public abstract void sendDocGetImage(XMLTree doc, Rectangle range);
public abstract void insertUser(EndUser user, boolean modifyNodeLocking);
public abstract EndUser findNeighbor(XMLTree doc);
```

4.2.1.4 gr.ntua.dblab.plbs.index.IndexInternalNode

Όπως αναφέρθηκε και παραπάνω, η κλάση αυτή αποτελεί υποκλάση της IndexNode και αναπαριστά τους εσωτερικούς κόμβους του δέντρου. Τα επιπλέον πεδία της κλάσης είναι τα:

```
private IndexNode[] childNodes;
```

Το πεδίο αυτό είναι ένας πίνακας από IndexNode's που περιέχει δείκτες προς τα τέσσερα παιδιά του κόμβου, που μπορεί να είναι είτε άλλοι εσωτερικοί κόμβοι είτε κόμβοι-φύλλα

```
private byte noOfOccupiedChildren;
```

Το πεδίο αυτό κρατάει τον αριθμό των παιδιών του κόμβου που περιέχουν χρήστες (δηλ. δεν είναι άδειοι). Χρησιμοποιείται για να γνωρίζουμε πότε πρέπει να γίνει σύμπτυξη του δέντρου (όταν το πεδίο αυτό πάρει την τιμή 0).

Οι μέθοδοι της κλάσης αυτής είναι οι ακόλουθες:

- `public` **`IndexInternalNode`**`(IndexLeaf sourceLeaf);`

  Δημιουργία ενός νέου εσωτερικού κόμβου για την αντικατάσταση του φύλλου sourceLeaf που διασπάται. Γίνονται τα απαραίτητα κλειδώματα που περιγράφονται στους αντίστοιχους αλγόριθμους και δημιουργούνται τα τέσσερα παιδιά-φύλλα που αντιστοιχούν στον κόμβο.

- `public` **`IndexInternalNode`**`(IndexTree indexTree, Rectangle quadrant);`

  Ο κατασκευαστής που χρησιμοποιείται για τη δημιουργία του κόμβου ρίζας του δέντρου και μόνο. Η δεύτερη παράμετρος είναι ένα αντικείμενο τύπου Rectangle που αναπαριστά το συνολικό χώρο που ευρετηριάζει το δέντρο.

- `public void` **`insertUser`**`(EndUser user);`

  Η μέθοδος αυτή αναλαμβάνει την εισαγωγή του χρήστη που παίρνει ως παράμετρο στο SPI tree σύμφωνα με τους αλγόριθμους και το πρωτόκολλο κλειδωμάτων που περιγράφεται στο προηγούμενο κεφάλαιο.

- `public void` **`moveUser`**`(EndUser user, double longitudeTo, double latitudeTo);`

  Η μέθοδος αυτή αποτελεί κομμάτι της υλοποίησης της μετακίνησης του χρήστη user στις νέες συντεταγμένες του (*longitudeTo*, *latitudeT*o). Η ενημέρωση γίνεται με τους αλγόριθμους που περιγράφονται σε προηγούμενο κεφάλαιο και τα κλειδώματα σύμφωνα με το πρωτόκολλο κλειδωμάτων που έχει παρουσιαστεί. Η μέθοδος αυτή χρησιμοποιεί και τη μέθοδο που ακολουθεί για να επιτελέσει τη λειτουργία της:

- `public IndexLeaf` **`getContainingLeaf`**`(double longitude, double latitude);`

  Η μέθοδος αυτή, που χρησιμοποιείται από τη moveUser, βρίσκει το φύλλο που περιέχει το σημείο (longitude, latitude) και το επιστρέφει. Πριν την εκτέλεση της μεθόδου αυτής, θεωρούμε ότι ο κόμβος έχει κλειδωθεί από την καλούσα μέθοδο. Η getContainingLeaf κλειδώνει και ξεκλειδώνει (όπως περιγράφεται στο πρωτόκολλο κλειδωμάτων) τους κόμβους από τον τρέχοντα κόμβο μέχρι το ζητούμενο φύλλο και επιστρέφει το φύλλο αυτό αν βρεθεί ή null διαφορετικά. Αν το φύλλο βρεθεί τόσο ο πατέρας του όσο και αυτό διατηρούνται κλειδωμένοι.

- `public byte` **`incrOccupiedChildren`**`();`

  Η συνάρτηση αυτή αυξάνει κατά ένα τη μεταβλητή noOfOccupiedChildren.

- `public void` **`decrOccupiedChildren`**`();`

Η μέθοδος αυτή μειώνει κατά ένα την τιμή της noOfOccupiedChildren. Αν η τιμή αυτή γίνει 0, τότε έχουμε σύμπτυξη του δέντρου και αντικατάσταση του κόμβου αυτού με φύλλο. Κατόπιν καλείται η decrOccupiedChildren του πατρικού κόμβου, ώστε η σύμπτυξη του δέντρου να συνεχιστεί προς τα πάνω αν είναι απαραίτητο. Καθ' όλη τη διαδικασία τα κλειδώματα των κόμβων γίνονται όπως περιγράφεται από το πρωτόκολλο κλειδωμάτων.

- `public void` **`sendDocument`** `(XMLTree doc, Rectangle boundingRectangle, DocumentAreaShape shape, Stack<EndUser> resultUsers);`

  Η μέθοδος αυτή υλοποιεί την εκτέλεση ενός ερωτήματος περιοχής στον κόμβο. Στο εσωτερικό της μεθόδου αυτής καλούνται αναδρομικά οι μέθοδοι sendDocument των παιδιών του κόμβου, οι οποίοι επικαλύπτονται με το boundingRectangle της περιοχής του ερωτήματος. Ο λόγος που χρησιμοποιείται το περιβάλλον ορθογώνιο της περιοχής είναι ότι αυτό επιταχύνει πολύ τους υπολογισμούς των επικαλύψεων, μεταξύ του Rectangle του ερωτήματος και των Rectangles-παιδιών του κόμβου. Στο Stack της παραμέτρου προστίθενται τα αποτελέσματα που προέρχονται από τους κόμβους-απογόνους του τρέχοντος κόμβου, ώστε τελικά στο stack αυτό να περιέχεται το αποτέλεσμα του ερωτήματος.

- `public void` **`sendDocGetImage`** `(XMLTree doc, Rectangle boundingRectangle);`

  Όπως και παραπάνω, αλλά ως αποτέλεσμα δημιουργείται ένα αντικείμενο IndexImage, που αποτελεί οπτικοποίηση του ερωτήματος και των αποτελεσμάτων του στο χώρο. Το αντικείμενο αυτό είναι static πεδίο του IndexTree (βλ. παρουσίαση IndexTree).

- `public EndUser` **`findNeighbor`** `(XMLTree doc);`

  Η μέθοδος αυτή χρησιμοποιείται κατά την αποτίμηση ενός ερωτήματος εγγύτερου γείτονα για τον εντοπισμό του αρχικού υποψηφίου. Λειτουργεί καλώντας αναδρομικά τη findNeighbor των παιδιών της μέχρι να βρεθεί κάποιος χρήστης, οποίος αν βρεθεί επιστρέφεται.

4.2.1.5 gr.ntua.dblab.plbs.index.IndexLeaf

Όπως αναφέρθηκε και παραπάνω, η κλάση αυτή αποτελεί υποκλάση της IndexNode και αναπαριστά τους κόμβους-φύλλα του δέντρου. Τα επιπλέον πεδία της κλάσης είναι τα:

`private EXFilterBasic` **`nodeFilter`**`;`

Το πεδίο nodeFilter είναι τύπου EXFilterBasic, που είναι η κλάση που αντιπροσωπεύει το NFA του φύλλου, όπως αυτό υλοποιείται στον κώδικα του YFilter. Πρόκειται δηλαδή για ένα δείκτη με τον οποίο έχουμε πρόσβαση στο NFA του φύλλου και τις μεθόδους του.

`public static int` **`MAX_NODE_USERS`**`;`

Το static πεδίο MAX_NODE_USERS είναι ένας ακέραιος που ορίζει τη χωρητικότητα του φύλλου, δηλαδή το μέγιστο αριθμό χρηστών, και άρα προφίλ, που μπορούν να εισαχθούν στο NFA του φύλλου.

```
private int nodeUsersNo;
```

Το πεδίο αυτό είναι ένας ακέραιος που κρατάει τον αριθμό των χρηστών που βρίσκονται εισηγμένοι στο NFA του φύλλου.

```
List<EndUser> nodeUsers;
```

Η λίστα αυτή κρατάει τους χρήστες που βρίσκονται στο φύλλο και ευρετηριάζονται από το NFA του φύλλου με βάση το προφίλ τους. Η λίστα αυτή διατηρείται για πιο γρήγορη πρόσβαση στους χρήστες του NFA.

Οι μέθοδοι της κλάσης IndexLeaf περιγράφονται ακολούθως:

- `public IndexLeaf (NodeLocation nodeLocation, IndexInternalNode fatherNode);`

  Δημιουργία ενός αντικειμένου IndexLeaf με πατρικό κόμβο το fatherNode, που αντιστοιχεί στο τεταρτημόριο του πατρικού κόμβου που περιγράφεται από την παράμετρο nodeLocation.

- `public void insertUser(EndUser user);`

  Εισαγωγή του χρήστη user στο φύλλο. Αυτό γίνεται προσθέτοντας το χρήστη στη λίστα nodeUsers, αυξάνοντας το nodeUsersNo και, κυρίως, προσθέτοντας το προφίλ του χρήστη στο NFA nodeFilter του φύλλου. Αν είναι ο πρώτος χρήστης που εισάγεται στο φύλλο, το φύλλο δεν έχει NFA, οπότε δημιουργείται ένα νέο αντικείμενο τύπου EXFilterBasic. Η πρόσθεση του χρήστη στο NFA γίνεται καλώντας τη μέθοδο add() της κλάσης EXFilterBasic, αφού πρώτα έχουμε εξασφαλίσει αποκλειστική πρόσβαση στο NFA με κλείδωμα του αντικειμένου nodeFilter. Αν ο χρήστης δε χωράει στο φύλλο, το φύλλο διασπάται, όπως περιγράφεται στους αλγόριθμους του προηγούμενου κεφαλαίου.

- `public IndexInternalNode removeUser (EndUser user);`

  Η μέθοδος αυτή υλοποιεί τη διαγραφή του χρήστη από το φύλλο, με διαγραφή του από τη λίστα nodeUsers και αφαίρεση του προφίλ του από το NFA του φύλλου. Η αφαίρεση αυτή από το NFA γίνεται αφού πρώτα έχουμε εξασφαλίσει αποκλειστική πρόσβαση στο NFA με κλείδωμα του αντικειμένου nodeFilter, καλώντας τη μέθοδο deleteQuery() της κλάσης EXFilterBasic. Αν το φύλλο αδειάσει από χρήστες μετά τη διαγραφή, καλείται η decrOccupiedChildren του πατρικού κόμβου.

- `public void updateProfile(EndUser user, Query oldProfile);`

  Ενημέρωση του προφίλ του χρήστη user. Διαγράφεται το oldProfile από το NFA του φίλτρου και εισάγεται το νέο προφίλ του χρήστη (το οποίο υπάρχει ήδη μέσα στο αντικείμενο EndUser).

- `public void **sendDocument** (XMLTree doc, Rectangle boundingRectangle, DocumentAreaShape shape, Stack<EndUser> resultUsers);`

  Η μέθοδος αυτή υλοποιεί το ταίριασμα του XML εγγράφου ενός ερωτήματος περιοχής με τα προφίλ των χρηστών που είναι εισηγμένοι στο NFA του φύλλου. Αυτό επιτυγχάνεται με διαδοχική κλήση των μεθόδων clear(), setXMLTree(doc), allocateStreams() και startParsing() της κλάσης EXFilterBasic, που εκτελούν το NFA βάση το έγγραφο doc. Στη συνέχεια με την getMatchedQueries() της ίδιας κλάσης παίρνουμε τα προφίλ που ταιριάζουν με το έγγραφο. Από κάθε προφίλ ακολουθώντας ένα δείκτη παίρνουμε το χρήστη στον οποίο ανήκει. Τελικά ελέγχουμε αν ο χρήστης όντως ανήκει στην περιοχή με σχήμα shape του ερωτήματος και περιβάλλον ορθογώνιο το boundingRectangle (υπενθυμίζεται ότι στο φύλλο οδηγηθήκαμε ελέγχοντας επικαλύψεις με το περιβάλλον ορθογώνιο μόνο). Στη λίστα resultUsers προστίθενται όλοι οι χρήστες που ικανοποιούν το ερώτημα.

- `public EndUser **findNeighbor**(XMLTree doc);`

  Η μέθοδος αυτή χρησιμοποιείται κατά την αποτίμηση ενός ερωτήματος εγγύτερου γείτονα για τον εντοπισμό του αρχικού υποψηφίου. Όπως και η sendDocument(), εκτελεί το NFA με βάση το XML έγγραφο και επιστρέφει *έναν* μόνο χρήστη από αυτούς που το ικανοποιούν ή null αν δεν υπάρχει τέτοιος.

- `public void **sendDocGetImage**(XMLTree doc, Rectangle boundingRectangle);`

  Η μέθοδος αυτή κάνει ότι και η sendDocument, αλλά τελικά προσθέτει το quadrant του φύλλου στη λίστα docRectangles του IndexTree (βλ. παρουσίαση IndexTree) και τους χρήστες που ταιριάζουν στο έγγραφο στην επίσης λίστα docWinners του IndexTree. Αμφότερες οι λίστες αυτές χρησιμοποιούνται από την κλάση IndexImage για την οπτικοποίηση του δέντρου και των αποτελεσμάτων ενός ερωτήματος.

4.2.1.6 gr.ntua.dblab.plbs.index.IndexTree

Η κλάση αυτή αποτελεί μια κλάση περιτύλιξης για τις λειτουργίες του SPI tree, δηλαδή παρέχει τις μεθόδους για τη δημιουργία και τη διαχείριση ενός στιγμιότυπου (instance) του δέντρου. Τα σηματικότερα πεδία της κλάσης είναι τα:

`private double **maxLatitude, maxLongitude, minLatitude, minLongitude;**`

Τα παραπάνω πεδία με την προφανή τους σημασία ορίζουν το συνολικό χώρο που ευρετηριάζει το δέντρο.

`private int **maxHeight;**`

Ακέραιος αριθμός που ορίζει το μέγιστο ύψος του δέντρου. Σε ένα δέντρο με τέτοιο ύψος, τα φύλλα θεωρούμε ότι έχουν άπειρη χωρητικότητα.

`private TreeMap **users;**`

Πρόκειται για μια δομή τύπου TreeMap, που αποτελεί ευρετήριο των χρηστών του συστήματος με βάση το αναγνωριστικό τους (id). Όπως περιγράφεται και στο προηγούμενο κεφάλαιο, αποτελεί υλοποίηση ενός Red-Black tree.

