ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Πρότυπο Σύστημα Διαχείρισης Ερωτήσεων Μονοπατιών σε Συναφείς Καταλόγους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΤΩΝΙΟΥ Ι. ΚΟΥΦΟΠΟΥΛΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2004

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

Πρότυπο Σύστημα Διαχείρισης Ερωτήσεων Μονοπατιών σε Συναφείς Καταλόγους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΝΤΩΝΙΟΥ Ι. ΚΟΥΦΟΠΟΥΛΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 19η Οκτωβρίου 2004.

.....	.....	.....
Τιμολέων Σελλής Καθηγητής Ε.Μ.Π.	Ιωάννης Βασιλείου Καθηγητής Ε.Μ.Π.	Ανδρέας–Γεώργιος Σταφυλοπάτης Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2004

.....

ΑΝΤΩΝΙΟΣ Ι. ΚΟΥΦΟΠΟΥΛΟΣ

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

Πρόλογος

Η διπλωματική αυτή εργασία εκπονήθηκε στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων του Εθνικού Μετσόβιου Πολυτεχνείου. Κατά την διάρκεια της εκπόνησης είχα την ευκαιρία να γνωρίσω και να ασχοληθώ με τα συστήματα διαχείρισης δεδομένων στο διαδίκτυο και να προχωρήσω ένα βήμα πιο μπροστά στον χώρο του προγραμματισμού. Θα ήθελα να ευχαριστήσω τον καθηγητή κ. Τιμολέοντα Σελλή για την επίβλεψη της διπλωματικής και για την ευκαιρία που μου έδωσε να εκπονήσω την διπλωματική εργασία στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων. Επίσης, θα ήθελα να ευχαριστήσω ιδιαίτερα τον Δρ. Θοδωρή Δαλαμάγκα για την πολύ μεγάλη βοήθεια που μου παρείχε, τις συμβουλές του και τις παρατηρήσεις του, για τον χρόνο που αφιέρωσε σε μένα, και γενικότερα για την άψογη συνεργασία μας. Τέλος, να ευχαριστήσω την οικογένειά μου, για την υποστήριξη που μου παρείχαν όλον αυτόν τον καιρό.

Στους γονείς μου, Ιωάννη και Μαργαρίτα

Περίληψη

Οι ιεραρχικοί κατάλογοι, ή απλά *ιεραρχίες*, είναι ένα δημοφιλές εργαλείο οργάνωσης των δεδομένων που παρέχονται σε ιστοσελίδες. Με την βοήθεια ενός ιεραρχικού καταλόγου, ο χρήστης ξεκινά από γενικές θεματικές κατηγορίες, οι οποίες εξειδικεύονται καθώς το βάθος της ιεραρχίας αυξάνει. Ερωτήσεις στις ιεραρχίες αυτές μπορούν να διατυπωθούν είτε με χρήση *ερωτήσεων μονοπατιών*, είτε με διαδικασίες διάσχισης των κόμβων των ιεραρχιών.

Η διπλωματική εργασία περιγράφει ένα σύστημα, με το οποίο ο χρήστης μπορεί να διατυπώσει ερωτήσεις μονοπατιών σε ιεραρχίες, οι οποίες, αν και ανήκουν σε κοινό γνωστικό πεδίο, έχουν διαφορετική οργάνωση. Κεντρικό ρόλο στο σύστημα αυτό παίζουν οι *Διαστάσεις*, που είναι σύνολα από κόμβους μιας ιεραρχίας που αναφέρονται σε σχετικές σημασιολογικά κατηγορίες, και ο *Γράφος Εξαρτήσεων*, που καθορίζει επιτρεπτές σειρές από διαστάσεις. Ο χρήστης, μέσω κατάλληλης διαπροσωπείας, διαλέγει διαστάσεις, με τις οποίες κατασκευάζει ένα *Κανονικοποιημένο Δένδρο*, δηλαδή ένα δένδρο, όπου κάθε επίπεδό του αποτελείται από κόμβους μιας διάστασης μόνο, από το οποίο ο χρήστης διατυπώνει ερωτήσεις μονοπατιών πάνω σε ιεραρχίες. Το σύστημα μετασχηματίζει τις ερωτήσεις αυτές ώστε να αποτιμηθούν αυτόματα από όλες τις ιεραρχίες, παρά τις δομικές τους διαφορές.

Λέξεις Κλειδιά:

Ιεραρχίες, ερωτήσεις μονοπατιών, Διαστάσεις, Γράφος Εξαρτήσεων, Κανονικοποιημένο Δένδρο

Abstract

Hierarchical catalogs, or simply *hierarchies*, are a popular means to organize data available on the Web. Querying capabilities on these hierarchies are provided either through browsing tasks or through path expression-based queries. This diploma thesis describes a prototype system to pose *path expression queries* to hierarchies that although they come from the same domain knowledge they use different organization for the data provided. The thesis introduces the notion of *Dimensions.* Dimensions are sets of semantically related nodes (i.e. categories). A *Dependency Graph* defines the allowed sequences of Dimensions, which can be used to create path queries. Users, guided by a GUI, choose sequences of Dimensions, which help them to construct a *Normalized Tree;* that is a tree every level of which consists of nodes that belong to a single Dimension. Using this tree as schema guise, users pose path queries. Those queries are transformed automatically and sent to hierarchies in order to be evaluated.

Key Words:

Hierarchies, path expression queries, Dimensions, Dependency Graph, Normalized Tree

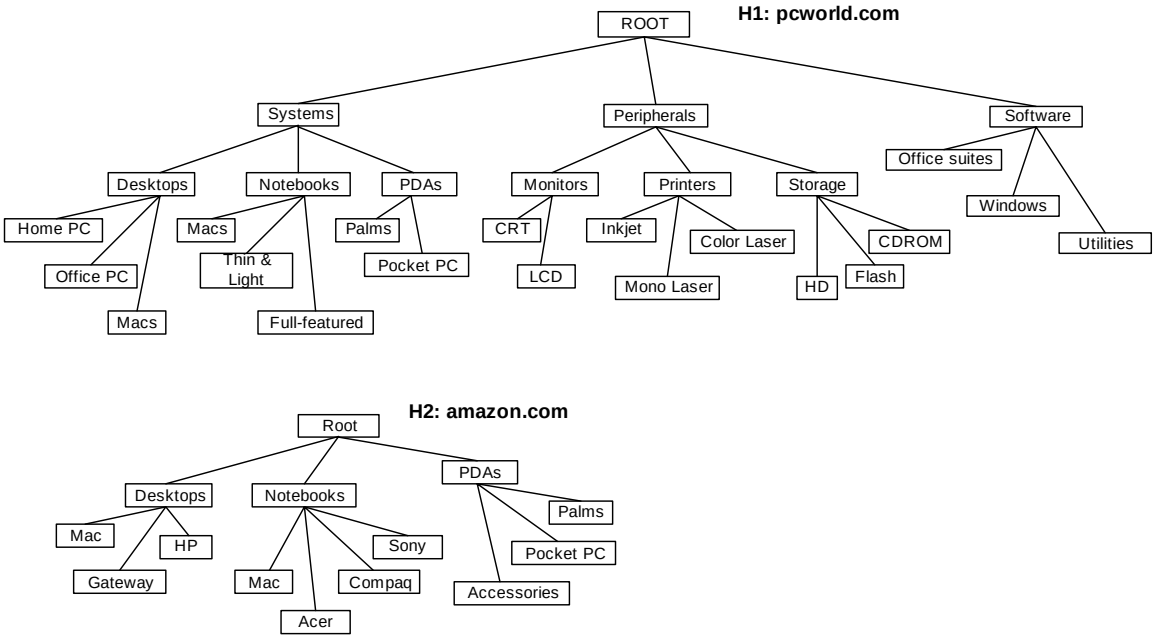
Πίνακας περιεχομένων

1

Εισαγωγή

Στις μέρες μας το Διαδίκτυο αποτελεί τον μεγαλύτερο τόπο εύρεσης πληροφορίας, καλύπτοντας τα περισσότερα γνωστικά θέματα και προσφέροντας την δυνατότητα εύρεσης πληροφορίας σε μεγάλο βάθος. Προκειμένου να γίνει εύκολη η εύρεση της πληροφορίας μέσα σε ένα τόσο μεγάλο χώρο αναζήτησης, έχουν δημιουργηθεί κάποιες ιστοσελίδες, που σκοπό έχουν να παρέχουν στους χρήστες πληροφορίες σχετικά με μια θεματική κατηγορία. Παραδείγματα τέτοια αποτελούν οι πύλες που περιέχουν υλικό για μουσεία, μουσική, αλλά και εμπορικές πύλες για πώληση ηλεκτρονικών υπολογιστών, βιβλίων κλπ. Οι ιεραρχικοί κατάλογοι, ή απλά ιεραρχίες, είναι ένα δημοφιλές εργαλείο οργάνωσης των δεδομένων που παρέχονται σε αυτές τις ιστοσελίδες του διαδικτύου. Με την βοήθεια ενός ιεραρχικού καταλόγου, ο χρήστης ξεκινά από γενικές θεματικές κατηγορίες, οι οποίες εξειδικεύονται καθώς το βάθος της ιεραρχίας αυξάνει.

Ένα κοινό χαρακτηριστικό που συνδέει τις ιστοσελίδες που προαναφέραμε είναι ότι η πληροφορία που παρέχουν είναι οργανωμένη σε ιεραρχίες. Το άλλο κοινό χαρακτηριστικό που έχουν είναι ότι η οργάνωση των ιεραρχιών είναι αυθαίρετη, ανάλογα με τις απαιτήσεις της κάθε ιστοσελίδας. Έτσι, πολλοί κατάλογοι, αν και αποθηκεύουν ίδιου τύπου αντικείμενα έχουν διαφορετική οργάνωση στην ιεραρχία τους. Για παράδειγμα, ακολουθεί μία εικόνα που παρουσιάζει την δομή των ιεραρχιών δύο Portal Catalog Web Sites (του pcworld.com και του amazon.com) που αποθηκεύουν πληροφορία σχετική με Ηλεκτρονικούς Υπολογιστές.



Εικόνα 1.1 Κομμάτια ιεραρχιών από 2 site καταλόγων.

Όπως φαίνεται και στην Εικόνα 1.1, αυτή η διαφορετικότητα στην οργάνωση των ιεραρχιών δεν μας επιτρέπει να εκτελούμε αναζητήσεις με έναν ενιαίο τρόπο χρησιμοποιώντας γλώσσες ερωτήσεων μονοπατιών. Για παράδειγμα, η ερώτηση `'/Root/PDAs/Palms/*'`, που σημαίνει ότι θέλουμε πληροφορίες που έχουν να κάνουν με τα Palms, που ανήκουν στα PDAs, μπορεί να αποτιμηθεί στην ιεραρχία οργάνωσης του amazon.com, αλλά δεν μπορεί να αποτιμηθεί στην ιεραρχία οργάνωσης του pcworld.com, γιατί σε αυτή την ιστοσελίδα, αμέσως κάτω από το Root δεν υπάρχει κόμβος με το όνομα PDAs, αν και αυτός βρίσκεται κάπου παρακάτω, δηλαδή πληροφορίες που να αφορούν τα Palms των PDAs υπάρχουν. Ένας τρόπος να λυθεί αυτό το πρόβλημα, όπως θα δούμε και σε επόμενο κεφάλαιο, είναι με την χρήση κανόνων μετασχηματισμού. Αυτό σημαίνει ότι ψάχνουμε την ερώτηση μονοπάτι που δίνει ο χρήστης, και ακολουθώντας κάποιους κανόνες, αντικαθιστούμε την σειρά με την οποία έχει γίνει η επιλογή, έτσι ώστε να μπορεί να αποτιμηθεί στην εξεταζόμενη ιεραρχία.

Πρέπει επίσης να παρατηρήσουμε ότι το πρόβλημα της διαφορετικής οργάνωσης δεν έγκειται μόνο σε τέτοιες παρόμοιες περιπτώσεις, αλλά και σε περιπτώσεις όπου 2 ή περισσότερες ιεραρχίες έχουν κόμβους που σημασιολογικά ταυτίζονται, αλλά δεν ταυτίζονται λεξικογραφικά, όπως για παράδειγμα συμβαίνει στα τρία παρακάτω μονοπάτια με τον τελευταίο κόμβο: `'/Root/Desktops/Macs'`, `'/Root/Desktops/MAC'`, `'/Root/Desktops/PC_MAC'`. Της ίδιας κατηγορίας πρόβλημα είναι, πχ. τα ονόματα cost και price, που μπορεί να χρησιμοποιούνται σε διάφορες ιεραρχίες και αναφέρονται στο ίδιο πράγμα. Αυτό αποτελεί ένα γενικότερο πρόβλημα, όσον αφορά την εννοιολογική ταύτιση

αντικειμένων με διαφορετικό όνομα, που είναι γνωστό ως πρόβλημα *ταιριάσματος σχημάτων* (schema matching).

Σημαντικό είναι επίσης να λάβουμε υπ'όψιν ότι σε πολλές περιπτώσεις κάποιες ιεραρχίες έχουν πληροφορία που δεν περιέχονται καθόλου σε άλλες ιεραρχίες, όπως στην προηγούμενη εικόνα η πύλη pcworld.com με τους κόμβους `'Systems'`, `'Peripherals'` και `'Software'`, που ορίζει κατηγοριοποίηση που δεν υπάρχει καθόλου στο amazon.com.

1.1 Αντικείμενο της διπλωματικής

Στην διπλωματική αυτή εργασία, σκοπό έχουμε να λύσουμε το πρόβλημα της διαφορετικής οργάνωσης των καταλόγων, όσον αφορά το κομμάτι της δημιουργίας και εφαρμογής ερωτήσεων μονοπατιών. Η ιδέα είναι ότι εφόσον δεν μπορούμε να επέμβουμε στην δομή των ιεραρχιών οργάνωσης των δεδομένων, θα υλοποιήσουμε μοντέλα με τα οποία θα φτιάξουμε δομές ανάλογες με αυτές τις ιεραρχίες, τις οποίες μπορούμε να χειριστούμε πολύ πιο ευέλικτα.

Πιο συγκεκριμένα, κατ'αρχήν θα ομαδοποιήσουμε τους κόμβους σε κατηγορίες (που θα τις καλούμε *διαστάσεις*), σύμφωνα με κάποια κοινά χαρακτηριστικά, όπως πχ. το αντικείμενο στο οποίο αναφέρονται, και κάποιους περιορισμούς, όπως πχ. το ότι δεν επιτρέπεται δύο κόμβοι που ανήκουν στην ίδια διάσταση να ανήκουν στο ίδιο μονοπάτι της αρχικής ιεραρχίας. Αυτό δεν μας απασχολεί και τόσο, γιατί υποθέτουμε ότι αυτή η κατηγοριοποίηση έχει γίνει από πριν. Το σύστημα που θα υλοποιήσουμε θα μας δίνει όμως την δυνατότητα να μεταφέρουμε όποιους κόμβους θέλουμε από την μια διάσταση σε κάποια άλλη. Τους λόγους για τους οποίους μπορεί να θελήσουμε να το κάνουμε αυτό θα τους αναλύσουμε στο Κεφάλαιο 3.

Έχοντας κατηγοριοποιήσει τους κόμβους σε διαστάσεις, μπορούμε για κάθε ιεραρχία να κατασκευάσουμε μία διαφορετική, νέα ιεραρχία (η οποία όμως είναι ανάλογη της αρχικής), στην οποία κάθε επίπεδο αποτελείται από κόμβους μίας μόνο διάστασης. Αυτή η νέα δομή καλείται *κανονικοποιημένο δένδρο*. Όπως θα δούμε και παρακάτω, το ενδιαφέρον είναι ότι κατά την κατασκευή αυτού του κανονικοποιημένου δένδρου μπορούμε να αλλάξουμε την σειρά με την οποία έχουμε επιλέξει τις διαστάσεις, και άρα να κατασκευάσουμε πολλά ομότιμα κανονικοποιημένα δένδρα.

Με βάση αυτό, και εφαρμόζοντας όσα είπαμε για κάθε ιεραρχία, κατασκευάζουμε ένα *καθολικό κανονικοποιημένο δένδρο* από τα επιμέρους κανονικοποιημένα δένδρα της κάθε ιεραρχίας. Το σημαντικό είναι ότι το καθολικό κανονικοποιημένο δένδρο κατασκευάζεται

δυναμικά, ανάλογα με τις επιλογές του χρήστη, δηλαδή την στιγμή που ο χρήστης κατασκευάζει την ερώτηση μονοπατιού που θέλει.

Το επόμενο βήμα είναι η εφαρμογή της ερώτησης μονοπατιού (κατασκευασμένη ως καθολικό κανονικοποιημένο δένδρο) στις επιμέρους ιεραρχίες. Σε αυτό το σημείο οι επιλεγμένες διαστάσεις και κόμβοι αλλάζουν θέση μέσα στην δομή, παίρνοντας όλες τις δυνατές μορφές, έως ότου να μπορούν να αποτιμηθούν στις αρχικές ιεραρχίες. Αν αυτό δεν γίνεται για κάποια ιεραρχία, τότε αποτυγχάνει η αποτίμηση της ερώτησης μονοπατιού στην συγκεκριμένη ιεραρχία.

Στην διπλωματική εργασία υλοποιήσαμε όλες αυτές τις δομές και γενικές λειτουργίες που περιγράψαμε παραπάνω, καθώς και ένα γραφικό περιβάλλον με το οποίο ο χρήστης μπορεί να κάνει όλες τις διαμορφώσεις των παραμέτρων που θέλει, και φυσικά, να δημιουργεί ερωτήσεις μονοπατιών και να παίρνει τα δεδομένα που θέλει από μια Βάση Δεδομένων.

1.2 Οργάνωση του τόμου

Οι λεπτομέρειες της διπλωματικής εργασίας αναλύονται στα εξής παρακάτω κεφάλαια.

Στο 2^{ο} Κεφάλαιο έχουμε την αναλυτική περιγραφή του θέματος της διπλωματικής εργασίας, που αποτελείται από μια αναφορά σε άλλες εργασίες που έχουν γίνει σχετικές με το θέμα, καθώς και από την αναλυτική παρουσίαση τόσο του προβλήματος, όσο και του στόχου αυτής της διπλωματικής.

Στο 3^{ο} Κεφάλαιο αναπτύσσεται το θεωρητικό υπόβαθρο πάνω στο οποίο στηριζόμαστε για την επίτευξη της λύσης του προβλήματός μας. Δίνονται σαφείς ορισμοί των δομών και των εννοιών που θα χρησιμοποιήσουμε παρακάτω.

Στο 4^{ο} Κεφάλαιο έχουμε την λεπτομερή περιγραφή της ανάλυσης των απαιτήσεων της διπλωματικής εργασίας. Περιγράφεται η αρχιτεκτονική και οι διάφορες λειτουργίες.

Στο 5^{ο} Κεφάλαιο έχουμε την σχεδίαση του συστήματος, το πώς υλοποιήθηκαν οι λειτουργίες, ποιες τεχνολογίες και δομές χρησιμοποιήθηκαν και από ποιά υποσυστήματα αποτελείται η όλη εφαρμογή μας.

Στο 6^{ο} Κεφάλαιο έχουμε την περιγραφή όλων των προγραμματιστικών εργαλείων που χρησιμοποιήθηκαν για την ανάπτυξη της εργασίας, τις λεπτομέρειες εγκατάστασης του συστήματος μας και την περαιτέρω αναφορά και επεξήγηση των κλάσεων και των αλγορίθμων που υλοποιήθηκαν.

Στο 7^{ο} Κεφάλαιο παρουσιάζεται η μέθοδος ελέγχου, καθώς και η παρουσίαση ενός χαρακτηριστικού παραδείγματος, με λεπτομερή αναφορά και παρουσίαση όλων των γραφικών παραθύρων που θα συναντήσει ο χρήστης.

Το 8^{ο} Κεφάλαιο είναι ο επίλογος της διπλωματικής. Εδώ έχουμε μια σύνοψη της εργασίας και αναφορά στα συμπεράσματα που καταλήξαμε. Επίσης αναφέρονται προτάσεις και συμβουλές για μελλοντικές επεκτάσεις.

Τέλος, στο 9^{ο} Κεφάλαιο έχουμε την αναφορά στην βιβλιογραφία που χρησιμοποιήθηκε κατά την εκπόνηση αυτής της διπλωματικής.

2

Περιγραφή του Θέματος

Σε αυτό το κεφάλαιο παραθέτουμε μια σύντομη περιγραφή μερικών εργασιών, σχετικών με το αντικείμενο αυτής της διπλωματικής. Επίσης σημειώνουμε τον στόχο της παρούσας διπλωματικής.

2.1 Σχετικές εργασίες

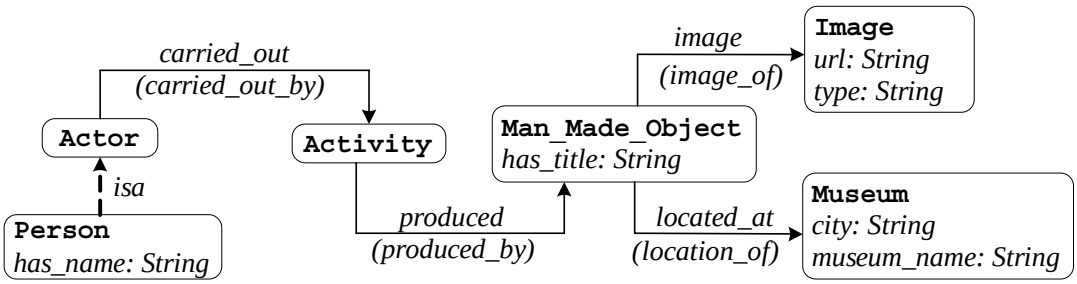
Θα παρουσιάσουμε μια σύντομη περιγραφή της γενικής ιδέας τεσσάρων εργασιών, και τον τρόπο με τον οποίο αντιμετωπίζουν το ίδιο, ή παρόμοια ζητήματα.

2.1.1 Ενοποίηση Οντολογιών

Στη εργασία αυτή, των Bernd Amann, Catriel Beeri, Irini Fundulaki και Michel Scholl, με τον τίτλο 'Ontology-based integration of XML Web Resources' [2], εξετάζεται η ανάγκη να υιοθετηθεί ένα υψηλού επιπέδου μοντέλο αναπαράστασης της σημασιολογίας που εμπεριέχεται στα XML δεδομένα, όταν θέλουμε να συγχωνεύσουμε πληροφορίες από διάφορες πηγές. Συγκεκριμένα προτείνεται η χρήση ενός οντολογικού σχήματος, το οποίο θα προκύπτει από την εννοιολογική συγχώνευση των XML δεδομένων, μέσα από το οποίο ο χρήστης θα μπορεί να κάνει τις ερωτήσεις του (query), οι οποίες προκειμένου να αποτιμηθούν σε όλες τις πηγές που παρέχουν την πληροφορία (και οι οποίες πρέπει να

υποστηρίζουν XPath [4] ή XQuery [3]), ακολουθούν κάποιους κανόνες αντικατάστασης επιμέρους κομματιών τους, ώστε η δομή του τελικού query να αντιστοιχεί στην εκάστοτε δομή της πηγής.

Η εργασία αυτή βασίζεται στην προσέγγιση *local as view* [5], δηλαδή υποθέτει την ύπαρξη μιας *καθολικής 'αποθήκης'* που περιέχει όλες τις πληροφορίες που ενδιαφέρουν τον χρήστη που περιγράφονται από ένα *καθολικό σχήμα* (*global schema*). Οι πηγές είναι μόνο μία τοπική αναπαράσταση αυτού του καθολικού σχήματος, με πιθανώς διαφορετική δομή. Το μοντέλο περιγραφής του καθολικού σχήματος δεν είναι ένα μοντέλο XML, αλλά μία *οντολογία*, δηλαδή ένα απλό αντικειμενοστραφές σχήμα που περιγράφει έννοιες με χαρακτηριστικά πολλαπλών τιμών, που συνδέονται με δυαδικές, συμμετρικές N:M σχέσεις. Επίσης υπάρχει η κληρονομική ιδιότητα (isa) μεταξύ των κλάσεων. Ένα παράδειγμα μιας οντολογίας με την χρήση ενός κατευθυνόμενου γράφου μπορεί να είναι το εξής:



Εικόνα 2.1 Μία οντολογία για πολιτιστικά αντικείμενα

Οι κόμβοι παρουσιάζουν τις έννοιες του σχήματος, με τα πιθανά χαρακτηριστικά τους (όνομα και τύπος). Οι δυαδικές σχέσεις μεταξύ των εννοιών φαίνονται ως βέλη, ενώ η κληρονομική σχέση φαίνεται ως βέλος με διακεκομμένη γραμμή. Κάθε σχέση έχει και αντίστροφη διεύθυνση, που φαίνεται μέσα σε παρενθέσεις.

Για να μπορεί να αποτιμηθεί ένα query του χρήστη που έγινε στην οντολογία στις αρχικές πηγές, πρέπει να μεταφραστεί σε ένα ή περισσότερα query για XML. Γι'αυτό κατασκευάζεται μία *αντιστοίχηση* (*mapping*) μεταξύ των πηγών και της οντολογίας, που περιλαμβάνει *κανόνες αντιστοίχησης* (*mapping rules*). Γενικότερα, αυτοί οι κανόνες αντιστοιχούν ένα υποσύνολο του XPath σε *μονοπάτια σχήματος* στην οντολογία.

Ένας κανόνας έχει την μορφή $r : u / q \ \ as \ \ v \leftarrow p$, όπου r είναι το λογότυπο του κανόνα, u είναι ή ένα URL ή μία μεταβλητή, v είναι μια μεταβλητή, q είναι τύπος του XPath και p είναι ένα μονοπάτι σχήματος της οντολογίας. Οι μεταβλητές v δεσμεύονται στην έκφραση που προηγείται, ενώ οι μεταβλητές u συνιστούν χρήση προηγούμενων δεσμευμένων μεταβλητών. Το μονοπάτι q καλείται *μονοπάτι πηγής* του κανόνα και είναι ένα XPath που χρησιμοποιεί

μόνο τον άξονα *παιδιών* και *απογόνων*. Το μονοπάτι p είναι μια έννοια ή μονοπάτι σχέσεων της οντολογίας, και καλείται *μονοπάτι σχήματος* του κανόνα.

Η διεργασία που γίνεται είναι η αντικατάσταση και δέσμευση ορισμένων μεταβλητών των queries του χρήστη, σύμφωνα με τους κανόνες αντιστοίχησης της κάθε πηγής με την οντολογία, και η αποτίμηση του νέου query στην εκάστοτε ιεραρχία. Ενδεχομένως να χρειαστεί ένα query να σπάσει σε κομμάτια, όταν το αρχικό query περιέχει πληροφορία που δεν αντιστοιχεί σε κάποιες πηγές.

Ένα θέμα είναι η συγχώνευση των πηγών XML. Ο πιο προφανής τρόπος συγχώνευσης θα ήταν να γίνει σε μορφή XML, όπως και γίνεται στα μοντέλα Xyleme [5], MIX [12],Nimble [20] and Agora [13]. Αν και στα MIX και Nimble χρησιμοποιείται η προσέγγιση του *global as view* (GAV), εδώ χρησιμοποιείται η προσέγγιση *local as view* (LAV), όπως και στα Xyleme και Agora, όπου είναι απαραίτητη η επανεγγραφή των queries. Πάντως, σε όλες τις περιπτώσεις που αναφέραμε, τα XML δεδομένα συγχωνεύονται πάλι σε XML. Ένας τρόπος να συγχωνευθούν χρησιμοποιώντας οντολογίες σε εννοιολογικό επίπεδο προτείνεται στο [7]. Η συγχώνευση επιτυγχάνεται θεωρώντας ότι όλα τα δεδομένα XML ακολουθούν ένα κανονικό DTD που προκύπτει από την οντολογία.

Κατά την συγχώνευση των XML δεδομένων μπορεί να προκύψουν διάφορα προβλήματα, όπως ένα element σε μία πηγή να έχει διαφορετική δομή απ'ότι σε μια άλλη, ένα δεδομένο να αναπαρίσταται σαν element στην μια πηγή και στην άλλη σαν attribute, τα διαφορετικά tags κα. Γι'αυτό προτιμάται να χρησιμοποιηθούν *έννοιες* για να αναπαρασταθούν οι τύποι οντοτήτων.

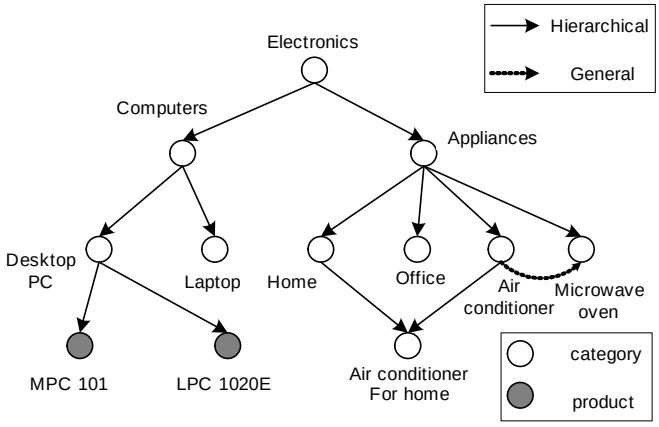
Κάποιες ιδιότητες του εννοιολογικού μοντέλου είναι οι εξής: υπάρχει διαχωρισμός μεταξύ των οντοτήτων (ή αντικειμένων) και των τιμών, οι οντότητες συσχετίζονται με *σχέσεις*, που συχνά είναι συμμετρικές, οι οντότητες έχουν *χαρακτηριστικά* (attributes), αναγνωρίζονται από τα *κλειδιά*, και υπάρχουν και σχέσεις κληρονομικότητας (isa). Όλα αυτά σε αντίθεση με το ιεραρχικό μοντέλο XML, όπου η μόνη σχέση που υπάρχει είναι πατέρας-παιδί.

Τέλος, η έννοια των σημασιολογικών κλειδιών είναι ο μοναδικός τρόπος ταυτοποίησης των δεδομένων που προέρχονται από δύο ή περισσότερες ανεξάρτητες και αυτόνομες πηγές δεδομένων. Τοπικά σε κάθε πηγή υπάρχουν κλειδιά για να ξεχωρίζουν οι εγγραφές, τα οποία όμως συνήθως δεν συμπίπτουν με τα τοπικά κλειδιά που υπάρχουν σε κάποια άλλη πηγή, γι'αυτό χρειάζεται να ορίσουμε κλειδιά πάνω στο καθολικό σχήμα.

2.1.2 Ενοποίηση Καταλόγων Ηλεκτρονικού Εμπορίου (I)

Σε αυτή την εργασία των Dongkyu Kim, Jaebum Kim και Sang-goo Lee, με τον τίτλο 'Catalog Integration for Electronic Commerce through Category-Hierarchy Merging Tecnique' [9], προτείνεται η χρήση ενός εκτεταμένου μοντέλου καταλόγου, που μοντελοποιεί

τα σύνολα των προϊόντων, των κατηγοριών και των σχέσεων μεταξύ αυτών, των αρχικών ιεραρχιών. Αυτό το μοντέλο ονομάζεται *Live Catalog*. Υπάρχουν δύο είδη αντικειμένων σε αυτό το μοντέλο, οι κατάλογοι προϊόντων, που περιέχουν τυπικές πληροφορίες για προϊόντα, όπως το όνομα του προϊόντος, τον κατασκευαστή, τον αριθμό προϊόντος κλπ, και οι κατάλογοι κατηγοριών, που ορίζεται ως ένα σύνολο προϊόντων που χαρακτηρίζονται από μια κοινή ιδιότητα. Μια κατηγορία μπορεί να έχει μία ή περισσότερες υπο-κατηγορίες, των οποίων οι ιδιότητες είναι περισσότερο συγκεκριμένες. Ένα παράδειγμα μιας ιεραρχίας καταλόγων φαίνεται παρακάτω:



Εικόνα 2.2 Μία Ιεραρχία Καταλόγων

Η σχέση μεταξύ δύο κόμβων μπορεί να είναι ιεραρχική, εάν πρόκειται για σχέση μεταξύ κατηγορίας και υπο-κατηγορίας, ή γενική, για οποιαδήποτε άλλη σχέση. Κάθε κόμβος του καταλόγου είναι μια δυάδα, το πρώτο στοιχείο της οποίας είναι μια δομή που περιέχει στοιχεία για το προϊόν ή την κατηγορία, και το δεύτερο είναι ένας κανόνας, δηλαδή ένα σύνολο από κατηγορήματα που υποδηλώνουν τα προϊόντα, ή τις υπο-κατηγορίες, που ανήκουν στον συγκεκριμένο κόμβο.

Στον Live Catalog χρησιμοποιείται ένα σύνολο από δηλωμένες διευκρινίσεις συνθηκών, με το οποίο μπορούμε να βρούμε τα αντικείμενα που ανήκουν σε έναν κόμβο, σε μια ιεραρχία κατηγοριών. Εφαρμόζοντας αυτές τις συνθήκες σαν queries μπορούμε να βρούμε ανά πάσα στιγμή τα αντικείμενα που ανήκουν σε έναν κόμβο. Επίσης, με αυτόν τον τρόπο είναι εύκολη η εισαγωγή και διαγραφή στοιχείων που ανήκουν σε έναν κόμβο, σε αντίθεση με τις χρησιμοποιούμενες τεχνικές, που υλοποιούν δείκτες από ή προς τις εκάστοτε εγγραφές. Αυτούς τους κανόνες τους χρησιμοποιούμε για την κατασκευή του Live Catalog, με την εφαρμογή μερικών επαναληπτικών βημάτων για όλες τις κατηγορίες.

Η συγχώνευση 2 καταλόγων γίνεται θεωρώντας τον ένα σαν κατάλογο-βάση, και τον άλλο σαν κατάλογο-στόχο, ο οποίος θα ενωθεί στον κατάλογο βάση, κάτι που γίνεται σε 3 στάδια. Το πρώτο αφορά την μείωση του χώρου ελέγχου του αλγορίθμου, που υλοποιεί την

συγχώνευση. Το δεύτερο βρίσκει το πιο κατάλληλο μέρος για να τοποθετηθεί ένας κόμβος, και το τρίτο εφαρμόζει την ένωση, ανάλογα με την κάθε περίπτωση.

Το πρώτο στάδιο ψάχνει για τις εξής κατηγορίες κόμβων και κανόνων που τους συνοδεύουν: 1) όλος ο κατάλογος-στόχος εντάσσεται στον κατάλογο-βάση, 2) όλος ο κατάλογος-βάση εντάσσεται στον κατάλογο-στόχο, 3) οι κανόνες του καταλόγου-βάση και του καταλόγου-στόχου υπερκαλύπτονται, και 4) ο κατάλογος-βάση και ο κατάλογος-στόχος είναι πλήρως διακριτοί. Έτσι μειώνουμε το πεδίο εφαρμογής του αλγορίθμου, ανάλογα με την περίπτωση.

Στο δεύτερο στάδιο, ψάχνουμε για όλους τους κόμβους του καταλόγου-στόχου να βρούμε το πιο κατάλληλο σημείο του καταλόγου-βάση, στο οποίο μπορούν να εισαχθούν. Αυτό γίνεται βρίσκοντας το μικρότερο άνω φράγμα για κάθε κόμβο του καταλόγου-στόχου στον κατάλογο-βάση, δηλαδή ψάχνουμε κόμβους, πάνω από τους οποίους δεν μπορεί να μπει ο εξεταζόμενος κόμβος, και διαλέγουμε τον μικρότερο (πιο χαμηλά στην ιεραρχία) από αυτούς.