```
private IndexInternalNode tree;
```

Πρόκειται για τον κόμβο ρίζα του δέντρου.

```
public Stack<EndUser> docWinners;
```

```
public Stack<Rectangle> docRectangles;
```
Οι δύο αυτές στοίβες χρησιμοποιούνται για την απεικόνιση του δέντρου και των αποτελεσμάτων ενός ερωτήματος περιοχής από την κλάση IndexImage. Στην docRectangles αποθηκεύονται όλα τα Rectangles των φύλλων στα οποία φτάνει ένα ερώτημα περιοχής κατά την αποτίμησή του, ενώ στη στοίβα docWinners αποθηκεύονται οι χρήστες που ικανοποιούν ένα ερώτημα.

Οι μέθοδοι της κλάσης συνοψίζονται παρακάτω:

- ```
  public IndexTree (double minLongitude, double maxLongitude,
                    double minLatitude, double maxLatitude,
                    int maxHeight, int maxNodeUsers);
  ```

  Κατασκευαστής του δέντρου. Οι παράμετροι έχουν τη σημασία των ομωνύμων τους πεδίων. Δημιουργείται η δομή TreeMap που περιέχει τους χρήστες και η ρίζα του δέντρου ανατίθεται στο δείκτη tree.

- ```
  public void registerUser (EndUser user);
  ```

  Καταγραφή ενός χρήστη στο σύστημα (και στη βάση δεδομένων αν αυτό είναι επιθυμητό). Προσθήκη του χρήστη στη δομή TreeMap.

- ```
  public boolean moveIn (int id, double longitude, double latitude)
                         throws MoveOutOfBoundsException;
  ```

  Εισαγωγή του χρήστη με αναγνωριστικό id στο δέντρο, στη θέση του χώρου (longitude, latitude) καλώντας τη συνάρτηση move() του αντίστοιχου αντικειμένου EndUser. Αν η θέση είναι εκτός του ευρετηριαζόμενου χώρου, εγείρεται εξαίρεση MoveOutOfBoundsException.

- ```
  public boolean moveOut (int id);
  ```

  Διαγραφή ενός χρήστη από το δέντρο, δηλαδή αφαίρεση του χρήστη από το φύλλο στο οποίο βρίσκεται. Υλοποιείται με κλήση της removeUser της κλάσης IndexLeaf.

- ```
  public boolean move (int id, double longitude, double latitude)
                       throws MoveOutOfBoundsException
  ```
  ```

Μετακίνηση του χρήστη με αναγνωριστικό id στη θέση (longitude, latitude) με κλήση της move() του αντίστοιχου αντικειμένου EndUser. Αν η θέση είναι εκτός του ευρετηριαζόμενου χώρου, εγείρεται εξαίρεση MoveOutOfBoundsException.

- `public boolean` **`updateProfile`** `(int id, Query profile);`

  Ενημέρωση του προφίλ του χρήστη με αναγνωριστικό id με κλήση της μεθόδου updateProfile του αντίστοιχου αντικειμένου EndUser.

- ```
  public void sendDocument (XMLTree doc, double minLongitude,
                            double minLatitude, double width, double height);
  public void sendDocument (XMLTree doc, double longitude,
                                        double latitude, double radius);
  ```

  Οι δύο αυτές μέθοδοι υλοποιούν τα ερωτήματα περιοχής. Η πρώτη από αυτές αντιστοιχεί σε ερώτημα σε ορθογωνική περιοχή με νοτιοδυτική κορυφή το σημείο (longitude, latitude),πλάτος width και ύψος height, ενώ η δεύτερη σε κυκλική περιοχή με κέντρο (longitude, latitude) και ακτίνα radius. Αμφότερες υλοποιούνται με κλήση της sendDocument του κόμβου-ρίζας του δέντρου (που βρίσκεται από το δείκτη tree). Στην πρώτη η παράμετρος shape έχει τιμή DocumentAreaShape.RECTANGLE, ενώ στη δεύτερη DocumentAreaShape.CIRCLE.

- `public EndUser` **`sendNNDocument`** `(XMLTree doc, int id);`

  Η μέθοδος αυτή υλοποιεί το ερώτημα εγγύτερου γείτονα. Δηλαδή επιστρέφει το χρήστη EndUser του οποίου το προφίλ ικανοποιεί το doc και η απόστασή του από το χρήστη με αναγνωριστικό id είναι η ελάχιστη. Αρχικά βρίσκεται ένας υποψήφιος χρήστης με χρήση της findNeighbor (που καλείται για το φύλλο στο οποίο βρίσκεται ο χρήστης με αναγνωριστικό id). Αν δε βρεθεί τέτοιος χρήστης, η αναζήτηση συνεχίζεται στον πατρικό του κόμβο, με κλήση της findNeighbor της κλάσης IndexInternalNode (οπότε η αναζήτηση «κατεβαίνει» σε όλα τα παιδιά του). Η διαδικασία αυτή επαναλαμβάνεται αναδρομική μέχρι να βρεθεί ένας χρήστης. Αν δε βρεθεί σταματάει. Διαφορετικά, όπως περιγράφεται και στο προηγούμενο κεφάλαιο, εκτελείται ερώτημα περιοχής με κλήση της sendDocument σε κυκλική περιοχή, για να διαπιστωθεί αν αυτός είναι όντως ο εγγύτερος χρήστης ή όχι.

- ```
  public IndexImage sendDocGetImage (XMLTree doc, double longitude,
                    double latitude, double radius, String format);
  public IndexImage sendNNDocGetImage (XMLTree doc, int id,
                                       String format);
  ```

  Οι δύο αυτές συναρτήσεις είναι αντίστοιχες με τις δύο προηγούμενες, μόνο που επιστρέφουν ένα αντικείμενο IndexImage, που αποτελεί οπτικοποίηση των αποτελεσμάτων των αντίστοιχων ερωτημάτων. Η οπτικοποίηση για το ερώτημα περιοχής επιτυγχάνεται όπως

μέσω των λιστών docWinners και docRectangles, όπως περιγράφεται παραπάνω, ενώ για το ερώτημα εγγύτερου γείτονα απλά «ζωγραφίζοντας» τα δύο σημεία που αντιστοιχούν στο χρήστη με αναγνωριστικό id και στον εγγύτερό του γείτονα.

4.2.2 Διεπαφή Υπηρεσιών Ιστού (Web Services Interface)

Για την υλοποίηση των Υπηρεσιών Ιστού (*Web Services*) χρησιμοποιήθηκε το πακέτο Apache Axis στην έκδοση 1.4, ενώ για τη διάθεσή τους στο Internet χρησιμοποιήσαμε ως εξυπηρετητή (Web Server & servlet engine) τον Apache Tomcat στην έκδοση 5.5.17. Οι Υπηρεσίες ιστού διακρίνονται σε αυτές που είναι διαθέσιμες στους εγγεγραμμένους χρήστες και στους παρόχους περιεχομένου αν και αυτή η διάκριση αυτή όπως έχει αναφερθεί και παραπάνω δεν είναι παρά τυπική. Εδώ παραθέτουμε τις μεθόδους που υλοποιούν τις Υπηρεσίες Ιστού όπως περιγράφονται στο αρχείο PLBS.wsdl του συνοδευτικού υλικού. Σημειώνεται πως η περιγραφή με τη γλώσσα WSDL είναι βέβαια ανεξάρτητη υλοποίησης που μπορεί να γίνει σε κάθε γλώσσα και πλατφόρμα που υποστηρίζει Υπηρεσίες Ιστού.

4.2.2.1 Υλοποίηση Υπηρεσιών χρηστών

Οι Υπηρεσίες Ιστού που υλοποιήθηκαν και είναι διαθέσιμες σε όλους του χρήστες του συστήματος (εγγεγραμμένοι χρήστες, πάροχοι περιεχομένου) είναι οι εξής:

```
void registerUser(int id, String profile)
```

Εγγραφή ενός νέου χρήστη στο σύστημα με αναγνωριστικό το id και προφίλ που περιγράφεται από το XPath ερώτημα που περιέχει το string profile. Ενημερώνεται η ΒΔ του κεντρικού εξυπηρετητή. Στη συνέχεια ο χρήστης με αναγνωριστικό id έχει τη δυνατότητα να εισέρχεται στο σύστημα και να χρησιμοποιεί τις υπηρεσίες του.

```
boolean moveIn(int id, double longitude, double latitude)
```

Με την κλήση της συγκεκριμένης υπηρεσίας ο χρήστης εισέρχεται στο σύστημα και αρχικοποιείται η θέση του και το προφίλ του. Ελέγχεται αρχικά αν ο χρήστης είναι εγγεγραμμένος στο σύστημα από το TreeMap που διατηρείται στην κύρια μνήμη με όλους τους εγγεγραμμένους χρήστες. Σε περίπτωση που ο χρήστης δεν είναι εγγεγραμμένος η είσοδος αποτυγχάνει και επιστρέφεται ένδειξη στον καλούντα της συνάρτησης (επιστρέφει false). Αν ο χρήστης είναι εγγεγραμμένος ανακτάται το προεπιλεγμένο προφίλ του (*default profile*) από τη ΒΔ. Στη συνέχεια τοποθετείται στο κατάλληλο φύλλο του SPI tree με βάση τις πληροφορίες θέσης (*longitude*, *latitude*) και προφίλ. Η Υπηρεσία επιστρέφει ένδειξη για την επιτυχημένη είσοδο του χρήστη στο σύστημα.

```
boolean move(int id, double longitude, double latitude)
```

Η κλήση της συγκεκριμένης υπηρεσίας γίνεται για την ενημέρωση της νέας θέσης με γεωγραφικές συντεταγμένες *longitude* και *latitude* του χρήστη με αναγνωριστικό id. Εντοπίζεται η εγγραφή του χρήστη με χρήση του TreeMap και από την εγγραφή η τρέχουσα τοποθέτησή του στη δομή του ευρετηρίου. Στη συνέχεια καλούνται οι κατάλληλες μέθοδοι του SPI tree. Σε περίπτωση μη εγγεγραμμένου χρήστη, χρήστη που δεν έχει εισέλθει στο σύστημα ή μετακίνησης εκτός των ορίων του ευρετηριαζόμενου χώρου η κλήση της υπηρεσίας αποτυγχάνει (επιστρέφει *false*). Διαφορετικά επιτυγχάνει (επιστρέφει *true*).

```
boolean updateProfile(int id, XPQuery newProfile)
```

Η χρήση της αντίστοιχης Υπηρεσίας Ιστού γίνεται για την ενημέρωση του Προφίλ του χρήστη με αναγνωριστικό id. Αν ο χρήστης είναι την τρέχουσα στιγμή εκτός συστήματος, τότε εξάγεται η XPath έκφραση που αντιστοιχεί το προφίλ και ενημερώνεται απλά η ΒΔ που διατηρούνται τα προεπιλεγμένα προφίλ. Διαφορετικά εντοπίζεται το φύλλο που ευρετηριάζει το χρήστη και γίνονται οι κατάλληλες τροποποιήσεις στη δομή του NFA του YFilter ώστε να γίνει η ενημέρωση του προφίλ του χρήστη. Παρατηρούμε ότι ο χρήστης καλείται να στείλει το νέο του προφίλ όχι ως συμβολοσειρά της XPAth αλλά ως αντικείμενο της τάξης που αναπαριστά ένα ερώτημα της XPath στο SPI tree. Η συγκεκριμένη επιλογή έγινε για την ελάφρυνση του υπολογιστικού φορτίου του κεντρικού εξυπηρετητή (αποφυγή parsing). Η μέθοδος επιστρέφει *false* σε περίπτωση μη εγγεγραμμένου χρήστη, διαφορετικά επιστρέφει *true*.

```
boolean moveOut(int id)
```

Με την κλήση αυτής της μεθόδου-Υπηρεσίας ο χρήστης με αναγνωριστικό *id* ενημερώνει το σύστημα για την έξοδό του από αυτό. Η κλήση αυτής της υπηρεσίας σε πραγματικό περιβάλλον γίνεται είτε εκούσια από το χρήστη είτε ακούσια (με κάποιο αυτοματοποιημένο μηχανισμό από την τερματική συσκευή όταν τερματίζει τη λειτουργία της). Εντοπίζεται η θέση του χρήστη στο ευρετήριο και εκτελείται η πράξη της διαγραφής. Αν ο χρήστης δεν έχει εισέλθει τότε η κλήση παραβλέπεται ως εσφαλμένη. Η μέθοδος επιστρέφει *false* αν ο χρήστης δεν είναι εγγεγραμμένος.