Στο τρίτο στάδιο έχουμε την συγχώνευση του εξεταζόμενου κόμβου, από τον κατάλογο-στόχο, με το μικρότερο άνω φράγμα από τον κατάλογο-βάση, όπως υπολογίστηκαν στο δεύτερο στάδιο, με διαφορετικές υλοποιήσεις αν οι κόμβοι είναι ίδιοι, ή αν ο ένας υπερκαλύπτει τον άλλο.

2.1.3 Ενοποίηση Καταλόγων (ΙΙ)

Σε αυτή την εργασία των Pedro Jose Marron, Georg Lausen και Martin Weber, με τον τίτλο ‘Catalog Integration Made Easy’ [14], αναλύεται λίγο περισσότερο η τεχνική της προσαρμοζόμενης αποτίμησης (Adaptive Query Evaluation), που περιγράφεται στο [11].

Μέχρι τώρα, στα μοντέλα που χρησιμοποιούνται για την επανεγγραφή των queries και την συγχώνευση των καταλόγων, ο χρήστης δίνει ένα query με βάση το καθολικό σχήμα που βλέπει, και αυτό τροποποιείται για την κάθε ιεραρχία, με βάση το εκάστοτε τοπικό σχήμα. Τα επιμέρους αποτελέσματα που επιστρέφουν συλλέγονται και σχηματίζουν την τελική απάντηση. Με την τεχνική της προσαρμοζόμενης αποτίμησης δεν χρειάζεται να επανεγγράψουμε τα queries, καθώς το αρχικό query μπορεί να αποτιμηθεί, ανεξάρτητα από τις διαφορές που υπάρχουν μεταξύ του καθολικού και των τοπικών σχημάτων.

Η υλοποίηση της τεχνικής γίνεται σε δύο στάδια. Ένα προκαταρκτικό στάδιο, όπου, εάν χρειάζεται, οι έννοιες που βρίσκονται στο query μεταφράζονται ανάλογα με τις υπάρχουσες οντότητες των τοπικών σχημάτων, και το στάδιο της επεξεργασίας του query.

Το πρώτο στάδιο, ουσιαστικά υποθέτει ότι υπάρχει μια αντιστοίχιση ανάμεσα στα τοπικά ονόματα, και τα ονόματα που χρησιμοποιούνται στο καθολικό σχήμα. Έτσι θεωρούμε ότι λύνεται εύκολα το πρόβλημα των ασυμφωνιών σε ονόματα οντοτήτων.

Στο δεύτερο στάδιο, ο αλγόριθμος υλοποιεί τα εξής δύο βήματα: την εφαρμογή του κάθε προσαρμοσμένου μετασχηματισμού στην είσοδο του κάθε subquery, και την αποτίμηση μιας

συνάρτησης κατάστασης σε κάθε βήμα για να βρούμε μια κατάταξη κόμβων, που μας βοηθάει να ξεχωρίσουμε το ένα σύνολο λύσεων από το άλλο. Οι ενέργειες που μπορεί να γίνουν κατά το πρώτο βήμα, είναι (1) καμία αλλαγή (n), (2) γενίκευση του subquery (g), όπου ένα κομμάτι του subquery μεταφέρεται από παιδί σε απόγονο, και από πατέρα σε πρόγονο, και (3) απαλοιφή του subquery (e), όταν το αποτέλεσμα είναι κενό.

Το σημαντικό είναι να καθορίσουμε την σειρά με την οποία θα γίνουν οι μετασχηματισμοί, ώστε να πάρουμε σωστά αποτελέσματα. Δεδομένου ενός subquery και της εισόδου του, κάνουμε αποτίμηση με κάθε μετασχηματισμό, δηλαδή βγάζουμε 3 αποτελέσματα. Εδώ χρησιμοποιείται και η συνάρτηση κατάστασης, για τον υπολογισμό της κατάστασης. Κάθε ενδιάμεση αποτίμηση του query θεωρείται σαν κόμβος, και παίρνει μια τιμή, με βάση τον μετασχηματισμό που έγινε, κάτι που μας επιτρέπει να βρούμε τους κόμβους που οδηγούν στις βέλτιστες λύσεις.

Έστω q_i το subquery που πρόκειται να αποτιμηθεί, και C_i η σχετική είσοδός του. Κάθε κόμβος $n \in C_i$ αναλαμβάνει και μία τιμή u_n που προκύπτει από την αποτίμηση των προηγούμενων subqueries $q_o,\ldots,q_{i-1}$. Η νέα είσοδος $C_i = C_{i+1}^n \bigcup C_{i+1}^g \bigcup C_{i+1}^e$ υπολογίζεται από την ένωση ανεξάρτητων συνόλων αποτελεσμάτων των τριών δυνατών τεχνικών αποτίμησης. Επίσης ξέρουμε ότι κάθε νέος κόμβος στο C_{i+1} είναι το αποτέλεσμα της αποτίμησης του query q_i σε έναν κόμβο $n \in C_i$. Η συνάρτηση κατάστασης αναθέτει μια τιμή u_m σε κάθε νέο κόμβο $m \in C_{i+1}$ που προκύπτει από έναν κόμβο $n \in C_i$ ως εξής: Αν $m \in C_{i+1}^n$, τότε $u_m = b^2 + u_n$, αν $m \in C_{i+1}^g$, τότε $u_m = b + u_n$, και αν $m \in C_{i+1}^e$, τότε $u_m = 1 + u_n$. Σε αυτόν τον ορισμό, το b είναι μια θετική σταθερά, που αναπαριστά την βάση της συνάρτησης κατάστασης. Αν το n έχει προκύψει από την εφαρμογή παραπάνω από μιας τεχνικής, πχ. $n \in C_{i+1}^n$ και $n \in C_{i+1}^g$, τότε το τελικό C_{i+1} θα περιέχει για το n την μεγαλύτερη τιμή u_n. Επίσης, αρχικά ισχύει $u_{root} = 1$

2.1.4 Διαχείριση Καταλόγων Ηλεκτρονικού Εμπορίου

Σε αυτή την εργασία των Jihye Jung, Dongkyu Kim, Sang-goo Lee, Chisu Wu και Kapsoo Kim, με τον τίτλο 'EE-Cat: Extended Electronic Catalog for Dynamic and Flexible Electronic Commerce' [8], προτείνεται η ένωση των ιεραρχιών προϊόντων και των ιεραρχιών κατηγοριών σε ένα είδος ιεραρχίας, που ονομάστηκε EE-Cat, προκειμένου να κατασκευαστεί μια πιο ευέλικτη δομή, με την οποία θα μπορούμε, χρησιμοποιώντας πχ. το κομμάτι των κατηγοριών, να μικρύνουμε το εύρος της αναζήτησης των προϊόντων. Επίσης, αυτή η δομή είναι αρκετά εύχρηστη, ευέλικτη και εύκολη στην τροποποίησή της. Ένα άλλο σημείο που

έχει σημασία, είναι ότι χρησιμοποιούνται οι ιεραρχίες που μας δίνουν οι κατασκευαστές, και από αυτές, κάθε ηλεκτρονικός κατάλογος διαλέγει πιο κομμάτι θα χρησιμοποιήσει, παίρνοντας ένα υποσύνολο της αρχικής ιεραρχίας.

Κάθε προϊόν έχει κάποια χαρακτηριστικά, τα οποία το προσδιορίζουν και διαφοροποιούν τα προϊόντα μεταξύ τους, πχ. ένα ψυγείο έχει κάποια χαρακτηριστικά για την χωρητικότητά του και για την κατανάλωση ενέργειας. Αυτά τα χαρακτηριστικά ονομάζονται *χαρακτηριστικά με βάση το προϊόν*. Μπορούμε να χρησιμοποιήσουμε την χωρητικότητα για να διαχωρίσουμε τα διάφορα ψυγεία. Από την άλλη, υπάρχουν και κάποια χαρακτηριστικά που είναι κοινά για όλα τα προϊόντα, όπως το όνομα της εταιρίας, αριθμός μοντέλου κλπ., τα οποία ονομάζουμε *γενικά χαρακτηριστικά*. Μπορούμε να θεωρήσουμε λοιπόν ότι ένα προϊόν χαρακτηρίζεται από αυτά τα δύο σύνολα χαρακτηριστικών. Πάνω σε αυτά μπορούμε να ορίσουμε διαστάσεις, οι οποίες είναι μια προβολή μερικών από όλα αυτά τα χαρακτηριστικά των προϊόντων, με βάση την σημασία της διάστασης, πχ. χρόνος, τοποθεσία, χρήστης κλπ.

Επειδή τα προϊόντα μπορούν να ανήκουν σε περισσότερες από μια κατηγορίες, μπορεί να γίνει αναζήτηση με διαφορετική σειρά αυτών των κατηγοριών, πχ. Nike/Tennis_Shoes/10-11-12 ή 10-11-12/Tennis_Shoes/Nike. Για το λόγο αυτό, αναπαριστούμε την ιεραρχία κατηγοριών σαν έναν γράφο, όπου κάθε κόμβος είναι μια κατηγορία, και οι ακμές εκφράζουν σχέσεις δύο ειδών, είτε μεταξύ κατηγορίας–υπο-κατηγορίας, είτε μια γενική σχέση, ότι δηλαδή δύο κατηγορίες σχετίζονται με κάποιον άλλον τρόπο. Για να δηλώσουμε την σχέση ανάμεσα στα προϊόντα και στις κατηγορίες, μέχρι τώρα χρησιμοποιείται ένα σύνολο δεικτών, δηλ. για την κάθε κατηγορία έχουμε δείκτες που δείχνουν σε όλα τα προϊόντα, ή δείκτες από κάθε προϊόν σε όλες τις κατηγορίες που ανήκει. Εδώ χρησιμοποιείται ένα σύνολο κανόνων για το κάθε προϊόν, δηλ. κάποια χαρακτηριστικά με την τιμή που χαρακτηρίζει την κατηγορία στην οποία ανήκουν. Έτσι, ένα προϊόν που ανήκει στην κατηγορία Tennis_Shoes, έχει έναν κανόνα που λέει ότι: Product_Family = “Athletic_Shoes” AND Purpose = “Tennis”.

Η κατασκευή μιας ιεραρχίας γίνεται αρχίζοντας από μερικές βασικές κατηγορίες, και εφαρμόζοντας είτε συγχωνεύσεις είτε εξειδικεύσεις καταλήγουμε σε νέες κατηγορίες, και οι κανόνες για την κάθε κατηγορία βγαίνουν αυτόματα, ανάλογα με τις κινήσεις που προκάλεσαν την δημιουργία τους. Το EE-Cat είναι ένας κατευθυνόμενος γράφος, του οποίου οι κόμβοι είναι ηλεκτρονικοί κατάλογοι για κατηγορίες ή προϊόντα, και οι ακμές είναι οι σχέσεις μεταξύ των καταλόγων.

Για την εύρεση υπο-γράφων του αρχικού, χρησιμοποιείται μια γλώσσα ερωτήσεων με όνομα EE-Cat query language, η οποία είναι βασισμένη στην SQL, και η οποία καθιστά το EE-Cat σε δομή αναζήτησης. Με τους υπο-γράφους βρίσκουμε τους καταλόγους που αναζητάμε, ξεκινώντας από έναν κόμβο, ή μια όψη του EE-Cat. Επίσης, με την EE-Cat query language μπορούμε να εξάγουμε και να αναδιοργανώσουμε πληροφορίες που προέρχονται από τους καταλόγους που ανακτήσαμε μέσω του query.

2.2 Στόχος

Στόχος αυτής της διπλωματικής είναι να συγχωνεύσει τα διάφορα δεδομένα XML των πηγών που συμμετέχουν και να επιτρέψει στον χρήστη να θέτει ερωτήσεις (queries) μονοπατιών, οι οποίες θα αποτιμώνται στις διάφορες πηγές. Δεν θα χρησιμοποιηθούν όμως κανόνες μετατροπής των ερωτήσεων, όπως αναφέρθηκε σε προηγούμενη εργασία, αλλά θα ορίσουμε και θα κατασκευάσουμε ευέλικτες δομές, όπως τις *διαστάσεις* (Dimensions), όπου και θα βάλουμε τα elements των XML δεδομένων ανάλογα με τις απαιτήσεις του χρήστη, τις οποίες θα χρησιμοποιεί ο χρήστης για να κατασκευάσει μία ερώτηση μονοπατιού. Οι διαστάσεις που επέλεξε ο χρήστης εναλλάσσονται μεταξύ τους, κατασκευάζοντας ανάλογες ερωτήσεις μονοπατιού με διαφορετική δομή, οι οποίες αποτιμούνται σε όλες τις ιεραρχίες (ή όπου μπορούν να αποτιμηθούν). Επίσης έχουμε μια ακόμη δομή, τον *γράφο εξαρτήσεων* (Dependency Graph), ο οποίος περιορίζει την δυνατότητα των διαστάσεων να παίρνουν όλες τις δυνατές θέσεις στο μονοπάτι, βάζοντας κάποιους κανόνες, όπως πχ. ότι μία διάσταση a δεν μπορεί να βρεθεί πάνω από μία διάσταση b (εξάρτηση του a από το b). Όλα αυτά μέσα από ένα φιλικό γραφικό περιβάλλον.

3

Θεωρητική Μελέτη

Σε αυτό το κεφάλαιο παραθέτουμε όλο το θεωρητικό υπόβαθρο πάνω στο οποίο βασίστηκε η διπλωματική αυτή, δηλαδή τους απαραίτητους ορισμούς καθώς και την γενική ιδέα στην οποία στηριχτήκαμε για την επίλυση του προβλήματός μας.

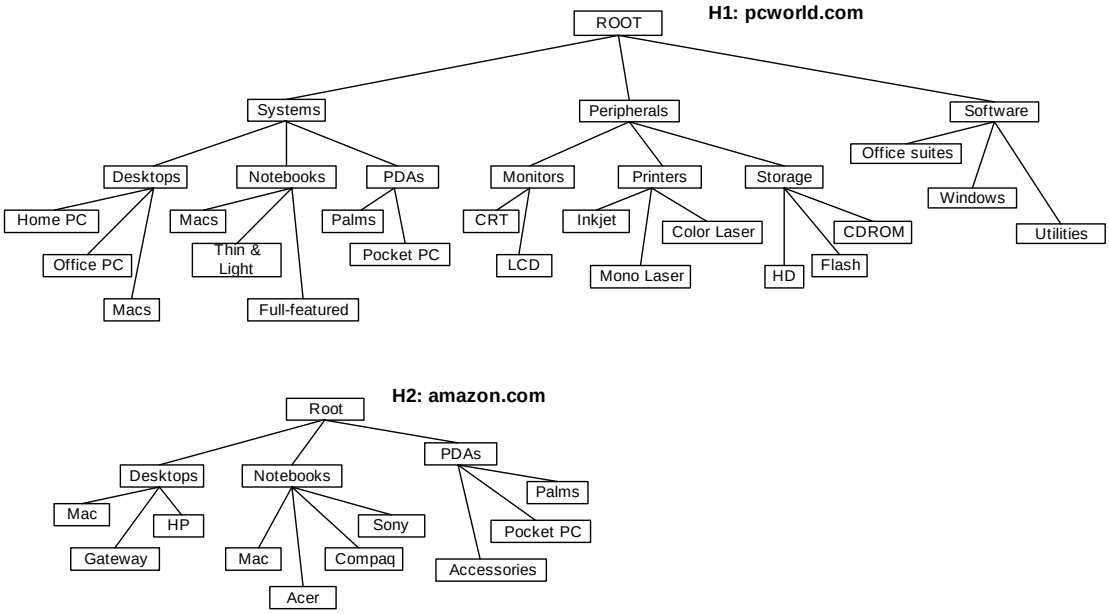
3.1 Βασικοί Ορισμοί

Αναπαριστούμε τις ιεραρχίες ως *δένδρα με ρίζα και ετικέτες* (rooted labeled trees), σύμφωνα με τον ακόλουθο ορισμό:

Ορισμός 3.1. *Ένα* δένδρο με ρίζα και ετικέτες, *ή απλά* δένδρο, *T είναι*

1. *ένας* κόμβος ρίζα *r, ή*
2. *ένας* κόμβος ρίζα *r και ένα σύνολο E από ακμές,* $E=\{e_1, e_2,\ldots, e_k\}$, *με* $e_1: r \rightarrow r_1$, $e_2: r \rightarrow r_2$, …. $e_k: r \rightarrow r_k$, *όπου* $r_1, r_2,\ldots r_k$, *είναι κόμβοι ρίζες k (υπό) δένδρων.*

Μία συνάρτηση label(n) αναθέτει ένα string label ως το αναγνωριστικό για κάθε κόμβο n του T. Ειδικά για το r, label(r) =root.



Εικόνα 3.1 Κομμάτια ιεραρχιών από 2 site καταλόγων.

Στην Εικόνα 3.1 φαίνονται παραδείγματα τέτοιων δένδρων, τα οποία θα χρησιμοποιήσουμε κατά την διάρκεια της αναφοράς ως χαρακτηριστικά παραδείγματα. Σημειώνουμε ότι όμοιες επιγραφές χρησιμοποιούνται για τους κόμβους ενός καταλόγου που ταιριάζουν με κόμβους ενός άλλου καταλόγου.

Παρουσιάζοντας τις ιεραρχίες σαν δένδρα, μπορούμε να ξεχωρίσουμε σε κατηγορίες κάποια σύνολα κόμβων, που τα αποκαλούμε *διαστάσεις (dimensions)*. Γενικά, μία διάσταση είναι σαν ένας τίτλος ενός συνόλου κόμβων, ένα γενικό χαρακτηριστικό που αντιπροσωπεύει κόμβους-κατηγορίες. Για παράδειγμα, στην ιεραρχία H_2, στην Εικόνα 3.1, μπορούμε να διακρίνουμε μία διάσταση για τα **product types**, χρησιμοποιώντας τους κόμβους *Desktops*, *Notebooks*, *PDAs*. Δεν είναι απαραίτητο οι κόμβοι μιας διάστασης να βρίσκονται στο ίδιο επίπεδο της ιεραρχίας. Έτσι μπορούμε πχ. να ορίσουμε τη διάσταση **brand names**, χρησιμοποιώντας τους κόμβους *Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*. Ακολουθεί ο ορισμός για τις διαστάσεις.

Ορισμός 3.2. *Έστω T ένα δένδρο που παριστάνει μία ιεραρχία. Μία* διάσταση *του T είναι ένα ζεύγος (id, V) όπου*

1. *id είναι ένα αναγνωριστικό που καλείται* διακριτικό χαρακτηριστικό *(discrimination property)*

2. *$V = \{u_0, u_1,\ldots, u_m\}$ είναι ένα σύνολο από m κόμβους του T, τέτοιους ώστε n_i και n_j δεν ανήκουν στο ίδιο μονοπάτι, $\forall\, n_i,n_j,\ i \neq j,\ i<m,\ j<m$ (root $\notin V$). Θεωρούμε ότι το V είναι το* σύνολο τιμών *για το* διακριτικό χαρακτηριστικό *id*.

Για οποιεσδήποτε 2 διαστάσεις (id_i, V_i) *και* (id_j, V_j)*, ισχύει ότι:* $V_i \cap V_j = \varnothing$.

Για παράδειγμα, οι δύο διαστάσεις που αναφέραμε στην προηγούμενη παράγραφο ορίζονται ως εξής: (*product_type*, {*Desktops*, *Notebooks*, *PDAs*}), και (*brand_name*, {*Mac*, *Gateway*, *Hp*, *Acer*, *Compaq*, *Sony*}).
Ο *διαμερισμός* (partitioning) αναφέρεται στην πράξη της αναγνώρισης-ορισμού των διαστάσεων σε ένα δένδρο που αναπαριστά μία ιεραρχία.

Ορισμός 3.3. *Έστω T ένα δένδρο με N κόμβους, στους οποίους δεν συμπεριλαμβάνουμε την ρίζα, που παριστάνει μία ιεραρχία. Ένας* διαμερισμός *(partitioning) του T είναι ένα σύνολο από διαστάσεις* $\{(P_1, V_1), (P_2, V_2), \ldots, (P_m, V_m)\}$ *του T, όπου* V_i *ένα σύνολο από τιμές για το διακριτικό χαρακτηριστικό* P_i $(1 \leq i \leq m)$*. Ένας* ολικός διαμερισμός *(total partitioning) είναι ένας διαμερισμός όπου* $|V_1| + |V_1| + \ldots + |V_m| = N$.

3.2 Κανονικοποιημένο Δένδρο

3.2.1 Περιγραφή

Βασισμένοι στους προηγούμενους ορισμούς, μπορούμε να κατασκευάσουμε ένα νέο δένδρο που καλείται *κανονικοποιημένο δένδρο* (normalized tree) με τις ακόλουθες ιδιότητες:

1. Κάθε επίπεδο αντιστοιχεί σε μία και μόνο διάσταση.

2. Δεν χάνεται η δομική πληροφορία του αρχικού δένδρου, δηλ. όλοι οι κόμβοι από την ρίζα μέχρι και τα φύλλα έχουν διατηρηθεί.

Η κατασκευή του κανονικοποιημένου δένδρου εξαρτάται από την ακολουθία με την οποία χρησιμοποιούνται τα διακριτικά χαρακτηριστικά. Για παράδειγμα, στην ιεραρχία H_2 που φαίνεται στην Εικόνα 3.1, μπορούμε να εφαρμόσουμε την ακόλουθη διαμέριση:

1. Διάσταση 1: (*product_type*, {*Desktops*, *Notebooks*, *PDAs*}).

2. Διάσταση 2: (*brand_name*, {*Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*}).

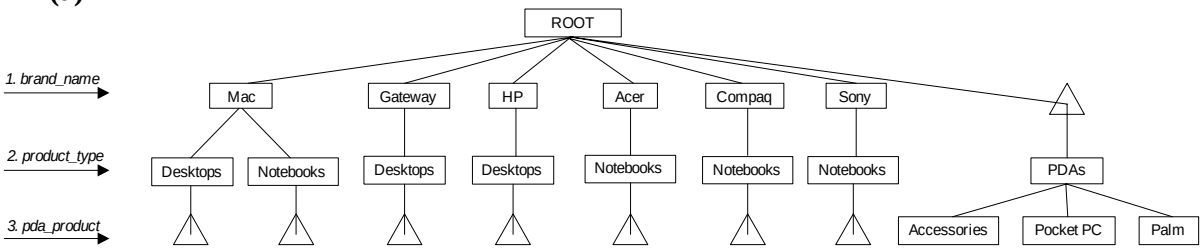
3. Διάσταση 3: (*pda_product*, {*Accessories*, *PocketPC*, *Palm*}).

Πρέπει να σημειώσουμε ότι αυτή η διαμέριση είναι *ολική*, δηλαδή όλοι οι κόμβοι (εκτός της ρίζας root) ανήκουν σε κάποια διάσταση. Στην Εικόνα 3.2(a) φαίνεται ένα κανονικοποιημένο

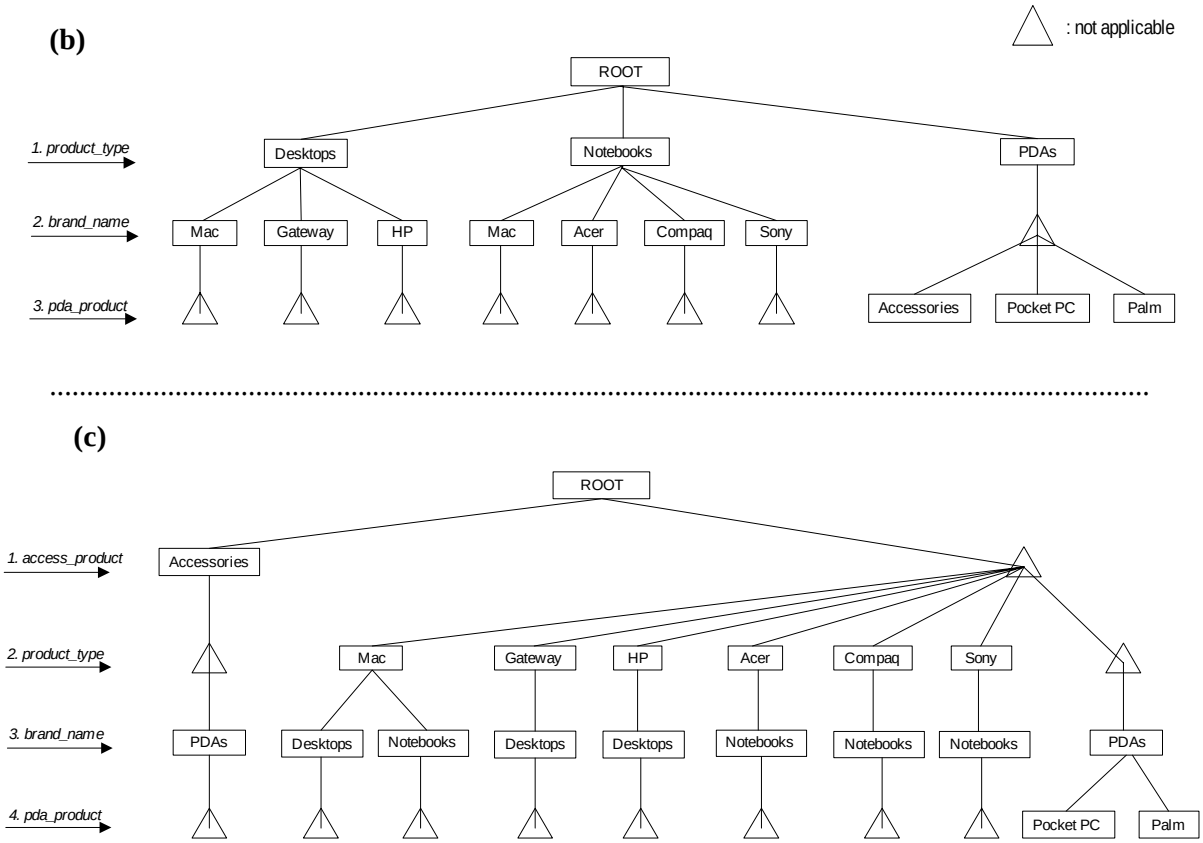
δένδρο για την ιεραρχία H_2, δεδομένης της ακολουθίας (*brand_name, product_type, pda_product*) των διακριτικών χαρακτηριστικών. Η κατασκευή του δένδρου αρχίζει με την επέκταση του κόμβου-ρίζα (επίπεδο 0) χρησιμοποιώντας τις τιμές *Mac, Gateway, HP, Acer, Compaq, Sony* του πρώτου διακριτικού χαρακτηριστικού *brand_name*. Αν υπάρχουν μονοπάτια στην αρχική ιεραρχία που δεν περιέχουν καμία από αυτές τις τιμές, τότε ένας νέος κόμβος, που δηλώνεται με Δ, εισάγεται για να δείξουμε ότι υπάρχουν κόμβοι για τους οποίους οι τιμές του νέου χαρακτηριστικού δεν είναι εφαρμόσιμες. Αυτοί οι κόμβοι θα εισαχθούν στο κανονικοποιημένο δένδρο σε επόμενα βήματα. Για παράδειγμα, το Δ εισάγεται στο πρώτο επίπεδο του κανονικοποιημένου δένδρου γιατί υπάρχουν μονοπάτια (όπως /*PDAs*/*Palm*) στην αρχική ιεραρχία H_2 που δεν περιέχουν καμία από τις τιμές *Mac, Gateway, HP, Acer, Compaq, Sony*.

Μετά, για κάθε επακόλουθο διακριτικό χαρακτηριστικό ελέγχουμε αν μπορούμε να επεκτείνουμε τους κόμβους στο τρέχον επίπεδο του κατασκευασμένου δένδρου χρησιμοποιώντας τις τιμές αυτού του χαρακτηριστικού. Η επέκταση ενός κόμβου *n* χρησιμοποιώντας μία τιμή *v*, είναι δυνατή μόνο όταν (α) το *v* και όλοι οι μη-Δ (μη-τριγωνικοί) κόμβοι στο μονοπάτι που αρχίζει από τον κόμβο *n* μέχρι και την ρίζα ανήκουν σε κάποιο μονοπάτι της αρχικής ιεραρχίας, με οποιαδήποτε σειρά, και (β) στην αρχική ιεραρχία δεν υπάρχει μονοπάτι, με οποιαδήποτε σειρά, που να σχηματίζεται από το *v* και όλους τους μη-Δ κόμβοι του κατασκευασμένου δένδρου, στο επίπεδο των οποίων έχει μπει Δ κάπου στο μονοπάτι που αρχίζει από τον κόμβο *n* μέχρι και την ρίζα. Στην αντίθετη περίπτωση, εισάγουμε έναν νέο κόμβο Δ για να δείξουμε ότι οι τιμές του νέου χαρακτηριστικού δεν μπορούν να εφαρμοστούν για την επέκταση του εξεταζόμενου κόμβου.

(a)



.....



Εικόνα 3.2. Μορφές του κανονικοποιημένου δένδρου

Για παράδειγμα, στην Εικόνα 3.2(a), χρησιμοποιώντας τις τιμές του χαρακτηριστικού *product_type*, ο κόμβος *Mac* (επίπεδο 1) επεκτείνεται (a) σε *Desktops*, καθώς οι κόμβοι *Mac* και *Desktops* ανήκουν στο ίδιο μονοπάτι στην αρχική ιεραρχία, και (b) σε *Notebooks*. Η τιμή *PDAs* δεν μπορεί να χρησιμοποιηθεί για την επέκταση του *Mac* καθώς η αρχική ιεραρχία δεν περιέχει το *PDAs* και το *Mac* στο ίδιο μονοπάτι. Ομοίως, χρησιμοποιώντας τις τιμές του διακριτικού χαρακτηριστικού *pda_product*, ο κόμβος *PDAs* (επίπεδο 2) επεκτείνεται στα *Accessories*, *PocketPC*, *Palm*. Αυτές οι τιμές δεν επεκτείνουν τους άλλους κόμβους του επιπέδου 2, καθώς δεν υπάρχουν μονοπάτια στην αρχική ιεραρχία που να περιέχουν κάποια από τις τιμές του *pda_product* μαζί με κόμβους όπως οι *Desktops*, *Notebooks*, κα.

Η Εικόνα 3.2(b) παρουσιάζει το κανονικοποιημένο δένδρο για την H_2, δεδομένης της ακολουθίας (*product_type*, *brand_name*, *pda_product*) των διακριτικών χαρακτηριστικών. Εφ'όσον η ακολουθία αντιστοιχεί με την δομή της αρχικής ιεραρχίας, το κανονικοποιημένο δένδρο είναι ίδιο με την H_2.

Στην Εικόνα 3.2(c) παρουσιάζεται το κανονικοποιημένο δένδρο για την H_2, δεδομένης της ακολουθίας (*access_product*, *brand_name*, *product_type*, *pda_product*) των διακριτικών χαρακτηριστικών. Η διαμέριση της H_2 έχει αλλάξει στην ακόλουθη:

1. Διάσταση 1: (*access_product*, {*Accessories*}).

2. Διάσταση 2: (*brand_name*, {*Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*}).

3. Διάσταση 3: (*product_type*, {*Desktops*, *Notebooks*, *PDAs*}).

4. Διάσταση 4: (*pda_product*, {*PocketPC*, *Palm*}).

Για να διευκρινίσουμε την χρήση του κόμβου Δ, σημειώνουμε ότι ο δεξιός Δ κόμβος στο επίπεδο 2 (για το διακριτικό χαρακτηριστικό *brand_name*) αναπαριστά τα προϊόντα τα οποία δεν μπορούν να ταξινομηθούν ούτε σαν *Accessories*, ούτε σαν *Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*. Αυτά είναι τα προϊόντα που ταξινομούνται μόνο σαν *PDAs/PocketPC* ή σαν *PDAs/Palm*.

3.2.2 Κατασκευή

Δεδομένης μιας διαμέρισης και μιας ακολουθίας των διαστάσεων, μπορούμε να κατασκευάσουμε ένα κανονικοποιημένο δένδρο ως εξής:

Έστω $D = \{(P_1, V_1), (P_1, V_1),\ldots, (P_m, V_m)\}$ μία σειρά από m διαστάσεις;
Έστω T το αρχικό δένδρο και T' ένα κενό δένδρο;
Δημιουργώ μία ρίζα για το T' ;
$E = \{T'.root\}$; /* Το E περιέχει τους κόμβους που θα επεκταθούν σε κάθε βήμα */
$E' = \{\ \}$ /* Το E' περιέχει τους κόμβους που προσθέτονται σε κάθε βήμα */

Για κάθε διάσταση (P, V) στο D :
    Για κάθε κόμβο n στο E :
        $W = \{\ \}$; /* Στο W έχουμε όλους τους μη-Δ κόμβους στο μονοπάτι με αρχή το n και τέλος την ρίζα του T' */
        $W' = \{\ \}$; /* Στο W' έχουμε όλους τους μη-Δ κόμβους (στα άλλα μονοπάτια του T'), στα επίπεδα των οποίων βάλαμε Δ κόμβους στο μονοπάτι που ορίζεται από τον κόμβο n και την ρίζα του T', για να δείξουμε ότι δεν μπορούμε να τους επεκτείνουμε με το νέο χαρακτηριστικό. Με άλλα λόγια, στο W' βάζουμε τους κόμβους-'αδέρφια' των Δ κόμβων */

        Για κάθε κόμβο i από το n έως την ρίζα του T' :
            Αν $i \neq \Delta$ τότε
                $W = W + \{ i \}$;
            αλλιώς /* $i = \Delta$ */
                Για κάθε παιδί-κόμβο k στο Parent(i) με $k \neq \Delta$
                $W' = W' + \{ k \}$; /* W': αδέρφια του Δ */
            end
        end

```
        /* Φάση επέκτασης του τρέχοντος κόμβου */
        Για κάθε τιμή v στο V
            Αν n ≠ Δ τότε
                Αν οι δύο κόμβοι n, v ανήκουν σε κάποιο μονοπάτι στο T, τότε
                    Βάλε έναν κόμβο v ως παιδί του n ;
                    E' = E' + { v } ;
                end
            αλλιώς   /* n = Δ */
                αν υπάρχει μονοπάτι στο T με τον κόμβο v και όλους τους κόμβους
                του W, αλλά χωρίς κανέναν κόμβο του W', τότε
                    Βάλε έναν κόμβο v ως παιδί του n ;
                    E' = E' + { v } ;
                end
            end
        end

        /* Έλεγχος αν πρέπει να βάλουμε κόμβο Δ σαν παιδί του κόμβου n */
        Αν υπάρχει μονοπάτι στο T με όλους τους κόμβους του W, αλλά χωρίς
        κανέναν από τους κόμβους του V ∪ W', τότε
            Βάλε έναν κόμβο Δ ως παιδί του n ;
            E' = E' + { Δ } ;
        end
    end
    E = E' ;
    E' = { } ;
end
```

Σαν παράδειγμα μπορούμε να δούμε την Εικόνα 3.2, όπου βλέπουμε τρία κανονικοποιημένα δένδρα, για την ίδια ιεραρχία.

Μπορεί να αποδειχθεί ότι ένα κανονικοποιημένο δένδρο περιλαμβάνει όλες τις απαραίτητες πληροφορίες σχετικά με τους κόμβους που υπάρχουν στα μονοπάτια των κανονικών δένδρων.