4.2.2.2 Υλοποίηση Υπηρεσιών παρόχων περιεχομένου

Οι Υπηρεσίες Ιστού που υλοποιήθηκαν και είναι διαθέσιμες στους παρόχους περιεχομένου αναφέρονται παρακάτω. Μέσω αυτών οι πάροχοι δημοσιεύουν περιεχόμενο σε ενδιαφερόμενους χρήστες που πληρούν τα γεωγραφικά κριτήρια που θέτουν οι πάροχοι.

```
List sendDocument (int id, XMLTree doc, double longitude,
```

```
                                        double latitude, double radius)
```

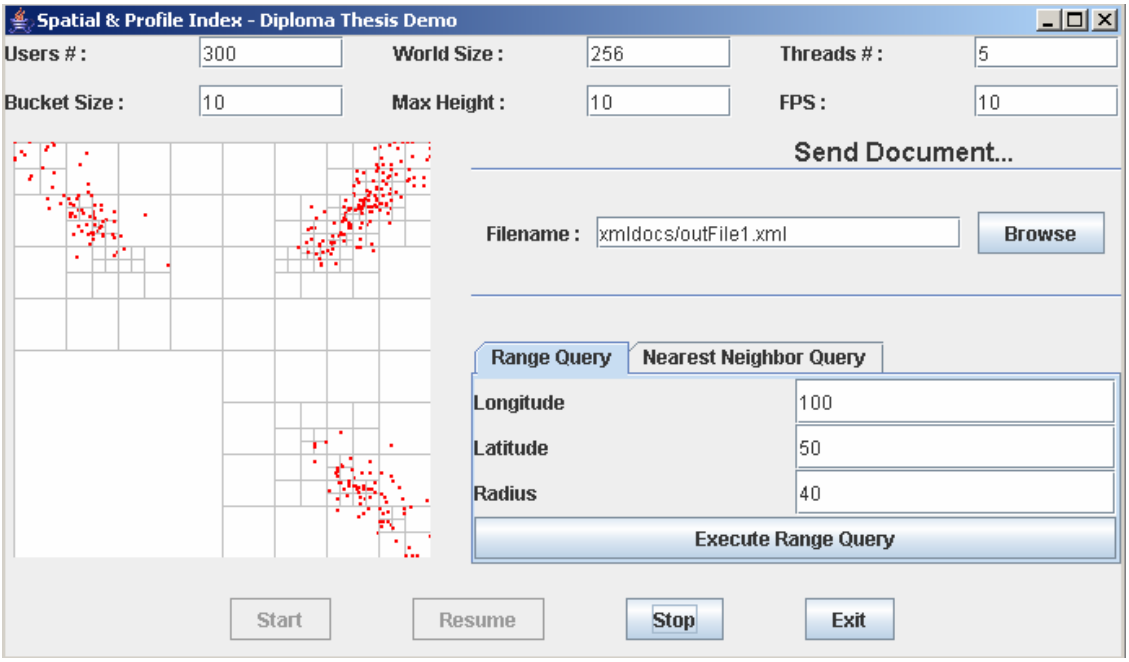
Αποστολή ενός XML εγγράφου στο σύστημα. Η αποστολή γίνεται από το χρήστη – πάροχο περιεχομένου με αναγνωριστικό *id*, το οποίο πιστοποιείται για να πάρει την απάντηση, αλλά και για λόγους πιθανής χρέωσης. Πρόκειται για ερώτημα περιοχής (*range query*), στο οποίο το XML έγγραφο στέλνεται στο σύστημα σε συντακτικά αναλυμένη μορφή (κλάση XML tree) ώστε να μην επιβαρύνεται υπολογιστικά ο κεντρικός εξυπηρετητής με τη διαδικασία της συντακτικής ανάλυσης. Το ερώτημα γίνεται στην κυκλική περιοχή με κέντρο τη θέση (longitude, latitude) και ακτίνα radius. Η κλήση αυτή επιστρέφει μια λίστα με τους χρήστες που παρέλαβαν μέρος ή όλο το XML έγγραφο.

```
EndUser sendNNDocument (int id, XMLTree doc)
```

Αποστολή ενός XML εγγράφου στον εγγύτερο γείτονα από το χρήστη με αναγνωριστικό id. Αναζητείται δηλαδή ο κοντινότερος χρήστης του *id* του οποίου το προφίλ ταιριάζει με το έγγραφο doc. Αν βρεθεί τέτοιος χρήστης, επιστρέφονται τα στοιχεία του, διαφορετικά επιστρέφεται *null*.

4.3 Εφαρμογή επίδειξης SPI tree

Για την επίδειξη της λειτουργικότητας του SPI tree αναπτύχθηκε μία μικρή εφαρμογή με γραφικό περιβάλλον (package gr.ntua.dblab.plbs.dtdemo). Το γραφικό περιβάλλον της εφαρμογής φαίνεται στο Σχήμα 62.



Σχήμα 62: Εφαρμογή επίδειξης

Ο χρήστης της εφαρμογής έχει τη δυνατότητα να καθορίσει τις τιμές των κύριων μεταβλητών του συστήματος και να έχει μια εποπτική εικόνα της λειτουργίας του συστήματος. Οι μεταβλητές που μπορεί να καθορίσει είναι οι εξής:

- Τρέχων αριθμός χρηστών συστήματος
- Μέγεθος κουβά (bucket size)
- Μέγεθος ευρετηριαζόμενου χώρου (μόνο μία διάσταση για τετραγωνικό χώρο στην εφαρμογή επίδειξης)
- Μέγιστο ύψος του Quad tree του SPI tree
- Αριθμός νημάτων ν που εξυπηρετούν τις αιτήσεις χρηστών για μετακίνηση
- Αριθμός εικόνων του συστήματος το δευτερόλεπτο

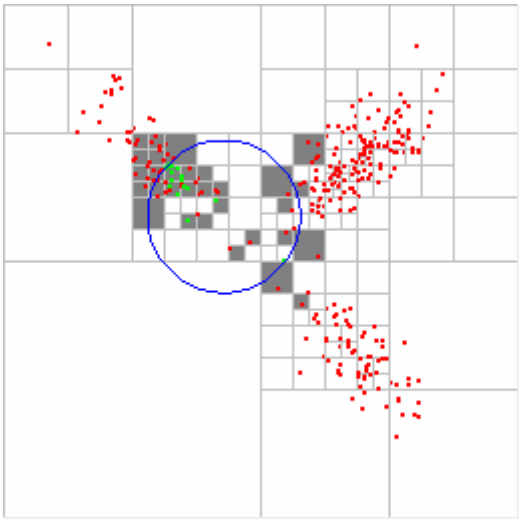
Με βάση τις τιμές των παραπάνω μεταβλητών δημιουργούνται χρήστες με τυχαίες θέσεις στο σύστημα και τυχαίο προφίλ. Για το δεύτερο χρησιμοποιείται μία γεννήτρια ερωτημάτων XPath (XPath query generator) που παίρνει σαν είσοδο ένα σχήμα XML εγγράφων. Στη συνέχεια ν νήματα τρέχουν ταυτόχρονα και επιλέγουν τυχαία χρήστες και ενημερώνουν τη θέση τους με τυχαίες μεταβολές.

Για την απεικόνιση της τρέχουσας κατάσταση του συστήματος χρησιμοποιείται η κλάση IdexImage με μεθόδους που διατρέχουν τους κόμβους του δέντρου ξεκινώντας από τη ρίζα και σχεδιάζουν τις κατάλληλες τομές στο αρχικό τετράγωνο με τη μεταβολή του ύψους των υπό εξέταση κόμβων. Ο ρυθμός με τον οποίο ζητούνται αυτές οι εικόνες από το σύστημα καθορίζεται, όπως αναφέρεται παραπάνω, από το χρήστη.

Τέλος, μέσα από το γραφικό περιβάλλον δίνεται η δυνατότητα στο χρήστη δημοσίευσης εγγράφων στο σύστημα και η οπτικοποίηση των χρηστών που τα λαμβάνουν. Αυτό γίνεται με τη βοήθεια δύο μεθόδων που προστέθηκαν στην κλάση IndexTree. Αυτές οι μέθοδοι είναι:

```
IndexImage sendDocGetImage(int id, XMLTree doc, double longitude,
                           double latitude, double radius, String format)
```

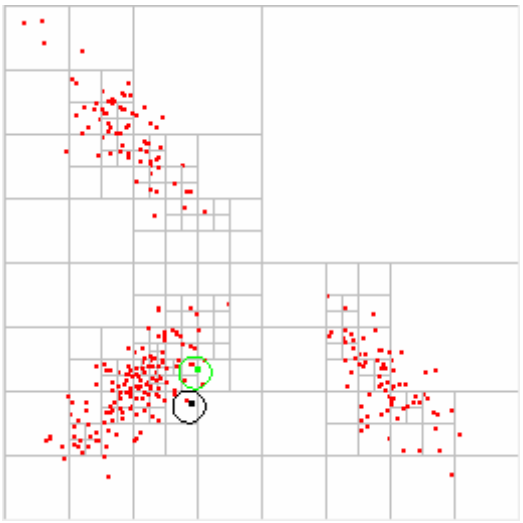
Χρησιμοποιείται κατ ουσία η μέθοδος για δημοσίευση εγγράφου σε περιοχή, Index.sendDocument. Σχεδιάζεται αρχικά η τρέχουσα Αποστολή ενός ερωτήματος στο σύστημα, όπως και παραπάνω, με τη διαφορά ότι ο υποβάλλων το ερώτημα εκτός από τη λίστα των χρηστών που ικανοποιούν το ερώτημα, λαμβάνει ως απάντηση και μια εικόνα σε format της επιλογής του, με τη θέση των χρηστών στο χώρο, όπου με κόκκινο φαίνονται όλοι οι χρήστες, ενώ με πράσινο αυτοί που ικανοποιούν το ερώτημα της κυκλικής μπλε περιοχής.



Σχήμα 63: Ερώτημα περιοχής στην Εφαρμογή Επίδειξης

```
IndexImage sendNNDocGetImage(int id, XMLTree doc, String format)
```

Όπως και παραπάνω, μόνο που στην περίπτωση αυτή επιστρέφεται εικόνα με το επιθυμητό format, που απεικονίζει τη θέση του χρήστη, αλλά και του κοντινότερου γείτονά του, όπως φαίνεται στο Σχήμα 64:



Σχήμα 64: Ερώτημα εγγύτερης γειτονίας στην Εφαρμογή Επίδειξης

Για τις παραπάνω οπτικοποιήσεις αναπτύχθηκε η κλάση IndexImage που περιέχει τα ακόλουθα πεδία:

```
private BufferedImage image;
```

Αντικείμενο τύπου BufferedImage που περιέχει την αναπαράσταση στη μνήμη της τελευταίας εικόνας που πήραμε από το δέντρο.

`private int **width;**`

Το πλάτος της εικόνας, που ισούται με το πλάτος του χώρου που ευρετηριάζει το δέντρο.

`private int **height;**`

Το ύψος της εικόνας, που ισούται με το ύψος του χώρου που ευρετηριάζει το δέντρο.

`private IndexTree **indexTree;**`

Το δέντρο, το οποίο απεικονίζει η εικόνα.

Οι σημαντικότερες μέθοδοι της κλάσης αυτής φαίνονται παρακάτω:

- `public **IndexImage**(IndexNode root);`

  Μέθοδος-κατασκευαστής των στιγμιότυπων της κλάσης που απεικονίζει το δέντρο με ρίζα τον κόμβο root. Χρησιμοποιεί τη μέθοδο redraw() που περιγράφεται παρακάτω.

- `public **IndexImage** (IndexNode root, Stack<Rectangle> docRectangles, Stack<EndUser> docWinners);`

  Μέθοδος-κατασκευαστής των στιγμιότυπων της κλάσης, που απεικονίζει τα αποτελέσματα ενός ερωτήματος περιοχής που έχει επισκεφθεί τα φύλλα που έχουν ως quadrant τα αντικείμενα τύπου Rectangle της στοίβας docRectangles και ικανοποιείται από τους χρήστες της στοίβας docWinners. Αφού δημιουργήσει το αντικείμενο IndexImage, ζωγραφίζει την εικόνα με λευκό χρώμα και κατόπιν ζωγραφίζει όλα τα Rectangles της αντίστοιχης στοίβας, ως ορθογώνια γεμισμένα με σκούρο γκρι χρώμα. Κατόπιν καλείται η traverse() που διασχίζει το δέντρο και απεικονίζει όλα τα ορθογώνια και τους χρήστες του δέντρου όπως εξηγείται παρακάτω. Τέλος, με πράσινο χρώμα ζωγραφίζει τους χρήστες που ικανοποιούν το ερώτημα (και βρίσκονται στη λίστα docWinners).

- `public void **traverse**(IndexNode node);`

  Η μέθοδος αυτή ξεκινώντας από τον κόμβο node του δέντρου, διασχίζει όλο το υποδέντρο που έχει ως ρίζα τον κόμβο αυτό, μέχρι να φτάσει στα φύλλα. Η διάσχιση αυτή γίνεται αναδρομικά. Όταν φτάσει σε ένα φύλλο του δέντρου ζωγραφίζει στο αντικείμενο graphics του IndexImage το σύνορο του Rectangle του φύλλου με χρώμα γκρι, καθώς και όλους τους χρήστες που περιέχονται στο φύλλο του δέντρου με χρώμα κόκκινο. Κατ' αυτόν τον τρόπο, όταν η μέθοδος αυτή κληθεί με παράμετρο τη ρίζα του δέντρου, απεικονίζει στο image του αντικειμένου IndexImage όλα τα Rectangles των φύλλων του δέντρου και με κόκκινο χρώμα όλους τους χρήστες του δέντρου.

- `public void **redraw**(IndexNode root);`

  Η μέθοδος αυτή καθαρίζει την περιοχή σχεδίασης, γεμίζοντας την εικόνα με λευκό χρώμα, και στη συνέχεια καλεί την traverse() με παράμετρο τη ρίζα του δέντρου. Ως αποτέλεσμα, αποτυπώνει στην εικόνα όλα τα Rectangles που αντιστοιχούν σε φύλλα του δέντρου και όλους τους χρήστες που βρίσκονται εισηγμένοι στο δέντρο.

- `public void` **`writeFile`**`(String extension);`

  Η μέθοδος αυτή δημιουργεί ένα αρχείο εικόνας , τύπου extension, στο δίσκο, όπου αποτυπώνει την τρέχουσα μορφή της εικόνας image.

5 Έλεγχος – Δοκιμές

Στο παρόν κεφάλαιο παρουσιάζεται η μεθοδολογία καθώς και τα αποτελέσματα των πειραματικών δοκιμών που έγιναν για την αξιολόγηση του συστήματος. Έγιναν δοκιμές για διάφορες τιμές παραμέτρων που υπεισέρχονται στην αποδοτικότητα του συστήματος και προέκυψαν ενδιαφέροντα αποτελέσματα τα οποία παρουσιάζουμε στη συνέχεια.

5.1 Μεθοδολογία Ελέγχου

Για την εκτίμηση της αποδοτικότητας του συστήματος που δημιουργήθηκε, σχεδιάσαμε και εκτελέσαμε μια σειρά από πειράματα, μεταβάλλοντας κάθε φορά τις παραμέτρους που επηρεάζουν τη λειτουργία του κεντρικού εξυπηρετητή. Στόχος των δοκιμών αυτών είναι να διαπιστωθεί η επίδραση που έχουν οι διάφορες παράμετροι του συστήματος στην αποδοτικότητα του και να εκτιμηθούν οι τιμές των παραμέτρων αυτών που εξασφαλίζουν βέλτιστη λειτουργία στο σύστημα καθώς και οι ακρότατες τιμές των παραμέτρων που μπορεί να υποστηρίξει το σύστημα. Κύριο αντικείμενο των πειραμάτων είναι η «καρδιά» του συστήματος, δηλαδή το SPI-tree, και ως εκ τούτου δεν ασχολούμαστε με περιορισμούς που υπεισέρχονται στο σύστημα εξαιτίας του δικτύου μέσα στο οποίο λειτουργεί ή της βάσης δεδομένων στην οποία αποθηκεύουμε επικουρικά δεδομένα.

Η εκτέλεση των πειραμάτων έγινε σε υπολογιστή Apple PowerMac G5 Dual με δύο επεξεργαστές PowerPC G5 χρονισμένους στα 2.5 Ghz και με 2.5 GB μνήμης, με λειτουργικό σύστημα Apple Mac OSX έκδοση 10.4.5κ αι Java Virtual Machine έκδοσης 1.5.0_06.

Οι σημαντικότερες παράμετροι που υπεισέρχονται στη λειτουργία του συστήματος είναι οι ακόλουθες:

size : Το μέγεθος του συνολικού χώρου που ευρετηριάζεται. Θεωρούμε ότι ο χώρος είναι τετράγωνος και size είναι το μέγεθος της πλευράς του.

users : Ο συνολικός αριθμός των χρηστών που είναι ενεργοί στο σύστημα.

bucket_size : Το μέγεθος (σε XPath ερωτήματα, που εδώ ταυτίζεται και με τον αριθμό χρηστών) του NFA ενός φύλλου του SPI δέντρου. Εδώ ισοδυναμεί με το μέγιστο αριθμό χρηστών που μπορούν να αποθηκευτούν σε ένα φύλλο.

max_height : Το μέγιστο ύψος του SPI δέντρου (όταν το δέντρο φτάσει αυτό το ύψος, τα φύλλα του μπορούν να αποθηκεύουν περισσότερους από bucket_size χρήστες).

user_threads : Ο αριθμός των νημάτων (threads) που μετακινούν τους χρήστες στο σύστημα.

document_threads : Ο αριθμός των νημάτων (threads) που υποβάλλουν ερωτήματα στο σύστημα.

distance: Η μέγιστη μετακίνηση ενός χρήστη, ως ποσοστό του μήκους όλου του χώρου

range : Η μέγιστη ακτίνα αποστολής ενός εγγράφου σε range query, ως ποσοστό του μήκους της διαγωνίου του χώρου

Σε κάθε πειραματική δοκιμή, αρχικά κατασκευάζεται το SPI tree με βάση τις τιμές των size, bucket_size και max_height. Στη συνέχεια δημιουργούνται και εισάγονται στο δέντρο οι χρήστες (το πλήθος τους καθορίζεται από την παράμετρο users), των οποίων τα αναγνωριστικά (*id*'s) είναι συνεχόμενοι ακέραιοι, δηλαδή ο πρώτος εισαγόμενος χρήστης έχει *id = 1* και ο τελευταίος έχει *id = users*. Το προφίλ του καθενός από τους χρήστες, δηλαδή το XPath ερώτημα που αντιστοιχεί στον κάθε χρήστη, παράγεται τυχαία με βάση κάποιο Document Type Definition (DTD). Στις πειραματικές δοκιμές που παρουσιάζονται παρακάτω χρησιμοποιούμε το News Industry Text Format Version 2.5 (http://www.nitf.org). Μερικά τυχαία XPath ερωτήματα που παράγονται με βάση αυτό φαίνονται παρακάτω:

```
//title
/nitf/head/*
/nitf/body/body.end/bibliography
/nitf/head/revision-history
/nitf/body/body.head/rights//rights.enddate
/nitf/*//pubdata
/nitf/*/docdata/doc.copyright
/nitf/body/body.end/tagline/virtloc/alt-code
/nitf[@class=4][head//meta]/*//credit/lang
/*//body.end/bibliography
/nitf[@baselang=2]//event
//nitf//body.head/*[text()=9]/*/*
```

```
/nitf[@class=9]/body[@class=3][@style=2]//body.head
/*/head/tobject/tobject.subject[@tobject.subject.matter=2]
//body[@background=4]/body.end/tagline/event//alt-code
/nitf[@baselang=2]/*//br
/nitf[@change.time=1]/head/*/tobject.property
/*/body[@style=14]/body.head/hedline/hl2[@class=3]/chron[@id=13]
/nitf[@uno=9]/body/*/hl2[@dir=4][money]/function/alt-code
/nitf[@change.date=1]/body//br
/nitf//body[@background=2]/body.head/rights/rights.owner
```

Η αρχική θέση στην οποίο εισάγεται κάθε χρήστης στο σύστημα αποφασίζεται επίσης με τυχαίο τρόπο, δηλαδή η αρχική τετμημένη και τεταγμένη του χρήστη είναι τυχαίοι αριθμοί από 0 ως size, που ακολουθούν την κανονική κατανομή, με αποτέλεσμα οι χρήστες να είναι αρχικά ομοιόμορφα κατανεμημένοι στο χώρο.

Στη συνέχεια δημιουργούνται δύο FIFO ουρές, που εξομοιώνουν τις ουρές αναμονής στο πραγματικό σύστημα για τη μετακίνηση των χρηστών και την εκτέλεση ερωτημάτων. Συγκεκριμένα, στην πρώτη ουρά (που ονομάζουμε *Ουρά Μετακινήσεων*) φθάνουν το id και οι συντεταγμένες της νέας θέσης των χρηστών που έχουν μετακινηθεί. Για το σκοπό αυτό δημιουργείται ένα ανεξάρτητο νήμα, το οποίο εκτελεί έναν ατέρμονα βρόχο, σε κάθε επανάληψη του οποίου:

α) Εκτελεί καθυστέρηση για χρόνο $-\mu_{User} \cdot \ln U$ όπου U ψευδοτυχαίος αριθμός και μ_{User} ο μέσος χρόνος μεταξύ δύο αφίξεων.

β) Επιλέγει τυχαία έναν από τους χρήστες του συστήματος και μια μετακίνηση προς τυχαία κατεύθυνση και για απόσταση που ισούται με τυχαία τιμή του διαστήματος $\left[0, \frac{size}{1000}\right]$.

γ) Εισάγει τη μετακίνηση αυτή στην Ουρά Μετακινήσεων.

Παρατηρούμε ότι με αυτό τον τρόπο, ο χρόνος που μεσολαβεί μεταξύ δύο αφίξεων για εξυπηρέτηση μετακίνησης χρήστη είναι τυχαία μεταβλητή που ακολουθεί την εκθετική κατανομή (βλ. Knu). Επομένως, οι αφίξεις (αιτήσεων προς μετακίνηση) στο σύστημα ακολουθούν κατανομή Poisson.

Με το ίδιο σκεπτικό, στη δεύτερη ουρά (που ονομάζουμε *Ουρά Ερωτημάτων*) φθάνουν το XML έγγραφο και η περιοχή όπου θέλουμε να γίνει το range query. Για το σκοπό αυτό δημιουργείται ένα ανεξάρτητο νήμα, το οποίο εκτελεί έναν ατέρμονα βρόχο, σε κάθε επανάληψη του οποίου:

α) Εκτελεί καθυστέρηση για χρόνο $-\mu_{Query} \cdot \ln U$ όπου U ψευδοτυχαίος αριθμός και μ_{Query} ο μέσος χρόνος μεταξύ δύο αφίξεων

β) Επιλέγει τυχαία ένα XML έγγραφο που συμφωνεί με το NITF DTD, από 10 διαφορετικά που υπάρχουν σε φάκελο του συστήματος. Παράδειγμα τέτοιου εγγράφου είναι το ακόλουθο:

```
<nitf baselang="14" change.date="1" change.time="1" id="17" version="1">
      <head id="9">
           <meta content="10" http-equiv="1" id="15"></meta>
           <meta content="1" http-equiv="1"></meta>
           <meta content="4"></meta>
           <tobject>
           <tobject.property tobject.property.type="15"> </tobject.property>
           <tobject.property tobject.property.type="2"> </tobject.property>
           <tobject.property tobject.property.type="1"> </tobject.property>
           <tobject.subject tobject.subject.code="13"
           tobject.subject.refnum="17"> </tobject.subject>
           </tobject>
           <iim id="13" ver="14"></iim>
           <docdata id="14">
           <key-list>
           <keyword id="6"></keyword>
           <keyword id="4"></keyword>
           <keyword></keyword>
           </key-list>
           <correction id="8" id-string="10" reg-src="17"></correction>
           </docdata>
           <pubdata date.publication="11" edition.area="20" ex-ref="16">
           </pubdata>
     </head>
     <body background="8" class="2" dir="1" id="2">
           <body.content> News Content… </body.content>
     </body>
</nitf>
```

γ) Επιλέγονται τυχαία οι συντεταγμένες ενός σημείου του χώρου (με τον ίδιο τρόπο που επιλέγεται η αρχική θέση ενός χρήστη). Το σημείο αυτό είναι το κέντρο της κυκλικής περιοχής του range query. Επίσης επιλέγεται τυχαία η ακτίνα της παραπάνω κυκλικής περιοχής, ως ένας τυχαίος αριθμός του διαστήματος $[0, range]$ και εκτελείται το range query.