Πρόταση 3.1. *Έστω T ένα δένδρο που αναπαριστά μία ιεραρχία,* $\{p_1, p_2,\ldots,p_m\}$ *το σύνολο των ξεχωριστών μονοπατιών από την ρίζα μέχρι τα φύλλα, και* $P=\{P_1, P_2,\ldots, P_m\}$ *το σύνολο των συνόλων των κόμβων για κάθε μονοπάτι* p_i $(1 \le i \le m)$*. Έστω T' ένα κανονικοποιημένο δένδρο βασισμένο σε μία ακολουθία διαστάσεων από μία καθολική διαμέριση του T,* $\{ p_1', p_2',\ldots, p_k' \}$ *το σύνολο των ξεχωριστών μονοπατιών από την ρίζα μέχρι τα φύλλα και* $P'=\{ P_1', P_2',\ldots, P_k' \}$ *το σύνολο των συνόλων των κόμβων για κάθε μονοπάτι* p_i' $(1 \le i \le k)$*, εξαιρουμένων όλων των Δ κόμβων. Τότε,* $P \subseteq P'$ *και* $P' \subseteq P$*, και άρα* $P=P'$*.*

3.3 Εξάρτηση Διαστάσεων και Γράφος Εξαρτήσεων

Κάποιες διαστάσεις από μια διαμέριση ενός δένδρου μπορούν να καθορίσουν άλλες διαστάσεις, υπό την έννοια ότι οι τιμές κάποιων διακριτικών χαρακτηριστικών πρέπει να είναι 'απόγονοι' των τιμών άλλων χαρακτηριστικών στο γνωστικό πεδίο ορισμού που περιγράφει το δένδρο. Για παράδειγμα, η τιμή *Inkjet* (διάσταση Δ_1) πρέπει πάντα να είναι απόγονος της τιμής *Printers* (διάσταση Δ_2) στο H_1 (Εικόνα 3.1). Σε αυτή την περίπτωση λέμε ότι το Δ_1 *εξαρτάται από το* Δ_2. Από την άλλη μεριά, η τιμή *Mac* θα μπορούσε να είναι πάνω από την τιμή *Notebooks* στο H_2, καθώς μία κατηγοριοποίηση που ξεκινάει με τα brand names και μετά συνεχίζει με τα hardware types, αντί να αρχίζει με τα hardware types και μετά να συνεχίζει με τα brand names, είναι εξίσου εφαρμόσιμη. Σε αυτή την περίπτωση λέμε ότι οι σχετικές διαστάσεις είναι *ανεξάρτητες*.

Οι εξαρτήσεις μεταξύ διαστάσεων περιορίζουν τον τρόπο με τον οποίο κατασκευάζονται τα κανονικοποιημένα δένδρα. Αν μία διάσταση Δ_1 εξαρτάται από μία διάσταση Δ_2, τότε η ακολουθία των διακριτικών χαρακτηριστικών που χρησιμοποιείται για την κατασκευή του κανονικοποιημένου δένδρου δεν μπορεί να περιέχει τα χαρακτηριστικά του Δ_2 μετά από αυτά του Δ_1. Για παράδειγμα, έστω ότι

1. (*peripheral_type*, {*Monitors*, *Printers*, *Storage*}) και

2. (*printer_type*, {*Inkjet*, *MonoLaser*, *ColorLaser*})

είναι 2 από τις διαστάσεις για την ιεραρχία H_1 στην Εικόνα 3.1. Τότε, το *peripheral_type* πρέπει πάντα να προηγείται του *printer_type* σε ένα κανονικοποιημένο δένδρο που βασίζεται σε μία διαμέριση του H_1.

Ένας γράφος εξαρτήσεων μπορεί να ορισθεί για να τυποποιήσουμε τις εξαρτήσεις των διαστάσεων.

Ορισμός 3.4. *Έστω T ένα δένδρο με N κόμβους, χωρίς την ρίζα r, που αναπαριστά μία ιεραρχία και* $\{(P_1, V_1), (P_2, V_2),..., (P_m, V_m)\}$ *μία ολική διαμέριση P του T, όπου* V_i *ένα σύνολο από τιμές για το διακριτικό χαρακτηριστικό* P_i $(1 \leq i \leq m)$. *Ο* γράφος εξαρτήσεων *G για το P κατασκευάζεται χρησιμοποιώντας κόμβους* $n_1, n_2,\ldots, n_k$ *και ακμές* $n_i \rightarrow n_j$ *για κάθε ζευγάρι διακριτικών χαρακτηριστικών* (P_i, P_j) *με το* P_i *να εξαρτάται από το* P_j $(1 \leq i \leq k, 1 \leq j \leq k, i \neq j, k \leq m)$. *Κάθε εξάρτηση που εμφανίζεται σε αυτόν τον γράφο πρέπει να υπάρχει και στο T, δηλαδή αν το* P_i *ορίζεται να εξαρτάται από το* P_j, *τότε οι τιμές του* P_i *πρέπει να είναι απόγονοι των τιμών του* P_j *στο T.*

Αν θεωρήσουμε την ρίζα *r* σαν την μόνη τιμή ενός διαχωριστικού χαρακτηριστικού *Root*, ο γράφος των εξαρτήσεων θα έπρεπε να περιέχει και τις ακμές $n_i \rightarrow r$, $(1 \leq i \leq k)$, δηλαδή ακμές από όλους τους άλλους κόμβους του γράφου προς την ρίζα *r*. Για λόγους απόδοσης, αποκλείουμε τους κόμβους-ρίζες *r* από τους γράφους εξαρτήσεων.
Τώρα θα καθορίσουμε εξαρτήσεις που προκύπτουν από τον γράφο εξαρτήσεων.

Ορισμός 3.5. *Έστω*

1. *T ένα δένδρο με N κόμβους, χωρίς την ρίζα r, που αναπαριστά μία ιεραρχία,*
2. *$\{(P_1, V_1), (P_2, V_2),..., (P_m, V_m)\}$ μία ολική διαμέριση P του T, όπου V_i ένα σύνολο από τιμές για το διακριτικό χαρακτηριστικό P_i $(1 \leq i \leq m)$, και*
3. *G ο γράφος εξαρτήσεων για το P.*

Το P_i εξαρτάται από το P_j *όταν υπάρχει μονοπάτι στον G που να ενώνει τους κόμβους n_i, n_j, με n_i, n_j να αναπαριστούν τα P_i και P_j αντίστοιχα $(1 \leq i \leq m, 1 \leq j \leq m, i \neq j)$.*

Ομοίως:

Ορισμός 3.6. *Έστω*

1. *T ένα δένδρο με N κόμβους, χωρίς την ρίζα r, που αναπαριστά μία ιεραρχία,*
2. *$\{(P_1, V_1), (P_2, V_2),..., (P_m, V_m)\}$ μία ολική διαμέριση P του T, όπου V_i ένα σύνολο από τιμές για το διακριτικό χαρακτηριστικό P_i $(1 \leq i \leq m)$, και*
3. *G ο γράφος εξαρτήσεων για το P.*

Το P_i δεν εξαρτάται από το P_j *όταν δεν υπάρχει μονοπάτι στον G που να ενώνει τους κόμβους n_i, n_j, με n_i, n_j να αναπαριστούν τα P_i και P_j αντίστοιχα $(1 \leq i \leq m, 1 \leq j \leq m, i \neq j)$.*

Όπως είναι λογικό, δεν μπορεί να υπάρχουν κύκλοι στον γράφο αυτόν, γιατί δεν μπορούν δύο διαστάσεις να αλληλεξαρτώνται.
Έγκυρα κανονικοποιημένα δένδρα είναι αυτά που δεν παραβιάζουν τον γράφο των εξαρτήσεων, όπως εκφράζει ο επόμενος ορισμός.

Ορισμός 3.7. *Έστω*

1. *T ένα δένδρο με N κόμβους, χωρίς την ρίζα r, που αναπαριστά μία ιεραρχία,*
2. *$\{(P_1, V_1), (P_2, V_2),..., (P_m, V_m)\}$ μία ολική διαμέριση P του T, όπου V_i ένα σύνολο από τιμές για το διακριτικό χαρακτηριστικό P_i $(1 \leq i \leq m)$,*
3. *G ο γράφος εξαρτήσεων για το P, και*

4. *S μία ακολουθία των διακριτικών χαρακτηριστικών του P.*

Ένα κανονικοποιημένο δένδρο είναι έγκυρο *όσον αφορά την ακολουθία S, όταν για κάθε ζεύγος* (P_i, P_j)*, με το* P_i *πριν το* P_j *στο S* $(1 \leq i \leq m, 1 \leq j \leq m, i \neq j)$*, δεν υπάρχει μονοπάτι στον G που να ενώνει τους κόμβους* n_i*,* n_j*, με* n_i*,* n_j *να αναπαριστούν τα* P_i *και* P_j *αντίστοιχα.*

Ένας γράφος εξαρτήσεων κατασκευάζεται είτε για να εκφράσει περιορισμούς που υποδηλώνονται από το ίδιο το γνωστικό πεδίο ορισμού, ή για περιορισμούς ορισμένους από τον χρήστη. Η περίπτωση που παρουσιάστηκε παραπάνω, στην αρχή αυτής της ενότητας, όπου η τιμή *Inkjet* (διάσταση Δ_1) έπρεπε πάντα να είναι απόγονος της τιμής *Printers* (διάσταση $\Delta 2$) στο H_1 (Εικόνα 3.1), είναι ένα παράδειγμα που παρουσιάζει τους περιορισμούς που υποδηλώνονται από το γνωστικό πεδίο ορισμού, καθώς δεν υπάρχουν αντικείμενα '*Inkjet*' πέραν από εκτυπωτές (printers) σε ένα πεδίο ορισμού που αναφέρεται σε hardware Η/Υ. Από την άλλη, θα μπορούσαμε να ορίσουμε μία εξάρτηση μεταξύ του *brand_name* και του *product_type* στο H_2 (Εικόνα 3.1, 3.2), για να εκφράσουμε έναν περιορισμό ορισμένο από τον χρήστη, έτσι ώστε σε κάθε περίπτωση, το κανονικοποιημένο δένδρο να έχει την κατηγοριοποίηση που βασίζεται στο *brand_name* μετά από αυτή που βασίζεται στο *product_type*. Σε αυτή την περίπτωση, το δένδρο (a) στην Εικόνα 3.2 δεν είναι ένα έγκυρο κανονικοποιημένο δένδρο.

3.4 Καθολικό Κανονικοποιημένο Δένδρο

Ό,τι περιγράψαμε παραπάνω έχει να κάνει με μία ιεραρχία κάθε φορά. Ο σκοπός είναι να κατασκευάσουμε ένα κανονικοποιημένο δένδρο, που είναι μία πολύ ευέλικτη δομή, για πολλές ιεραρχίες μαζί. Το πρώτο βήμα είναι να έχουμε για όλες τις ιεραρχίες που θα συμμετέχουν μία πλήρη διαμέριση, δηλαδή τις διαστάσεις και τους κόμβους που τις αποτελούν, με όλους τους κόμβους να ανήκουν σε κάποια διάσταση.

Το επόμενο βήμα είναι η συγχώνευση των διαστάσεων. Εδώ παίζει ρόλο η ονομασία των ετικετών των κόμβων της κάθε ιεραρχίας, καθώς μπορεί 2 ή περισσότερες ιεραρχίες να αναφέρονται στην ίδια έννοια χρησιμοποιώντας διαφορετικά ονόματα. Προς το παρόν θεωρούμε το πρόβλημα λυμένο, αν και δεν απαιτείται για την λειτουργία του συστήματος οι ιεραρχίες να έχουν τα ίδια ονόματα στις ετικέτες που ταυτίζεται η έννοιά τους (θα δούμε παρακάτω στο Κεφάλαιο 5 πως έχουμε κωδικοποιήσει την λύση). Όμως, αυτό που έχει σημασία είναι 2 κόμβοι διαφορετικών ιεραρχιών που ταυτίζονται εννοιολογικά να ανήκουν στην ίδια διάσταση. Για παράδειγμα, στην ιεραρχία H_1 στην Εικόνα 3.1, πρέπει ο κόμβος με το όνομα *Mac* να ανήκει στην διάσταση *brand_name*, τόσο στην διαμέριση της H_1, όσο και

στην διαμέριση της H_2. Σε αυτό το σημείο πρέπει να τονίσουμε ότι θεωρούμε πως δεν υπάρχει πρόβλημα χρησιμοποίησης διαφορετικών ονομάτων για ταυτόσημες διαστάσεις σε διαφορετικές ιεραρχίες. Άρα χρησιμοποιούνται τα ίδια ονόματα για τις διαστάσεις σε όλες τις ιεραρχίες.

Υπό την προϋπόθεση, λοιπόν, ότι ταυτόσημοι κόμβοι ανήκουν στην ίδια διάσταση, εφαρμόζουμε την συγχώνευση. Το καινούργιο σύνολο των διαστάσεων θα αποτελείται από τις διαστάσεις που προέρχονται από τις διαμερίσεις όλων των ιεραρχιών, δηλαδή την ένωση των επιμέρους συνόλων των διαστάσεων. Προφανώς σε αυτό το σύνολο μπορεί να ανήκουν διαστάσεις που δεν υπάρχουν σε κάποιες ιεραρχίες. Όσον αφορά τα στοιχεία του συνόλου της κάθε διάστασης, και εδώ εφαρμόζεται μια απλή ένωση όλων των συνόλων των ταυτόσημων διαστάσεων. Ομοίως και εδώ, μπορεί σε κάποιες διαστάσεις να υπάρχουν κόμβοι που δεν υπάρχουν σε κάποιες ιεραρχίες, ως αποτέλεσμα της ένωσης. Για παράδειγμα, έστω μια ιεραρχία H_1 στην οποία έχουμε ορίσει τις δύο αυτές διαστάσεις:

1. Διάσταση 1: (*brand_name*, {*Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*}).
2. Διάσταση 2: (*product_type*, {*Desktops*, *Notebooks*, *PDAs*}).

Και μια ιεραρχία H_2, στην οποία ορίζουμε τις ακόλουθες 3 διαστάσεις:

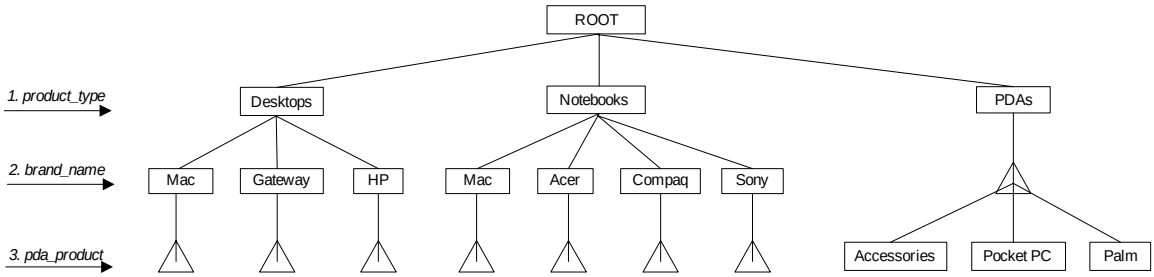
1. Διάσταση 1: (*brand_name*, {*Gateway*, *HP*, *Compaq*, *Sony*}).
2. Διάσταση 2: (*product_type*, {*Desktops*, *Notebooks*, *Printers*}).
3. Διάσταση 3: (*printer_type*, {*Black_and_White*, *Color*}).

Εάν συγχωνεύσουμε αυτά τα 2 σύνολα διαστάσεων θα πάρουμε το εξής τελικό:

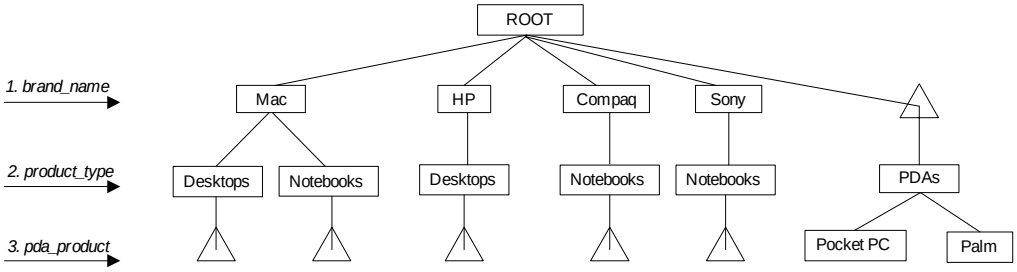
1. Διάσταση 1: (*brand_name*, { *Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony* }).
2. Διάσταση 2: (*product_type*, {*Desktops*, *Notebooks*, *PDAs*, *Printers*}).
3. Διάσταση 3: (*printer_type*, {*Black_and_White*, *Color*}).

Έχοντας το καθολικό σύνολο των διαστάσεων, όπως κάναμε και στις ιεραρχίες μεμονωμένα με τα κανονικοποιημένα δένδρα, και εδώ μπορούμε να κατασκευάσουμε το *καθολικό κανονικοποιημένο δένδρο*, δηλαδή ένα κανονικοποιημένο δένδρο όπως και τα προηγούμενα, μόνο που τώρα λαμβάνονται υπ'όψιν όλοι οι κόμβοι και οι διαστάσεις όλων των ιεραρχιών. Σημασία έχει να προσέξουμε ότι το καθολικό κανονικοποιημένο δένδρο υπερκαλύπτει όλα τα κανονικοποιημένα δένδρα των ιεραρχιών, δηλαδή δεν υπάρχει περίπτωση να υπάρχει σε κάποιο κανονικοποιημένο δένδρο πληροφορία που δεν υπάρχει στο καθολικό

κανονικοποιημένο δένδρο. Για παράδειγμα, ακολουθούν δύο σχήματα. Στο πρώτο φαίνεται το καθολικό κανονικοποιημένο δένδρο της Εικόνας 3.2 (a), και στο δεύτερο ένα τοπικό κανονικοποιημένο δένδρο. Παρατηρούμε ότι το τοπικό κανονικοποιημένο δένδρο αποτελεί (πάντα) υποσύνολο του καθολικού.



Εικόνα 3.3 Ένα Καθολικό Κανονικοποιημένο Δένδρο



Εικόνα 3.4 Ένα τοπικό Κανονικοποιημένο Δένδρο

Στις δύο προηγούμενες εικόνες βλέπουμε ότι το τοπικό κανονικοποιημένο δένδρο περιέχει λιγότερη πληροφορία από το καθολικό, και δεν υπάρχουν μονοπάτια στο τοπικό δένδρο που να μην υπάρχουν στο καθολικό, μιλώντας πάντα για ανεξάρτητη σειρά των κόμβων. Επίσης είναι διαφορετική η σειρά επιλογής των διαστάσεων, κάτι που δεν παίζει ρόλο. Εδώ λοιπόν, το τοπικό κανονικοποιημένο δένδρο υπερκαλύπτεται από το καθολικό κανονικοποιημένο δένδρο.

Αυτό που έχει σημασία είναι να κατασκευάσουμε *έγκυρα* καθολικά κανονικοποιημένα δένδρα, όπου δεν παραβιάζονται οι εξαρτήσεις που ο γράφος εξαρτήσεων ορίζει, ακριβώς όπως και στα απλά κανονικοποιημένα δένδρα.

4

Ανάλυση Απαιτήσεων

Στο Κεφάλαιο αυτό παρουσιάζουμε την ανάλυση των απαιτήσεων του συστήματός μας, δηλαδή περιγράφουμε την αρχιτεκτονική του συστήματος και τις λειτουργικές εργασίες που πρέπει να εκτελούνται στα ξεχωριστά μέρη του συστήματός μας.

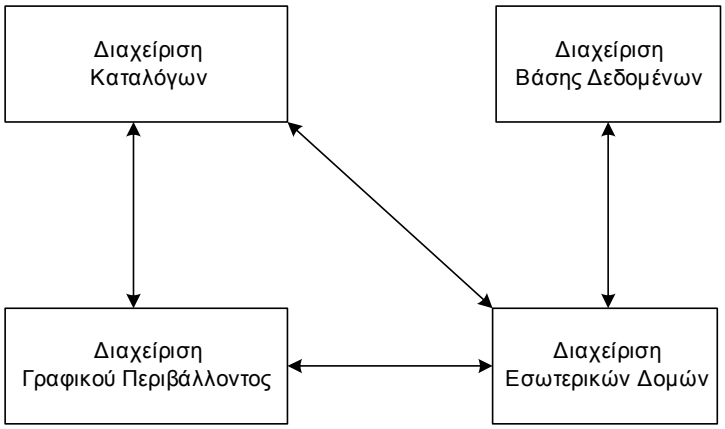
4.1 Περιγραφή Αρχιτεκτονικής

Για την άρτια λειτουργία του συστήματος πρέπει να εκτελούνται επιμέρους λειτουργίες που αφορούν την διαχείριση κάποιων συγκεκριμένων τμημάτων του συστήματος. Τα τμήματα αυτά αναφέρονται εδώ, και οι λειτουργίες που απαιτούνται από το καθένα αναφέρονται στην επόμενη ενότητα.

- **Διαχείριση Καταλόγων:** Αφορά τις λειτουργίες με τις οποίες χειριζόμαστε τις ιεραρχίες που συμμετέχουν στο σύστημα.
- **Διαχείριση Εσωτερικών Δομών:** Αφορά τις λειτουργίες με τις οποίες χειριζόμαστε τις εσωτερικές δομές, τις οποίες αναφέραμε στο προηγούμενο κεφάλαιο, και αυτές, πέρα από τις αρχικές ιεραρχίες, είναι οι διαστάσεις, ο γράφος εξαρτήσεων και το καθολικό κανονικοποιημένο δένδρο.

- **Διαχείριση της Βάσης Δεδομένων:** Εδώ ανήκουν οι λειτουργίες με τις οποίες εφαρμόζουμε τις ερωτήσεις στην Βάση Δεδομένων, στην οποία φυλάμε τις εγγραφές και τα δεδομένα των ιεραρχιών, και οι λειτουργίες με τις οποίες παίρνουμε τα αποτελέσματα από αυτήν.
- **Διαχείριση Γραφικού Περιβάλλοντος:** Αφορά όλες τις λειτουργίες που έχουν να κάνουν με την γραφική απεικόνιση εσωτερικών δομών, καθώς και με την όλη παρουσίαση της διαπροσωπείας χρήστη.

Μία σχηματική αναπαράσταση αυτών των τμημάτων του συστήματος μπορούμε να δούμε αμέσως παρακάτω, στην Εικόνα 4.1. Βλέπουμε και τον τρόπο με τον οποίο συσχετίζονται τα επιμέρους τμήματα. Η διαχείριση της Βάσης Δεδομένων γίνεται μέσα από την επεξεργασία των εσωτερικών δομών, ενώ τα υπόλοιπα τμήματα αλληλεπιδρούν μεταξύ τους.



Εικόνα 4.1 Σχηματική Αναπαράσταση της Αρχιτεκτονικής

4.2 Περιγραφή Λειτουργιών

Σε αυτή την ενότητα παρουσιάζουμε όλες τις λειτουργίες και τις απαιτήσεις, οι οποίες εκτελούνται από τα τμήματα που περιγράψαμε στην αμέσως προηγούμενη ενότητα.

4.2.1 Λειτουργίες Διαχείρισης Καταλόγων

Όπως αναφέραμε προηγουμένως, εδώ ανήκουν οι απαιτήσεις και οι λειτουργίες που έχουν να κάνουν με την διαχείριση των ιεραρχιών. Πιο συγκεκριμένα, οι λειτουργίες αυτές περιγράφονται ως εξής:

- **Αναπαράσταση Ιεραρχιών σε Αρχείο:** Δηλαδή ο τρόπος με τον οποίο οι ιεραρχίες που συμμετέχουν στο σύστημα αναπαρίστανται σε αρχεία στον σκληρό δίσκο.
- **Αναπαράσταση Ιεραρχιών στην Μνήμη:** Εδώ αναφερόμαστε στην λειτουργία μεταφοράς των ιεραρχιών σε δομές στην μνήμη.
- **Αναπαράσταση Δεδομένων των Ιεραρχιών:** Οι ιεραρχίες πρέπει στα φύλλα τους να αναφέρονται με κάποιον τρόπο στα δεδομένα στα οποία αναφέρονται, τόσο στα αρχεία του σκληρού δίσκου, όσο και στις δομές της μνήμης.
- **Επεξεργασία των Ιεραρχιών:** Εδώ αναφερόμαστε στον τρόπο με τον οποίο κάνουμε διάσχιση στις ιεραρχίες και παίρνουμε την πληροφορία των κόμβων.

4.2.2 Λειτουργίες Διαχείρισης Εσωτερικών Δομών

Οι απαιτήσεις και οι λειτουργίες της διαχείρισης των εσωτερικών δομών είναι αρκετές, και κυρίως έχουν να κάνουν με την επεξεργασία και την τροποποίηση της δομικής τους κατασκευής.

4.2.2.1 Διαχείριση των Διαστάσεων

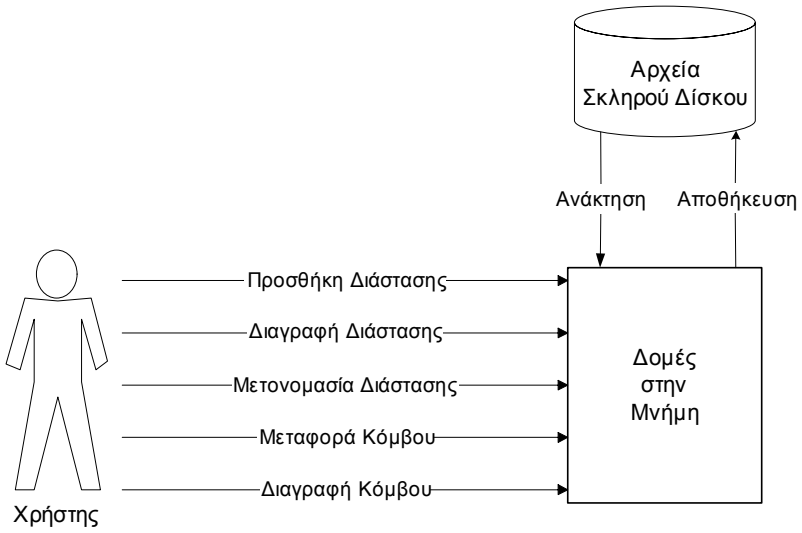
Οι λειτουργίες αυτές συνίστανται στα εξής:

- **Αναπαράσταση Διαστάσεων σε Αρχεία:** Αφορά τον τρόπο με τον οποίο τα σύνολα των διαστάσεων της κάθε ιεραρχίας δηλώνονται με αρχεία στον σκληρό δίσκο.
- **Αναπαράσταση Διαστάσεων στην Μνήμη:** Αυτό το κομμάτι αναφέρεται στο γεγονός της αναπαράστασης των διαστάσεων, ως δομές, στην μνήμη. Έχουμε μία τέτοια δομή για κάθε σύνολο διαστάσεων, και μία ακόμη, που είναι η συγχώνευση όλων των επιμέρους συνόλων.
- **Ανάκτηση από τον Σκληρό Δίσκο:** Αφορά την λειτουργία μεταφοράς των δεδομένων από τα αρχεία που αναπαριστούν τα σύνολα των διαστάσεων σε δομές στην μνήμη.
- **Αποθήκευση στον Σκληρό Δίσκο:** Εδώ μιλάμε για τις λειτουργίες μεταφοράς σε αρχεία του σκληρού δίσκου των δομών της μνήμης, που αναπαριστούν τα σύνολα των διαστάσεων των ιεραρχιών.
- **Τροποποίηση των Διαστάσεων:** Εδώ αναφερόμαστε στις λειτουργίες με τις οποίες μεταβάλλουμε την δομή του συνόλου διαστάσεων που έχει προκύψει από την συγχώνευση των επιμέρους συνόλων, κάτι που γίνεται με διάφορους τρόπους. Οι μεταβολές μεταφέρονται από τη γενική δομή του συνόλου των διαστάσεων στις

επιμέρους δομές, όταν οι αλλαγές που γίνονται στην γενική δομή επηρεάζουν και τις τοπικές δομές. Αυτές οι ενέργειες μπορούν να χωρισθούν ως εξής:

- Προσθήκη μιας νέας διάστασης
- Μεταφορά ενός κόμβου από τη μια διάσταση σε μια άλλη
- Διαγραφή ενός κόμβου από μια διάσταση
- Διαγραφή μιας υπάρχουσας διάστασης
- Αλλαγή ονόματος μιας διάστασης

Ένας καλύτερος τρόπος να αντιληφθούμε τις λειτουργίες που εκτελούνται από το τμήμα διαχείρισης των διαστάσεων είναι το παρακάτω σχήμα:



Εικόνα 4.2 Σχηματική Αναπαράσταση της Διαχείρισης Διαστάσεων

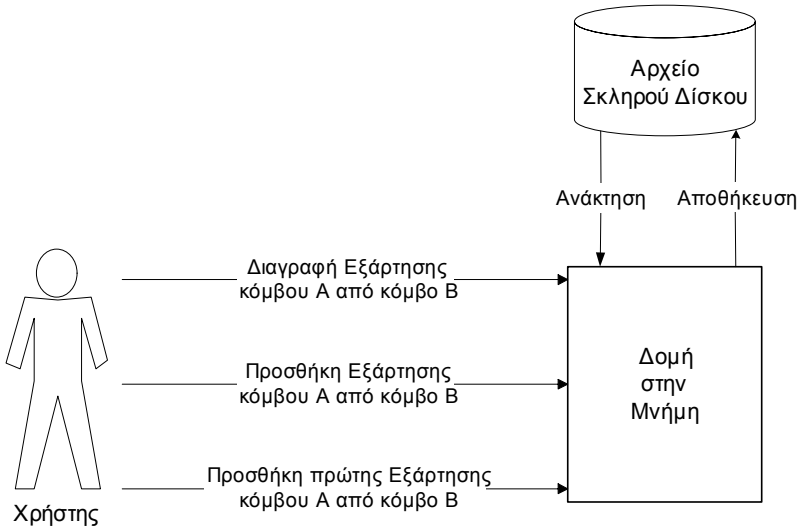
4.2.2.2 Διαχείριση του Γράφου Εξαρτήσεων

Αντίστοιχες είναι και οι απαιτήσεις και οι λειτουργίες για την διαχείριση του γράφου εξαρτήσεων, και άρα έχουμε:

- **Αναπαράσταση του Γράφου Εξαρτήσεων σε Αρχείο:** Εδώ αναφερόμαστε στο γεγονός ότι ο γράφος αυτός πρέπει με κάποιον τρόπο να αναπαρίσταται σε ένα αρχείο, ώστε να φορτώνεται κατά την εκκίνηση του συστήματος.
- **Αναπαράσταση του Γράφου Εξαρτήσεων στην Μνήμη:** Αναφερόμαστε στο γεγονός ότι ο γράφος εξαρτήσεων πρέπει να φορτώνεται σε μια δομή στην μνήμη, ώστε να είναι ευέλικτη η αναζήτηση εξαρτήσεων, και οι διάφορες τροποποιήσεις.

- **Ανάκτηση από τον Σκληρό Δίσκο:** Εδώ αναφερόμαστε στην λειτουργία μεταφοράς του γράφου εξαρτήσεων από το αρχείο στην δομή, με την οποία τον αναπαριστούμε στην μνήμη.
- **Αποθήκευση στον Σκληρό Δίσκο:** Αναφερόμαστε στην λειτουργία μεταφοράς της δομής από την μνήμη στο αρχείο, ώστε να φυλάμε τις αλλαγές που τυχόν κάνουμε στην δομή του γράφου.
- **Τροποποίηση του Γράφου Εξαρτήσεων:** Αυτό το κομμάτι αναφέρεται στις λειτουργίες, με τις οποίες μπορούμε να αλλάξουμε την δομή του γράφου. Αυτές οι λειτουργίες μπορούν να χωριστούν στις εξής:
    - Διαγραφή της εξάρτησης ενός κόμβου Α του γράφου, από έναν κόμβο Β (δηλ. μιας διάστασης Α από μια διάσταση Β, καθώς οι κόμβοι στον γράφο εξαρτήσεων αναφέρονται σε διαστάσεις)
    - Προσθήκη εξάρτησης ενός κόμβου Α από έναν κόμβο Β, όταν ο Α ήδη εξαρτάται από κάποιον άλλο κόμβο Γ (δηλαδή προσθήκη ακμής που ξεκινάει από τον Α και καταλήγει σε έναν <ίσως και νέο> κόμβο Β).
    - Προσθήκη εξάρτησης ενός κόμβου Α από έναν κόμβο Β, όπου ο Α δεν υπήρχε προηγουμένως στον γράφο (δηλαδή δημιουργία πρώτης εξάρτησης για τον κόμβο Α).

Και εδώ, προκειμένου να γίνουν πιο σαφείς οι λειτουργίες αυτού του τμήματος, παραθέτουμε μια σχηματική αναπαράσταση των λειτουργιών που εκτελούνται:



Εικόνα 4.3 Σχηματική Αναπαράσταση της Διαχείρισης του Γράφου Εξαρτήσεων

4.2.2.3 Διαχείριση του Καθολικού Κανονικοποιημένου Δένδρου

Στο κομμάτι αυτό αναφερόμαστε στις απαιτήσεις και τις λειτουργίες που πρέπει να εκτελούνται κατά την διαχείριση του καθολικού κανονικοποιημένου δένδρου. Να σημειώσουμε ότι επειδή το καθολικό κανονικοποιημένο δένδρο υπερκαλύπτει όλα τα τοπικά κανονικοποιημένα δένδρα, δεν χρειάζεται να κατασκευαστούν αυτά, αλλά απλά χρησιμοποιούμε από το καθολικό κανονικοποιημένο δένδρο τα κομμάτια που εφαρμόζονται σε κάθε ιεραρχία. Οι λειτουργίες και οι απαιτήσεις μπορούν να χωριστούν στις εξής:

- **Αναπαράσταση του Καθολικού Κανονικοποιημένου Δένδρου στην Μνήμη:** Αυτό το κομμάτι αναφέρεται στην απαίτηση της αναπαράστασης της δομής αυτής στην μνήμη.
- **Δυναμική Κατασκευή του Καθολικού Κανονικοποιημένου Δένδρου στην Μνήμη:** Εδώ αναφερόμαστε στο γεγονός ότι η κατασκευή πρέπει να γίνεται δυναμικά, όπως μας υπαγορεύει και η θεωρία που αναλύσαμε στο Κεφάλαιο 3, καθώς ο χρήστης είναι αυτός που με τις επιλογές του κατευθύνει την κατασκευή του καθολικού κανονικοποιημένου δένδρου.
- **Κατασκευή ανάλογων Καθολικών Κανονικοποιημένων Δένδρων:** Αναφερόμαστε στην λειτουργία όπου το καθολικό κανονικοποιημένο δένδρο παίρνει διάφορες μορφές, προκειμένου να αποτιμηθεί στις επιμέρους ιεραρχίες.