δ) Εισάγει τη μετακίνηση αυτή στην Ουρά Ερωτημάτων.

Όπως και για τις μετακινήσεις των χρηστών, ο χρόνος που μεσολαβεί μεταξύ δύο αφίξεων για εξυπηρέτηση ερωτήματος παρόχου είναι τυχαία μεταβλητή που ακολουθεί την εκθετική κατανομή και ως εκ τούτου οι αφίξεις (ερωτημάτων προς αποτίμηση) στο σύστημα ακολουθούν κατανομή Poisson.

Στη συνέχεια, δημιουργείται *user_threads* αριθμός από ανεξάρτητα νήματα, καθένα από τα οποία εκτελεί ατέρμονα βρόχο, σε κάθε επανάληψη του οποίου διαβάζει την κεφαλή της ουράς Μετακινήσεων και την αφαιρεί (από την ουρά) και στη συνέχεια εκτελεί την περιγραφόμενη από την κεφαλή μετακίνηση. Εντελώς αντίστοιχα, δημιουργείται *document_threads* αριθμός από ανεξάρτητα νήματα, καθένα από τα οποία εκτελεί ατέρμονα βρόχο, σε κάθε επανάληψη του οποίου διαβάζει την κεφαλή της ουράς Ερωτημάτων και την αφαιρεί (από την ουρά) και στη συνέχεια εκτελεί το ερώτημα που περιγράφει η κεφαλή της ουράς.

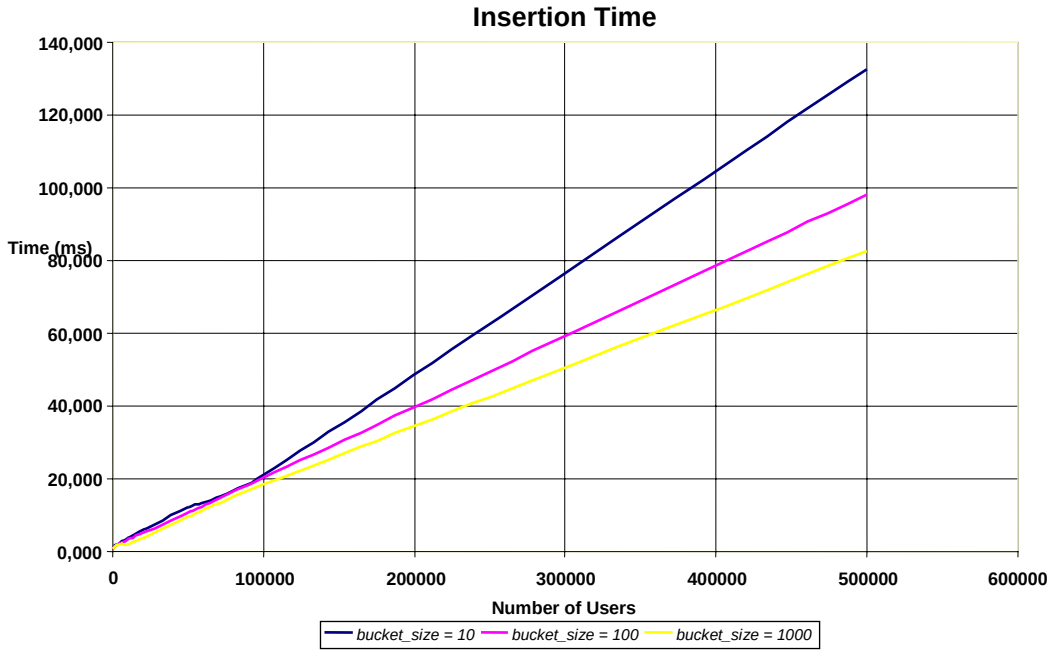
Ακολουθούν τα αποτελέσματα των πειραματικών δοκιμών.

5.2 Αποτελέσματα Δοκιμών

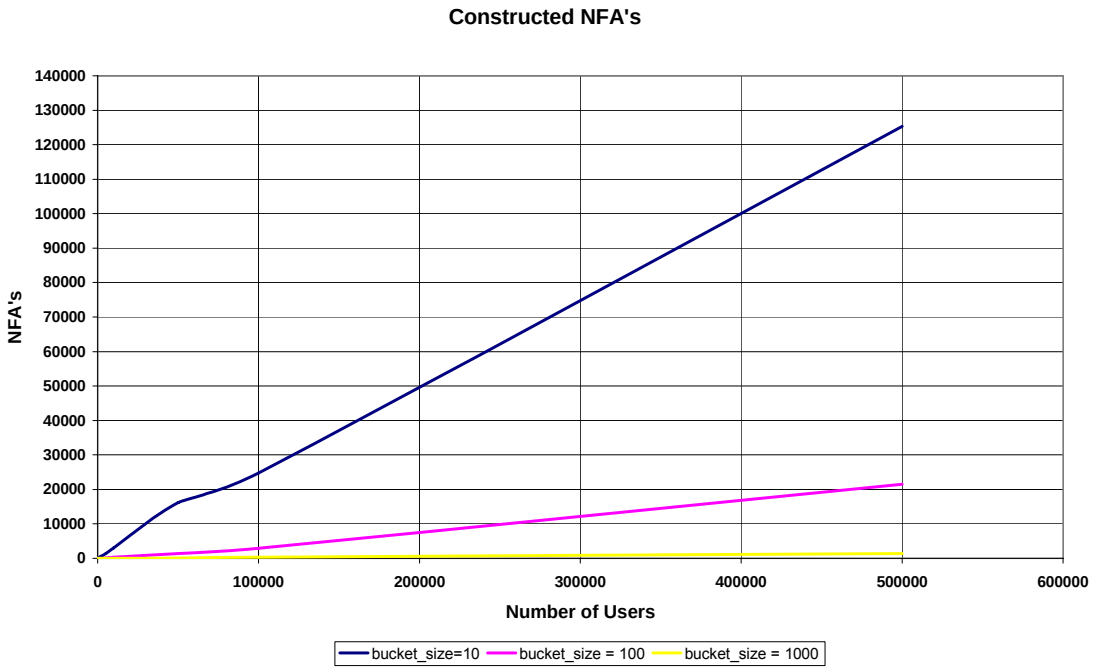
5.2.1 Δοκιμές Εισαγωγής

Αρχικά εκτελέσαμε δοκιμές για να μετρήσουμε την απόδοση του συστήματος στην εισαγωγή χρηστών. Το μέγεθος του χώρου είναι σε όλες τις περιπτώσεις 1024. Οι εισαγωγές γίνονται από ένα νήμα, ενώ δε γίνεται κανένα ερώτημα προς το σύστημα. Το μέγιστο ύψος του δέντρου είναι 10. Οι δοκιμές γίνονται για αριθμό χρηστών 100, 500, 1000, 5000, 10000, 50000, 100000 και 500000 και χωρητικότητα φύλλου (bucket_size) 10, 100 και 1000.

Σε κάθε περίπτωση, εισάγουμε όλους τους χρήστες επαναληπτικά στο δέντρο και μετράμε α) το χρόνο που χρειάστηκε το σύστημα για τις εισαγωγές και β) το συνολικό αριθμό από NFA που χρειάστηκε να κατασκευαστούν.



Σχήμα 65: Χρόνος εισαγωγής χρηστών για μεταβλητό μέγεθος κουβά



Σχήμα 66: NFA που κατασκευάζονται με την εισαγωγή χρηστών για μεταβλητό μέγεθος κουβά

Από τις ανωτέρω δοκιμές φαίνεται ότι ο χρόνος εισαγωγής των χρηστών αυξάνεται μόνο γραμμικά σε σχέση με το πλήθος τους. Είναι επίσης εμφανές ότι παρά τη μεγάλη μείωση του αριθμού των NFA όταν αυξάνεται η χωρητικότητα του φύλλου, η μείωση του χρόνου εισαγωγής δεν είναι εξίσου θεαματική, γεγονός που σημαίνει ότι κατά την εισαγωγή ενός χρήστη, ο χρόνος για την εισαγωγή του προφίλ του στο NFA του κατάλληλου φύλλου είναι της ίδιας τάξης

μεγέθους με το χρόνο που απαιτείται για τον εντοπισμό και το κλείδωμα του κατάλληλου φύλλου.

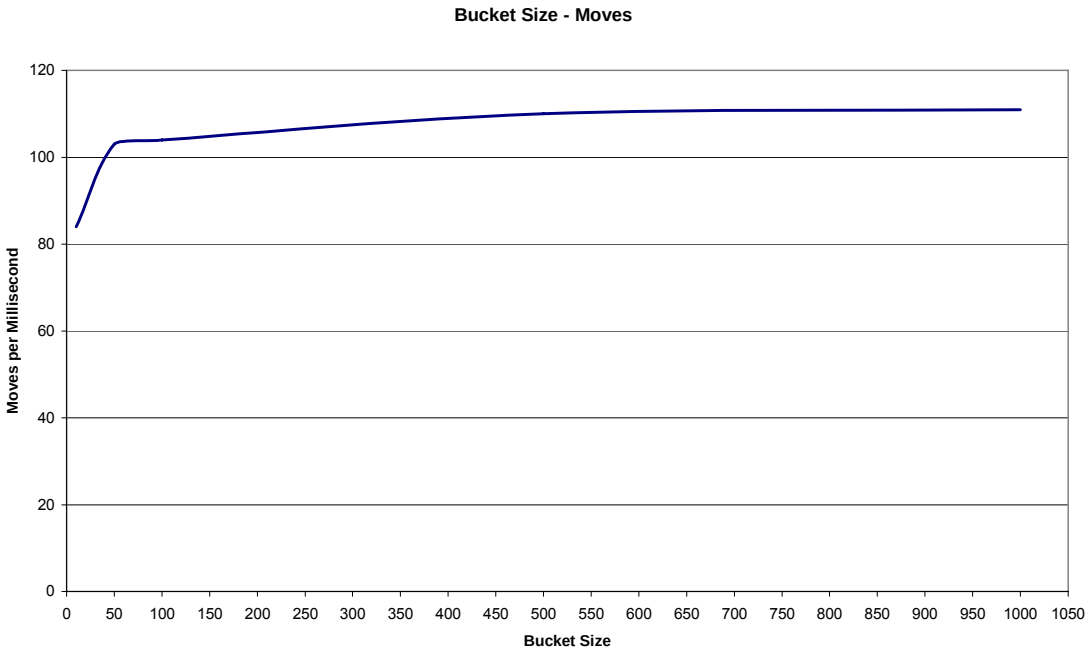
Σημειώνεται επίσης ότι για το μέγεθος του SPI-Tree μετά την είσοδο 500000 χρηστών σε αυτό ήταν περίπου 1.25 GB, γεγονός που υποδηλώνει ότι η μνήμη που καταλαμβάνεται από τη δομή επιτρέπει την παροχή των υπηρεσιών σε μεγάλη κλίμακα.

5.2.2 Δοκιμές Μετακίνησης Χρηστών

Στη συνέχεια εκτελέσαμε δοκιμές με στόχο την εκτίμηση της αποδοτικότητας του συστήματος στις μετακινήσεις χρηστών. Στις περιπτώσεις αυτές χρησιμοποιήσαμε ένα νήμα το οποίο περιοδικά «βλέπει» πόσες μετακινήσεις έχουν εξυπηρετηθεί και υπολογίζει το μέσο αριθμό εξυπηρέτησης ανά millisecond.

Αρχικά, χρησιμοποιήσαμε 100000 χρήστες σε δέντρο με ύψος 15. Οι χρήστες μετακινούνταν από 10 νήματα, ενώ κάθε μετακίνηση είχε μέγιστη απόσταση 0,001 του μεγέθους του χώρου.

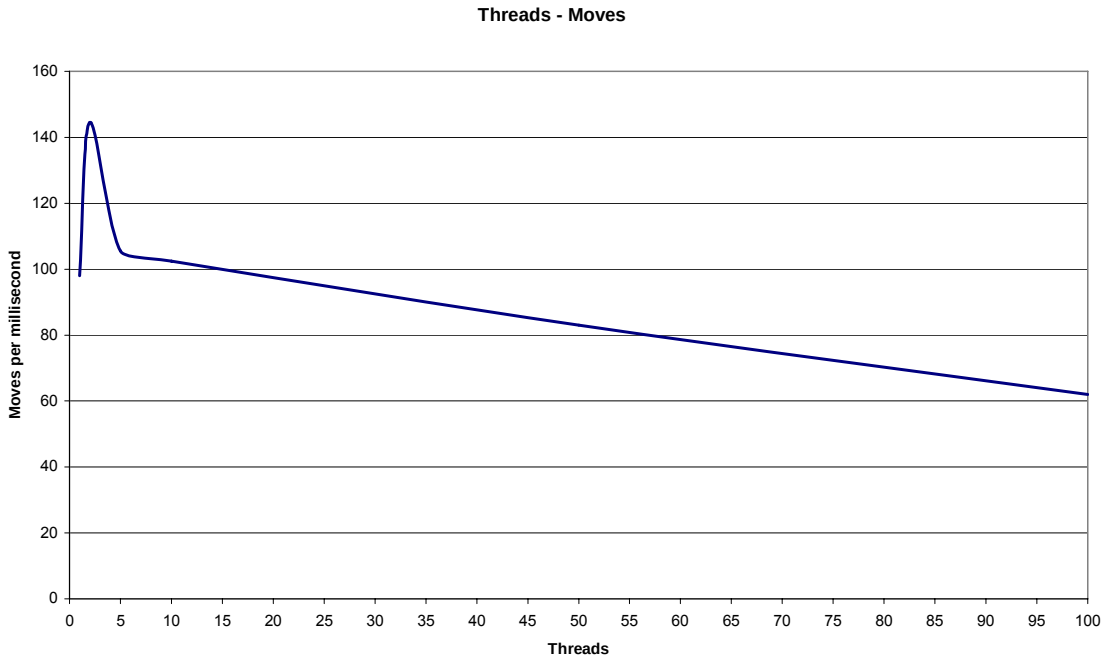
Μετρήσαμε το μέσο αριθμό μετακινήσεων που εξυπηρετήθηκαν όταν η χωρητικότητα του φύλλου του δέντρου ήταν 10, 50 , 100, 500 και 1000. Πήραμε τα ακόλουθα αποτελέσματα:



Σχήμα 67: Κινήσεις – Μέγεθος κουβά

Από το παραπάνω διάγραμμα φαίνεται ότι για τόσο μεγάλο αριθμό χρηστών, η πολύ μικρή χωρητικότητα του φύλλου έχει εμφανείς αρνητικές συνέπειες. Αυτό είναι αναμενόμενο λόγω των συχνότερων διασπάσεων και συμπτύξεων του δέντρου. Παρατηρούμε επίσης ότι από ένα σημείο και μετά, περίπου όταν η χωρητικότητα είναι 50, η αύξησή της δεν επιφέρει ουσιαστικές συνέπειες στη λειτουργία του δέντρου.

Στη συνέχεια, εστιάσαμε στην επίδραση που έχει ο αριθμός των νημάτων (threads) στην αποδοτικότητα των μετακινήσεων. Για 100000 χρήστες, με χωρητικότητα φύλλου 50 χρήστες, μέγιστο βάθος δέντρου 15 και μέγιστη απόσταση μετακίνησης 0.001 του μεγέθους του συνολικού χώρου, μετρήσαμε το μέσο αριθμό μετακινήσεων στη μονάδα του χρόνου, όταν τις μετακινήσεις εκτελούν παράλληλα 1, 2, 5, 10, 50 και 100 threads. Πήραμε τη γραφική παράσταση που φαίνεται στο Σχήμα 68.



Σχήμα 68: Κινήσεις – Αριθμός νημάτων

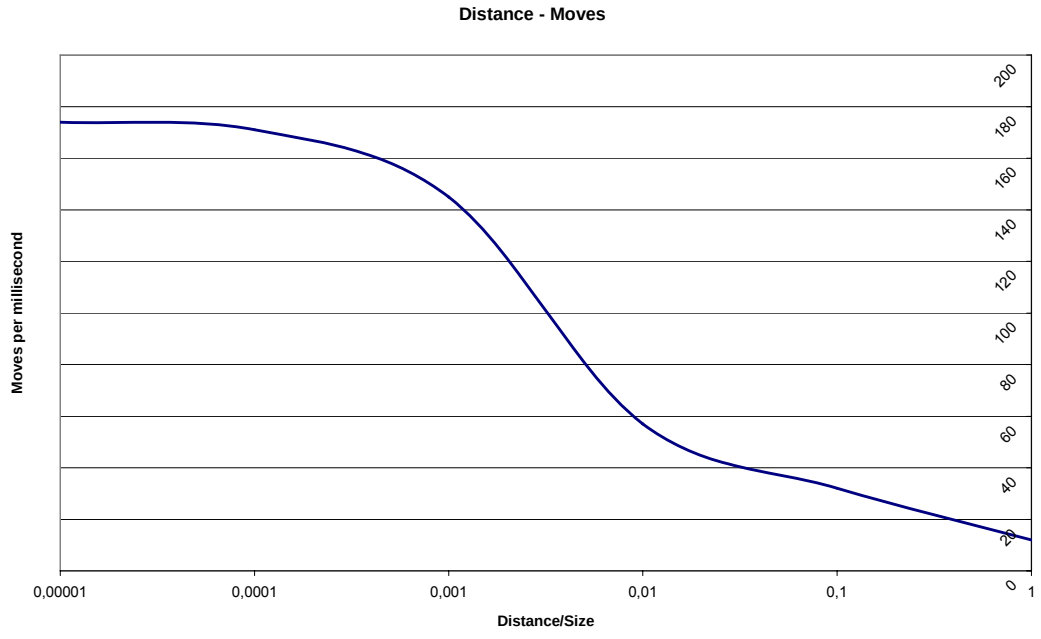
Δεδομένου ότι οι δοκιμές έγιναν σε υπολογιστή με δύο επεξεργαστές, είναι αναμενόμενο μέγιστη απόδοση να παρατηρείται για αριθμό νημάτων γύρω από το 2. Σε μεγαλύτερο πλήθος νημάτων έχουμε την επιβάρυνση της εναλλαγής μεταξύ των νημάτων και κυρίως των αντικρουόμενων κλειδωμάτων, γεγονός που επιβαρύνει σημαντικά την αποδοτικότητα του συστήματος. Βλέπουμε πάντως ότι η χρήση παράλληλων τεχνικών και κλειδωμάτων μας εξασφαλίζει 45% περίπου καλύτερη ταχύτητα όταν έχουμε δύο νήματα, απ' ότι αν τρέχαμε μόνο ένα νήμα.

Χρησιμοποιώντας δύο νήματα και τις ίδιες υπόλοιπες ρυθμίσεις, μελετήσαμε τη συμπεριφορά των μετακινήσεων όταν μεταβάλλαμε το μέγιστο ύψος του δέντρου σε 5, 10, 15 20:

Σχήμα 69: Κινήσεις – Μέγιστο ύψος δέντρου

Η συμπεριφορά αυτή δικαιολογείται από το γεγονός ότι σε πολύ μικρό ύψος δέντρου έχουμε πολύ μεγάλα φύλλα.

Κατόπιν, μελετούμε την επίδραση της απόστασης της μετακίνησης στην απόδοση του συστήματος. Με 100000 χρήστες, μέγεθος χώρου 1024, μέγιστο βάθος δέντρου 15, νήματα 2 και χωρητικότητα φύλλου 50 δοκιμάζουμε ως μέγιστη απόσταση μετακίνησης του κάθε χρήστη το 0.00001, 0.0001, 0.001, 0.01, 0.1 και 1 της πλευράς του ολικού χώρου και μετράμε το μέσο ρυθμό εξυπηρέτησης των μετακινήσεων. Το διάγραμμα φαίνεται στο Σχήμα 70.



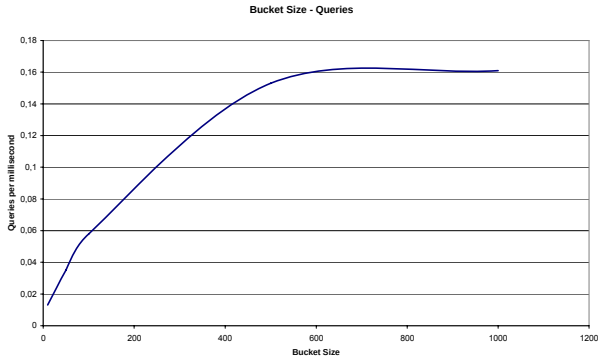
Σχήμα 70: Κινήσεις – Μέγιστη απόσταση μετακίνησης χρηστών

Προφανώς, όσο μεγαλύτερη γίνεται η μέγιστη απόσταση μετακίνησης, τόσο αυξάνεται και η μέση απόσταση μετακίνησης. Αυτό όμως έχει ως αποτέλεσμα κάθε μετακίνηση να έχει πολύ μεγαλύτερη πιθανότητα να απαιτεί πρόσβαση σε κάποιον κόμβο που βρίσκεται πιο κοντά στη ρίζα. Ως εκ τούτου απαιτείται αφενός περισσότερος χρόνος για την πρόσβαση στον κόμβο-προορισμό, όσο και κλείδωμα μεγαλύτερου κομματιού του δέντρου, απ' ότι για μια μετακίνηση μικρότερης απόστασης. Για το λόγο αυτό και οι μετακινήσεις με μεγαλύτερη μέση απόσταση απαιτούν περισσότερο χρόνο, όπως φαίνεται ξεκάθαρα στο παραπάνω σχήμα. Στο σχήμα φαίνεται ότι όσο μικραίνει η μέση απόσταση των μετακινήσεων, τόσο αποδοτικότερα εξυπηρετούνται οι μετακινήσεις. Ωστόσο, από μια τιμή της απόστασης (0.00001) και κάτω δε βλέπουμε ουσιαστική διαφορά στην απόδοση, καθώς από την απόσταση αυτή και κάτω όλες σχεδόν οι μετακινήσεις γίνονται εντός του φύλλου στο οποίο βρίσκεται ο χρήστης.

5.2.3 Δοκιμές Μετακίνησης Χρηστών

Στη συνέχεια εκτελέσαμε δοκιμές για να εκτιμήσουμε την αποδοτικότητα του συστήματος στην εκτέλεση ερωτημάτων. Στις δοκιμές αυτές ασχολούμαστε αποκλειστικά με ερωτήματα περιοχής (η αποδοτικότητα των ερωτημάτων κοντινότερου γείτονα εξαρτάται από αυτήν). Στις παρακάτω δοκιμές οι χρήστες του συστήματος, αφού εισαχθούν στο SPI Tree, στη συνέχεια παραμένουν ακίνητοι. . Στις περιπτώσεις αυτές χρησιμοποιήσαμε ένα νήμα το οποίο περιοδικά «βλέπει» πόσα ερωτήματα έχουν εξυπηρετηθεί και υπολογίζει το μέσο αριθμό εξυπηρέτησης ανά millisecond.