4.2.3 Λειτουργίες Διαχείρισης της Βάσης Δεδομένων

Στο κομμάτι αυτό αναφερόμαστε στις απαιτήσεις και λειτουργίες που έχουν να κάνουν με την Βάση Δεδομένων. Το σύστημα που κατασκευάσαμε δεν προβλέπει μεταβολή των δεδομένων αυτών, δηλαδή η Βάση Δεδομένων χρησιμοποιείται μόνο για την φύλαξη των δεδομένων της κάθε ιεραρχίας, τα οποία φυσικά και μπορούμε να πάρουμε όταν ζητηθούν από τον χρήστη. Άρα μία απαίτηση είναι να έχουμε στην Βάση Δεδομένων αποθηκευμένες τις εγγραφές και τα δεδομένα στα οποία αναφέρεται η κάθε ιεραρχία. Οι μόνες λειτουργίες που έχουν να κάνουν με την Βάση Δεδομένων είναι η εφαρμογή των ερωτήσεων και η συλλογή των αποτελεσμάτων σε δομές, τις οποίες θα εξετάσουμε στο παρακάτω κεφάλαιο

4.2.4 Λειτουργίες Διαχείρισης του Γραφικού Περιβάλλοντος

Το γραφικό περιβάλλον είναι το κομμάτι που ενώνει όλα τα προηγούμενα κομμάτια, καθώς μέσα από αυτό ο χρήστης κάνει όλες τις ενέργειες. Η κυριότερη – και πιο γενική – απαίτηση είναι να παρέχει την είσοδο και την έξοδο των προαναφερόμενων λειτουργιών, και τα κουμπιά με τα οποία αρχίζουν να εκτελούνται. Πέρα από την έναρξη των λειτουργιών και το πέρασμα των παραμέτρων μέσα από τις επιλογές του χρήστη, υπάρχουν και κάποιες άλλες απαιτήσεις. Όλα αυτά μπορούμε να τα κατηγοριοποιήσουμε στα εξής:

- **Είσοδος και Έξοδος των Λειτουργιών – Πέρασμα παραμέτρων:** Όπως αναφέραμε και πιο πάνω, πρέπει να δίνεται στον χρήστη η δυνατότητα να διαλέγει τις παραμέτρους για τις λειτουργίες που το απαιτούν, και να επιλέγει τις λειτουργίες που θέλει να εκτελεστούν, με τα διάφορα κουμπιά εκκίνησης.
- **Φιλικό προς τον Χρήστη Γραφικό Περιβάλλον:** Είναι αυτονόητο ότι το γραφικό περιβάλλον πρέπει να είναι εύχρηστο, προσιτό, όχι ιδιαίτερα πολύπλοκο, εύκολο και γρήγορο στην μάθηση του. Επίσης πρέπει να προσέχουμε τις επιλογές που δίνουμε στον χρήστη, δηλαδή να μην εμφανίζουμε καν επιλογές που μπορούν να οδηγήσουν σε σφάλμα του συστήματος. Επίσης, απαραίτητο είναι τα μηνύματα λάθους που δίνονται να είναι κατατοπιστικά.
- **Εμφάνιση-Αναπαράσταση των Εσωτερικών Δομών:** Άλλη μία λειτουργία του γραφικού περιβάλλοντος είναι η γραφική αναπαράσταση και παρουσίαση των βασικών εσωτερικών δομών που χρησιμοποιούνται από το σύστημα, δηλαδή η αναπαράσταση των ιεραρχιών, των συνόλων των διαστάσεων, του γράφου εξαρτήσεων, και του καθολικού κανονικοποιημένου δένδρου. Επίσης, απαιτείται και η παρουσίαση των αποτελεσμάτων των ερωτήσεων, που φορτώνονται σε εσωτερικές δομές.

Περισσότερες πληροφορίες για τον τρόπο υλοποίησης όλων αυτών των λειτουργιών ακολουθούν στο επόμενο κεφάλαιο, που αναφέρεται στην Σχεδίαση του Συστήματος.

5

Σχεδίαση Συστήματος

Εδώ αναφερόμαστε στα υποσυστήματα τα οποία αποτελούν το όλο σύστημα, και εξηγούμε τον τρόπο με τον οποίο εκτελούν τις λειτουργίες που αναφέραμε στο προηγούμενο κεφάλαιο.

5.1 Υποσύστημα Διαχείρισης Καταλόγων

Το υποσύστημα αυτό υλοποιεί τις λειτουργίες και τις απαιτήσεις που περιγράψαμε στο προηγούμενο κεφάλαιο για την διαχείριση των καταλόγων.

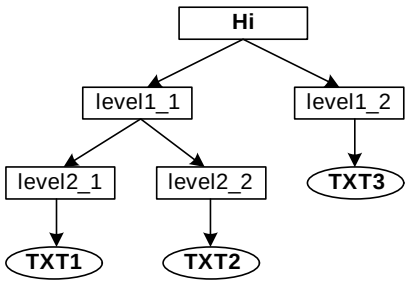
5.1.1 Διαδικασία Διαχείρισης των Καταλόγων ως Αρχεία

Αυτό το υπο-υποσύστημα αναλαμβάνει την αναπαράσταση των ιεραρχιών σε αρχεία, και την διαχείρισή τους.

Πιο συγκεκριμένα, η αναπαράσταση των ιεραρχιών γίνεται με αρχεία XML, τα οποία δεν ακολουθούν κάποια συγκεκριμένη δομή που να περιγράφεται με κάποιο DTD, αλλά είναι απαραίτητο να αντανακλούν την (δενδρική) δομή των ιεραρχιών. Δηλαδή, αυτά τα αρχεία XML δεν χρησιμοποιούνται τόσο για την σημασιολογική πληροφορία που παρέχουν, όσο για την μορφολογία της δομής. Όπως ξέρουμε, τα XML αρχεία μπορούν να θεωρηθούν σαν

δενδρικές δομές, και αυτό είναι που ζητάμε. Χρησιμοποιώντας φωλιασμένα tags έχουμε την δυνατότητα να αναπαραστήσουμε πλήρως τις ιεραρχίες. Εάν υπάρχει πληροφορία μέσα στα tags, σαν attributes, αυτή δεν λαμβάνεται υπ'όψιν από το συγκεκριμένο σύστημα. Για παράδειγμα, ένα αρχείο XML που αναπαριστά μία ιεραρχία θα μπορούσε να έχει την παρακάτω μορφή, η οποία καλύπτει ακριβώς τις απαιτήσεις, χωρίς περιττές πληροφορίες και χωρίς ελλιπή δομή. Δίπλα είναι το σχήμα της δομής που περιγράφει.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Hi>
    <level1_1>
      <level2_1>TXT1</level2_1>
      <level2_2>TXT2</level2_2>
    </level1_1>
    <level1_2>TXT3</level1_2>
</Hi>
```



Εικόνα 5.1

Με **έντονη** γραφή είναι τα στοιχεία που πρέπει να υπάρχουν σε κάθε αρχείο XML που αναπαριστά ιεραρχίες. Το πρώτο αφορά το πρώτο tag σε κάθε αρχείο, το οποίο έχει το όνομα της κάθε ιεραρχίας (για την εφαρμογή του συστήματός μας αυτά τα ονόματα είναι Hi, με i=1,2,3,… και είναι ο τρόπος για να ξεχωρίσουμε τις ιεραρχίες μεταξύ τους). Δεν επιτρέπεται να υπάρχουν πάνω από μία ιεραρχία με το ίδιο όνομα, οπότε τα i μπορούν να πάρουν οποιοδήποτε μη χρησιμοποιημένο φυσικό αριθμό.

Το δεύτερο στοιχείο είναι το κείμενο που βρίσκεται ως πληροφορία στα tags των φύλλων, αυτών δηλαδή που δεν έχουν άλλα φωλιασμένα tags. Την πληροφορία αυτή την αναπαριστούμε με **TXT** στο παράδειγμα που δείξαμε, και αναφέρεται στις εγγραφές που χαρακτηρίζονται από το εκάστοτε μονοπάτι. Έχει σημασία να αναφέρουμε ότι αυτή η πληροφορία βρίσκεται ανάμεσα στα tags, και όχι σαν attribute μέσα στα tags των φύλλων, και ότι επιτρέπεται να υπάρχει μόνο ανάμεσα στα tags των φύλλων.

Το συμπέρασμα είναι ότι από τα XML αρχεία χρειαζόμαστε μόνο την δενδρική πληροφορία, και την πληροφορία για τις εγγραφές, που υπάρχει μόνο στα φύλλα των ιεραρχιών. Αυτή η πληροφορία, το **TXT** που αναφέραμε παραπάνω, είναι ένα SQL query, το οποίο μας επιστρέφει τα δεδομένα που ανήκουν στο εκάστοτε μονοπάτι. Αυτό προτιμήθηκε, αντί εγγραφών στα φύλλα, για την διευκόλυνση των πειραμάτων.

5.1.2 Λειτουργίες Αναπαράστασης των Ιεραρχιών στην Μνήμη

Το υπο-υποσύστημα αυτό αναφέρεται στην επεξεργασία των ιεραρχιών ως δομών στην μνήμη. Καταρχήν, πρέπει να μεταφέρουμε τις πληροφορίες που θέλουμε από τα αρχεία XML των ιεραρχιών που μας δίνονται στην μνήμη. Επειδή μιλάμε για ιεραρχίες με δενδρική μορφή, η εσωτερική δομή που χρησιμοποιήθηκε είναι αυτή ενός δένδρου. Κάθε tag αντιστοιχεί σε έναν κόμβο του δένδρου. Όλα τα άλλα στοιχεία που μπορεί να μας δίνει το XML αρχείο αγνοούνται, εκτός από τα τελευταία επίπεδα, όπου φορτώνουμε στους κόμβους την πληροφορία των TXTs (queries). Περισσότερες πληροφορίες για τον τρόπο υλοποίησης και τις συναρτήσεις που υποστηρίζει η δομή αυτή θα αναφερθούν στο επόμενο κεφάλαιο.
Η τεχνολογία που χρησιμοποιήθηκε για την διάσχιση των XML αρχείων είναι το SAX, με το οποίο κάνουμε διάσχιση των tags, και μπορούμε να επιλέγουμε τι θα φορτώσουμε στην μνήμη, καλώντας κατάλληλες συναρτήσεις. Με τον τρόπο αυτό μπορούμε να επιλέξουμε τα στοιχεία που θα φορτώσουμε στην δομή του δένδρου, και να μην πάρουμε όλες τις πληροφορίες που ενδεχομένως να περιέχονται στα attributes των tags, κάτι που θα γινόταν αν χρησιμοποιούσαμε το DOM. Αυτός είναι και ο κυριότερος λόγος που προτιμήθηκε το SAX έναντι του DOM.
Όσον αφορά τον τρόπο αναπαράστασης των εγγραφών στην μνήμη, όπως είπαμε και προηγουμένως, αυτά τα SQL queries που αναφέρονται στα εκάστοτε δεδομένα και βρίσκονται σαν πληροφορία στα φύλλα των δένδρων, φροντίζουμε να τα φυλάμε σε κάποιες μεταβλητές, οι οποίες σχετίζονται με τους κόμβους των τελευταίων επιπέδων σε κάθε ιεραρχία. Έτσι, ούτε τις εγγραφές χρειάζεται να φορτώνουμε στα φύλλα των δένδρων στην μνήμη, ούτε χρειάζεται να ψάχνουμε στο εκάστοτε αρχείο για τα κατάλληλα queries.

5.2 Υποσύστημα Διαχείρισης Εσωτερικών Δομών

Το υποσύστημα αυτό είναι υπεύθυνο για τις λειτουργίες που περιγράψαμε στο προηγούμενο κεφάλαιο, στην υποενότητα 4.2.2. Θα εξηγήσουμε με ποιον τρόπο υλοποιούνται οι λειτουργίες αυτές, και πως πληρούνται οι απαιτήσεις που είχαν αναφερθεί.

5.2.1 Διαχείριση των Διαστάσεων

Το υπο-υποσύστημα αυτό ικανοποιεί όλες τις απαιτήσεις που προκύπτουν κατά την διαχείριση των διαστάσεων, και μπορεί να χωριστεί σε περαιτέρω μέρη, τα οποία αναφέρουμε και εξηγούμε παρακάτω.

5.2.1.1 Αναπαράσταση των Διαστάσεων σε Αρχεία

Αυτό το υπο-υπο-υποσύστημα διαχειρίζεται την δήλωση των διαστάσεων σε αρχεία. Όπως συμβαίνει με τις ιεραρχίες, έτσι και με τις διαστάσεις, πρέπει το σύστημα να μπορεί να φυλάει την πληροφορία στον σκληρό δίσκο σε αρχεία, ώστε να μπορεί να φορτώνει κατά την εκκίνηση τις δομές. Η διαφορά των διαστάσεων με τις ιεραρχίες είναι ότι οι διαστάσεις μπορούν να τροποποιηθούν από τον χρήστη, ενώ οι ιεραρχίες όχι, και γι'αυτό, όπως θα δούμε λίγο παρακάτω, υπάρχει και ένα υπο-υπο-υποσύστημα για την μεταφορά των δομών σε αρχεία. Ας δούμε τώρα πως πρέπει να είναι αυτά τα αρχεία.

Τα αρχεία των διαστάσεων προτιμήθηκε να αναπαριστώνται σε μορφή XML, γιατί και οι διαστάσεις μπορούν να θεωρηθούν ιεραρχικές, με την εξής έννοια: Για κάθε ιεραρχία, μπορούμε να θεωρήσουμε μια ρίζα, κάτω από την οποία βρίσκονται ως κόμβοι όλες οι διαστάσεις της ιεραρχίας. Κάτω από κάθε τέτοιο κόμβο, βρίσκονται οι κόμβοι της ιεραρχίας, που ανήκουν στην κάθε διάσταση. Σαν παράδειγμα μπορούμε να δούμε την Εικόνα 5.2, που βρίσκεται λίγο πιο κάτω, και θα εξηγηθεί στην επόμενη ενότητα.

Έτσι, και επειδή έχουμε ήδη έτοιμη την ευέλικτη σε διαμορφώσεις δομή του δένδρου, που κατασκευάσαμε, αναπαριστούμε τις ιεραρχίες σαν δένδρα. Αυτό δεν είναι κακή επιλογή, γιατί έτσι έχουμε ευκολία στην διάσχιση, και επίσης ο ορισμός των διαστάσεων δεν ορίζει κάποια σχέση που να αντιβαίνει στον ορισμό των δένδρων. Τονίζουμε ότι η επιλογή των δένδρων έγινε για λόγους ευκολίας στον χειρισμό της δομής, και όχι ως δομικής αντιστοιχίας της έννοιας των διαστάσεων.

Τα XML αυτά αρχεία θεωρούμε ότι έχουν όνομα αυτής της μορφής: dimHi.xml, όπου όπως αναφέραμε και προηγουμένως, i = 1,2,3,..., δηλαδή ο αριθμός που αντιστοιχεί στην κάθε ιεραρχία. Τα αρχεία αυτά μπορούν να οριστούν σύμφωνα με ένα DTD, του οποίου η μορφή είναι η

```
<!ELEMENT dimHi (dimNames+)>
<!ELEMENT dimNames (DIM_NAME)>
<!ELEMENT DIM_NAME (attributes)>
<!ELEMENT attributes (NODE_NAME+)>
<!ELEMENT NODE_NAME EMPTY>
```

όπου dimHi είναι η 'ταυτότητα' του συνόλου διαστάσεων, και DIM_NAME και NODE_NAME είναι οι μεταβλητές, δηλαδή τα ονόματα των διαστάσεων και των κόμβων που τις αποτελούν, αντίστοιχα.

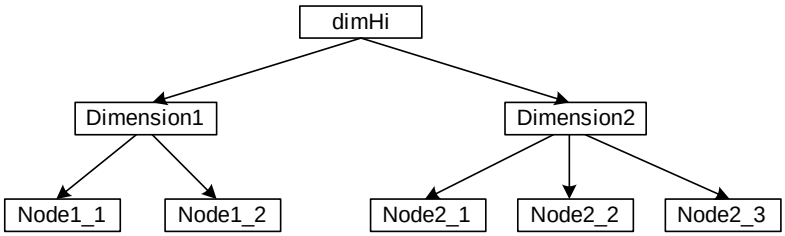
Οπότε, ένα αρχείο dimHi.xml θα έχει την ακόλουθη μορφή, αν θεωρήσουμε ότι ορίζει 2 διαστάσεις (DIMENSION1, {Node1_1, Node1_2}) και (DIMENSION2, {Node2_1, Node2_2, Node2_3}):

```
<?xml version="1.0" encoding="UTF-8" ?>
<dimHi>
  <dimNames>
    <DIMENSION_1>
      <attributes>
        <Node1_1></Node1_1>
        <Node1_2></Node1_2>
      </attributes>
    </DIMENSION_1>
  </dimNames>
  <dimNames>
    <DIMENSION_2>
      <attributes>
        <Node2_1></Node2_1>
        <Node2_2></Node2_2>
        <Node2_3></Node2_3>
      </attributes>
    </DIMENSION_2>
  </dimNames>
</dimHi>
```

5.2.1.2 Αναπαράσταση των Διαστάσεων στην Μνήμη

Όπως αναφέραμε και πιο πάνω, η αναπαράσταση στην μνήμη γίνεται με την δομή του δένδρου, την οποία κατασκευάσαμε και θα αναλύσουμε στο επόμενο κεφάλαιο. Προκειμένου να φορτώσουμε την πληροφορία από τα αρχεία στην μνήμη, όπως έγινε και στην περίπτωση των ιεραρχιών, όπου και εκεί είχαμε XML αρχεία, έτσι και εδώ προτιμήθηκε η τεχνολογία SAX, ώστε να φορτώνουμε τα πεδία που θέλουμε και να τα φυλάμε στις θέσεις που προτιμάμε (αυτό το τελευταίο έχει να κάνει με την τιμή που παίρνουν οι μεταβλητές της δομής του δένδρου). Φροντίζουμε στο δένδρο να μην φορτώσουμε τα επίπεδα που συμβολίζονται με τα tags <dimNames> και <attributes>, γιατί είναι πλεονάζουσα πληροφορία και χρησιμοποιούνται μόνο για τον ορισμό του DTD. Επίσης θα επιβαρύναμε τα δένδρα μας με 2 επιπλέον επίπεδα, κάτι που θα μείωνε την απόδοση και την ευελιξία της δομής, στην περίπτωση που είχαμε αρκετές διαστάσεις και κόμβους.

Οπότε, αν είχαμε το προηγούμενο αρχείο, δηλαδή 2 διαστάσεις, με 2 και 3 κόμβους αντίστοιχα στην κάθε μια, η δομή στην μνήμη, μιας και αναπαρίσταται σαν δένδρο, θα είχε το εξής σχήμα:



Εικόνα 5.2 Απεικόνιση ενός Δένδρου Διαστάσεων

Το υπο-υπο-υποσύστημα, λοιπόν, το οποίο περιγράφουμε τώρα, είναι υπεύθυνο για την μεταφορά της πληροφορίας από τα αρχεία στην μνήμη, και την δημιουργία της δομής του δένδρου κατά την αρχικοποίηση του συστήματος. Προφανώς κατασκευάζεται μία τέτοια δομή για κάθε ιεραρχία που συμμετέχει στο σύστημα, και επίσης άλλη μια δενδρική δομή, η οποία προκύπτει από την συγχώνευση όλων αυτών των δένδρων, και είναι αυτή που θα χρησιμοποιηθεί για την κατασκευή του καθολικού κανονικοποιημένου δένδρου, όπως έχουμε αναφέρει και στην Θεωρητική Μελέτη στο Κεφάλαιο 3.

5.2.1.3 Μεταφορά της Δομής σε Αρχεία.

Αυτό το υπο-υπο-υποσύστημα είναι υπεύθυνο για την σωστή μεταφορά της δενδρικής δομής των διαστάσεων από την μνήμη στα κατάλληλα αρχεία. Σημειώνουμε εδώ ότι τα αρχεία στον σκληρό δίσκο κρατάνε τις μόνιμες αλλαγές που γίνονται στην δομή των διαστάσεων, δηλαδή, όταν πχ. ο χρήστης μεταφέρει κάποιους κόμβους από μια διάσταση σε μια άλλη, οι αλλαγές θα γίνουν στις δομές στην μνήμη. Αν θελήσει να πάρει πίσω τις αλλαγές, οι δομές θα αρχικοποιηθούν ξανά από τα αρχεία στον σκληρό δίσκο, τα οποία περιέχουν τις τελευταίες έγκυρες μορφές των δομών. Αν ο χρήστης θελήσει να κάνει τις αλλαγές μόνιμες, μπορεί να πατήσει το σχετικό κουμπί, και οι αλλαγές θα περάσουν και στα αρχεία του δίσκου.
Το υπο-υπο-υποσύστημα λοιπόν αυτό φροντίζει και γράφει στα σχετικά αρχεία τα στοιχεία που πρέπει, με την σωστή μορφή, ακολουθώντας δηλαδή τους κανόνες του DTD που ορίσαμε πιο πάνω.

5.2.1.4 Τροποποίηση των Διαστάσεων

Εδώ αναφερόμαστε στο υπο-υπο-υποσύστημα που διαχειρίζεται την τροποποίηση των διαστάσεων. Όπως αναφέραμε και στο προηγούμενο κεφάλαιο, στην υποενότητα 4.2.2.1, ο χρήστης ουσιαστικά κάνει τις μετατροπές που θέλει στην δομή των συγχωνευμένων

διαστάσεων, εκεί που ελέγχει όλες τις διαστάσεις και όλους τους κόμβους. Επειδή είναι απαραίτητο να υπάρχει συνέπεια στις κατηγορίες των διαστάσεων, όποιες αλλαγές γίνουν στην δομή των συγχωνευμένων διαστάσεων αυτομάτως περνάνε και στις τοπικές δομές. Ο χρήστης έχει δύο επιλογές, να φυλάξει τις αλλαγές που έκανε, ή να τις αναιρέσει. Εάν φυλάξει τις αλλαγές, τότε οι μετατροπές θα περάσουν και στα αρχεία, ενώ εάν θελήσει να αναιρέσει τις αλλαγές, οι δομές θα ξαναφορτωθούν από τα αρχεία

Μια πιο λογική ιδέα θα ήταν να γίνονται οι αλλαγές στην συγχωνευμένη δομή των διαστάσεων, και αν ο χρήστης αποφάσιζε να τις φυλάξει, τότε να πέρναγαν στις τοπικές δομές, και στο τέλος της εφαρμογής να περνάνε οι αλλαγές στα αρχεία, ενώ αν ο χρήστης αποφάσιζε να αναιρέσει τις αλλαγές, τότε να ξανακατασκευαζόταν η συγχωνευμένη δομή των διαστάσεων από τις τοπικές. Αυτή θα ήταν η πιο λογική λύση, ώστε να μην έχουμε επικοινωνία με τον σκληρό δίσκο και τα αρχεία σε τακτική βάση. Η προηγούμενη λύση προτιμήθηκε γιατί είναι πολύ πιο αποδοτική αλγοριθμικά. Πιο συγκεκριμένα, είναι πολύ πιο εύκολο όταν γίνει μια αλλαγή, πχ μεταφορά ενός κόμβου από μια διάσταση Α σε μια άλλη Β, να ελεγχθεί αν ο κόμβος αυτός ανήκε στην διάσταση Α της εκάστοτε ιεραρχίας (ώστε να γίνει η αλλαγή), από το να πάρουμε το αλλαγμένο συγχωνευμένο δένδρο και να ελέγχουμε εάν ένας κόμβος που δεν υπήρχε σε μια διάσταση πριν, και υπάρχει τώρα, υπάρχει στην εκάστοτε ιεραρχία. Αυτό θα απαιτούσε διασχίσεις (ίσως και) σε ολόκληρα τα δένδρα των ιεραρχιών, κάτι που δεν είναι αποδοτικό.

Οι αλλαγές που μπορούν να γίνουν στην δομή των διαστάσεων είναι η προσθήκη νέας διάστασης, η μεταφορά ενός κόμβου από την μια διάσταση σε μια άλλη, η διαγραφή ενός κόμβου από την διάσταση στην οποία ανήκει, η διαγραφή μιας υπάρχουσας διάστασης, και η αλλαγή ονόματος μιας διάστασης. Η αναλυτική περιγραφή των πράξεων αυτών ακολουθεί παρακάτω:

- **Προσθήκη νέας Διάστασης:** Κατά την προσθήκη μιας νέας διάστασης, έχουμε την δημιουργία ενός νέου κόμβου με το όνομα που επιλέγει ο χρήστης. Αυτός ο κόμβος μπαίνει κάτω από τον κόμβο dimHi, που φαίνεται στην Εικόνα 5.2. Αυτό σημαίνει ότι έχουμε μια νέα διάσταση, η οποία δεν έχει κόμβους που να ανήκουν σε αυτήν ακόμη.
- **Μεταφορά ενός κόμβου από μια Διάσταση σε μια άλλη:** Όταν πραγματοποιείται αυτή η πράξη, ο συγκεκριμένος κόμβος που επιλέγεται μεταφέρεται από τον κόμβο-πατέρα στον οποίο ανήκει, στον επιλεγμένο νέο κόμβο-πατέρα. Εδώ φυσικά γίνεται και κάποιος έλεγχος για την εγκυρότητα της μεταφοράς του κόμβου, καθώς δεν επιτρέπεται να σχηματίζει μονοπάτι με κάποιον άλλο κόμβο που ήδη ανήκει στην νέα διάσταση σε οποιαδήποτε ιεραρχία.

- **Διαγραφή ενός κόμβου από μια Διάσταση:** Αυτή η πράξη έχει συνέπεια την διαγραφή του κόμβου-φύλλου από την δομή των διαστάσεων. Το ερώτημα είναι τι θα γίνει με αυτόν τον κόμβο, καθώς δεν ανήκει σε καμιά διάσταση, αλλά πρέπει να ξέρουμε ότι υπάρχει ελεύθερος, ώστε να μπει σε κάποια άλλη διάσταση. Γι'αυτό τον σκοπό έχει δημιουργηθεί μια συμπληρωματική δενδρική δομή, με το όνομα free, στην οποία έχουμε μια ρίζα με το όνομα free, και κάτω από αυτήν κρέμονται όλοι οι 'ορφανοί' κόμβοι. Έτσι ο χρήστης μπορεί ανά πάσα στιγμή να κοιτάξει στην δομή αυτή και να βρει όλους τους κόμβους που δεν έχουν ανατεθεί σε κάποια διάσταση. Άρα, όταν διαγράφεται ένας κόμβος από μια διάσταση, μεταφέρεται στην συμπληρωματική αυτή δομή.
- **Διαγραφή μια υπάρχουσας Διάστασης:** Όταν ο χρήστης επιλέξει αυτή την πράξη, τότε όλοι οι κόμβοι που ανήκουν στην επιλεγμένη διάσταση μεταφέρονται στην δομή free, και η επιλεγμένη διάσταση διαγράφεται από την δομή των διαστάσεων.
- **Αλλαγή ονομασίας σε μια Διάσταση:** Αυτή η πράξη είναι αρκετά απλή. Δομικά δεν αλλάζει τίποτα, απλά αλλάζει η πληροφορία που φέρει το όνομα της διάστασης που επιλέχθηκε να αλλάξει όνομα.

Σημειώνουμε και πάλι, ότι οι πράξεις που μόλις αναφέραμε εκτελούνται και στις τοπικές δομές των διαστάσεων, εφ'όσον βέβαια υπάρχουν οι σχετικοί κόμβοι ή διαστάσεις.

5.2.2 Διαχείριση του Γράφου Εξαρτήσεων

Το συγκεκριμένο υπο-υποσύστημα είναι αυτό που χειρίζεται και εκτελεί όλες τις λειτουργίες σχετικά με τον γράφο εξαρτήσεων. Εδώ θα εξηγήσουμε τον τρόπο με τον οποίο εκτελούνται αυτές, καθώς και τις λεπτομέρειες αναπαράστασης και των πράξεων που ορίσαμε για αυτόν τον γράφο. Όπως και στην προηγούμενη περίπτωση, μπορούμε να χωρίσουμε σε περαιτέρω υπο-υπο-υποσυστήματα.

5.2.2.1 Αναπαράσταση του Γράφου Εξαρτήσεων σε Αρχείο

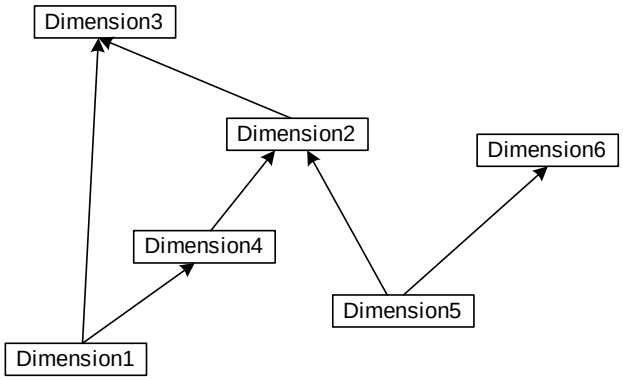
Το υπο-υπο-υποσύστημα αυτό είναι υπεύθυνο για την αναπαράσταση του γράφου εξαρτήσεων σε αρχείο στον σκληρό δίσκο. Όπως συμβαίνει με τις ιεραρχίες και τις διαστάσεις, έτσι και ο γράφος εξαρτήσεων πρέπει να φυλάσσεται στον σκληρό δίσκο, ώστε να μπορεί να αρχικοποιείται η δομή αυτή κατά την εκκίνηση του συστήματος, και για να φυλάσσουμε τις αλλαγές που κάνουμε στον γράφο. Προτιμήθηκε η μορφή απλών αρχείων TXT καθώς δεν χρειάζεται κάτι παραπάνω από μια απλή δήλωση των εξαρτήσεων.

Το αρχείο αυτό έχει το όνομα dependency.txt, και τα δεδομένα που περιέχει είναι κάπως έτσι:

```
X (depending from) y z w

Dimension1 Dimension3 Dimension4
Dimension2 Dimension3
Dimension4 Dimension2
Dimension5 Dimension6 Dimension2
```

Όπως βλέπουμε, το αρχείο αυτό έχει 2 γραμμές στην αρχή που δεν χρησιμοποιούνται για παροχή πληροφορίας. Βασικά στην πρώτη γραμμή περιγράφεται ο τρόπος αναπαράστασης των εξαρτήσεων. Κάθε γραμμή περιγράφει την εξάρτηση μιας διάστασης με το όνομα της πρώτης λέξης της γραμμής, από τα ονόματα των διαστάσεων που ακολουθούν. Άρα, η πρώτη γραμμή μας λέει ότι η διάσταση Dimension1 εξαρτάται από τις διαστάσεις Dimension3 και Dimension4, δηλαδή για να μπορέσει ο χρήστης να επιλέξει κόμβο αυτής της διάστασης, πρέπει πρώτα να έχει γίνει επιλογή των κόμβων των διαστάσεων με τα ονόματα Dimension3 και Dimension4. Είναι σαφές πως ορίζουμε απλή και πολλαπλή εξάρτηση μιας διάστασης. Στις γραμμές 1 και 4 έχουμε πολλαπλή εξάρτηση, ενώ στις γραμμές 2 και 3 έχουμε απλή.
Ο γράφος εξαρτήσεων που περιγράφεται στο αρχείο που δείξαμε θα έχει την εξής μορφή:



Εικόνα 5.3 Γράφος Εξαρτήσεων

Εδώ αναπαριστάνουμε την εξάρτηση με ένα βέλος που ξεκινάει από την εξαρτώμενη διάσταση και καταλήγει στην διάσταση από την οποία εξαρτάται. Όπως αναφέραμε και στην ενότητα 3.3, μετά των ορισμό των εξαρτήσεων, δεν μπορούμε να έχουμε κύκλους στον γράφο αυτόν.

5.2.2.2 Αναπαράσταση του Γράφου Εξαρτήσεων στην Μνήμη

Αυτό το υπο-υπο-υποσύστημα χειρίζεται την αναπαράσταση του γράφου εξαρτήσεων σαν δομή στην μνήμη. Ένα θέμα είναι με ποιον τρόπο μπορούμε να αναπαραστήσουμε αυτόν τον γράφο. Προτιμήθηκε να χρησιμοποιηθεί ένας πίνακας Hash, κυρίως γιατί είναι εύκολος ο έλεγχος εξάρτησης μιας διάστασης.

Πιο συγκεκριμένα, ο πίνακας αυτός έχει για κλειδιά τα ονόματα των διαστάσεων που έχουν εξάρτηση, και αυτά τα κλειδιά δείχνουν σε ένα Vector (διάνυσμα) που περιέχει όλες τις διαστάσεις από τις οποίες εξαρτώνται. Έτσι, όταν ζητάμε αν υπάρχει εξάρτηση για την διάσταση Dimension1, καλούμε την σχετική συνάρτηση του HashTable, και αν επιστρέψει null τότε δεν υπάρχει εξάρτηση για αυτή την διάσταση, αλλιώς επιστρέφει το διάνυσμα με τις διαστάσεις από τις οποίες εξαρτάται η Dimension1. Σαν αναπαράσταση αυτού του HashTable, και για το παράδειγμα που δείξαμε πιο πάνω, θα είχαμε τον εξής πίνακα στην μνήμη:

Λέξεις Κλειδιά	**Διάνυσμα Εξαρτήσεων**
Dimension1 →	[Dimension3, Dimension4]
Dimension2 →	[Dimension3]
Dimension4 →	[Dimension2]
Dimension5 →	[Dimension6, Dimension2]

Πίνακας 5.1

Η μεταφορά των δεδομένων από το αρχείο στην δομή HashTable γίνεται πολύ εύκολα, με μια συνάρτηση που διαβάζει το αρχείο γραμμή-γραμμή (αγνοώντας τις δυο πρώτες) και φτιάχνει το κλειδί από την πρώτη λέξη, και με τα υπόλοιπα κατασκευάζει ένα Vector, στο οποίο δείχνει το εκάστοτε κλειδί.

5.2.2.3 Μεταφορά του Γράφου Εξαρτήσεων σε Αρχείο

Το υπο-υπο-υποσύστημα αυτό είναι υπεύθυνο για την μεταφορά του γράφου εξαρτήσεων στο αρχείο στο οποίο περιγράφεται. Αυτό γίνεται για να αποθηκεύουμε μόνιμα τις αλλαγές που κάνουμε στον γράφο, έτσι ώστε να μπορεί να βρίσκει το σύστημα έγκυρα δεδομένα κατά την εκκίνηση του. Και πάλι, δεν πρόκειται για κάποια σύνθετη εργασία. Μια συνάρτηση φροντίζει να γράψει στο αρχείο τα κλειδιά και τις λέξεις που υπάρχουν στο αντίστοιχο Vector, στην μορφή που περιγράψαμε πιο πάνω.

5.2.2.4 Έλεγχος των Εξαρτήσεων

Αυτό το υπο-υπο-υποσύστημα εκτελεί τους ελέγχους εξαρτήσεων στον γράφο. Δίνοντας το όνομα μιας διάστασης ελέγχεται στον HashTable αν υπάρχει η σχετική εγγραφή ως κλειδί, και ανάλογα επιστρέφεται το σύνολο των διαστάσεων από τις οποίες εξαρτάται άμεσα. Πιο πολύπλοκο είναι να βρεθούν όλες οι διαστάσεις, από τις οποίες εξαρτάται άμεσα και έμμεσα μια διάσταση Α, ή ακόμη να βρεθούν οι εξαρτώμενες διαστάσεις μιας διάστασης Α. Αυτές οι τελευταίες ενέργειες είναι λίγο πιο πολύπλοκες, καθώς έχουμε επαναληπτικούς ελέγχους στον πίνακα HashTable. Προφανώς, υπάρχουν κατάλληλες συναρτήσεις που επιτελούν αυτές τις ενέργειες.

5.2.2.5 Τροποποίηση του Γράφου Εξαρτήσεων

Το υπο-υπο-υποσύστημα αυτό εκτελεί τις λειτουργίες διαμόρφωσης του γράφου εξαρτήσεων. Αυτές, όπως αναφέραμε και στο προηγούμενο κεφάλαιο, έχουν να κάνουν με την διαγραφή της εξάρτησης μιας διάστασης Α από μια διάσταση Β, την προσθήκη εξάρτησης μιας διάστασης Α από μια διάσταση Β, όταν η Α ήδη εξαρτάται από κάποια άλλη διάσταση Γ, και την προσθήκη εξάρτησης μιας διάστασης Α από μια διάσταση Β, όπου η Α δεν υπήρχε προηγουμένως στον γράφο. Πιο αναλυτικά εξηγούμε παρακάτω:

- **Διαγραφή της Εξάρτησης μιας Διάστασης Α από μια Διάσταση Β:** Σε αυτή την περίπτωση, η διαδικασία που εκτελείται απλά διαγράφει από το Vector στο οποίο έχουμε τις εξαρτήσεις της διάστασης Α το όνομα της διάστασης Β. Εάν το Vector μείνει κενό, τότε αυτό σημαίνει ότι η Α είναι ανεξάρτητη. Εάν δεν προστεθεί εξάρτηση στην Α, τότε αυτή διαγράφεται από τον γράφο, και προφανώς και από τον Hashtable.
- **Προσθήκη Εξάρτησης μιας Διάστασης Α από μια Διάσταση Β, όταν η Α ήδη εξαρτάται από μια Διάσταση Γ:** Εδώ απλά προστίθεται στο Vector που περιέχει τις διαστάσεις από τις οποίες εξαρτάται η Α, το όνομα της Γ. Πρέπει να σημειώσουμε όμως ότι πρώτα ελέγχεται αν με αυτή την κίνηση δημιουργείται κύκλος στον γράφο εξαρτήσεων.
- **Προσθήκη Εξάρτησης μιας Διάστασης Α από μια Διάσταση Β, όταν η Α είναι ανεξάρτητη:** Όταν εκτελείται αυτή η λειτουργία, γίνεται ένας έλεγχος αν δημιουργείται κύκλος στον γράφο, και αν όχι, τότε απλά προστίθεται στον HashTable μία ακόμη γραμμή, με κλειδί την διάσταση Α, που δείχνει σε έναν Vector με μοναδικό αντικείμενο την διάσταση Β.

Πρέπει να σημειώσουμε ότι εάν σε κάποια περίπτωση βρεθεί ότι δημιουργείται κύκλος στον γράφο, η ενέργεια λαμβάνεται ως αντικανονική, δεν εκτελείται, και ένα κατάλληλο μήνυμα ενημερώνει τον χρήστη για την λανθασμένη του ενέργεια.

5.2.3 Διαχείριση του Καθολικού Κανονικοποιημένου Δένδρου

Αυτό το υπο-υποσύστημα αναλαμβάνει τις λειτουργίες που σχετίζονται με το καθολικό κανονικοποιημένο δένδρο. Αυτές τις περιγράψαμε στην υποενότητα 4.2.2.3, και περισσότερες λεπτομέρειες γι'αυτές ακολουθούν παρακάτω.

5.2.3.1 Αναπαράσταση της Δομής στην Μνήμη

Η αναπαράσταση του καθολικού κανονικοποιημένου δένδρου στην μνήμη είναι ένα σημαντικό στοιχείο, γιατί πρέπει να είναι αρκετά ευέλικτη για να μπορούμε να την χειριστούμε εύκολα, και επίσης να περιέχει όλες τις πληροφορίες που χρειαζόμαστε. Ουσιαστικά χρησιμοποιήσαμε την δενδρική δομή που έχουμε χρησιμοποιήσει και πιο πάνω, μόνο που εδώ την υλοποιούμε σε μια εκφυλισμένη εκδοχή, όπου δεν έχουμε δένδρο, αλλά μονοπάτι.

Για να το εξηγήσουμε αυτό, πρέπει να σημειώσουμε ότι αυτό που έχει σημασία δεν είναι η κατασκευή όλου του καθολικού κανονικοποιημένου δένδρου όπως την ορίζει η θεωρητική μελέτη, αλλά να κρατήσουμε τις επιλογές του χρήστη όσον αφορά το μονοπάτι που επιλέγει ως επερώτηση. Δηλαδή κρατάμε πληροφορία για την επιλογή της διάστασης σε κάθε επίπεδο, και για την κάθε διάσταση κρατάμε τον κόμβο που επιλέχθηκε από αυτήν. Έτσι έχουμε ένα μονοπάτι κόμβων, όπου κάθε κόμβος έχει σαν πληροφορία το όνομα της διάστασης και του κόμβου που επιλέχθηκε.

Φυσικά, επειδή το καθολικό κανονικοποιημένο δένδρο είχε ανάγκη πρόσθετων συναρτήσεων, η πραγματική δομή που υλοποιήθηκε γι'αυτό είναι πιο σύνθετη από αυτή της δενδρικής δομής που χρησιμοποιήθηκε πιο πάνω. Ουσιαστικά, ένα πεδίο της δομής αυτής είναι και η δενδρική δομή, όπου και φυλάμε το μονοπάτι.

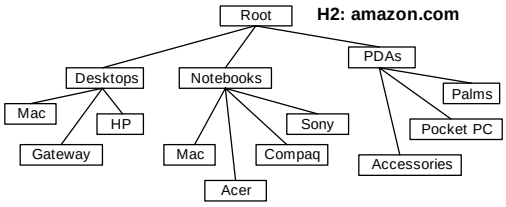
5.2.3.2 Δυναμική Κατασκευή της Δομής

Αυτό το υπο-υπο-υποσύστημα υλοποιεί την κατασκευή του καθολικού κανονικοποιημένου δένδρου, όπως την περιγράψαμε αμέσως πιο πάνω. Αυτό που έχει σημασία είναι η δυναμική κατασκευή, δηλαδή πρέπει να παρέχουμε στον χρήστη την δυνατότητα πολλαπλών επιλογών και ανακάλεσης αυτών, αν κάπου θελήσει να διορθώσει την επιλογή του.

Πιο συγκεκριμένα, η διαδικασία που εκτελείται για την δημιουργία ενός κόμβου στο καθολικό κανονικοποιημένο δένδρο είναι η εξής: Ο χρήστης επιλέγει την διάσταση, σύμφωνα με την οποία θα επεκτείνει τις επιλογές του, δηλαδή θα εμφανιστούν οι κόμβοι που ανήκουν στην διάσταση αυτή, και σχηματίζουν κάποιο μονοπάτι με τις μέχρι τώρα επιλογές του. Αν δεν υπάρχει κανένας τέτοιος κόμβος, τότε εμφανίζεται το τριγωνάκι (Δ) που είδαμε στην θεωρητική μελέτη. Στην πράξη δεν εμφανίζεται τριγωνάκι, αλλά ένας κόμβος που μας ειδοποιεί ότι δεν μπορούμε να επεκτείνουμε τις επιλογές μας σύμφωνα με την επιλεγμένη διάσταση. Έτσι λοιπόν κατασκευάζεται στο καθολικό κανονικοποιημένο δένδρο ένας κόμβος που έχει στο πεδία διάστασης αυτή που διάλεξε ο χρήστης. Τώρα, εάν δεν εμφανίστηκε τριγωνάκι, τότε ο χρήστης πρέπει να επιλέξει έναν κόμβο. Όταν επιλέξει, τότε το όνομα του κόμβου γράφεται στο συγκεκριμένο επίπεδο στο καθολικό κανονικοποιημένο δένδρο. Έτσι προχωράει η κατασκευή της δομής.

Για παράδειγμα, δείχνουμε τα βήματα που θα ακολουθούσε ο χρήστης για να κατασκευάσει ένα κανονικοποιημένο δένδρο σύμφωνα με την εικόνα 3.2(a).

<u>Παράδειγμα:</u>

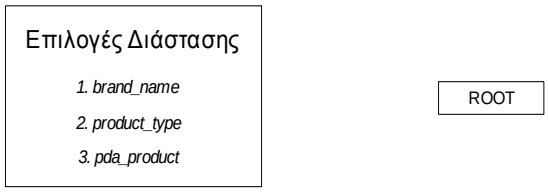


<u>Διάσταση 1:</u>
(*product_type*, {*Desktops*, *Notebooks*, *PDAs*}).

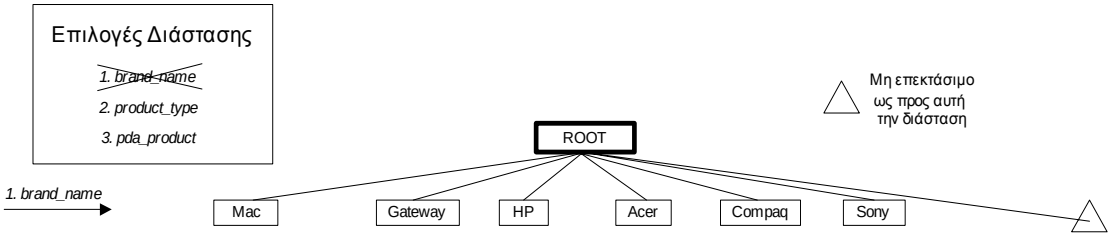
<u>Διάσταση 2:</u>
(*brand_name*, {*Mac*, *Gateway*, *HP*, *Acer*, *Compaq*, *Sony*}).

<u>Διάσταση 3:</u>
(*pda_product*, {*Accessories*, *PocketPC*, *Palm*}).

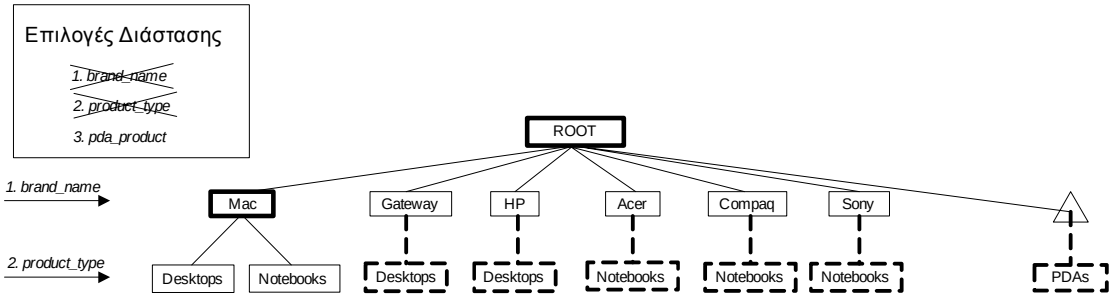
Εικόνα 5.4
Αρχική Ιεραρχία και Διαστάσεις



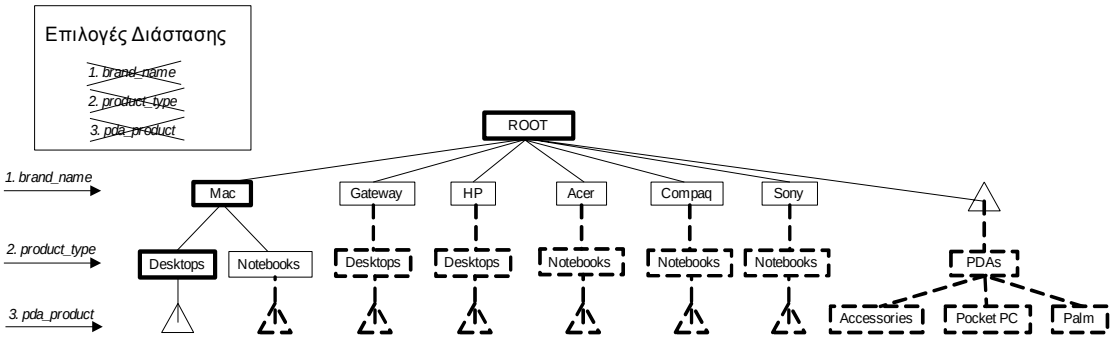
Εικόνα 5.5
Αρχική Εικόνα



Εικόνα 5.6
Βήμα 1ο: 1η Επιλογή διάστασης: *brand_name*



Εικόνα 5.7
Βήμα 2ο και 3ο:
Επιλογή κόμβου: *Mac* (με bold) και 2η Επιλογή διάστασης: *product_type*
(με διακεκομμένη γραμμή είναι οι κόμβοι που κατασκευάζονται αόρατα από τον χρήστη και δεν κρατούνται στην δομή).



Εικόνα 5.8
Βήμα 4ο και 5ο:
Επιλογή κόμβου: *Desktop* (με bold) και 3η Επιλογή διάστασης: *pda_product*
(με διακεκομμένη γραμμή είναι οι κόμβοι που κατασκευάζονται αόρατα από τον χρήστη. Η νέα διάσταση *pda_product* δεν προσφέρει τίποτα στο μονοπάτι που έχουμε επιλέξει)

Η δομή του καθολικού κανονικοποιημένου δένδρου κρατάει μόνο τις επιλογές του χρήστη και την διάσταση στην οποία ανήκουν, δηλαδή τους **έντονους** κόμβους μαζί με το όνομα της διάστασής τους. Επίσης κρατάει και τα τυχόν τριγωνάκια που θα βρεθούν στο μονοπάτι αυτό.

Ανά πάσα στιγμή ο χρήστης μπορεί να πάρει πίσω την τελευταία του επιλογή, δηλ. στην επιλογή κόμβου ή στην επιλογή διάστασης. Όταν πρόκειται για το δεύτερο, τότε στο καθολικό κανονικοποιημένο δένδρο διαγράφεται το τελευταίο επίπεδο, γιατί είχε κατασκευαστεί κατά την επιλογή της διάστασης, ενώ όταν πρόκειται για το πρώτο, τότε απλά σβήνεται το όνομα του επιλεγμένου κόμβου στο τελευταίο επίπεδο. Πρόκειται για πολύ απλές λειτουργίες.

5.2.3.3 Κατασκευή Ανάλογων Κανονικοποιημένων Δένδρων

Αυτό το υπο-υπο-υποσύστημα είναι υπεύθυνο για την κατασκευή των ομότιμων κανονικοποιημένων δένδρων, τα οποία αποτιμούνται στις αρχικές ιεραρχίες. Ουσιαστικά, η ιδέα είναι η αναδιάταξη των επιπέδων του μονοπατιού που κατασκεύασε ο χρήστης. Αυτό που αλλάζει είναι μόνο η σειρά των διαστάσεων, και όχι οι κόμβοι που επιλέχθηκαν για την κάθε διάσταση. Ουσιαστικά, γίνεται μια εξαντλητική αναδιάταξη των διαστάσεων, σύμφωνα πάντα με τον γράφο εξαρτήσεων. Μαζεύουμε κάπου αυτά τα δένδρα, και βλέπουμε ποιο από αυτά ταιριάζει σε κάθε ιεραρχία. Ενδεχομένως να μην ταιριάζει κανένα, αν έχει γίνει επιλογή διάστασης ή κόμβου που δεν υπάρχει σε αυτή την ιεραρχία.

Αυτό που είναι σημαντικό είναι ότι ο χρήστης μπορεί να σταματήσει την επιλογή των επιπέδων πριν φτάσει στο κατώτερο επίπεδο, δηλαδή πριν εξαντλήσει όλες τις διαστάσεις. Επειδή οι SQL ερωτήσεις για τα δεδομένα βρίσκονται στο επίπεδο των φύλλων, πρέπει εμείς από το μονοπάτι του χρήστη να κατασκευάσουμε όλα τα δυνατά μονοπάτια που θα μπορούσε να επιλέξει. Αυτό σημαίνει ότι από εκεί που σταμάτησε και κάτω, γίνεται μια εξαντλητική συμπλήρωση όλων των διαστάσεων (δηλαδή με όλες τις δυνατές σειρές επιλογής), και για κάθε διάσταση επιλέγονται όλοι οι κόμβοι. Αυτό είναι αρκετά πολύπλοκο στην εφαρμογή του, οπότε χρησιμοποιούμε τον γράφο εξαρτήσεων για να περιορίσουμε τις επιλογές, καθώς και μία μικρή παραλλαγή αυτού που μόλις περιγράψαμε:

Επειδή υπάρχει η περίπτωση σε μια διάσταση να ανήκουν κόμβοι που δεν υπάρχουν σε όλες τις ιεραρχίες, δεν έχει νόημα να γίνεται η επιλογή αυτών όταν κατασκευάζονται όλα τα μονοπάτια. Έτσι, αυτή την διαδικασία την εφαρμόζουμε για κάθε ιεραρχία ξεχωριστά. Άρα, στην κατασκευή των μονοπατιών συμμετέχουν μόνο οι διαστάσεις και οι κόμβοι που ανήκουν σε αυτή την ιεραρχία. Εννοείται ότι οι επιλογές του χρήστη δεν επηρεάζονται από αυτά, καθώς όλα αυτά γίνονται για την συμπλήρωση του μονοπατιού.

Το αποτέλεσμα είναι ότι για κάθε ιεραρχία βρίσκουμε τα μονοπάτια που μπορούν να αποτιμηθούν.

5.2.4 Διαχείριση Ταύτισης Κόμβων

Αν και δεν έχουμε αναφερθεί σε αυτό καθόλου, πρέπει να σημειώσουμε ότι χρησιμοποιούνται δυο ακόμη εσωτερικές δομές, οι οποίες μας βοηθάνε να λύσουμε το πρόβλημα του ταιριάσματος ταυτόσημων κόμβων με διαφορετικό όνομα. Για το λόγο αυτό χρησιμοποιείται ένα αρχείο, που υποθέτουμε ότι είναι πάντα ενημερωμένο. Το αρχείο αυτό έχει όνομα SimNodes.txt και έχει την εξής μορφή:

```
Name1,h1,h2 Name2,h3,h4
Name3,h1,h2,h3 Name4,h4
Name5,h1 Name6,h2 Name7,h3,h4
```

Κάθε γραμμή έχει την εξής σημασία: Οι ενότητες χωρίζονται με κενά. Σε κάθε ενότητα η πρώτη λέξη είναι το όνομα του κόμβου, και χωριζόμενες με κόμματα είναι οι ιεραρχίες στις οποίες χρησιμοποιείται το συγκεκριμένο όνομα. Επίσης, σε κάθε γραμμή, το πρώτο όνομα είναι η γενική έκφραση για τις συνωνυμίες κόμβων, δηλαδή στην δομή των συγχωνευμένων διαστάσεων χρησιμοποιούνται τα ονόματα της πρώτης στήλης. Άρα έχει σημασία ποιό όνομα θα μπει πρώτο, ενώ τα άλλα δεν έχουν κάποιο περιορισμό.
Άρα, η πρώτη γραμμή μας λέει ότι στις ιεραρχίες h1 και h2 χρησιμοποιείται για κάποιον κόμβο το όνομα Name1, ενώ στις h3 και h4 χρησιμοποιείται το όνομα Name2. Στις συγχωνευμένες διαστάσεις χρησιμοποιείται μόνο το Name1.
Έχει μεγάλη σημασία να τονίσουμε ότι σε αυτό το αρχείο πρέπει να δηλώνονται μετά από ένα όνομα όλες οι ιεραρχίες που το χρησιμοποιούν, ακόμη και αν πρόκειται για όλες εκτός από μιας. Πχ, ακόμη και έτσι, θα'πρεπε να υπάρχει μια γραμμή της μορφής αυτής:

```
Name1,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11,h12 Name2,h13
```

Να σημειώσουμε ότι αυτή η πληροφορία φυλάσσεται σε δύο πίνακες HashTable, για διαφορετική λειτουργία ο καθένας. Ο πρώτος χρησιμοποιείται απλά για την αντιστοιχία των ταυτονυμιών, και από το προηγούμενο κείμενο, θα έχει την εξής μορφή:

Λέξεις Κλειδιά	**Αντιστοιχία**
Name2 →	Name1
Name4 →	Name3
Name6 →	Name5
Name7 →	Name5

Πίνακας 5.2

Ο δεύτερος HashTable είναι αρκετά πιο πολύπλοκος, και σκοπό έχει να μας δίνει την δυνατότητα, βάζοντας σαν κλειδί την γενική ονομασία ενός κόμβου, να μας δίνει τις ειδικές ονομασίες και τις ιεραρχίες στις οποίες χρησιμοποιείται η κάθε μια. Οι λέξεις κλειδιά είναι οι γενικές ονομασίες, και δείχνουν σε ένα Vector που αποτελείται από άλλα Vectors. Κάθε ένα από αυτά τα Vectors περιέχει την εκάστοτε ονομασία σαν πρώτο στοιχείο, ακολουθούμενη από τις ιεραρχίες που την χρησιμοποιούν. Αυτό συμβαίνει και για την γενική ονομασία. Για να κάνουμε τα πράγματα πιο κατανοητά, αυτός ο HashTable, για το προηγούμενο παράδειγμα, θα είχε την παρακάτω μορφή:

Λέξεις Κλειδιά	**Αντιστοιχία**
Name1 →	[[Name1, h1, h2] , [Name2, h3, h4]]
Name3 →	[[Name3, h1, h2, h3] , [Name4, h4]]
Name5 →	[[Name5, h1] , [Name6, h2] , [Name7, h3, h4]]

Πίνακας 5.3

Αυτός ο πίνακας χρησιμοποιείται όταν, έχοντας το μονοπάτι του χρήστη, το οποίο αποτελείται από κόμβους με το γενικό τους όνομα, θέλουμε να αλλάξουμε τα γενικά ονόματα σε τοπικά, ξέροντας σε ποια ιεραρχία θα αναφερθούμε κάθε φορά. Απλά ψάχνουμε αν το όνομα του κόμβου υπάρχει σε αυτόν τον πίνακα. Αν όχι, σημαίνει ότι δεν υπάρχουν παραλλαγές, ενώ στην αντίθετη περίπτωση ψάχνουμε σειριακά τα Vectors για να βρούμε σε ποιόν ανήκει το όνομα της ιεραρχίας που εξετάζουμε. Όταν βρεθεί αυτό, τότε ξέρουμε ότι στην πρώτη θέση αυτού του Vector βρίσκεται και το τοπικό όνομα. Φυσικά υπάρχουν οι σχετικές συναρτήσεις που φροντίζουν από το αρχείο SimNodes.txt να φτιάξουν τους πίνακες που αναφέραμε.

5.3 Υποσύστημα Διαχείρισης της Βάσης Δεδομένων

Αυτό το υποσύστημα είναι υπεύθυνο για την επικοινωνία με την Βάση Δεδομένων, δηλαδή για την δημιουργία της σύνδεσης, για την εκτέλεση των ερωτήσεων, για το φόρτωμα των αποτελεσμάτων σε ειδικές δομές, και για το κλείσιμο της σύνδεσης με την Βάση Δεδομένων. Επίσης είναι υπεύθυνο για την κατασκευή του query που όταν εφαρμοστεί στην βάση θα φέρει τα αποτελέσματα που αφορούν την εκάστοτε ιεραρχία. Αποτελείται από 2 μέρη.

Το πρώτο κομμάτι έχει να κάνει με την ένωση των SQL ερωτήσεων που παίρνουμε από τα φύλλα των ιεραρχιών, όταν ο χρήστης δεν έχει κατασκευάσει ένα πλήρες καθολικό

κανονικοποιημένο δένδρο, και οπότε εμείς πρέπει να βρούμε όλες τις SQL ερωτήσεις που βρίσκονται στα φύλλα που μπορούμε να φτάσουμε ακολουθώντας το μερικό μονοπάτι του χρήστη. Για κάθε ιεραρχία μαζεύουμε αυτές τις SQL ερωτήσεις και τις ενώνουμε με ένα UNION, έτσι ώστε η SQL ερώτηση που θα προκύψει να φέρνει όλα τα αποτελέσματα από όλα τα σχετικά φύλλα. Σε μια δομή φυλάμε τις τελικές SQL ερωτήσεις που προκύπτουν και τις εφαρμόζουμε για κάθε ιεραρχία χωριστά.

Το δεύτερο μέρος είναι η εφαρμογή των ερωτήσεων και η συλλογή των αποτελεσμάτων. Ουσιαστικά εδώ χειριζόμαστε κάποια Exceptions, και φροντίζουμε για το σωστό άνοιγμα και κλείσιμο της σύνδεσης. Η εφαρμογή δεν προβλέπει αλλαγή στα δεδομένα της Βάσης, οπότε περιοριζόμαστε απλά στη εφαρμογή των ερωτήσεων, και στην συλλογή των εγγραφών. Αυτές επιστρέφουν σε μια δομή που ονομάζεται ResultSet, και έχει υλοποιηθεί στο πακέτο java.sql. Υλοποιεί έναν πίνακα, όπου οι γραμμές είναι τα records που επέστρεψαν από την Βάση Δεδομένων, και στήλες είναι τα πεδία που ζητήσαμε να επιστραφούν, από το “select” της SQL ερώτησης που εφαρμόσαμε.

5.4 Υποσύστημα Διαχείρισης Γραφικού Περιβάλλοντος

Όπως αναφέραμε και στην υποενότητα 4.2.4 το υποσύστημα διαχείρισης του γραφικού περιβάλλοντος είναι η εικόνα του συστήματος. Είναι από τα πιο σημαντικά κομμάτια, γιατί μέσα από αυτό ο χρήστης αλληλεπιδρά με το σύστημα, και έχει πολύ μεγάλη σημασία να είναι φιλικό, εύχρηστο, και ευχάριστο. Όλα αυτά έχουν ληφθεί υπ’όψιν σε όλα τα μέρη αυτού του υποσυστήματος, οπότε δεν χρειάζεται να τα αναφέρουμε ξεχωριστά, ή σε κάθε περίπτωση. Γενικά, μπορούμε να χωρίσουμε αυτό το κομμάτι στα εξής μέρη:

5.4.1 Είσοδος Παραμέτρων

Αυτό το υπο-υποσύστημα εκτελεί όλες τις λειτουργίες εισόδου των παραμέτρων. Λειτουργίες και μέρη αυτού βλέπουμε σε όλα τα γραφικά παράθυρα, καθώς για τις περισσότερες λειτουργίες υπάρχουν πολλαπλές επιλογές, οπότε καλείται ο χρήστης να κάνει τις επιλογές του, προκειμένου να εκτελεστούν οι λειτουργίες. Γι’αυτό το λόγο σε αυτό το υπο-υποσύστημα εντάσσονται όλα τα παράθυρα.

Πιο συγκεκριμένα, στο πέρασμα των παραμέτρων εντάσσονται όλες οι λίστες επιλογής, τα checkboxes κτλ. Στην έναρξη λειτουργιών εντάσσονται όλα τα κουμπιά, καθώς με αυτά εκκινούμε λειτουργίες, ή οριστικοποιούμε το πέρασμα των παραμέτρων σε συναρτήσεις. Ακριβώς επειδή το σύστημα στηρίζεται στην δυναμική των πολλαπλών επιλογών και τις επιλογές του χρήστη, όλα τα μέρη του γραφικού περιβάλλοντος ανήκουν εδώ.

5.4.2 Παρουσίαση Εσωτερικών Δομών

Ένα σημαντικό κομμάτι του γραφικού περιβάλλοντος είναι η αναπαράσταση των εσωτερικών δομών. Το σημαντικό είναι να είναι ευδιάκριτες και να μοιάζουν με αυτό που περιμένει να δει ο χρήστης, ανεξάρτητα με τον τρόπο που έχει υλοποιηθεί η δομή στην μνήμη. Πιο συγκεκριμένα, μπορούμε να το χωρίσουμε στις περαιτέρω κατηγορίες.

5.4.2.1 Παρουσίαση Δενδρικών Δομών

Αυτό το υπο-υπο-υποσύστημα είναι αρκετά απλό. Χρησιμοποιείται μια συνάρτηση, με την οποία μπορούμε να προβάλλουμε στην οθόνη μόνο τα στοιχεία που θέλουμε. Δηλαδή, αν και έχουμε δομές με αρκετά στοιχεία, στο γράφημα δείχνουμε επιλεκτικά μόνο τα ονόματα των κόμβων. Επίσης χρησιμοποιείται και ένας αλγόριθμος ώστε να έχουμε αρκετά λογική και ευδιάκριτη τοποθέτηση των κόμβων του δένδρου. Το πρόβλημα αυτό προέκυπτε λόγω των διαφορετικών μεγεθών του κάθε κόμβου.
Σε αυτή την κατηγορία, των δενδρικών δομών, ανήκουν οι ιεραρχίες και οι διαστάσεις. Και οι δύο δομές αναπαρίστανται λοιπόν σαν δένδρα.

5.4.2.2 Παρουσίαση του Κανονικοποιημένου Δένδρου

Αυτό το υπο-υπο-υποσύστημα χειρίζεται την αναπαράσταση του καθολικού κανονικοποιημένου δένδρου. Ουσιαστικά, όπως έχουμε πει, πρόκειται για μια δενδρική δομή εκφυλισμένη σε μονοπάτι. Η δομή που αναπαριστά το καθολικό κανονικοποιημένο δένδρο είναι αρκετά σύνθετη, οπότε και πάλι γίνεται επιλεκτική παρουσίαση της πληροφορίας.
Το σημαντικότερο κομμάτι είναι ότι εδώ ο χρήστης αλληλεπιδρά με το γράφημα, καθώς επιλέγει κόμβους και διαστάσεις. Επίσης μπορεί ανά πάσα στιγμή να αναιρεί μια-μια τις κινήσεις. Καταρχήν, σε κάθε επίπεδο παρουσιάζεται και το όνομα τις διάστασης που επιλέχθηκε, και δίπλα από αυτή όλοι οι σχετικοί κόμβοι. Ο χρήστης μπορεί να επιλέγει έναν κόμβο κάθε φορά, και αυτός που τελικά επιλέγεται έρχεται στην πρώτη θέση. Έτσι σιγά-σιγά κατασκευάζεται το μονοπάτι που θέλει ο χρήστης. Αυτή η αναπαράσταση δεν είναι δένδρο, αλλά ένα μονοπάτι, όπου δίπλα στους επιλεγμένους κόμβους βλέπουμε και αυτούς που δεν επιλέχθηκαν.

5.4.2.3 Παρουσίαση του Γράφου Εξαρτήσεων

Αυτό το υπο-υπο-υποσύστημα καλείται να παρουσιάσει έναν γράφο, τη στιγμή που στην μνήμη ο γράφος είναι αποθηκευμένος ως HashTable. Σε αυτό το μέρος του συστήματος λοιπόν ανήκουν οι λειτουργίες που παίρνουν τον HashTable και κατασκευάζουν τους σχετικούς κόμβους και τις ακμές. Επίσης εδώ ανήκει και ο αλγόριθμος τοποθέτησης των κόμβων. Ουσιαστικά κρύβουμε από τον χρήστη τις εσωτερικές δομές που χρησιμοποιούνται.

5.4.2.4 Παρουσίαση Αποτελεσμάτων

Αυτό το υπο-υπο-υποσύστημα είναι υπεύθυνο για την παρουσίαση των αποτελεσμάτων που προκύπτουν από την εφαρμογή των ερωτήσεων στην Βάση Δεδομένων. Εδώ ανήκουν οι συναρτήσεις που παίρνουν ένα ResultSet (που αναφέραμε στην ενότητα 5.3) και φτιάχνουν έναν πίνακα με μορφή ανάλογη με αυτή της Βάσης Δεδομένων, ο οποίος παρουσιάζεται στην συνέχεια στον χρήστη. Σημασία έχει τα αποτελέσματα να είναι ευδιάκριτα και να συγκρίνονται εύκολα μεταξύ τους.

5.4.3 Έλεγχος Λαθών και Εμφάνισης Μηνυμάτων

Τέλος, πρέπει να αναφερθούμε στο κομμάτι του γραφικού περιβάλλοντος που το καθιστά ασφαλές. Σε αυτό το μέρος του συστήματος ανήκουν οι έλεγχοι που γίνονται εσωτερικά για τις επιλογές του χρήστη, οι οποίοι μπορεί να παρουσιάσουν μηνύματα λάθους. Αυτά βοηθούν τον χρήστη να καταλάβει ποιό σφάλμα έγινε, και που οφείλεται. Σημαντικά επίσης είναι και τα μηνύματα πληροφοριών που εμφανίζονται σε κάποιες περιπτώσεις, για να ενημερώσουν τον χρήστη για κάποια βήματα που πρέπει να κάνει, ή για κάποια βήματα που πρέπει να αποφεύγει. Γενικά, αυτά τα μηνύματα βοηθούν στην παρακολούθηση της σωστής λειτουργίας του συστήματος.

6

Υλοποίηση

Στο κεφάλαιο αυτό αναφέρουμε τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη του συστήματος και τις ενέργειες που απαιτούνται να γίνουν ώστε να λειτουργεί σε οποιαδήποτε πλατφόρμα. Επίσης περιγράφουμε όλες τις λεπτομέρειες υλοποίησης του συστήματος, δηλαδή τους βασικούς αλγόριθμούς (με ψευδοκώδικα) που χρησιμοποιήθηκαν σε διάφορα μέρη του συστήματος, καθώς και τις κλάσεις, τις οποίες κατασκευάσαμε και υλοποιούν το σύστημα.

6.1 Πλατφόρμες και Προγραμματιστικά Εργαλεία

Σε αυτή την ενότητα παρουσιάζουμε τα προγραμματιστικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής, καθώς και τις πλατφόρμες στις οποίες αυτή εκτελείται. Επίσης, αναφέρουμε με ποιόν τρόπο μπορεί κάποιος να εγκαταστήσει την εφαρμογή στο σύστημά του.

6.1.1 Γενικά

Η εφαρμογή που υλοποιήθηκε μπορεί να εκτελεστεί τόσο σε περιβάλλον Windows, όσο και σε Linux, και κατά την ανάπτυξη της εργασίας χρησιμοποιήθηκαν και τα δύο. Η Βάση Δεδομένων που χρησιμοποιείται είναι η Postgres 7.3, και δεν είναι απαραίτητο να βρίσκεται στο ίδιο σύστημα με αυτό στο οποίο εκτελείται η εφαρμογή. Έτσι, προτιμήθηκε η εφαρμογή μας να τρέξει σε περιβάλλον WindowsXP ή Windows2000, και η Βάση Δεδομένων να

βρίσκεται σε έναν server SuSe Linux, τον οποίο καθορίζουμε από την IP διεύθυνσή του. Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java2, και πιο συγκεκριμένα το j2sdk1.4.1_02 (να σημειώσουμε ότι παρατηρήθηκαν κάποιες ασυμβατότητες με την έκδοση j2sdk.1.4.1_01 στην υλοποίηση των γραφικών). Οι editors που χρησιμοποιήθηκαν είναι ο Eclipse, για το περιβάλλον των Windows, και το Sun ONE Studio, για το περιβάλλον του Linux. Για την επικοινωνία των εφαρμογών με την Βάση Δεδομένων χρησιμοποιήθηκε η γέφυρα JDBC, με κατάλληλο driver για την επικοινωνία με την Postgres. Να σημειώσουμε επίσης ότι για την διαχείριση των XML αρχείων χρησιμοποιήσαμε το πακέτο sax και τις συναρτήσεις που μας προσφέρει. Τέλος, για την εφαρμογή των ερωτήσεων στην Βάση Δεδομένων χρησιμοποιήθηκε η γλώσσα ερωτήσεων SQL.

6.1.2 Εγκατάσταση Συστήματος

Για την πλήρη εγκατάσταση της εφαρμογής χρειάζεται να γίνουν κάποια βήματα. Καταρχήν, είναι απαραίτητο να έχει εγκατασταθεί η Java2, κατά προτίμηση η έκδοση που αναφέρθηκε πιο πάνω ή μεγαλύτερη. Επίσης, πρέπει να υπάρχει κάπου εγκατεστημένη και η Postgres, αν όχι στο ίδιο σύστημα, τότε κάπου που να μπορεί να προσδιορισθεί με ένα IP address. Για την σωστή επικοινωνία της εφαρμογής με την εκάστοτε Βάση Δεδομένων, χρειάζεται η αλλαγή του IP address, του username και του password σε μια συγκεκριμένη κλάση που έχει υλοποιηθεί για αυτόν τον σκοπό, την οποία θα αναφέρουμε παρακάτω, και φυσικά απαραίτητο είναι να μεταφραστεί εκ νέου η κλάση αυτή.

Το επόμενο βήμα είναι η τοποθέτηση των απαραίτητων βιβλιοθηκών κλάσεων που χρησιμοποιούνται (των αρχείων .jar) στα σημεία που πρέπει, ώστε να μπορούν να χρησιμοποιηθούν από το Virtual Machine της Java. Τα αρχεία αυτά είναι τα

- xerces.jar
- pg73jdbc3.jar

και τοποθετούνται στον εξής φάκελο: C:\Program Files\Java\j2re1.4.1_02\lib\ext. Το πρώτο σύνολο κλάσεων είναι για την διαχείριση των XML αρχείων, και το δεύτερο για την υλοποίηση του driver για την επικοινωνία μέσω JDBC με την Postgres 7.3.

Πρέπει να σημειώσουμε ότι για την γραφική αναπαράσταση των δενδρικών δομών και των γράφων χρησιμοποιήσαμε δύο πακέτα κλάσεων με το όνομα JGraph και JGraphT. Αυτά τα πακέτα μας δίνουν την δυνατότητα να παρουσιάσουμε γράφους, δηλαδή κόμβους και ακμές, με ελευθερία στην τοποθέτησή τους. Κάναμε κάποιες μετατροπές σε μερικές κλάσεις, τις οποίες θα αναφέρουμε μετά από την επεξήγηση των κλάσεων που εμείς κατασκευάσαμε. Αυτές οι κλάσεις ανήκουν σε ένα πακέτο με το όνομα “org”.

Οι εφαρμογή που κατασκευάσαμε βρίσκεται σε έναν φάκελο με το όνομα “diploma”. Μέσα, σε αυτόν τον φάκελο έχουμε άλλον έναν φάκελο με το όνομα “classes”, ο οποίος περιέχει τα αρχεία του κώδικα που γράψαμε, και τα μεταγλωττισμένα “.class” αρχεία. Άρα, οι κλάσεις που φτιάξαμε ανήκουν στο πακέτο “diploma.classes”. Δίπλα στον φάκελο “classes” έχουμε άλλους δυο φακέλους, έναν με το όνομα “dimensions” και έναν με το όνομα “hierarchies”. Στον πρώτο από αυτούς τους δυο φακέλους έχουμε όλα τα αρχεία “dimHi.xml”, το καθένα από τα οποία αντιστοιχεί στην ιεραρχία που δηλώνεται στο αρχείο με το όνομα “Hi.xml”, τα οποία βρίσκονται στον δεύτερο φάκελο που αναφέραμε. Σημειώνουμε ότι δεν είναι απαραίτητο να έχουμε συνεχόμενη αρίθμηση για τις ιεραρχίες. Στον πρώτο φάκελο έχουμε επιπλέον δύο αρχεία. Το ένα έχει όνομα “dimensions.xml” και περιέχει, σε μορφή ανάλογη με τα αρχεία των διαστάσεων, τις συγχωνευμένες διαστάσεις. Το άλλο αρχείο έχει όνομα “free.xml” και εδώ, σε απλή δενδρική δομή δύο επιπέδων, μπαίνουν οι κόμβοι που είναι ‘ορφανοί’, δηλαδή δεν ανήκουν σε καμιά διάσταση.

Δίπλα στους τρεις φακέλους που περιγράψαμε υπάρχουν τα δυο αρχεία “.txt” που περιγράψαμε σε προηγούμενες ενότητες, στο προηγούμενο κεφάλαιο (υποενότητες 5.2.2.1 και 5.2.4), δηλαδή τα “dependency.txt” και “SimNodes.txt”.

Για να τρέξει η εφαρμογή σωστά, δίπλα στον φάκελο με το όνομα “diploma” πρέπει να τοποθετηθεί ο φάκελος με το όνομα “org”, ο οποίος περιέχει τις κλάσεις για το JGraph JgraphT, όπως περιγράψαμε παραπάνω.

Τέλος, πρέπει να αναφέρουμε και τον τρόπο με τον οποίο χρησιμοποιούμε την Βάση Δεδομένων. Ουσιαστικά έχουμε κατασκευάσει μια Βάση Δεδομένων με το όνομα “MoviesDB” (εάν αλλάξει το όνομα αυτό, πρέπει να το αλλάξουμε και στην κλάση, όπου υλοποιείται η σύνδεση με την Βάση Δεδομένων, και να ξανα-μεταγλωττίσουμε την κλάση). Μέσα σε αυτήν υποθέτουμε ότι κάθε ιεραρχία έχει αποθηκευμένα τα δεδομένα της σε έναν πίνακα η κάθε μια, οι οποίοι πίνακες φέρουν το όνομα “MOVIESi”, όπου i ο αριθμός της εκάστοτε ιεραρχίας. Φυσικά αυτό είναι μια σύμβαση που εμείς κάναμε, γιατί κανονικά δεν πρέπει να υπάρχει δέσμευση για το όνομα των πινάκων, καθώς όπως έχουμε πει και στην υποενότητα 5.1.1, τα queries που επιστρέφουν τα δεδομένα βρίσκονται στα φύλλα των ιεραρχιών. Εκεί αναφέρεται το όνομα του πίνακα στον οποίο γίνεται το query. Άρα, αν κάποια ιεραρχία αναφέρεται σε πίνακα με διαφορετικό όνομα από το “MOVIESi”, αυτό το όνομα πρέπει να χρησιμοποιείται και στα queries. Όμως, για λόγους ευκολίας έχουμε υποθέσει ότι όλα τα ονόματα είναι της μορφής “MOVIESi”.

Με το ίδιο σκεπτικό, θα μπορούσαμε να πούμε ότι δεν είναι ανάγκη να περιορίσουμε τα δεδομένα της κάθε ιεραρχίας σε έναν μόνο πίνακα, αλλά να είχαμε κατάλληλα queries, που να αναφέρονται σε διαφορετικούς πίνακες. Αυτό όμως δεν ισχύει, και θα δούμε τον λόγο παρακάτω, στην υποενότητα 6.2.2.

Μετά από αυτά τα βήματα, η εφαρμογή είναι έτοιμη για εκτέλεση.

6.2 *Λεπτομέρειες Υλοποίησης*

Σε αυτή την ενότητα θα περιγράψουμε με αρκετές λεπτομέρειες τον τρόπο με τον οποίο υλοποιήθηκαν οι περισσότερες ενέργειες και λειτουργίες που εκτελεί το σύστημα, κυρίως όσον αφορά την αρχικοποίηση του συστήματος και την επικοινωνία με την Βάση Δεδομένων. Αναλυτική εξέταση και περιγραφή γίνεται και για τους βασικότερους αλγόριθμους, κυρίως με ψευδοκώδικα. Τέλος, ακολουθούν η περιγραφή των κλάσεων που κατασκευάσαμε και μια μικρή αναφορά στις αλλαγές που κάναμε στον κώδικα των πακέτων JGraph και JGraphT.

6.2.1 Αρχικοποίηση Συστήματος

Η αρχικοποίηση του συστήματος επιτελεί κυρίως την ανάκτηση των δομών που χρησιμοποιούμε, δηλαδή την μεταφορά από τον σκληρό δίσκο και τα αρχεία στην μνήμη των δομών των ιεραρχιών, των διαστάσεων, του γράφου εξαρτήσεων και της βοηθητικής δομής free (που κρατάει τους ‘ορφανούς’ κόμβους).

Πρώτα απ’όλα γίνεται ένας έλεγχος για την Βάση Δεδομένων, αν δηλαδή είναι δυνατόν να επιτευχθεί η σύνδεση με αυτήν. Σε περίπτωση που δεν καταστεί δυνατή η σύνδεση, η εφαρμογή τερματίζεται, δίνοντας και ένα κατατοπιστικό μήνυμα.

Το επόμενο βήμα είναι η αρχικοποίηση της δομής του HashTable που κρατάει τους ταυτόσημους κόμβους, με την μορφή που περιγράφτηκε στο Κεφάλαιο 5, ο έλεγχος για την ύπαρξη των απαραίτητων αρχείων (free.xml), και η αρχικοποίηση των δομών των ιεραρχιών, ελέγχοντας τα αρχεία του φακέλου με το όνομα hierarchies που περιγράψαμε πιο πάνω. Το επόμενο βήμα είναι η αντιστοίχιση των αρχείων των διαστάσεων με τα αρχεία των ιεραρχιών. Εάν κάποια ιεραρχία δεν συνοδεύεται από το αρχείο των διαστάσεών της, τότε η εφαρμογή τερματίζει, με το κατάλληλο μήνυμα λάθους, καθώς είναι υποχρεωτικό να υπάρχουν αυτά τα αρχεία.

Η επόμενη λειτουργία είναι η αρχικοποίηση της δομής του γράφου εξαρτήσεων, δηλαδή του HashTable που αναφέραμε στο Κεφάλαιο 5. Κατά την αρχικοποίηση αυτή πραγματοποιείται και η συγχώνευση των δομών των διαστάσεων με τον τρόπο που περιγράψαμε στο προηγούμενο κεφάλαιο, καθώς και η μεταφορά αυτής της δομής των συγχωνευμένων διαστάσεων στο αρχείο “dimensions.xml”, που βρίσκεται στον φάκελο dimensions. Παράλληλα φορτώνεται και μια δομή παρόμοια με αυτή που ορίσαμε για την ταύτιση των κόμβων, καθώς χρειάζεται και εδώ. Δεν χρησιμοποιείται η προηγούμενη δομή που αναφέραμε, γιατί εκείνη αποτελεί πεδίο μιας συγκεκριμένης κλάσης, και δεν είναι γενικώς προσβάσιμη.

Τέλος, ορίζονται και αρχικοποιούνται τα αντικείμενα που χειρίζονται το γραφικό περιβάλλον

6.2.2 Ερωτήσεις στην Βάση Δεδομένων

Όπως αναφέραμε και σε προηγούμενες ενότητες, η επικοινωνία με την Βάση Δεδομένων γίνεται με την γέφυρα JDBC και με κατάλληλα υλοποιημένο driver για την Postgres.Οι ερωτήσεις στην Βάση Δεδομένων γίνονται με SQL queries, τα οποία βρίσκονται στα φύλλα των ιεραρχιών.

Ο χρήστης μπορεί όταν κατασκευάζει το query του μέσα από το καθολικό κανονικοποιημένο δένδρο να μην κατασκευάσει όλα τα επίπεδα. Αυτό, αν μιλάγαμε για μία μόνο ιεραρχία, θα ήταν σαν να ακολουθούσαμε ένα μονοπάτι, και να σταματάγαμε προτού φτάσουμε στα φύλλα, όπου βρίσκονται τα queries. Αν γινόταν αυτό, τότε θα περιμέναμε να πάρουμε τις εγγραφές που αντιστοιχούν σε όλα τα φύλλα που είναι απόγονοι του κόμβου, στον οποίο σταματήσαμε. Το ίδιο συμβαίνει και στην περίπτωση των πολλών ιεραρχιών, όπου το καθολικό κανονικοποιημένο δένδρο έχει κατασκευαστεί μερικώς από τον χρήστη.

Σε αυτή την περίπτωση, προσπαθούμε να μαζέψουμε τα queries που βρίσκονται στα φύλλα των μονοπατιών που αντιστοιχούν στις επιλογές του χρήστη. Όταν τα μαζέψουμε αυτά, τα ενώνουμε με την δεσμευμένη λέξη “UNION” της SQL, και τα αποτελέσματα που παίρνουμε είναι η ένωση των επιμέρους συνόλων εγγραφών που θα παίρναμε με τα ξεχωριστά queries. Αυτός είναι και ο λόγος για τον οποίο δεν επιτρέπεται να έχουμε πολλαπλούς πίνακες στην Βάση Δεδομένων για την κάθε ιεραρχία, γιατί το UNION στην SQL δεν μπορεί να γίνει αν τα επιμέρους queries αναφέρονται σε διαφορετικούς πίνακες.

Τέλος, οι εγγραφές από την Βάση Δεδομένων επιστρέφουν στην δομή ResultSet, την οποία ορίζει και υλοποιεί το πακέτο java.sql. Με αυτή την δομή μπορούμε να μετρήσουμε τον αριθμό των εγγραφών που επέστρεψε το query, και να τις πάρουμε μια-μια, κάτι που γίνεται στην διαδικασία παρουσίασης των αποτελεσμάτων.

6.2.3 Βασικοί Αλγόριθμοι

Στην ενότητα αυτή περιγράφουμε τους σημαντικότερους αλγόριθμους που υλοποιούνται κατά την εκτέλεση της εφαρμογής, κυρίως με ψευδοκώδικα. Εκτός από τον αλγόριθμο κατασκευής του καθολικού κανονικοποιημένου δένδρου, που παρουσιάστηκε στο θεωρητικό κομμάτι, αυτοί είναι οι αλγόριθμοι συγχώνευσης των διαστάσεων, της εύρεσης των διαθέσιμων διαστάσεων, και της κατασκευής των queries από τα μονοπάτια του χρήστη.

6.2.3.1 Συγχώνευση των Διαστάσεων

Τα αποτελέσματα του αλγορίθμου αυτού τα δείξαμε στην υποενότητα 3.4, όταν αναφέραμε την έννοια του καθολικού κανονικοποιημένου δένδρου. Δείξαμε το αποτέλεσμα της συγχώνευσης 2 συνόλων διαστάσεων. Επειδή στην μνήμη τα σύνολα των διαστάσεων αναπαριστούνται ως δενδρικές δομές, εφαρμόζουμε τον εξής αλγόριθμο για την συγχώνευση των διαστάσεων:

Είσοδος: Ένα Vector με όλα τα σύνολα των διαστάσεων, δηλαδή ένα Vector που τα στοιχεία του είναι τα δένδρα με ρίζες τα dimHi, όπου i είναι ο αριθμός της ιεραρχίας.

Έξοδος: Ένα δένδρο, που περιέχει τις συγχωνευμένες διαστάσεις.

Αλγόριθμος:

```
/* Θεωρούμε ότι το Vector με τις διαστάσεις λέγεται dims, και το τελικό δένδρο globdims
*/
Αν dims.size()==0 τότε exit;
globdims = dims.get(0);
Για κάθε στοιχείο του dims, δηλαδή για κάθε δένδρο διαστάσεων dimHi:
        για κάθε διάσταση d του dimHi (δηλ. dimHi→d):
                αν το d δεν υπάρχει σαν παιδί του globdims (αν ¬∃ globdims→d), τότε:
                        -πρόσθεσέ το, μαζί με τα παιδιά του, στο globdims;
                αλλιώς:
                        -για κάθε παιδί n του dimHi→d που δεν υπάρχει στο globdims→d,
                        πρόσθεσέ το στο globdims→d σαν παιδί (globdims→d→n)
                        }
        }
}
/*** Τέλος***/
```

6.2.3.2 Εύρεση των Διαθέσιμων Διαστάσεων

Αυτός ο αλγόριθμος επιστρέφει ένα σύνολο από διαστάσεις, τις οποίες μπορεί να επιλέξει ο χρήστης κατά την διάρκεια της κατασκευής του καθολικού κανονικοποιημένου δένδρου. Λαμβάνοντας υπ'όψιν τις διαστάσεις που ήδη έχει επιλέξει ο χρήστης και τον γράφο εξαρτήσεων, βρίσκει τις διαθέσιμες για επιλογή διαστάσεις. Αυτή η διαδικασία γίνεται κάθε φορά που ο χρήστης πρόκειται να επεκτείνει το καθολικό κανονικοποιημένο δένδρο για ένα επιπλέον επίπεδο. Να σημειώσουμε ότι γίνεται χρήση μιας συνάρτησης, η οποία βρίσκει τις εξαρτήσεις, άμεσες και έμμεσες, μιας διάστασης. Ο αλγόριθμος έχει ως εξής:

Είσοδος: Ο γράφος εξαρτήσεων και το μέχρι εκείνη τη στιγμή κατασκευασμένο καθολικό κανονικοποιημένο δένδρο.

Έξοδος: Ένα σύνολο σε μορφή Vector που περιέχει τα ονόματα των διαθέσιμων για επιλογή διαστάσεων.

Αλγόριθμος:

```
/* Το καθολικό κανονικοποιημένο δένδρο αναφέρεται ως gnt και ο γράφος εξαρτήσεων
ως dg. Το σύνολο των διαθέσιμων διαστάσεων θα μπει σε ένα Vector με το όνομα avdims
*/
Για κάθε διάσταση d:
        Vector dep= βρες τις εξαρτήσεις της d;
        αν έχει επιλεχθεί η διάσταση d στο gnt, τότε:
                -προχώρα παρακάτω;
        αλλιώς:
                αν το dep είναι κενό, τότε:
                        -πρόσθεσε στο avdim το d;
                αλλιώς:
                        αν όλες οι διαστάσεις που βρίσκονται στο dep έχουν επιλεγεί
                        στην κατασκευή του gnt, τότε:
                                -πρόσθεσε στο avdim το d;
                        αλλιώς προχώρα παρακάτω;
                        }
                }
        }
}
/*** Τέλος***/
```

6.2.3.3 Κατασκευή των SQL Ερωτήσεων από το Μονοπάτι

Αυτός ο αλγόριθμος είναι ίσως ο πιο σημαντικός. Σκοπός είναι να πάρουμε το καθολικό κανονικοποιημένο δένδρο που έχει κατασκευάσει ο χρήστης, και από αυτό να καταφέρουμε να μαζέψουμε σε μια δομή τα queries που θα μας δώσουν τα αποτελέσματα, ξεχωριστά για κάθε δένδρο. Αυτό σε βήματα σημαίνει ότι πρέπει να συμπληρώσουμε εμείς το καθολικό κανονικοποιημένο δένδρο ώστε να εξαντλήσουμε όλους τους δυνατούς συνδυασμούς επιλογής διαστάσεων και κόμβων (ώστε να φτάνουμε πάντα στα φύλλα των αρχικών ιεραρχιών, γιατί εκεί βρίσκονται τα επιμέρους queries), δηλαδή κατασκευάζοντας πάρα πολλά μονοπάτια, και μετά, για κάθε ιεραρχία να προσπαθήσουμε να ταυτίσουμε όλα τα μονοπάτια που κατασκευάσαμε με την ιεραρχία. Όσα πραγματικά ταυτίζονται, μας δίνουν τα queries στα φύλλα τους, και έτσι τα μαζεύουμε σε ένα String, ενωμένα με την λέξη 'UNION', και έτσι φτιάχνουμε την SQL ερώτηση που θα μας επιστρέψει όλες τις εγγραφές που ζητάει ο χρήστης. Χωρίς να μπαίνουμε σε μεγάλη λεπτομέρεια υλοποίησης στις δομές της μνήμης, ο αλγόριθμος έχει την εξής μορφή:

Είσοδος: Το μονοπάτι που έχει κατασκευάσει ο χρήστης.

Έξοδος: Ένας μονοδιάστατος πίνακα μεγέθους n, όπου n ο αριθμός των ιεραρχιών, στις θέσεις του οποίου βρίσκονται τα queries που επιστρέφουν τις εγγραφές για κάθε ιεραρχία. Το

i είναι ο αριθμός της σειράς της ιεραρχίας (πχ. αν είχαμε τις ιεραρχίες H1, H2, H5, η H5 θα βρισκόταν στην θέση 3 του πίνακα, δηλ. με i=2)

Αλγόριθμος:

/* Το αρχικό μονοπάτι του χρήστη θα το λέμε userpath */
Για κάθε ιεραρχία h:
-συμπλήρωσε το userpath σύμφωνα με τις διαστάσεις και τους κόμβους της h, με όλους τους δυνατούς τρόπους, και βάλε τα σε ένα Vector paths, αφού σβήσεις τους Δ κόμβους (κόμβοι με όνομα "none");
-Από αυτό το Vector σβήσε τις διπλοεγγραφές;

/* Διπλοεγγραφές υπάρχουν όταν επιλεχθούν πχ. δύο διαστάσεις που προκαλούν την εισαγωγή κόμβου Δ (τρίγωνο), το οποίο αναπαρίσταται στο μονοπάτι με κόμβο ονόματος "none", με διαφορετική σειρά. Πχ, το μονοπάτι /Spielberg/none/Paramount/none/Drama μπορεί να προκύψει με (ίσως και όχι μόνο) 2 διαφορετικούς τρόπους, εάν το πρώτο "none" οφείλεται στην επιλογή μιας διάστασης Δ1, και το δεύτερο στην επιλογή μιας διάστασης Δ2. Εάν γινόταν ανάποδη επιλογή στην σειρά των Δ1 και Δ2, πάλι το ίδιο μονοπάτι θα είχαμε (διπλοεγγραφή). */

Για κάθε μονοπάτι του Vector paths:
-άλλαξε τα γενικά ονόματα των κόμβων σε τοπικά;
-φτιάξε ομογενή μονοπάτια, δηλαδή ανακάτεψε την επιλογή των διαστάσεων με όλους τους δυνατούς τρόπους, και κράτα από αυτά τα μονοπάτια μόνο όσα δεν αντιβαίνουν τους περιορισμούς του γράφου εξαρτήσεων, τα οποία φύλαξέ τα σε ένα Vector homopaths;
}
}

/* Έχουμε φτάσει στο σημείο να έχουμε στο homopaths όλα τα ομογενή μονοπάτια που μπορούν να αποτιμηθούν στην ιεραρχία h, δηλαδή σαν να κατασκευάσαμε όλα τα ομογενή κανονικοποιημένα δένδρα για την ιεραρχία h, με πλήρη επέκταση σε διαστάσεις και όλες τις δυνατές επιλογές κόμβων από το σημείο που σταμάτησε ο χρήστης και μετά. Τώρα πρέπει να βρούμε ποιά από αυτά τα κανονικοποιημένα δένδρα αποτιμούνται στην ιεραρχία h */

Για κάθε στοιχείο (μονοπάτι) του homopaths:
αν αυτό μπορεί να αποτιμηθεί στην ιεραρχία h (δηλ. οδηγεί σε κάποιο φύλλο με πλήρη αντιστοιχία κόμβων), τότε:
-πρόσθεσε το στοιχείο (μονοπάτι) σε ένα Vector με όνομα finalpaths;
-πρόσθεσε το query, στο οποίο μας οδηγεί το μονοπάτι, σε ένα Vector qr;
}
}

/* Στο Vector finalpaths έχουμε όλα τα μονοπάτια που οδηγούν σε query, δηλαδή που αποτιμούνται σε αποτέλεσμα, και στο qr έχουμε όλα τα επιμέρους queries */

-Βάλε σε έναν πίνακα qrArch, στην θέση i (όπως εξηγήσαμε στην εισαγωγή) το finalpaths;
-Σε κάθε στοιχείο του qr (σε κάθε query) πρόσθεσε τις επιπλέον επιλογές του χρήστη που αφορούν την συγκεκριμένη ιεραρχία (πχ. "and ID>10 and PRICE<1000", που δηλώνονται μέσα από το γραφικό περιβάλλον);
-Κατασκεύασε ένα String s που φτιάχνεται από την συνένωση όλων των στοιχείων του qr, όπου ανάμεσά τους μπαίνει η φράση " UNION ", και φύλαξέ το σε έναν πίνακα με το όνομα query, στην θέση i (αντίστοιχα με το qrArch);

/* Άρα, έχουμε 2 πίνακες, τον qrArch και τον query, οι οποίοι στην θέση i περιέχουν, ο πρώτος ένα Vector με όλα τα μονοπάτια που αποτιμήθηκαν στην ιεραρχία με σειρά i+1, και ο δεύτερος, το query που επιστρέφει όλες τις εγγραφές από τα φύλλα που βρίσκονται στο τέλος των μονοπατιών του Vector, που μόλις αναφέραμε, για την ίδια ιεραρχία */
}

-Τέλος, για κάθε στοιχείο του query:
-εφάρμοσε το query στην Βάση Δεδομένων, και με το αποτέλεσμα φτιάξε έναν πίνακα, που βλέπει ο χρήστης;
}

-Μάζεψε τους πίνακες και εμφάνισέ τους όλους μαζί;
/*** Τέλος***/

Από αυτά διαπιστώνουμε ότι ο τελευταίος αλγόριθμος κάνει την σημαντικότερη δουλειά. Κατασκευάζει όλα τα δυνατά μονοπάτια, και από αυτά τελικά κρατάει, αρχικά, όσα υπακούουν στους περιορισμούς του γράφου εξαρτήσεων, και τελικά, όσα μπορούν να εφαρμοστούν στην εκάστοτε ιεραρχία, για την οποία κατασκευάστηκαν. Έτσι, σύντομα βρίσκουμε τα queries που ζητάμε, και κρατάμε και σε ένα πίνακα τα μονοπάτια που μας οδήγησαν σε αυτά, όταν ο χρήστης δεν έχει κάνει πλήρη επιλογή των διαστάσεων. Αυτό το τελευταίο είναι χρήσιμο για την εποπτεία της λειτουργίας του συστήματος.

Πρέπει να σημειώσουμε ότι για την εκτέλεση των αλγορίθμων που περιγράψαμε συνεργάζονται αρκετές συναρτήσεις, οι οποίες εκτελούν το κάθε βήμα ξεχωριστά.

Επίσης να σημειώσουμε ότι υπάρχουν και κάποιοι αλγόριθμοι για την παρουσίαση των δενδρικών δομών και του γράφου εξαρτήσεων. Αυτοί φροντίζουν για την σωστή τοποθέτηση των κόμβων, δηλαδή να μην υπερκαλύπτει ένας κόμβος κάποιον άλλο, επειδή δεν έχουν όλοι το ίδιο μέγεθος. Περισσότερες λεπτομέρειες θα δοθούν, κυρίως για την γενική ιδέα, κατά την αναφορά των μεθόδων που επιτελούν τις συγκεκριμένες λειτουργίες.

6.2.4 Περιγραφή Κλάσεων

Σε αυτή την ενότητα θα αναφέρουμε όλες τις κλάσεις που υλοποιήθηκαν, τον σκοπό που εξυπηρετούν, και όλες τις μεταβλητές και μεθόδους που ανήκουν σε αυτές, με σύντομα σχόλια. Σε κάποιες συγκεκριμένες συναρτήσεις θα δοθεί και μια περιγραφική επεξήγηση.

6.2.4.1 public class Datab

Αυτή η κλάση υλοποιεί την σύνδεση με την Βάση Δεδομένων μέσω JDBC γέφυρας. Προσφέρει μεθόδους με τις οποίες ο χρήστης εκτελεί τα queries και δομές οι οποίες κρατάνε τα αποτελέσματα.

Πεδία

- `Connection con`
  Η σύνδεση με την Βάση Δεδομένων.
- `Statement st`
  Το query που θα περάσει για εκτέλεση στην Βάση Δεδομένων.
- `ResultSet rst`
  Σε αυτή την δομή επιστρέφουν τα αποτελέσματα (αν υπάρχουν) των queries.

Μέθοδοι

- `public void init() throws Exception`
  Η αρχικοποίηση των μεταβλητών, σύνδεση με την Βάση Δεδομένων.
- `public void execQ(String s) throws SQLException`
  Εκτελεί το (select) query s.
- `public void execUp(String s) throws SQLException`
  Εκτελεί το (update) query s.
- `public ResultSet getRst()`
  Επιστρέφει το rst, που κρατάει τα αποτελέσματα.
- `public void end() throws Exception`
  Κλείνει την σύνδεση με την Βάση Δεδομένων.

6.2.4.2 public class DeleteDim

Αυτή η κλάση υλοποιεί την διαπροσωπεία χρήστη για την διαγραφή μιας διάστασης, και ενεργοποιεί τις συναρτήσεις που το κάνουν αυτό. Αποτελεί προέκταση της κλάσης JDialog, και υλοποιεί τα interfaces ActionListener, WindowListener.

Πεδία

- `static JPanel pan`
  Η μεταβλητή για το panel.
- `static JButton cancel`
  Το κουμπί cancel.
- `static JButton ok`
  Το κουμπί ok.
- `static JLabel info1`
  Μία ετικέτα με πληροφορία, που φαίνεται στο παράθυρο.
- `static JLabel info2`
  Μία δεύτερη ετικέτα με πληροφορία.
- `static JLabel pick`
  Μία τρίτη ετικέτα με πληροφορία.
- `static JComboBox d`
  Μία μεταβλητή ComboBox.
- `static JLabel empty`
  Άλλη μια ετικέτα.

Μέθοδοι

- `public DeleteDim()`

Συνάρτηση Κατασκευής της κλάσης, αρχικοποίηση πεδίων.

- `public static void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Συνάρτηση που χρησιμοποιείται για την τοποθέτηση των συστατικών (components) στο παράθυρο (panel).
- `public static void refreshDims(Vector v)`
  Ανανεώνει τις επιλογές των διαστάσεων.
- `public void actionPerformed(ActionEvent evt)`
  Η συνάρτηση χειρισμού των γεγονότων πατήματος κουμπιών. Από εδώ καλούνται οι διάφορες συναρτήσεις που εκτελούν τις εντολές του χρήστη.
- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Αυτές οι συναρτήσεις χειρίζονται τα γεγονότα που συμβαίνουν την στιγμή που έχουμε ένα γεγονός παραθύρου. Αντίστοιχα, καλούνται όταν το παράθυρο κλείνει, όταν ενεργοποιείται, όταν απενεργοποιείται, όταν έχει κλείσει, όταν έχει ανοίξει, όταν ελαχιστοποιείται, και όταν μεγιστοποιείται. Δεν θα ξανα-αναφερθούμε με μεγάλη λεπτομέρεια σε αυτές, καθώς υπάρχουν σε κάθε κλάση που υλοποιεί γραφικό παράθυρο.

6.2.4.3 public class DependenceGraph

Αυτή η κλάση είναι μια από τις σημαντικότερες κλάσεις που έχουν υλοποιηθεί, γιατί χειρίζεται τον γράφο εξαρτήσεων, περιέχει ως πεδίο το καθολικό κανονικοποιημένο δένδρο, και υλοποιεί με μεθόδους πολλές από τις ενέργειες που έχουν να κάνουν με τον αλγόριθμο κατασκευής των ομογενών κανονικοποιημένων δένδρων, που περιγράφτηκε στην ενότητα 6.2.3.3. Να σημειώσουμε ότι εδώ, το καθολικό κανονικοποιημένο δένδρο υπάρχει σαν ξεχωριστή δομή, η οποία πέρα από κάποια βοηθητικά πεδία (που θα δούμε αργότερα) περιέχει το δένδρο σαν μονοπάτι επιλογών του χρήστη, το οποίο θα αναφέρουμε σαν 'Μονοπάτι'. Περισσότερες λεπτομέρειες ακολουθούν παρακάτω:

Πεδία

- `private NormTree nt`
  Το πεδίο που κρατάει το Μονοπάτι.
- `private Tree dims`
  Ένα δένδρο που κρατάει τις συγχωνευμένες διαστάσεις.
- `private Hashtable depgr`
  Ο γράφος εξαρτήσεων, που φυλάσσεται σε δομή HashTable.
- `private Hashtable hash`
  Μία δομή HashTable που κρατάει τους ταυτόσημους σε νόημα κόμβους, με τα διαφορετικά ονόματα που χρησιμοποιούν, σύμφωνα με την δομή που περιγράψαμε στην ενότητα 5.2.4, Πίνακας 5.3.
- `private Vector dep`
  Ένα Vector που κρατάει τους κόμβους που έχουν εξάρτηση, για ευκολία αναζήτησης.

Μέθοδοι

- `DependenceGraph()`
  Κατασκευαστής της κλάσης.
- `DependenceGraph(Vector v)`
  Κατασκευαστής της κλάσης, με παράμετρο ένα Vector με τα δένδρα των διαστάσεων, για να γίνει η συγχώνευση.

- `public Tree getDims()`
  Επιστρέφει το δένδρο των συγχωνευμένων διαστάσεων.
- `public void setDims(Tree t)`
  Θέτει τις συγχωνευμένες διαστάσεις σύμφωνα με το t.
- `public NormTree getNt()`
  Επιστρέφει το Μονοπάτι.
- `public void setNt(NormTree t)`
  Θέτει το Μονοπάτι ίσο με το t.
- `public Hashtable getDepGr()`
  Επιστρέφει τον γράφο εξαρτήσεων.
- `public Hashtable getHash()`
  Επιστρέφει τον HashTable με τους ταυτόσημους κόμβους.
- `public void setHash(Hashtable h)`
  Θέτει αυτόν τον HashTable ίσο με την είσοδο h.
- `public Vector getDep()`
  Επιστρέφει το Vector με τις εξαρτώμενες διαστάσεις.
- `public void setDep(Vector v)`
  Θέτει αυτό το Vector ίσο με την είσοδο v.
- `public void initDepGr()`
  Αρχικοποιεί την δομή του γράφου εξαρτήσεων από το αρχείο.
- `public void setDepGr(Hashtable t)`
  Θέτει τον γράφο εξαρτήσεων ίσο με την είσοδο t.
- `public Tree concatDimensions(Vector dims)`
  Η συνάρτηση που πραγματοποιεί την συγχώνευση των διαστάσεων, που υπάρχουν σαν στοιχεία της εισόδου (Vector dims), αλγόριθμος 6.2.3.1.
- `public Vector getAvailDims()`
  Επιστρέφει σε Vector τις διαθέσιμες διαστάσεις, σύμφωνα με την δομή των συγχωνευμένων διαστάσεων, που μπορεί να επιλέξει ο χρήστης, σύμφωνα με τον γράφο εξαρτήσεων και τις επιλογές που έχουν ήδη γίνει για την κατασκευή του καθολικού κανονικοποιημένου δένδρου, αλγόριθμος 6.2.3.2.
- `public Vector getAvailDims(Tree dims2)`
  Επιστρέφει ότι και η προηγούμενη συνάρτηση, σύμφωνα όμως με το δένδρο διαστάσεων dims2, και όχι τις συγχωνευμένες διαστάσεις.
- `public Vector findWords2(String s)`
  Βοηθητική συνάρτηση για την ανάγνωση των γραμμών του αρχείου dependency.txt. Το s είναι η κάθε γραμμή αυτού του αρχείου.
- `public void initHash()`
  Αρχικοποιεί τον HashTable hash που περιέχει τους ταυτόσημους κόμβους, σύμφωνα με τον Πίνακα 5.3.
- `public Vector findWords(String s)`
  Βοηθητική συνάρτηση για την ανάγνωση των γραμμών του αρχείου SimNodes.txt. Το s είναι η κάθε γραμμή του αρχείου αυτού.
- `public void calcAllPaths(Vector paths, Vector v, String h)`
- `public void homoPaths(NormTree in, Vector v, String h)`
- `public NormTree homoPaths(NormTree in, String h)`
  Αυτές οι τρεις συναρτήσεις αλλάζουν τα γενικά ονόματα των κόμβων σε τοπικά. Ουσιαστικά χρησιμοποιείται μόνο η τελευταία, όπου σαν είσοδο παίρνει το Μονοπάτι in και το όνομα της ιεραρχίας h (σαν “hi”, i=1,2,…) και επιστρέφει το Μονοπάτι με τις τοπικές ονομασίες κόμβων. Οι δύο πρώτες συναρτήσεις κάνουν την ίδια δουλειά, όταν στο Vector paths έχουμε πολλά Μονοπάτια, οπότε καλείται κατάλληλα η δεύτερη συνάρτηση, και τα αποτελέσματα για κάθε ένα από αυτά μπαίνουν στο Vector v.
- `public void addNewDep(String which, String from)`
  Προσθέτει μια νέα εξάρτηση στον γράφο εξαρτήσεων. Οι κόμβοι δηλώνονται με τα γενικά τους ονόματα.
- `public void delDep(String which, String from)`
  Διαγράφει μια εξάρτηση. Οι κόμβοι δηλώνονται με τα γενικά τους ονόματα.
- `public void saveDG(String file)`
  Αποθηκεύει τον γράφο εξαρτήσεων στο αρχείο file.
- `public void cleanDims(Tree dimi)`

Από το Μονοπάτι σβήνει τις επιλογές κόμβων διαστάσεων που έγιναν κατά την συμπλήρωση διαστάσεων και κόμβων αυτόματα, που δεν ανήκουν στο δένδρο διαστάσεων dimi. Δηλαδή, προετοιμάζει το Μονοπάτι ώστε να μπορεί να αποτιμηθεί σε μια τοπική ιεραρχία.

- `public void fillPath(Vector hi, Tree dimi, Vector results)`
  Γεμίζει αυτόματα το Μονοπάτι με όλες τις υπόλοιπες δυνατές επιλογές διαστάσεων και κόμβων. Ο hi περιέχει την ιεραρχία για την οποία γίνεται η διαδικασία αυτή, το dimi είναι το δένδρο διαστάσεων για την συγκεκριμένη ιεραρχία, και τα αποτελέσματα μπαίνουν στον Vector results.
- `public boolean isDependent(String s1, String s2)`
  Επιστρέφει TRUE αν το s1 εξαρτάται άμεσα ή έμμεσα από το s2, όπου s1,s2 είναι τα ονόματα δύο διαστάσεων.
- `public boolean isNotNoneDimension(Vector hi)`
  Ελέγχει αν η τρέχουσα διάσταση που έχει επιλεχθεί για το Μονοπάτι πρέπει να είναι τριγωνάκι (Δ), για τις ιεραρχίες που υπάρχουν στον Vector hi (όλες). Επιστρέφει TRUE αν δεν είναι διάσταση Δ, αλλιώς επιστρέφει FALSE.
- `public Vector getPathNodes(Vector hi)`
  Αυτή η συνάρτηση καλείται όταν η τρέχουσα διάσταση δεν είναι Δ. Επιστρέφει σε ένα Vector τα ονόματα των κόμβων από την τρέχουσα διάσταση που, μαζί με το τρέχον Μονοπάτι, υπάρχουν σε κάποιο μονοπάτι, στις ιεραρχίες που ανήκουν στο Vector hi (όλες). Αυτό γίνεται για να δίνονται στον χρήστη οι επιλογές που μπορούν να οδηγήσουν σε ένα φύλλο, σε κάποια ιεραρχία.
- `public void getDependencies(String s, Vector v)`
  Μαζεύει σε ένα Vector v όλες τις διαστάσεις από τις οποίες εξαρτάται άμεσα ή έμμεσα η διάσταση με όνομα s.

6.2.4.4 public class DGGraph

Αυτή η κλάση που είναι υπεύθυνη για την εμφάνιση του γράφου εξαρτήσεων και για την τοποθέτηση των κόμβων στα διάφορα σημεία τους. Ουσιαστικά, βρίσκει από το HashTable όλες τις διαστάσεις που συμμετέχουν, δημιουργεί κόμβους με το όνομά τους και ακμές, σύμφωνα με τις εξαρτήσεις, και μετά ακολουθεί ένας αλγόριθμος για την τοποθέτησή τους στο επίπεδο. Ο αλγόριθμος αυτός δεν παρουσιάστηκε προηγουμένως, γιατί δεν ανήκει τόσο στην ουσία της εφαρμογής, αλλά απλά στην παρουσίαση. Επειδή το HashTable αποθηκεύει τις διαστάσεις με αυθαίρετο τρόπο, σκεφτήκαμε να τις βάλουμε σε κυκλική διάταξη, με τυχαία σειρά. Έτσι, χρησιμοποιώντας πολικές συντεταγμένες, και υπολογίζοντας από τον αριθμό των κόμβων την γωνία μεταξύ αυτών, πήραμε ένα αυθαίρετο κέντρο στο panel, και τοποθετήσαμε γύρω γύρω τους κόμβους. Η κλάση αυτή επεκτείνει το JPanel και έχει ως εξής:

Πεδία

- `private JGraphModelAdapter m_jgAdapter`
  Αυτό είναι η μεταβλητή για τον χειρισμό του γράφου μέσω του JGraph.
- `private final Color DEFAULT_BG_COLOR`
  Μια σταθερά που φέρει την τιμή ενός χρώματος.
- `JGraph jgraph`
  Η μεταβλητή του γράφου.
- `Vector nodes`
  Ένα Vector που κρατάει όλους τους κόμβους που θα μπουν στον γράφο.

Μέθοδοι

- `public DGGraph()`
  Η συνάρτηση κατασκευαστής της κλάσης.

- `public void init()`
  Εδώ αρχικοποιούνται οι δομές και μπαίνουν οι κόμβοι στον γράφο (χωρίς συγκεκριμένη θέση).
- `public void positionVertexAt(Object vertex, int x, int y)`
  Η συνάρτηση αυτή τοποθετεί τον κόμβο vertex στην θέση με συντεταγμένες x και y.
- `public int calcWidth(String s)`
  Εδώ γίνεται ο υπολογισμός του πλάτους του κουτιού που δηλώνει τον κόμβο, σύμφωνα με το μέγεθος του ονόματός του s.
- `public JGraph getJGraph()`
  Επιστρέφει τον γράφο.
- `public void fix (Vector v)`
  Αυτή η συνάρτηση υπολογίζει για κάθε κόμβο την θέση στην οποία θα μπει, και καλεί για κάθε κόμβο την συνάρτηση positionVertexAt(…).

6.2.4.5 public class EditDep

Αυτή η κλάση είναι που υλοποιεί την διαπροσωπεία χρήστη για τον χειρισμό των εξαρτήσεων. Ουσιαστικά είναι το παράθυρο που δείχνει στον χρήστη τις διάφορες εξαρτήσεις, και μέσα από αυτό ο χρήστης μπορεί να προσθέσει ή να αφαιρέσει εξαρτήσεις, ενεργοποιώντας τις επιλογές του με το πάτημα κάποιων κουμπιών. Πιο συγκεκριμένα, η κλάση αυτή επεκτείνει την κλάση JDialog, υλοποιεί τα interfaces ActionListener, ListSelectionListener και WindowListener, και έχει ως εξής:

Πεδία

- `static NewDimDep ndd`
  Αυτή η μεταβλητή δηλώνει το αντικείμενο τύπου NewDimDep, το οποίο είναι ένα μικρό παράθυρο για την δήλωση μιας νέας εξάρτησης. Θα το δούμε παρακάτω.
- `static JPanel pan`
- `static JPanel panel1`
- `static JPanel panel2`
- `static JPanel pan1`
  Δήλωση κάποιων panels για την εμφάνιση των στοιχείων.
- `static JButton cancel`
- `static JButton ok`
- `static JButton del1`
- `static JButton add1`
- `static JButton del2`
- `static JButton add2`
  Δήλωση κάποιων κουμπιών, που εκκινούν λειτουργίες. Με την σειρά, πρόκειται για την ακύρωση των αλλαγών του χρήστη, την επικύρωση αυτών, την διαγραφή των εξαρτήσεων κάποιου κόμβου, την προσθήκη νέου εξαρτώμενου κόμβου, την διαγραφή εξάρτησης για έναν κόμβο, και την προσθήκη εξάρτησης για έναν κόμβο.
- `static JLabel fr`
- `static JLabel info`
- `static JLabel info1`
- `static JLabel info2`
- `static JLabel info3`
  Κάποια labels με μηνύματα και πληροφορίες.
- `static DefaultListModel listModel1`
- `static DefaultListModel listModel2`
- `static JList vList1`
- `static JList vList2`
  Με αυτές τις μεταβλητές δηλώνουμε 2 λίστες και τα μοντέλα για την διαχείρισή τους με δυναμικό τρόπο.
- `static JScrollPane scroll1`

- `static JScrollPane scroll2`
  Δηλώνουμε δύο μπάρες κύλισης.
- `static Vector arr2`
  Ένα Vector που χρησιμοποιείται για την αποθήκευση των πολλαπλών επιλογών του χρήστη από την λίστα 2.
- `JButton editDims`
- `JButton editDep`
- `JButton res`
- `JButton gnt`
- `JButton gr`
- `JButton restart`
- `JButton quit`
- `JToolBar tb`
  Με αυτές τις μεταβλητές δηλώνουμε ένα ToolBar και τα κουμπιά που μπαίνουν σε αυτό. Αυτά απλά μεταφέρουν τον χρήστη σε άλλα παράθυρα, εκτός από το quit, που κλείνει την εφαρμογή, και το restart, που επαναρχικοποιεί τις δομές από τα αρχεία. Δεν θα αναφερθούμε σε αυτό το ToolBar με λεπτομέρειες, γιατί αυτό εμφανίζεται στα περισσότερα παράθυρα διαπροσωπείας.

Μέθοδοι

- `public EditDep()`
  Η συνάρτηση κατασκευαστής της κλάσης. Τοποθετεί τα γραφικά στοιχεία στην θέση που πρέπει και καλεί τις αρχικοποιήσεις των λιστών.
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Συνάρτηση τοποθέτησης των γραφικών στοιχείων στην θέση που πρέπει.
- `public void initList1()`
- `public void initList2(String s)`
  Συναρτήσεις αρχικοποίησης των λιστών. Η λίστα 1 περιέχει τους κόμβους που έχουν εξάρτηση, και η λίστα 2 περιέχει τις εξαρτήσεις του κόμβου με όνομα s. Αυτή ενεργοποιείται κάθε φορά που από την πρώτη λίστα επιλέγεται ένας κόμβος.
- `public void actionPerformed(ActionEvent evt)`
  Χειρίζεται τα γεγονότα που ενεργοποιούνται με το πάτημα των κουμπιών. Από εδώ καλούνται οι συναρτήσεις εκτέλεσης των εντολών του χρήστη.
- `public void cleanEmptyDep()`
  Σβήνει από τον γράφο εξαρτήσεων του κόμβους που άφησε ο χρήστης χωρίς εξάρτηση, και δεν τους έχει σβήσει από τον γράφο.
- `public void valueChanged(ListSelectionEvent e)`
  Χειρισμός των γεγονότων που προκαλούνται όταν ο χρήστης κάνει κάποια επιλογή σε μια από τις δυο λίστες.
- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.4.6 public class EditDim

Αυτή η κλάση υλοποιεί την διαπροσωπεία χρήστη για τον χειρισμό των διαστάσεων, τις ενέργειες των οποίων περιγράψαμε στο Κεφάλαιο 5. Ο χρήστης μπορεί να δημιουργήσει ή να σβήσε διαστάσεις, να μεταφέρει κόμβους μεταξύ διαστάσεων κλπ. Η κλάση αυτή επεκτείνει την JDialog και υλοποιεί τα interfaces ActionListener, ListSelectionListener και WindowListener, και έχει ως εξής:

Πεδία

- `static RenameDims rd`
- `static DeleteDim dd`
  Αυτές οι δύο μεταβλητές αντιστοιχούν σε αντικείμενα δύο κλάσεων που θα δούμε παρακάτω, μέσω των οποίων γίνεται η μετονομασία και η διαγραφή μιας διάστασης.
- `static JPanel pan`
- `static JPanel pan1`
  Δήλωση δύο panels
- `JButton editDims`
- `JButton editDep`
- `JButton res`
- `JButton gnt`
- `JButton gr`
- `JButton restart`
- `JButton quit`
- `static JToolBar tb`
  Δήλωση του ToolBar και των κουμπιών του, όπως είπαμε και στην προηγούμενη ενότητα.
- `static JButton cancel`
- `static JButton ok`
- `static JButton moveright`
- `static JButton moveleft`
- `static JButton newDim`
- `static JButton del`
- `static JButton ren`
  Δήλωση των κουμπιών αυτού του παραθύρου. Ενεργοποιούν τις λειτουργίες ακύρωσης των αλλαγών, επικύρωσης των αλλαγών, μεταφορά κόμβου από την επιλεγμένη διάσταση στα αριστερά (από λίστα 1) προς την επιλεγμένη διάσταση στα δεξιά (από λίστα 2), διαγραφή ενός κόμβου από την δεξιά διάσταση, δημιουργία νέας διάστασης, διαγραφή διάστασης και μετονομασία διάστασης, αντίστοιχα.
- `static JLabel fr`
- `static JLabel t`
- `static JLabel empty`
- `static JLabel empty1`
- `static JLabel empty2`
  Δήλωση κάποιων labels με μηνύματα και πληροφορίες.
- `static JComboBox from`
- `static JComboBox to`
  Δήλωση δύο ComboBoxes για την επιλογή διάστασης, από και προς (χρησιμοποιείται στην μεταφορά κόμβου).
- `static DefaultListModel listModel1`
- `static DefaultListModel listModel2`
- `static JList vList1`
- `static JList vList2`
  Δήλωση των δύο λιστών και των χειριστών τους.
- `static JScrollPane scroll1`
- `static JScrollPane scroll2`
  Δήλωση δύο μπαρών κυλίσεως.
- `static Vector arr1`
- `static Vector arr2`
  Δήλωση δύο μεταβλητών Vector, για να κρατάμε κάπου την πολλαπλή επιλογή των λιστών. Το arr1 είναι για την πρώτη λίστα και το arr2 για την δεύτερη.

Μέθοδοι

- `public EditDim()`
  Συνάρτηση κατασκευαστής της κλάσης, τοποθέτηση των γραφικών στοιχείων, αρχικοποίηση των λιστών και μεταβλητών.
- `public static void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`

Ορίζει τις συντεταγμένες του κάθε γραφικού στοιχείου.

- `public static void refreshDims(Vector v)`
  Με είσοδο ένα Vector v που περιέχει τις υπάρχουσες διαστάσεις, αρχικοποιεί τα ComboBoxes που περιέχουν τις επιλογές διαστάσεων.
- `public static void refreshList1(String s)`
- `public static void refreshList2(String s)`
  Συναρτήσεις που αλλάζουν το περιεχόμενο των λιστών σύμφωνα με την επιλογή διαστάσεων που έχει γίνει για την κάθε μια. Δείχνουν τους κόμβους της κάθε διάστασης.
- `public void actionPerformed(ActionEvent evt)`
  Συνάρτηση χειρισμού των γεγονότων που προκαλούνται από το πάτημα κουμπιών.
- `public static boolean isValid(String s)`
  Ελέγχει αν το όνομα μιας καινούργιας διάστασης είναι κενό.
- `public void valueChanged(ListSelectionEvent e)`
  Συνάρτηση χειρισμού των γεγονότων που προκαλούνται από την αλλαγή στην επιλογή στοιχείων σε κάποια από τις δύο λίστες.
- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.4.7 public class ExtraOptions

Αυτή η κλάση υλοποιεί το γραφικό κομμάτι του παραθύρου που ο χρήστης κατασκευάζει τα μονοπάτια, το οποίο δίνει τις επιλογές για περαιτέρω περιορισμούς στις εγγραφές. Πιο συγκεκριμένα, βρίσκει τα πεδία του πίνακα που αντιπροσωπεύει τις εγγραφές της κάθε ιεραρχίας, και ανάλογα με τον τύπο του κάθε πεδίου του πίνακα προσφέρει στον χρήστη την δυνατότητα να επιλέξει αν θέλει πχ το ID των εγγραφών που θα του επιστραφούν να είναι μεγαλύτερο από το 10. Αυτοί οι επιπλέον περιορισμοί προστίθενται στα queries που εφαρμόζονται στην κάθε ιεραρχία. Ένα στιγμιότυπο αυτής της κλάσης κατασκευάζεται για κάθε ιεραρχία. Η κλάση αυτή επεκτείνει το JPanel, και έχει ως εξής:

Πεδία

- `GridBagLayout gridbag`
  Είναι ο τύπος της διάταξης των γραφικών στοιχείων στο panel.
- `GridBagConstraints constraints`
  Βοηθητική μεταβλητή για την τοποθέτηση των στοιχείων στο panel.
- `Vector cols`
  Ένα Vector που κρατάει σαν checkboxes τα ονόματα των στηλών του πίνακα της ιεραρχίας, στην οποία αντιστοιχεί το αντικείμενο της κλάσης αυτής.
- `Vector cb`
  Ένα Vector που κρατάει για κάθε στήλη το ComboBox με τις προτεινόμενες επιλογές (πχ >,<,= κλπ), ανάλογα με τον τύπο των δεδομένων της κάθε στήλης.
- `Vector text`
  Το Vector που κρατάει τα textfields στα οποία ο χρήστης δηλώνει την σύγκριση που θέλει να γίνει (πχ αν έχει επιλεγεί η σύγκριση '<' να δηλώσει μετά το '10').
- `Vector types`
  Το Vector που κρατάει με την σειρά τον τύπο των δεδομένων της κάθε στήλης του πίνακα που εξετάζουμε. Υπάρχει αντιστοίχιση με την σειρά των στηλών.

Μέθοδοι

- `public ExtraOptions(int tabid)`
  Η συνάρτηση κατασκευαστής της κλάσης. Φροντίζει για την αρχικοποίηση των μεταβλητών που περιγράψαμε, σύμφωνα με τον αριθμό της ιεραρχίας (και του πίνακα) tabid.
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Ορίζει την τοποθέτηση των γραφικών στοιχείων στο panel.
- `public void initPan()`
  Αρχικοποιεί τα ComboBoxes με τις επιλογές σύγκρισης, και τοποθετεί τα γραφικά στοιχεία στο panel.
- `public String getExtra()`
  Από τις επιλογές του χρήστη για τους περιορισμούς αυτούς, κατασκευάζει το String που θα προστεθεί στα queries.

6.2.4.8 public class GraphNode

Αυτή η κλάση είναι μια βοηθητική κλάση, για την γραφική απεικόνιση των κόμβων των δενδρικών δομών μέσω του JGraph. Ουσιαστικά, είναι μια κλάση που έχει ως πεδία της τον κόμβο, τον οποίο αντιπροσωπεύει, και τις συντεταγμένες στις οποίες θα εμφανιστεί. Αυτό γίνεται ώστε με κάποιον αλγόριθμο να υπολογίσουμε την θέση του κάθε κόμβου, την οποία θέση φυλάσσει μέσα του κάθε αντικείμενο αυτής της κλάσης, και μετά τα παίρνουμε όλα και τα εμφανίζουμε. Αυτό γίνεται γιατί θα ήταν αρκετά πιο πολύπλοκο να διορθώσουμε τις θέσεις των κόμβων αφού ζωγραφιστούν στο panel. Αυτή η κλάση έχει υλοποιηθεί ως εξής:

Πεδία

- `public TreeNode node`
  Είναι ο κόμβος της δενδρικής δομής, τον οποίο αναπαριστά.
- `public GraphNode parent`
  Είναι ο GraphNode που αναπαριστά τον πατέρα του κόμβου αυτού του στιγμιότυπου GraphNode.
- `public int width`
  Το πλάτος, σε pixels, του κόμβου.
- `public int xpos`
  Η θέση x του κόμβου.
- `public int ypos`
  Η θέση y του κόμβου.

Μέθοδοι

- `GraphNode(TreeNode n, GraphNode p, int x, int y)`
  Η συνάρτηση κατασκευαστής της κλάσης. Αρχικοποιεί τις μεταβλητές σύμφωνα με τις εισόδους.
- `public void setNode(TreeNode n)`
- `public void setParent(GraphNode p)`
- `public void setWidth(int w)`
- `public void setXpos(int x)`
- `public void setYpos(int y)`
- `public TreeNode getNode()`
- `public GraphNode getParent()`
- `public int getWidth()`
- `public int getXpos()`
- `public int getYpos()`
  Οι συναρτήσεις που θέτουν και επιστρέφουν τα πεδία της κλάσης.
- `public int calcWidth()`

Η συνάρτηση υπολογισμού του πλάτους του κόμβου, ανάλογα με το μήκος του ονόματος που θα φέρει.

6.2.4.9 public class NewDimDep

Αυτή η κλάση χειρίζεται την προσθήκη εξαρτήσεων στον γράφο εξαρτήσεων. Χρησιμοποιείται στην κλάση EditDep, όπως είδαμε και παραπάνω, και λειτουργεί τόσο για την προσθήκη νέων εξαρτώμενων κόμβων, όσο και για την προσθήκη νέων εξαρτήσεων. Όπως και κάθε κλάση που υλοποιεί γραφικά, έτσι και αυτή επεκτείνει μια κλάση γραφικών, την JDialog, υλοποιεί τα interfaces ActionListener και WindowListener, και έχει ως εξής:

Πεδία

- `static JPanel pan`
  Ένα panel για την υποδοχή γραφικών στοιχείων.
- `static JButton cancel`
- `static JButton ok`
  Δύο κουμπιά, ένα για την ακύρωση και ένα για την επικύρωση των αλλαγών.
- `static JLabel info1`
- `static JLabel pick`
- `static JLabel empty`
  Ετικέτες με πληροφορίες.
- `static JComboBox d`
  Ένα ComboBox για την επιλογή της διάστασης.
- `static int c`
  Μια μεταβλητή για να ξεχωρίζουμε αν πρόκειται για την προσθήκη νέου εξαρτώμενου κόμβου, ή για την προσθήκη νέας εξάρτησης.
- `static String sdep`
  Το όνομα της διάστασης της νέας εξάρτησης.

Μέθοδοι

- `public NewDimDep()`
  Συνάρτηση κατασκευαστής της κλάσης. Τοποθέτηση των γραφικών στοιχείων στο panel.
- `public static void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Ορίζει την τοποθέτηση των γραφικών στοιχείων στο panel.
- `public void refreshDims(Vector v,int cas)`
  Ανανεώνει τις επιλογές για την επιλογή διάστασης. Αν cas=0, πρόκειται για επιλογή νέου εξαρτώμενου κόμβου, αλλιώς αν cas=1 πρόκειται για νέα εξάρτηση.
- `public void actionPerformed(ActionEvent evt)`
  Χειρισμός των γεγονότων που προκύπτουν από το πάτημα των κουμπιών. Ενεργοποιούνται οι ενέργειες προσθήκης νέων εξαρτήσεων στον γράφο εξαρτήσεων.
- `public void setSdep(String s)`
  Θέτει το πεδίο sdep ίσο με s.
- `public void windowClosing(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.4.10 public class NormTree

Πρόκειται για την κλάση που συμπληρώνει την κλάση DependenceGraph, όσον αφορά τους χειρισμούς γύρω από το Μονοπάτι και τις διάφορες συναρτήσεις που απαιτούνται για την υλοποίηση όλων των απαιτήσεων της διαχείρισης του καθολικού κανονικοποιημένου δένδρου. Αντικείμενο αυτής της κλάσης υπάρχει ως πεδίο στην κλάση DependenceGraph, όπως είδαμε και παραπάνω. Το πεδίο path που βλέπουμε εδώ, είναι το 'Μονοπάτι' που αναφέραμε στην υποενότητα 6.2.4.3 Η κλάση αυτή έχει ως εξής:

Πεδία

- `private Tree path`
  Είναι το μοναδικό πεδίο της κλάσης. Είναι μία δομή δένδρου, εκφυλισμένη όμως σε μονοπάτι.

Μέθοδοι

- `NormTree()`
- `NormTree(NormTree n)`
  Δύο συναρτήσεις κατασκευαστές. Η δεύτερη κατασκευάζει το NormTree ίδιο με την είσοδο n.
- `public void setPath(Tree p)`
- `public Tree getPath()`
  Απλές συναρτήσεις που θέτουν και επιστρέφουν το path.
- `public void addInPath(TreeNode n)`
  Απλή συνάρτηση που προσθέτει ένα αντίγραφο του n στο τέλος του path.
- `public void delLastNode()`
  Σβήνει τον τελευταίο κόμβο από το path.
- `public TreeNode findLastNode(TreeNode n)`
- `public TreeNode findLastNode()`
  Επιστρέφουν τον τελευταίο κόμβο του path. Η πρώτη συνάρτηση επιστρέφει το πρώτο φύλλο του TreeNode n.
- `public void findPaths(Hashtable dg, Vector v)`
- `public void findPaths(NormTree t,Hashtable dg, Vector v)`
  Οι δύο αυτές συναρτήσεις καλούν την επόμενη, και είναι αυτές που διερευνούν όλους τους πιθανούς συνδυασμούς ανακατανομής των διαστάσεων μέσα στο Μονοπάτι. Η πρώτη το κάνει με δεδομένο μονοπάτι το path της κλάσης, ενώ η δεύτερη χρησιμοποιώντας την είσοδο t. Το dg είναι ο γράφος εξαρτήσεων, και το v είναι το Vector που θα περιέχει όλα τα αποτελέσματα.
- `public void explorePaths(NormTree stand, NormTree inPath, Vector curPaths, Hashtable dg)`
  Αυτή η συνάρτηση, επειδή καλεί αναδρομικά τον εαυτό της, παίρνει ως είσοδο stand το Μονοπάτι που αυτή έχει κατασκευάσει, το inPath που είναι το αρχικό Μονοπάτι, τον Vector curPaths, που είναι τα αποτελέσματα, και τον γράφο εξαρτήσεων dg.
- `public boolean isValidPath(NormTree t, Hashtable dg)`
  Ελέγχει αν το Μονοπάτι t είναι σύμφωνο με τους περιορισμούς του γράφου εξαρτήσεων dg.
- `public boolean isValidPath(Hashtable dg)`
  Ακριβώς ότι και η προηγούμενη, χρησιμοποιώντας το πεδίο path.
- `public void showPath()`
- `public String getStringPath()`
- `public void showPathComplete()`
- `public String getPathComplete()`
  Αυτές οι συναρτήσεις εμφανίζουν το μονοπάτι. Οι μεν showPath() το τυπώνουν στην έξοδο, και οι getStringPath() το επιστρέφουν σαν String. Οι συναρτήσεις που έχουν και την λέξη Complete σημαίνει ότι επιστρέφουν και το όνομα της διάστασης, στην οποία ανήκει ο κάθε κόμβος.
- `public int getPathSize()`
  Επιστρέφει το βάθος του Μονοπατιού.

- `public boolean hasNode(String s)`
  Ελέγχει αν το path περιέχει κόμβο με τοπικό όνομα ίσο με το s.
- `public boolean hasNodePub(String s)`
  Ελέγχει αν το path περιέχει κόμβο με γενικό όνομα ίσο με το s.
- `public boolean checkPath(Vector hierarchies)`
  Αυτή η συνάρτηση καλείται όταν γίνει η επιλογή νέας διάστασης για την επέκταση του Μονοπατιού. Ελέγχει αν σε κάποια από τις ιεραρχίες που υπάρχουν στην είσοδο hierarchies, υπάρχει κόμβος της διάστασης που μόλις επιλέχθηκε, που να δημιουργεί μονοπάτι με τις μέχρι τώρα επιλογές. Με άλλα λόγια, ελέγχει αν για την επιλογή της νέας διάστασης πρέπει να μπει τριγωνάκι (Δ).
- `public void findLeaves(TreeNode start, Vector leaves)`
  Επιστρέφει στο Vector leaves όλα τα φύλλα ενός TreeNode start.
- `public boolean existsPath(TreeNode tn)`
  Ελέγχει αν στους προγόνους του TreeNode tn υπάρχουν όλοι οι κόμβοι του Μονοπατιού, με οποιαδήποτε σειρά.
- `public boolean hasAncestor(TreeNode tn, String s)`
  Ελέγχει αν το TreeNode tn έχει πρόγονο με γενικό όνομα το s.