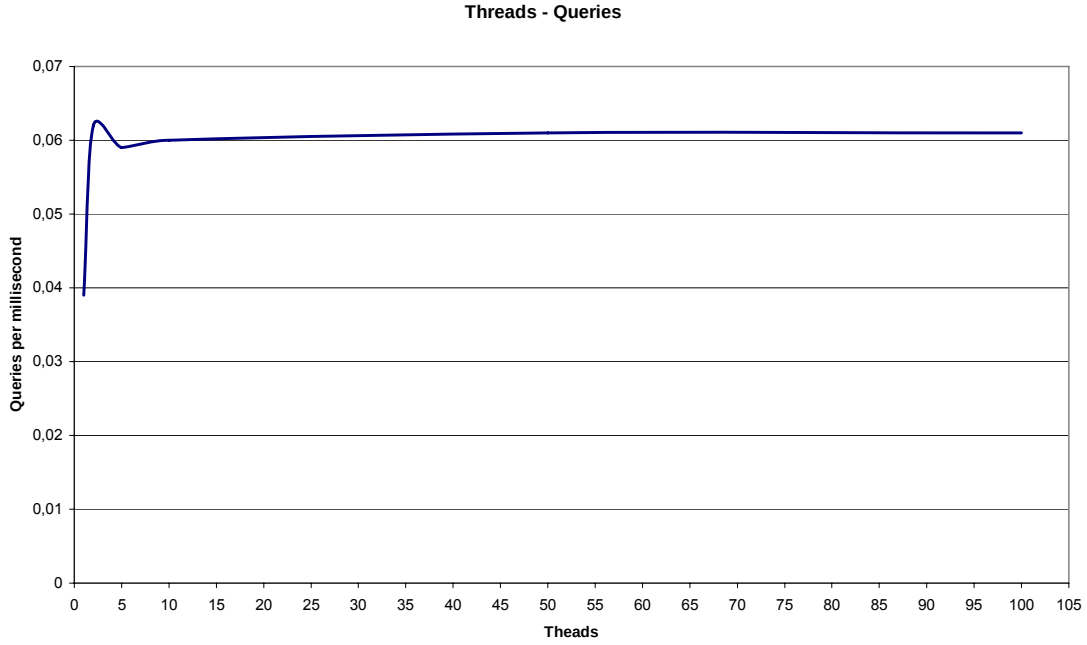
Αρχικά, μελετούμε τη συμπεριφορά του συστήματος σε σχέση με τη χωρητικότητα των φύλλων του δέντρου, για τις χωρητικότητες 10, 50, 100, 500, 1000. Στο σύστημα έχουμε εισάγει 100000 χρήστες, το μέγιστο βάθος του δέντρου είναι 15, ενώ χρησιμοποιούμε 2 νήματα που εκτελούν ερωτήματα περιοχής με μέγιστη ακτίνα το 1/20 της διαγωνίου του χώρου. Η καμπύλη που παίρνουμε φαίνεται παρακάτω:



Σχήμα 71: Δημοσιεύσεις – Μέγεθος κουβά

Είναι εμφανές ότι όσο μεγαλύτερη είναι η χωρητικότητα των φύλλων, τόσο μικρότερος αριθμός από NFA πρέπει να εκτελεστούν για την αποτίμηση ενός ερωτήματος σε συγκεκριμένη γεωγραφικά περιοχή. Βέβαια καθένα από τα NFA αυτά είναι σημαντικά μεγαλύτερο σε μέγεθος. Αυτό όμως δεν έχει ιδιαίτερη σημασία, καθώς τα NFA εκμεταλλεύονται κοινά προθέματα των προφίλ, με αποτέλεσμα να μη σημειώνεται «έκρηξη» στον αριθμό των καταστάσεων τους. Με βάση αυτά, είναι αναμενόμενο το γεγονός ότι η αύξηση της χωρητικότητας των φύλλων συνεπάγεται μεγάλη αύξηση της αποδοτικότητας, τουλάχιστον μέχρι την τιμή των 500 χρηστών ανά φύλλο.

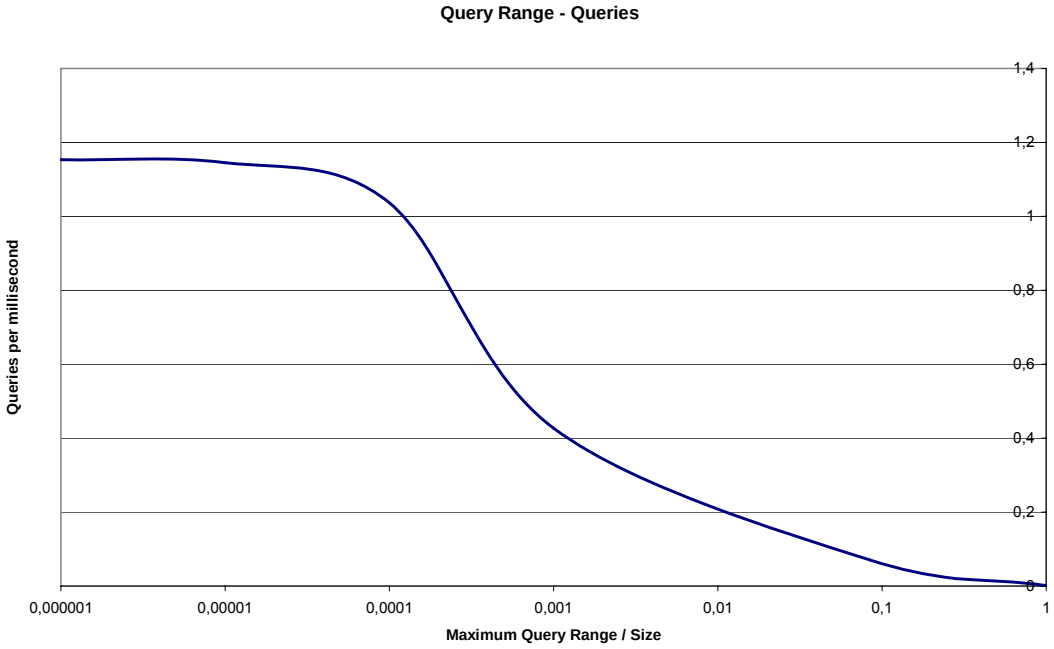
Στη συνέχεια εκτελέσαμε δοκιμές για την εκτίμηση της αποδοτικότητας του συστήματος σε σχέση με τον αριθμό των νημάτων που εκτελούν ταυτόχρονα ερωτήματα περιοχής. Μελετήθηκε η συμπεριφορά του συστήματος για αριθμό νημάτων 1, 2, 5, 10, 50 ,100. Οι εκτελέσεις έγιναν για μέγεθος κόσμου 1024, αριθμό χρηστών 100000, μέγιστο ύψος δέντρου 15, χωρητικότητα φύλλου 100 και μέγιστη ακτίνα περιοχής του ερωτήματος ίση με το 1/20 της διαγωνίου του κόσμου. Πήραμε το ακόλουθο:



Σχήμα 72: Δημοσιεύσεις - Νήματα

Όπως είναι αναμενόμενο, βέλτιστη απόδοση έχουμε στις τιμές γύρω από το 2 (όσες και οι επεξεργαστικές μονάδες του συστήματος). Παρατηρούμε ότι για σημαντικά μεγαλύτερο αριθμό νημάτων, η απόδοση δεν φθίνει με μεγάλους ρυθμούς, όπως στην περίπτωση της μετακίνησης χρηστών. Αυτό οφείλεται κυρίως στο γεγονός ότι η εκτέλεση των ερωτημάτων κλειδώνει τελικά μόνο το φύλλο στο οποίο φτάνει και όχι κάποιον από τους παραπάνω κόμβους (όπως η μετακίνηση), με αποτέλεσμα οι αναγνώσεις στα γειτονικά φύλλα να γίνονται ανεμπόδιστα από άλλα νήματα.

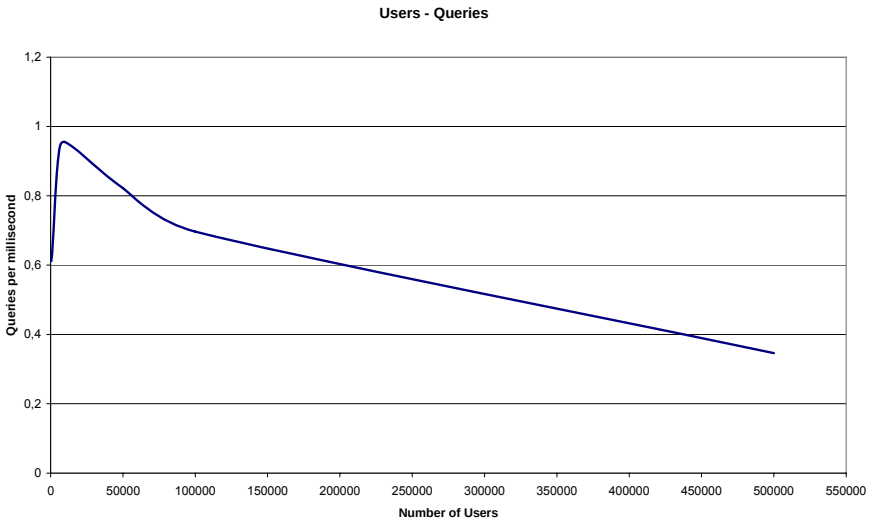
Ακολουθεί γραφική παράσταση των ερωτημάτων που εκτελούνται ανά millisecond σε σχέση με τη μέγιστη ακτίνα περιοχής σε κάθε ερώτημα περιοχής. Εκτελέσαμε μετρήσεις για μέγιστη ακτίνα ίση με τη διαγώνιο του κόσμου (range = 1), ίση με το 0.1 της διαγωνίου, το 0.01 της διαγωνίου, το 0.001 της διαγωνίου, το 0.0001 της διαγωνίου, το 0.00001 της διαγωνίου και το 0.000001 της διαγωνίου. Τα ερωτήματα γίνονταν από 2 νήματα σε συνολικό τετραγωνικό χώρο πλευράς 1024 με 100000 χρήστες, χωρητικότητα φύλλου 100 και μέγιστο βάθος δέντρου 15. Πήραμε το ακόλουθο:



Σχήμα 73: Δημοσιεύσεις – Χωρικό εύρος δημοσίευσης

Όπως είναι αναμενόμενο, για μέγεθος περιοχής της τάξης του μεγέθους του συνολικού χώρου, κάθε ερώτημα απαιτεί πολύ χρόνο για να εκτελεστεί, αφού πρέπει να εκτελεστούν όλα τα NFA του συστήματος. Όσο η μέγιστη ακτίνα του κάθε ερωτήματος μικραίνει, η αποδοτικότητα του συστήματος αυξάνεται, μέχρι την τιμή του 0,000001 της διαγωνίου όπου σταθεροποιείται, αφού για τόσο μικρή απόσταση, το ερώτημα φτάνει σε ένα μόνο NFA. Είναι εμφανές ότι η μέγιστη ακτίνα του κάθε ερωτήματος είναι πολύ καθοριστικός παράγοντας για την αποδοτικότητά του, καθώς και ότι για ρεαλιστικές τιμές της μέγιστης ακτίνας (μικρότερες του 1/1000 του κόσμου) η αποδοτικότητα του συστήματος κυμαίνεται σε πολύ ικανοποιητικά επίπεδα.

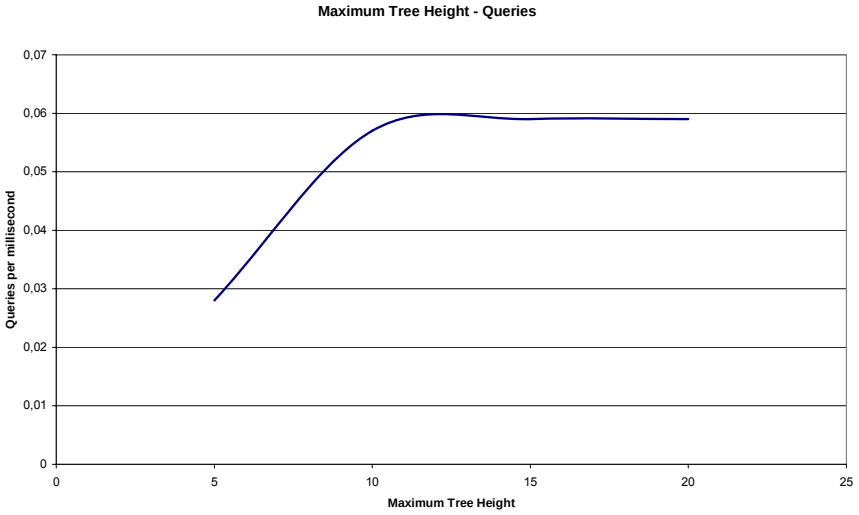
Κατόπιν μελετάμε την αποδοτικότητα του συστήματος σε σχέση με τον αριθμό των χρηστών που βρίσκονται εισηγμένοι στο σύστημα. Ο κόσμος έχει μέγεθος 1024 x 1024, η χωρητικότητα του φύλλου είναι 100, το μέγιστο βάθος του δέντρου 15, ενώ 2 νήματα εκτελούν ερωτήματα και η μέγιστη ακτίνα του κάθε ερωτήματος είναι ίση με το 0,005 της διαγωνίου. Εκτελέσαμε πειράματα για πληθυσμό χρηστών 100, 500, 1000, 5000, 10000, 50000, 100000 και 500000. Πήραμε τα ακόλουθα αποτελέσματα:



Σχήμα 74: Δημοσιεύσεις – Αριθμός χρηστών

Από τα αποτελέσματα είναι εμφανές ότι το πλήθος των χρηστών και άρα η πυκνότητά του στο χώρο επηρεάζει την αποδοτικότητα του συστήματος στην εκτέλεση ερωτημάτων. Για πολύ μικρό πλήθος, η μειωμένη αποδοτικότητα εμφανίζεται λόγω του πολύ μικρού ύψους του δέντρου (1 ή 2) και άρα των πολλών μπλοκαρισμάτων λόγω κλειδωμάτων. Σε μεγάλες τιμές πλήθους χρηστών η μείωση της αποδοτικότητας οφείλεται στο μεγάλο αριθμό χρηστών που υπάρχει σε κάθε NFA, και άρα στο μεγάλο αριθμό καταστάσεων κατά την εκτέλεσή του.

Ακολουθεί η δοκιμή του συστήματος σε εκτελέσεις ερωτημάτων σε σχέση με το μέγιστο βάθος του δέντρου. Μελετήθηκε η συμπεριφορά του συστήματος για μέγιστο βάθος δέντρου 5, 10, 15, 20. Οι εκτελέσεις έγιναν για μέγεθος κόσμου 1024, αριθμό χρηστών 100000, χωρητικότητα φύλλου 100 και μέγιστη ακτίνα περιοχής του ερωτήματος ίση με το 1/20 της διαγωνίου του κόσμου, ενώ τα ερωτήματα εκτελούσαν ταυτόχρονα 2 threads. Τα αποτελέσματα φαίνονται παρακάτω:



Σχήμα 75: Δημοσιεύσεις – Μέγιστο ύψος δέντρου

Εντελώς ενδεικτικά, μετρήσαμε ότι για μέγεθος χώρου 1024, 100000 χρήστες, μέγιστο βάθος δέντρου 15, χωρητικότητα φύλλου 100, μέγιστη μετακίνηση 0.0001 του size και μέγιστη ακτίνα ερωτήματος 0.001 της διαγωνίου, το σύστημα μπορεί και εξυπηρετεί 110.000 μετακινήσεις χρηστών το δευτερόλεπτο και ταυτόχρονα να εκτελεί περίπου 610 ερωτήματα το δευτερόλεπτο.

5.3 Συμπεράσματα

Από τα αποτελέσματα των δοκιμών που προηγήθηκαν, προκύπτει το συμπέρασμα ότι SPI Tree, και γενικότερα ολόκληρο το σύστημα είναι ικανό να παρέχει Profile and Location Based Services σε μεγάλη κλίμακα. Υπάρχουν φυσικά πολλές παράμετροι που πρέπει να ληφθούν υπόψη για την αποδοτικότερη λειτουργία του. Με δεδομένη την απαίτηση για εξυπηρέτηση συγκεκριμένου πλήθους χρηστών, πολύ σημαντικό ρόλο παίζει η χωρητικότητα του φύλλου του SPI-Tree. Για πληθυσμό 50.000 – 100000 χρηστών η καλύτερη τιμή φαίνεται να είναι μερικές δεκάδες (π.χ. 50 ή 100). Σημαντικό ρόλο παίζει και το μέγιστο ύψος δέντρου, που για τέτοιο πλήθος χρηστών πρέπει να είναι τουλάχιστο 10 ή 12. Ο αριθμός των νημάτων που θα χρησιμοποιήσουμε πρέπει να είναι της τάξης μεγέθους του αριθμού των επεξεργαστών του συστήματος που χρησιμοποιούμε. Είναι εμφανές όμως ότι πρωτεύοντα ρόλο στην απόδοση του συστήματος παίζει α) η απόσταση κατά την οποία μετακινούνται οι χρήστες σε κάθε τους μετακίνηση και β) το μέγιστο εύρος κάθε ερωτήματος περιοχής. Σε ότι αφορά την απόσταση, αυτή είναι συνάρτηση τόσο της ταχύτητας με την οποία κινούνται οι χρήστες, όσο και της συχνότητας με την οποία αναφέρουν τη θέση τους στο σύστημα. Σε ότι αφορά το μέγιστο εύρος περιοχής, αυτό μπορεί να περιορίζεται από μια μέγιστη τιμή ή να περιορίζεται με έμμεσους τρόπους (π.χ. όσο μεγαλύτερο το εύρος ενός ερωτήματος κάποιου χρήστη, τόσο μεγαλύτερη να είναι η χρέωσή του). Συνολικά, βλέπουμε ότι υπάρχει πλήθος παραμέτρων με τις οποίες μπορεί να ρυθμιστεί το σύστημα ανάλογα με τις συνθήκες στις οποίες χρησιμοποιείται και να εστιάσει έτσι την απόδοσή του σε συγκεκριμένες πτυχές των υπηρεσιών που παρέχει.

6 Επίλογος

6.1 Σύνοψη και συμπεράσματα

Σε αυτήν Διπλωματική Εργασίας αναπτύξαμε μία πρότυπη Πλατφόρμας για Υπηρεσίες με Βάση τη Θέση και το Προφίλ των χρηστών τους. Για την ικανοποίηση των απαιτήσεων και των προδιαγραφών του συστήματος μεταχειριστήκαμε σύνθετες δομές δεδομένων και πρόσφατα ερευνητικά αποτελέσματα στο πεδίο του ταιριάσματος προφίλ. Συνθέτοντας όλα τα παραπάνω αναπτύξαμε μία καινούρια δομή ευρετηρίου, το SPI δέντρο, που είναι ικανή να ανταποκριθεί στις απαιτήσεις Παροχής Υπηρεσιών με Βάση τη Θέση και το Προφίλ σε μεγάλη κλίμακα.

Για την παραπάνω δομή παραθέσαμε τόσο θεωρητικά στοιχεία που δείχνουν την αποδοτικότητά της αλλά πραγματοποιήσαμε και ρεαλιστικά σενάρια προσομοίωσης στα οποία ανταποκρίθηκε με επιτυχία. Τέλος, με την υλοποίηση και της διεπαφής Υπηρεσιών Ιστού (Web Services Interface) παραδίδουμε στην εκπαιδευτική κοινότητα ένα σύστημα με ανοιχτή αρχιτεκτονική το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών και περαιτέρω έρευνα

6.2 Μελλοντικές επεκτάσεις

Κατά την εκπόνηση της Διπλωματικής Εργασίας προέκυψαν θέματα ερευνητικού ενδιαφέροντος αλλά και υλοποίησης με τα οποία δεν θα μπορούσαμε να ασχοληθούμε στα χρονικά περιθώρια που έπρεπε να ολοκληρωθεί. Μερικά από αυτά έχουν εξαιρετικό ενδιαφέρον και ενδεχομένως θα μας απασχολήσουν το προσεχές διάστημα.

Ενδιαφέρον θα παρουσίαζε η ανάλυση με τη χρήση στατιστικών μοντέλων και της θεωρίας πιθανοτήτων για τη μέση πολυπλοκότητα της λειτουργίας της ενημέρωσης ενός χρήστη. Σε αυτή θα λαμβάνεται υπόψη η πιθανότητα μία μετακίνηση να οδηγήσει σε αλλαγή κόμβου, η πρόβλεψη της μέσης διάσπασης του δέντρου και η πολυπλοκότητα της κάθε μετακίνησης.

Κορυφαίο μεταξύ των θεμάτων που χρήζουν μελλοντικής έρευνας θεωρούμε την τροποποίηση της υλοποίησης του ευρετηρίου SPI ώστε να λαμβάνονται υπόψη οι ιεραρχίες μνήμης ενός υπολογιστικού συστήματος. Αυτό θα επιφέρει μεταβολή στην οργάνωση των κουβάδων (*buckets*) ώστε να γίνεται αποδοτικά η φόρτωση τους στην κρυφή μνήμη (*cache memory*).

Στην ανάπτυξη του συστήματος παραβλέψαμε επίσης το κομμάτι της δικτυακής υποδομής, μέσω της οποίας θα μεταφέρονται οι ενημερώσεις από τους χρήστες και οι δημοσιεύσεις των παρόχων. Θα πρέπει να γίνει έρευνα στον τρόπο οργάνωσης του συστήματος γύρω από τον κεντρικό εξυπηρετητή ώστε η ροή των δεδομένων να είναι τέτοια που να αξιοποιεί πλήρως τις δυνατότητες του νέου ευρετηρίου.

Στο πεδίο των εφαρμογών, τέλος, θα πρέπει να γίνει μελέτη και ανάπτυξη Γραφικών Εφαρμογών για την προσφιλή στο χρήστη διατύπωση του προφίλ του και την εξαγωγή από αυτή τη διατύπωση των κατάλληλων XPath ερωτημάτων. Το αντίστοιχο πρέπει να γίνει και στην πλευρά των παρόχων με τις δημοσιεύσεις. Θα πρέπει να μελετηθούν επίσης μοντέλα κοστολόγησης των υπηρεσιών που μπορούν να παρέχονται από το σύστημα και αναζητηθούν νέα πεδία στα οποία μπορεί να αξιοποιηθεί.

7 Βιβλιογραφία

[AF00] Mehmet Altinel, and Michael J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In Proceedings of VLDB 2000.

[AJ00] Mehmet Altinel, Michael J. Franklin: Efficient Filtering of XML Documents for Selective Dissemination of Information. VLDB 2000: 53-64

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'90),

[BPW98] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, Carola Wenk. On Map-Matching Vehicle Tracking Data. VLDB 2005: 853-864

[BS77] R. Bayer, M. Schkolnick. Concurrency of Operations on B-Trees, Acta Informatica 9, 1-21, 1977

[CC05] Hyung-Ju Cho, Chin-Wan Chung. An Efficient and Scalable Approach to CNN Queries in a Road Network. VLDB 2005: 865-876

[CDT+00] Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. SIGMOD Conference 2000: 379-390

[CFZ01] Mitch Cherniack, Michael J. Franklin, Stan Zdonik. Expressing User Profiles For Data Recharging, IEEE Personal Communications, 2001

[CRW01] Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19(3): 332-383 (2001)

[DAF+03] Yanlei Diao, Mehmet Altinel, Michael J. Franklin, Hao Zhang, and Peter Fischer. Path Sharing and Predicate Evaluation for High-Performance XML

Filtering. TODS .

[DF03] Yanlei Diao, and Michael J. Franklin. Query Processing for High-Volume XML Message Brokering. In Proceedings of VLDB 2003.

[DF03] Yanlei Diao, and Michael J. Franklin. High-Performance XML Filtering: An Overview of YFilter. IEEE Data Engineering Bulletin, 2003.

[DFFT02] Yanlei Diao, Peter Fischer, Michael Franklin, and Raymond To. YFilter: Efficient and Scalable Filtering of XML Documents. Demo paper, in Proceedings of ICDE 2002.

[DRF04] Yanlei Diao, Shariq Rizvi, and Michael J. Franklin. Towards an Internet-Scale XML Dissemination Service. In Proceedings of VLDB 2004.

[GG97] V. Gaede and O. Gunther. Multidimensional Access Methods. ACM Computing Surveys, 1997

[Gut84] Antonin Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD Conference 1984, p. 47-57

[HS02] Gísli R. Hjaltason, Hanan Samet: Speeding up construction of PMR quadtree-based spatial indexes. VLDB Journal. 11 2002

[Knu98] Donald E. Knuth: The Art of Computer Programming, Volume II: Seminumerical Algorithms, 3rd Edition Addison-Wesley 1998

[LY81] Philip L. Lehman, S. Bing Yao. Efficient Locking for Concurrent Operations on B-Trees. ACM Transactions on Database Systems, Vol. 6, No. 4, December 1981

[MPBT05] Kyriakos Mouratidis, Dimitris Papadias, Spiridon Bakiras, Yufei Tao. "A Threshold- Based Algorithm for Continuous Monitoring of k Nearest Neighbors" IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 11, pp. 1451-1464, November, 2005.

[PC05] Olga Papaemmanouil, Ugur Çetintemel. SemCast: Semantic Multicast for Content-based Data Dissemination. ICDE 2005

[PFP03] Shrideep Pallickara, Geoffrey Fox, Marlon E. Pierce. Incorporating an XML Matching Engine in Distributed Brokering Systems. PDPTA 2003: 1511-1517

[Samet89] Hanan Samet. The Design and Analysis of Spatial Data Structures, Addison-Wesley 1989

[SCZ05] http://doi.acm.org/10.1145/1107499.1107504Michael Stonebraker, Ugur Çetintemel, Stanley B. Zdonik. The 8 requirements of real-time stream processing. SIGMOD Record 34(4): 42-47 (2005)

[Sel00] Timos K. Sellis: Review - The R*-Tree. An Efficient and Robust Access Method for Points and Rectangles. ACM SIGMOD Digital Review 2, 2000

[SJL00] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In Proc. ACM SIGMOD, pp. 331--342, 2000.

[SNE05] Stefan Steiniger, Moritz Neun and Alistair Edwardes. Foundations of Location Based Services, Department of Geography, University of Zurich, 2005

[SRF87] Timos K. Sellis, Nick Roussopoulos, Christos Faloutsos. The R+-Tree. A Dynamic Index for Multi-Dimensional Objects. VLDB 1987

[Tat01] Nesime Tatbul. Index Structures and Algorithms for Efficient Profile Matching, Brown University, April 17, 2001

[TPS03] Tao, Y., Papadias, D., Sun, J. The TPR*-Tree. An Optimized Spatio-Temporal Access Method for Predictive Queries. Proceedings of the Very Large Data Bases Conference (VLDB), pp. 790-801, Berlin, September 9-12, 2003.