- `public void cleanNone()`
  Σβήνει από το path τους τριγωνικούς κόμβους (για να γίνει η αποτίμηση).
- `public boolean samePath(Tree t)`
  Ελέγχει αν ένα (μονοπάτι) Tree t είναι ίδιο με το πεδίο path.
- `public boolean pathIncluded(NormTree in)`
  Ελέγχει αν το μονοπάτι in περιέχεται με κάποιον τρόπο στο πεδίο path, με οποιαδήποτε σειρά.
- `public void clearIncluded(Vector v)`
  Από το Vector v, που περιέχει μονοπάτια, καθαρίζει τα στοιχεία που περιέχονται ήδη σε κάποιο άλλο στοιχείο του v.
- `public Vector getNones()`
  Επιστρέφει σε ένα Vector τα ονόματα των διαστάσεων που προκάλεσαν την εισαγωγή τριγωνικού κόμβου (Δ) στο Μονοπάτι.

6.2.4.11 public class NTGraph

Η κλάση αυτή εξυπηρετεί την γραφική απεικόνιση και κατασκευή του Μονοπατιού. Υλοποιεί τις ενέργειες γραφικών, όπως την απεικόνιση του επιλεγμένου κόμβου, ή της διάστασης, των κόμβων Δ, και κρατάει σε μια δομή Vector από Vectors με την σειρά, τις διαστάσεις του Μονοπατιού, και τους κόμβους που επέλεξε ο χρήστης, πέραν από την κλάση NormTree, προκειμένου να γίνεται πιο εύκολα η απεικόνιση και ο έλεγχος, αν ο χρήστης επέλεξε σωστό κόμβο (που να ανήκει στην τελευταία διάσταση, δηλαδή). Η κλάση αυτή επεκτείνει την JPanel, και έχει ως εξής:

Πεδία

- `public JGraphModelAdapter m_jgAdapter;`
  Αυτό είναι η μεταβλητή για τον χειρισμό του γράφου μέσω του JGraph.
- `public final Color DEFAULT_BG_COLOR`
  Μια σταθερά που φέρει την τιμή ενός χρώματος.
- `ListenableGraph g`
  Η μεταβλητή ενός δυναμικού γράφου.
- `JGraph jgraph`
  Η μεταβλητή του γράφου.
- `int MIN_DIST`
- `int VDIST`
- `int DIM_OFFSET`

Μεταβλητές που κρατούν την ελάχιστη οριζόντια απόσταση, την ελάχιστη κατακόρυφη απόσταση (μεταξύ των επιπέδων) και την απόσταση από τις ετικέτες που κρατούν τα ονόματα των διαστάσεων, αντίστοιχα.

- `Vector levels`
  Αυτό το Vector περιέχει άλλα Vectors. Κάθε ένα από τα εσωτερικά Vectors αναπαριστά την επιλογή μιας διάστασης, με όλους τους διαθέσιμους κόμβους. Στην πρώτη θέση καθενός από αυτά τα εσωτερικά Vectors βρίσκεται ο επιλεγόμενος κόμβος. Αν δεν έχει γίνει επιλογή, τότε βρίσκονται τοποθετημένοι αλφαβητικά.
- `String chNode`
  Κρατάει το όνομα του τελευταίου επιλεγμένου κόμβου.
- `Vector vdims`
  Κρατάει με την σειρά τα ονόματα των επιλεγόμενων διαστάσεων (την σειρά με την οποία βρίσκονται τα εσωτερικά Vectors στο Vector levels).

Μέθοδοι

- `public NTGraph()`
  Συνάρτηση κατασκευαστής της κλάσης. Αρχικοποίηση μεταβλητών.
- `public void positionVertexAt(Vector l, int size)`
- `public void positionVertexAt(Vector l, int size, String s, int m)`
  Αυτές οι δύο συναρτήσεις τοποθετούν τους κόμβους το γράφου εκεί που πρέπει. Η πρώτη παίρνει ως είσοδο ένα Vector l που περιέχει τους νέους κόμβους της επιλεγμένης διάστασης, και την σειρά που επιλέχθηκε η διάσταση αυτή. Καλείται όταν παρουσιάζουμε τους κόμβους της νέας διάστασης. Η δεύτερη συνάρτηση παίρνει επιπλέον δύο ορίσματα: το όνομα του επιλεγόμενου κόμβου, και μια ακέραια τιμή 0 ή 1. Καλείται με το 0 για την επιλογή του νέου κόμβου, τον οποίο και φέρνει στην πρώτη θέση των κόμβων της διάστασής του, και με 1 όταν πρόκειται για την ακύρωση της επιλογής κόμβου. Επίσης, αν η δεύτερη κληθεί με s=null και m=0, τότε πρόκειται για την ακύρωση της επιλογής της τελευταίας διάστασης, οπότε ο γράφος σχηματίζεται όπως ήταν πριν.
- `public int calcWidth(String s)`
  Συνάρτηση υπολογισμού του πλάτους του κάθε κόμβου, σύμφωνα με το μήκος του ονόματός του s.
- `public void addLevel(String d)`
  Η συνάρτηση που υλοποιεί την προσθήκη ενός νέου επιπέδου στο Μονοπάτι, δηλαδή μετά την επιλογή διάστασης, και πριν την επιλογή του κόμβου.
- `public void chooseNode(String s)`
  Η συνάρτηση που υλοποιεί την επιλογή του κόμβου.
- `public void refreshGraph(String temp,int m)`
  Αυτή η συνάρτηση προκαλεί την ανανέωση του γραφήματος, σύμφωνα με τα νέα δεδομένα. Καλεί μια από τις δύο positionvertexAt, όπως περιγράψαμε και παραπάνω.
- `public boolean isValid(String s)`
  Ελέγχει αν η επιλογή του κόμβου ήταν έγκυρη, δηλαδή αν ανήκε στο τελευταίο επίπεδο.
- `public boolean checkNewDim()`
  Ελέγχει αν η νέα διάσταση πρέπει να προκαλέσει την είσοδο κόμβου Δ.
- `public void cancelLastNode()`
  Ακυρώνει την τελευταία επιλογή κόμβου.
- `public void cancelLastDim()`
  Ακυρώνει την τελευταία επιλογή διάστασης.
- `public void restart(int s)`
  Ξεκινάει τον γράφο από την αρχή, σβήνοντας όλες τις επιλογές.
- `public void resetPositions()`
  Αν χρειαστεί να αλλάξει κάποια από τις αρχικές τιμές των ελάχιστων αποστάσεων, προκαλεί την ανανέωση της παρουσίασης του γράφου.

6.2.4.12 public class Qmanage

Αυτή η κλάση είναι και η εκτελέσιμη, καθώς σε αυτήν υλοποιείται η main. Επίσης, σε αυτήν δηλώνονται αντικείμενα των ενεργών γραφικών παραθύρων, των εσωτερικών δομών, και

αρκετών συναρτήσεων που επιδρούν άμεσα στις δομές αυτές. Αυτό, γιατί οι περισσότερες κλάσεις δεν έχουν πρόσβαση σε όλες τις δομές και τις συναρτήσεις ταυτόχρονα, ενώ από αυτή την κλάση μπορούν να ελεγχθούν όλα. Δηλαδή, μπορεί στην κλάση που υλοποιεί το παράθυρο διαχείρισης διαστάσεων, το πάτημα ενός κουμπιού να ενεργοποιεί την μεταφορά ενός κόμβου από την μια διάσταση σε μια άλλη, αλλά σε αυτή την κλάση υλοποιείται αυτή η ενέργεια. Περισσότερες λεπτομέρειες ακολουθούν αμέσως παρακάτω:

Πεδία

- `static Vector hierarchies`
  Ένα Vector που περιέχει τις ιεραρχίες.
- `static Tree [] dims`
  Ένας πίνακας που περιέχει τα δένδρα των διαστάσεων.
- `static DependenceGraph dg`
  Αντικείμενο της κλάσης DependenceGraph.
- `static Tree fre`
  Ένα βοηθητικό δένδρο, που κρατάει τους 'ορφανούς' κόμβους.
- `static ResultRecords rr`
  Αντικείμενο της κλάσης ResultRecords.
- `static String chosenNode`
  String που κρατάει το όνομα του τελευταίου επιλεγμένου κόμβου.
- `static EditDim ed`
  Αντικείμενο της κλάσης EditDim.
- `static EditDep edep`
  Αντικείμενο της κλάσης EditDep.
- `static ViewTrees vt`
  Αντικείμενο της κλάσης ViewTrees.
- `static QueryPath qp`
  Αντικείμενο της κλάσης QueryPath.
- `static Vector activeH`
  Ένα Vector που κρατάει τις ενεργές ιεραρχίες (ιεραρχίες στις οποίες θα αποτιμηθεί το query).
- `static Vector queryArch`
  Ένα Vector που κρατάει τα μονοπάτια που αποτιμήθηκαν στην κάθε ιεραρχία.
- `static Tree t`
  Βοηθητικό αντικείμενο της κλάσης Tree.
- `static DummyFrame df`
  Ένα αντικείμενο της βοηθητικής κλάσης DummyFrame.
- `static String[] dirHs`
  Πίνακας που περιέχει τα ονόματα των αρχείων που βρίσκονται στον φάκελο "hierarchies".
- `static int[] Hids`
  Πίνακας που περιέχει τους αριθμούς των ιεραρχιών με αύξουσα σειρά, οι οποίοι δεν είναι απαραίτητο να είναι συνεχόμενοι.

Μέθοδοι

- `public static void main(String[] args)`
  Η βασική συνάρτηση που εκτελείται με την έναρξη του προγράμματος. Μέσα από αυτήν καλούνται οι αρχικοποιήσεις των περισσοτέρων δομών και αντικειμένων των υπολοίπων κλάσεων.
- `public static Tree[] initDims(int k)`
  Επιστρέφει έναν πίνακα που περιέχει τα δένδρα των διαστάσεων. Η είσοδος k είναι ο αριθμός των ιεραρχιών που συμμετέχουν στο σύστημα.
- `public static void initIds(int k)`
  Αρχικοποιεί τον πίνακα Hids, που περιέχει τα ονόματα των ιεραρχιών, παίρνοντάς τα από τα αρχεία στα οποία δηλώνονται (πχ. το H15.xml θα δώσει το όνομα H15).
- `public static Vector initDG(int k, Tree[] dims)`

Επιστρέφει σε ένα Vector όλα τα ονόματα των ριζών των δένδρων διαστάσεων, που βρίσκονται στον πίνακα εισόδου dims. Το k είναι το μήκος του πίνακα.

- `public static Vector getDimNodes()`
  Επιστρέφει τα ονόματα των δένδρων διαστάσεων (τα ονόματα των ριζών τους) με αλφαβητική σειρά.
- `public static String[] sortString(String[] s)`
  Ταξινομεί έναν πίνακα από String s με αλφαβητική σειρά.
- `public static Vector getDimsChildren(String s)`
  Επιστρέφει τα πεδία της συγχωνευμένης διάστασης με όνομα s.
- `public static void renameDims(String old, String n)`
  Μετονομάζει το όνομα της διάστασης από old σε n, σε όλες τις δομές που υπάρχει αυτή.
- `public static void delDim(String dim, int pos)`
  Σβήνει την διάσταση με όνομα dim, που βρίσκεται στην θέση pos στο δένδρο συγχωνευμένων διαστάσεων.
- `public static void changeDim(String old, String n, Vector arr)`
  Μεταφέρει τους κόμβους που βρίσκονται στο Vector arr από την παλιά διάσταση old στην n, σε όλες τις δομές που υπάρχει κάποιος από τους κόμβους του arr.
- `public static void moveNode(String old, String n, Vector arr)`
- `public static void moveNode2(int k, String old, String n, Vector arr)`
  Υλοποιούν την μετακίνηση των κόμβων, και καλούνται κατάλληλα από την συνάρτηση changeDim. Η δεύτερη συνάρτηση διαφέρει από την πρώτη στο ότι η μεταφορά γίνεται για το τοπικό δένδρο διαστάσεων που βρίσκεται στην θέση k του πίνακα που τα περιέχει.
- `public static boolean testDim(String n, Vector arr)`
  Ελέγχει αν οι κόμβοι που υπάρχουν στο Vector arr σχηματίζουν μονοπάτι σε κάποια ιεραρχία, με τους κόμβους που ήδη υπάρχουν στην διάσταση με όνομα n. Αυτό γίνεται πάντα πριν την μεταφορά κόμβων.
- `public static void showResults (String nodeName, ResultSet rst, String path, int tabl)`
  Ενεργοποιεί την εμφάνιση αποτελεσμάτων της κλάσης ResultRecords, την οποία θα δούμε παρακάτω. Το nodeName είναι το όνομα που θα φανεί στο Tab του αποτελέσματος, το rst φέρει τα αποτελέσματα, το path είναι το μονοπάτι που μας οδήγησε στα αποτελέσματα και το tabl είναι ο αριθμός του πίνακα από τον οποίο πήραμε τα αποτελέσματα (“MOVIESi”). Αυτή η συνάρτηση είναι για την παρουσίαση εγγραφών που βρίσκονται στα φύλλα ενός δένδρου, και όχι για τα αποτελέσματα των queries που κάνει ο χρήστης.
- `public static void end()`
  Τερματίζει την εφαρμογή.
- `public static void setChosenNode(String s)`
  Θέτει την μεταβλητή chosenNode ίση με το s.
- `public static void createQueries(String[] s)`
  Αυτή είναι μια από τις σημαντικότερες συναρτήσεις, όσον αφορά την λειτουργία της κατασκευής των Μονοπατιών. Εδώ συγχρονίζονται και καλούνται όλες οι συναρτήσεις που έχουν να κάνουν με αυτό το θέμα, και παρουσιάστηκαν σε προηγούμενες κλάσεις, από την συμπλήρωση των Μονοπατιών, μέχρι την κατασκευή των queries, την εκτέλεση αυτών και την κλήση των καταλλήλων συναρτήσεων που θα παρουσιάσουν τα αποτελέσματα. Η παράμετρος s είναι ένας πίνακας που περιέχει για κάθε ιεραρχία το String με τους επιπλέον περιορισμούς που θα μπουν σε κάθε query, για την εκάστοτε ιεραρχία.
- `public static void execPath(Tree hi, Vector paths, Vector queries)`
  Αυτή η συνάρτηση παίρνει τα μονοπάτια paths που έχουν προκύψει για την ιεραρχία hi, και όποια από αυτά εφαρμόζονται στην ιεραρχία αυτή, τότε παίρνει τα queries των φύλλων τους και τα βάζει στο Vector queries.
- `public static void extQueryRestart()`
  Καλείται στην περίπτωση που πατηθεί από κάποιο παράθυρο το κουμπί που αρχικοποιεί τις δομές, και επανεκκινεί το Μονοπάτι. Αυτό γίνεται από εδώ, γιατί μόνο αυτή η κλάση έχει πρόσβαση στην συγκεκριμένη συνάρτηση επανεκκίνησης του Μονοπατιού.
- `public static void resShow()`
  Φέρνει στην μπροστινή όψη το παράθυρο με τα αποτελέσματα.
- `public static void restart()`
  Εκτελεί τις ενέργειες επανεκκίνησης.

public static class DummyFrame

Αυτή η κλάση είναι εντελώς βοηθητική, και ανήκει στην Qmanage. Απλά εμφανίζει ένα μικρό παράθυρο με το μήνυμα “Restarting the System” ή “Initializing”, μέχρι να αρχικοποιηθούν οι δομές. Επεκτείνει την JFrame, και αποτελείται απλά από τα εξής:

Πεδία

- `JPanel pan`
- `JLabel lab`
  Απλά πεδία υποδοχής γραφικών στοιχείων και ετικέτας με ένα μήνυμα.

Μέθοδοι

- `public DummyFrame()`
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Απλές συναρτήσεις κατασκευαστή και τοποθέτησης των γραφικών στοιχείων.

6.2.4.13 public class QueryPath

Και αυτή η κλάση είναι από τις πιο σημαντικές, γιατί υλοποιεί την διαπροσωπεία χρήστη, με την οποία ο χρήστης αλληλεπιδρά για την κατασκευή του Μονοπατιού. Άρα εδώ εμφανίζεται δυναμικά το γράφημα, μέσω του οποίου ο χρήστης διαλέγει διαστάσεις και κόμβους. Επίσης, εμφανίζονται και τα πεδία του κάθε πίνακα, που περιέχει τις εγγραφές, και έτσι ο χρήστης μπορεί να βάλει τους περαιτέρω περιορισμούς. Η κλάση αυτή επεκτείνει την JFrame, και υλοποιεί τα interfaces ActionListener και WindowsListener, και έχει ως εξής:

Πεδία

- `JPanel pan`
- `JPanel pan1`
- `JPanel pan2`
- `JPanel pan3`
- `JPanel panel`
  Δήλωση μεταβλητών JPanel, που θα υποδεχθούν τα γραφικά στοιχεία.
- `JButton ok1`
- `JButton ok2`
- `JButton back`
- `JButton exec`
  Δήλωση μεταβλητών κουμπιών, που ενεργοποιούν την αποδοχή της διάστασης, την αποδοχή του κόμβου, την αναίρεση της τελευταίας στην σειρά επιλογής, και την εκτέλεση του query, αντίστοιχα.
- `JLabel info`
- `JLabel info1`
- `JLabel info2`
- `JLabel empty`
  Ετικέτες με κάποια μηνύματα.
- `JComboBox cb`
  Το ComboBox που περιέχει τις προς επιλογήν διαστάσεις.
- `NTGraph ntg`
  Μια μεταβλητή της κλάσης NTGraph, που είδαμε και πιο πάνω. Είναι η αναπαράσταση του Μονοπατιού.
- `ExtraOptions eo[]`

Ένας πίνακας που περιέχει τους περαιτέρω περιορισμούς για τους πίνακες των δεδομένων, σε αντικείμενα ExtraOptions, που είδαμε παραπάνω.

- `boolean endDims`
  Μια τιμή Boolean, που είναι TRUE αν δεν υπάρχουν άλλες προς επιλογή διαστάσεις, αλλιώς είναι FALSE.
- `JButton editDims`
- `JButton editDep`
- `JButton res`
- `JButton gnt`
- `JButton grep`
- `JButton quit`
- `JButton restart`
- `JButton sysrestart`
- `JButton shPaths`
- `JToolBar tbr`
  Η δήλωση του ToolBar και των κουμπιών που θα μπουν σε αυτό. Τα κουμπιά που δεν έχουμε δει είναι το restart, που επανεκκινεί το query (Μονοπάτι) και το shPaths, που μας δείχνει σε ένα παράθυρο τα μονοπάτια που εφαρμόστηκαν σε κάθε ιεραρχία. Η επανεκκίνηση του συστήματος γίνεται με το κουμπί sysrestart.
- `JComboBox eoh`
  Το ComboBox που περιέχει τα ονόματα των ιεραρχιών, και που καθορίζει ποιανού πίνακα τα πεδία θα δούμε σε ένα panel, ώστε να μπορούμε να ορίσουμε τους περαιτέρω περιορισμούς.
- `JScrollPane temps`
  Κρατάει την μπάρα κύλισης των πεδίων του πίνακα, που έχει επιλεχθεί να φαίνεται.
- `JScrollPane[] scroll2`
  Εδώ υπάρχουν όλες οι μπάρες κύλισης για τα πεδία όλων των πινάκων. Το πεδίο temps κρατάει την ενεργό μπάρα κύλισης.
- `GridBagLayout gridbag`
- `GridBagConstraints constraints`
  Βοηθητικά πεδία για την τοποθέτηση των γραφικών στοιχείων μέσα στα panels.

Μέθοδοι

- `public QueryPath()`
  Η συνάρτηση κατασκευαστής της κλάσης, η οποία αρχικοποιεί κάποιες μεταβλητές και τοποθετεί τα γραφικά στοιχεία στην θέση που πρέπει μέσα στους υποδοχείς.
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Η βοηθητική συνάρτηση τοποθέτησης των γραφικών στοιχείων.
- `public void initDims()`
  Εμφανίζει με αλφαβητική σειρά τις διαθέσιμες προς επιλογή διαστάσεις. Ανανεώνεται κάθε φορά που επιλέγεται μια διάσταση.
- `public void changeState()`
  Θεωρούμε ότι η διαδικασία κατασκευής του Μονοπατιού γίνεται σε δύο στάδια καταστάσεων. Πρώτον, είναι η επιλογή της διάστασης που θέλουμε να επεκτείνουμε, και δεύτερον, η επιλογή κάποιου κόμβου από την επιλεγμένη διάσταση. Προκειμένου να μην επιτρέψουμε λάθη, πρέπει να απενεργοποιούμε και να ενεργοποιούμε κάποια κουμπιά από την μια κατάσταση στην άλλη. Αυτή η συνάρτηση φροντίζει για την σωστή εναλλαγή των καταστάσεων.
- `public void actionPerformed(ActionEvent evt)`
  Η συνάρτηση χειρισμού των γεγονότων που εγείρονται όταν ο χρήστης πατήσει κάποιο κουμπί. Από εδώ καλούνται οι συναρτήσεις που εκτελούν τις διάφορες ενέργειες.
- `public void chooseH(int pos)`
  Αυτή η συνάρτηση φροντίζει για την ανανέωση του panel που δείχνει τα πεδία κάποιου πίνακα εγγραφών, όταν ο χρήστης ζητήσει να δει τα πεδία κάποιου άλλου πίνακα, με τον αριθμό pos.
- `public void restartQuery(int state)`
  Επανεκκινεί την διαδικασία κατασκευής του Μονοπατιού, περνώντας σαν παράμετρο την τωρινή κατάσταση.
- `public void showAllPaths()`

Εμφανίζει ένα μικρό παράθυρο με τα μονοπάτια που αποτιμήθηκαν στις ενεργές ιεραρχίες.

- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`

  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.
- `public void extRestart()`

  Αυτή η συνάρτηση καλείται όταν εξαιτίας κάποιου άλλου γεγονότος, όπως η αλλαγή κάποιων εξαρτήσεων, πρέπει να προκαλέσει την επανεκκίνηση της κατασκευής του Μονοπατιού.

6.2.4.14 public class RenameDims

Αυτή η κλάση υλοποιεί την διαπροσωπεία χρήστη που διαχειρίζεται την μετονομασία μιας διάστασης. Πρόκειται για ένα απλό παράθυρο, στο οποίο ο χρήστης καλείται να επιλέξει μια διάσταση και να συμπληρώσει σε ένα πεδίο το καινούργιο όνομα που θέλει να της δώσει. Η κλάση αυτή επεκτείνει την JDialog, υλοποιεί τα interfaces ActionListener και WindowListener, και έχει ως εξής:

Πεδία

- `static JPanel pan`

  Η μεταβλητή του υποδοχέα των γραφικών στοιχείων.
- `static String txt`

  Εδώ θα φυλαχτεί το νέο όνομα της επιλεγμένης διάστασης.
- `static JButton cancel`
- `static JButton ok`

  Δυο κουμπιά για την αναίρεση και την επικύρωση της επιλογής.
- `static JLabel pick`
- `static JLabel to`
- `static JLabel empty`

  Ετικέτες που δείχνουν κάποια πληροφορία.
- `static JTextField text`

  Σε αυτό το TextField συμπληρώνεται το νέο όνομα.
- `static JComboBox d`

  Αυτό είναι το ComboBox που περιέχει τα ονόματα των διαστάσεων προς επιλογή.

Μέθοδοι

- `public RenameDims()`

  Η συνάρτηση κατασκευαστής της κλάσης, όπου αρχικοποιούνται κάποιες μεταβλητές και τα γραφικά στοιχεία τοποθετούνται στις θέσεις που πρέπει.
- `public static void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`

  Όπως και σε κάθε κλάση γραφικών, αυτή είναι η βοηθητική συνάρτηση για την τοποθέτηση των γραφικών στοιχείων.
- `public static void refreshDims(Vector v)`

  Η συνάρτηση που ανανεώνει τα ονόματα για τις διαστάσεις προς επιλογή, οι οποίες περιέχονται στο Vector v.
- `public  void actionPerformed(ActionEvent evt)`

  Η συνάρτηση χειρισμού των γεγονότων που εγείρονται με το πάτημα των κουμπιών.
- `public static boolean isValid(String s)`

  Επιστρέφει FALSE αν το όνομα s που δόθηκε είναι κενό ή αρχίζει με κενό.
- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`

- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.4.15 public class ResultRecords

Αυτή η κλάση είναι η υλοποίηση της διαπροσωπείας χρήστη για την απεικόνιση των αποτελεσμάτων, οποιουδήποτε query, είτε αυτό πρόκειται για την εφαρμογή ερώτησης μονοπατιού στις ιεραρχίες, είτε πρόκειται για το query που επιστρέφει τις εγγραφές ενός φύλλου σε κάποια ιεραρχία. Σε ένα μεγάλο panel, εμφανίζονται σε μορφή tabs τα αποτελέσματα μιας αίτησης κάθε φορά. Έτσι μπορεί να γίνει και σύγκριση μεταξύ των αποτελεσμάτων. Η κλάση αυτή επεκτείνει την JFrame, υλοποιεί τα interfaces ActionListener και WindowListener, και έχει ως εξής:

Πεδία

- `JPanel pan`
- `JPanel pan1`
  Τα panels που υποδέχονται τα γραφικά στοιχεία.
- `JButton close`
  Το κουμπί με το οποίο σβήνουμε το εν ενεργεία tab.
- `JTabbedPane tb`
  Ο υποδοχέας των tabs.
- `JButton editDims`
- `JButton editDep`
- `JButton res`
- `JButton gnt`
- `JButton grep`
- `JButton restart`
- `JButton quit`
- `JToolBar tbr`
  Το ToolBar και τα κουμπιά που μπαίνουν σε αυτό.

Μέθοδοι

- `public ResultRecords()`
  Η συνάρτηση κατασκευαστής της κλάσης, στην οποία αρχικοποιούνται κάποιες μεταβλητές, και τοποθετούνται τα γραφικά στοιχεία στην θέση που πρέπει.
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Η βοηθητική συνάρτηση τοποθέτησης των γραφικών στοιχείων.
- `public void actionPerformed(ActionEvent evt)`
  Η συνάρτηση χειρισμού των γεγονότων που εγείρονται κατά το πάτημα των κουμπιών.
- `public void addInTab(String name, ResultSet rst, String s, int table)`
  Προσθέτει ένα tab στο panel, με όνομα το name, και περιεχόμενο ένα αντικείμενο TabItem, που θα δούμε παρακάτω, το οποίο κατασκευάζεται από τις υπόλοιπες τρεις παραμέτρους.
- `public void addInTab(String name, TabItem ti)`
  Προσθέτει στο panel ένα tab με όνομα το name και περιεχόμενο την παράμετρο ti.
- `public void refresh()`
  Αρχικοποιεί το παράθυρο, σβήνοντας όλα τα tabs.
- `public void windowClosing(WindowEvent evt)`
- `public void windowActivated(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`

- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.4.16 public class SAXparse

Αυτή η κλάση υλοποιεί την διάσχιση των αρχείων XML, και αποκωδικοποιεί την πληροφορία που αυτά κρύβουν, σύμφωνα με τις δικές μας ανάγκες. Υλοποιεί το interface DefaultHandler, και εμείς, με τις συναρτήσεις που γράφουμε, ορίζουμε με ποιόν τρόπο να χειριστεί τα αρχεία και πως να κρατήσει την πληροφορία που θέλουμε. Πιο συγκεκριμένα, η κλάση αυτή έχει ως εξής:

Πεδία

- `private Stack stack`
  Η στοίβα, στην οποία φυλάμε τις αντιστοιχίες των οντοτήτων του αρχείου XML που διαβάζουμε, με την σειρά με την οποία τα διαβάζουμε. Αυτές οι αντιστοιχίες είναι απλοί ακέραιοι αριθμοί. Αυτό υλοποιεί ένα είδος κατά βάθους διάσχισης.
- `private long id`
  Είναι ο ακέραιος που ταυτίζεται με κάθε οντότητα, και ο οποίος μόνο αυξάνει.
- `private Tree t`
  Το δένδρο το οποίο κατασκευάζεται σύμφωνα με την δομή του αρχείου.

Μέθοδοι

- `public SAXparse()`
  Η συνάρτηση κατασκευαστής της κλάσης, στην οποία αρχικοποιούνται τα πεδία της κλάσης.
- `public Tree list(String file) throws Exception`
  Η συνάρτηση που εκκινεί την διαδικασία ανάλυσης του αρχείου με όνομα file.
- `public Tree list2(String file) throws Exception`
  Η συνάρτηση που εκκινεί την διαδικασία ανάλυσης του αρχείου με όνομα file2, και η οποία χρησιμοποιείται μόνο για τα αρχεία των διαστάσεων, καθώς αυτά έχουν ένα συγκεκριμένο DTD.
- `public Tree getTree()`
  Επιστρέφει το δένδρο που κατασκευάστηκε.

Πέρα από αυτές τις μεθόδους, υλοποιούνται και δύο κλάσεις που αναλαμβάνουν την ανάλυση των XML αρχείων. Βασικά και οι δύο κλάσεις κάνουν τα ίδια πράγματα, αλλά επειδή τα αρχεία των διαστάσεων ακολουθούν ένα DTD, ενώ αυτά των ιεραρχιών κανένα, έπρεπε να έχουμε ξεχωριστή ανάλυση για τα δύο διαφορετικά αυτά είδη αρχείων. Αυτές οι κλάσεις δεν έχουν πεδία, αλλά απλά υλοποιούν με διαφορετικό τρόπο τις συναρτήσεις ανάλυσης των αρχείων.

class ElemHandler1 extends DefaultHandler (για τα απλά XML αρχεία)

Μέθοδοι

- `public void startElement(String nsURI, String strippedName, String tagName, Attributes attributes) throws SAXException`
  Ενεργοποιείται όταν ανοίγει ένα tag στο αρχείο XML, και προσθέτει στο δένδρο έναν κόμβο με το όνομα της οντότητας του tag, στην κατάλληλη θέση (έχοντας σαν πατέρα το

αμέσως προηγούμενο tag που δεν έχει κλείσει). Γι'αυτό φροντίζει η στοίβα που έχουμε υλοποιήσει, και οι αριθμοί ταυτοποίησης των οντοτήτων του αρχείου XML.

- `public void characters(char[] text, int first, int length)`
  Ενεργοποιείται όταν ανάμεσα στο άνοιγμα και το κλείσιμο ενός tag υπάρχει κείμενο. Χρησιμοποιείται για την αποθήκευση των queries που βρίσκονται στα φύλλα των ιεραρχιών σε τέτοια μορφή.
- `public void endElement(String nsURI, String strippedName, String tagName) throws SAXException`
  Ενεργοποιείται όταν κλείσει ένα tag, και φροντίζει να βγάλει από την στοίβα τον αριθμό ταυτοποίησης του.
- `public void addInTree(String tagName, long id) throws SAXException`
  Φροντίζει για την σωστή τοποθέτηση του τρεχούμενο tag σαν κόμβο TreeNode στο δένδρο.
- `public void addText(String text, long id)`
  Προσθέτει την πληροφορία του κειμένου που βρίσκεται ανάμεσα στο άνοιγμα και το κλείσιμο των tags στο πεδίου του αντίστοιχου κόμβου TreeNode.

class ElemHandler2 extends DefaultHandler (για τα XML αρχεία των διαστάσεων)

Σε αυτή την κλάση έχουμε ακριβώς τις ίδιες συναρτήσεις, με λίγο διαφορετική υλοποίηση στις συναρτήσεις startElement και endElement, οι οποίες δεν κάνουν τίποτα όταν βρουν tag με όνομα "dimNames" και "attributes", γιατί δεν θέλουμε αυτούς τους κόμβους στα δένδρα των διαστάσεων. Επίσης, επειδή δεν υπάρχει κείμενο ανάμεσα στα ανοίγματα και κλεισίματα των tags, δεν έχει υλοποιηθεί οι συναρτήσεις characters και addtext. Γι'αυτό απλά αναφέρουμε τις μεθόδους, χωρίς περισσότερα σχόλια.

Μέθοδοι

- `public void startElement(String nsURI, String strippedName, String tagName, Attributes attributes) throws SAXException`
- `public void endElement(String nsURI, String strippedName, String tagName) throws SAXException`
- `public void addInTree(String tagName, long id) throws SAXException`

6.2.4.17 public class TabItem

Αυτή η κλάση αποτελεί επέκταση του JPanel και είναι ουσιαστικά ένα panel που περιέχει τα αποτελέσματα των queries, δομημένα σε πίνακες. Όπως περιγράψαμε και στην κλάση ResultRecords, και λίγο στην Qmanage, έχουμε δύο ειδών αποτελέσματα: (a) αυτά που αφορούν queries που βρίσκονται στα φύλλα μιας ιεραρχίας, και τα οποία προέρχονται από μόνο έναν πίνακα της Βάσης Δεδομένων, και (b) αυτά που προέρχονται από την εφαρμογή του Μονοπατιού, που κατασκεύασε ο χρήστης, στις ενεργές ιεραρχίες, και τα οποία προέρχονται από διαφορετικούς πίνακες της Βάσης Δεδομένων. Η ιδέα είναι να παίρνουμε τα αποτελέσματα που αφορούν κάθε πίνακα ξεχωριστά, και να τα δείχνουμε σε διαφορετικούς πίνακες. Αυτό είναι απλό όταν πρόκειται για αποτελέσματα τύπου (a) που αναφέραμε παραπάνω, γιατί έχουμε μόνο έναν πίνακα. Τα αποτελέσματα τύπου (b) απαιτούν να εμφανιστούν περισσότεροι του ενός πίνακες. Έτσι έχουμε υλοποιήσει μια συνάρτηση, την addElem, που προσθέτει πίνακες στο panel που χειρίζεται η κλάση αυτή, και η οποία καλείται όσες φορές χρειαστεί, με τα κατάλληλα αποτελέσματα κάθε φορά. Η κλάση λοιπόν χειρίζεται

το panel στο οποίο εμφανίζονται οι πίνακες. Μέσα στην κλάση αυτή ορίζεται μια δεύτερη κλάση, η MyTableModel, η οποία χειρίζεται και κατασκευάζει τους πίνακες αποτελεσμάτων , ανάλογα με τα αποτελέσματα. Αρχίζουμε από την κλάση TabItem, η οποία έχει ως εξής:

Πεδία

- `String path`
  Είναι το μονοπάτι που παριστάνει, είτε το Μονοπάτι που έδωσε ο χρήστης για αποτίμηση, είτε το μονοπάτι που οδηγεί στο φύλλο, του οποίου τις εγγραφές ζήτησε να δει ο χρήστης.
- `JScrollPane scroll1`
  Είναι η μπάρα κύλισης που εφαρμόζεται στο όλο panel.
- `MyTableModel myModel`
  Είναι ένα αντικείμενο της κλάσης που θα περιγράψουμε παρακάτω.
- `GridBagLayout gridbag`
- `GridBagConstraints constraints`
  Βοηθητικά πεδία, για την τοποθέτηση των γραφικών στοιχείων.
- `boolean empty`
  Είναι TRUE όταν τα αποτελέσματα από κάποιον πίνακα της Βάσης Δεδομένων είναι κενά, και χρησιμοποιείται κατάλληλα για την εμφάνιση ενός μηνύματος.

Μέθοδοι

- `public TabItem(String s, ResultSet rst, int tab)`
  Είναι η συνάρτηση κατασκευαστής της κλάσης, στην οποία αρχικοποιούνται οι διάφορες μεταβλητές και τοποθετούνται τα γραφικά στοιχεία εκεί που πρέπει. Μέσα από αυτήν καλούνται οι συναρτήσεις που κατασκευάζουν τον πίνακα με τα αποτελέσματα. Αυτός ο κατασκευαστής είναι για την περίπτωση αποτελεσμάτων τύπου (a). Το s είναι το μονοπάτι που οδηγεί στο φύλλο που επιλέχθηκε, το rst περιέχει τα αποτελέσματα, και το tab είναι ο αριθμός του πίνακα στην Βάση Δεδομένων (ή, αλλιώς, ο αριθμός της ιεραρχίας).
- `public TabItem(String s)`
  Μια άλλη συνάρτηση κατασκευαστής, η οποία απλά θέτει μια ετικέτα. Καλείται όταν έχουμε αποτελέσματα τύπου (b), και επαφίεται στην επόμενη συνάρτηση για την κατασκευή των πινάκων αποτελεσμάτων που χρειάζεται. Το s είναι το Μονοπάτι του χρήστη.
- `public void addElem(String s, int pos, ResultSet rst, int tab)`
  Αυτή η συνάρτηση καλείται τόσες φορές, όσες χρειάζεται να κατασκευαστεί ένας πίνακας αποτελεσμάτων, δηλαδή όσες είναι οι ενεργές ιεραρχίες. Το s είναι εδώ το όνομα της ιεραρχίας της οποίας τα αποτελέσματα βρίσκονται στο rst, και είναι αυτή που εξετάζουμε τώρα, pos είναι η σειρά της ιεραρχίας στην παρουσίαση των αποτελεσμάτων, και tab ο αριθμός της (και αριθμός του πίνακά της στην Βάση Δεδομένων).
- `public void buildConstraints(GridBagConstraints gbc, int gx,` int gy, int `gw, int gh, int wx, int wy, int fl, int anc)`
  Η βοηθητική συνάρτηση για την τοποθέτηση των γραφικών στοιχείων.
- `public String getPath()`
- `public void setPath(String s)`
  Δύο απλές συναρτήσεις που θέτουν και επιστρέφουν την τιμή του Path, αντίστοιχα.
- `public JScrollPane getScroll()`
  Η συνάρτηση που επιστρέφει την μπάρα κύλισης, στην οποία ανήκει όλο το panel.
- `public void initTable(ResultSet rst,int tab)`
  Η συνάρτηση αυτή ελέγχει αν τα αποτελέσματα rst είναι κενά. Αν είναι, τότε η μεταβλητή empty γίνεται TRUE, αλλιώς προχωράει η κατασκευή του πίνακα αποτελεσμάτων, για τον οποίο θα δούμε περισσότερες λεπτομέρειες αμέσως παρακάτω.

class MyTableModel extends AbstractTableModel

Πεδία

- `String[] columnNames`
  Το πεδίο στο οποίο φορτώνουμε τα ονόματα των στηλών.
- `Object[][] data`
  Ο πίνακας στον οποίο μπαίνουν τα αποτελέσματα.

- `Vector cols`
  Η δομή στην οποία φυλάμε προσωρινά τα ονόματα των στηλών, όπως τα παίρνουμε από την Βάση Δεδομένων, επειδή δεν ξέρουμε τον αριθμό τους.
- `Vector types`
  Το ίδιο για τον τύπο των δεδομένων αυτών.

Μέθοδοι

- `public MyTableModel(ResultSet rst, int tab)`
  Η συνάρτηση κατασκευαστής της κλάσης αυτής. Φροντίζει για την κατάλληλη αρχικοποίηση των μεταβλητών που προαναφέραμε, παίρνοντας πληροφορίες από την Βάση Δεδομένων, για τον εκάστοτε πίνακα με νούμερο tab. Μετά, παίρνει τα αποτελέσματα που βρίσκονται στο rst και τα βάζει στις γραμμές του πίνακα, με μορφή που να ανταποκρίνεται στον τύπο του κάθε πεδίου (πχ τα τύπου bit, που αντιπροσωπεύουν boolean τιμές, εμφανίζονται με τικαρισμένα ή μη checkboxes).
- `public int getColumnCount()`
- `public int getRowCount()`
- `public String getColumnName(int col)`
- `public String getColumnName(int col)`
- `public Class getColumnClass(int c)`
  Αυτές οι συναρτήσεις είναι σαφές και πολύ απλό το τι κάνουν.

6.2.4.18 public class Tree

Αυτή η κλάση, μαζί με την κλάση TreeNode που θα δούμε λίγο πιο κάτω, είναι από τις πιο σημαντικές, όσον αφορά την δομική βάση του συστήματός μας. Είναι η κλάση που, μαζί με την κλάση TreeNode, υλοποιεί την δενδρική δομή, μαζί με αρκετές συναρτήσεις που επιτελούν πολλές λειτουργίες, και καθιστούν τον χειρισμό των δένδρων αρκετά εύκολο και ευέλικτο. Το μοναδικό πεδίο που έχει αυτή η κλάση είναι η ρίζα, και πρέπει να σημειώσουμε ότι όλα τα δένδρα που χρησιμοποιούνται και έχουμε αναφέρει μέχρι τώρα, έχουν μία ρίζα, που δεν φαίνεται στις αναπαραστάσεις. Αυτή η ρίζα, με όνομα “root”, υπάρχει για να μην κατασκευάζονται κενά δένδρα (δηλαδή πάντα έχουν έστω και μια ρίζα), και για να μπορούμε να χειριστούμε όλα τα δένδρα με τον ίδιο τρόπο, οπότε ήταν απαραίτητο να γίνει αυτή η σύμβαση. Αυτό σημαίνει ότι, αν ένα δένδρο, πχ. διαστάσεων, λέμε ότι έχει ρίζα έναν κόμβο με όνομα “dimHi”, στην πραγματικότητα αυτός ο κόμβος είναι παιδί του root. Άρα, όπου ορίζεται ένα δένδρο, αρχικοποιείται με την εντολή:

Tree t=new Tree(new TreeNode(”root”));

Η κλάση αυτή έχει ως εξής:

Πεδία

- `private TreeNode root`
  Η ρίζα που περιγράψαμε παραπάνω. Κάτω από αυτή θα μπουν οι υπόλοιποι κόμβοι.

Μέθοδοι

- `public Tree()`
  Συνάρτηση κατασκευαστής. Δεν κάνει τίποτα.
- `public Tree(TreeNode r)`
  Συνάρτηση κατασκευαστής, όπου θέτει την ρίζα root ίση με το r.
- `public Tree(TreeNode r, long id)`
  Συνάρτηση κατασκευαστής, όπου θέτει την ρίζα root ίση με το r και της βάζει το πεδίο id ίσο με την παράμετρο id.

- `public void addNode(TreeNode which, TreeNode where)`
- `public void addNode(TreeNode which, TreeNode where, String useless)`
  Αυτές οι δύο συναρτήσεις κάνουν ακριβώς το ίδιο πράγμα. Προσθέτουν τον κόμβο which σαν παιδί του where, τον οποίο where τον βρίσκουν μέσα στο δένδρο.
- `public void addNode(TreeNode which, long whereId)`
  Ακριβώς ότι και η προηγούμενη συνάρτηση, μόνο που τώρα σαν είσοδο δίνουμε το id του κόμβου στον οποίο θα προστεθεί ο κόμβος which, αντί για τον ίδιο τον κόμβο.
- `public TreeNode findNode(TreeNode start, TreeNode which)`
- `public TreeNode findNodePub(TreeNode start, TreeNode which)`
  Συναρτήσεις που επιστρέφουν τον κόμβο που φέρει το όνομα του which, και βρίσκεται κάτω από τον κόμβο start. Η πρώτη συνάρτηση χρησιμοποιεί τα τοπικά ονόματα, ενώ η δεύτερη τα γενικά. Αν δεν βρεθεί, επιστρέφεται null.
- `public TreeNode findNode(TreeNode start, long id)`
  Το ίδιο με τις προηγούμενες, αλλά η αναζήτηση γίνεται με βάση το id του κόμβου που ψάχνουμε.
- `public TreeNode getRoot()`
  Επιστρέφει τον κόμβο ρίζα.
- `public void setRoot(TreeNode r)`
  Θέτει τον κόμβο ρίζα ίσο με το r.
- `public void printTree(TreeNode start)`
- `public void printTreePub(TreeNode start)`
  Δύο συναρτήσεις που τυπώνουν στην γραμμή εντολών το δένδρο, με αναζήτηση κατά βάθος, τυπώνοντας πρώτα την ετικέτα του κόμβου, και μετά το id του. Η πρώτη συνάρτηση τυπώνει τα τοπικά ονόματα, και η δεύτερη τα γενικά.
- `public void treePubInFile(TreeNode start, String file)`
- `public void savePubTree(TreeNode start, PrintWriter pw, int depth)`
- `public void treePubInFile2(TreeNode start, String file)`
- `public void savePubTree2(TreeNode start, PrintWriter pw, int depth)`
- `public void treeInFile(TreeNode start, String file)`
- `public void saveTree(TreeNode start, PrintWriter pw, int depth)`
- `public void treeInFile2(TreeNode start, String file)`
- `public void saveTree2(TreeNode start, PrintWriter pw, int depth)`
  Όλες αυτές οι συναρτήσεις είναι για την αποθήκευση των δένδρων σε αρχεία, ξεκινώντας από τον κόμβο start, τυπώνουν σε κατάλληλη μορφή XML στο αρχείο με όνομα file. Η συνάρτηση που καλείται είναι η treeInFile, με τα ορίσματα που αναφέραμε, και μέσα από αυτήν καλείται η saveTree. Σε αυτές τις συναρτήσεις χρησιμοποιούνται τα τοπικά ονόματα των κόμβων, ενώ στις treePubInFile και savePubTree χρησιμοποιούνται τα γενικά. Τώρα, επειδή τα αρχεία διαστάσεων ακολουθούν ένα συγκεκριμένο DTD που πρέπει να τηρηθεί, έχουν υλοποιηθεί και οι συναρτήσεις treeInFile2, saveTree2, treePubInFile2 και savePubTree2 για τις αντίστοιχες λειτουργίες για τα δένδρα και τα αρχεία των διαστάσεων.
- `public void removeNode(TreeNode start, TreeNode which)`
- `public void removeNode(TreeNode start, long id)`
  Αφαιρούν τον κόμβο which από το υποδένδρο που έχει ρίζα το start. Στην δεύτερη συνάρτηση χρησιμοποιείται το id του αφαιρούμενου κόμβου.
- `public boolean isSamePath(TreeNode n1, TreeNode n2, TreeNode start)`
  Ελέγχει αν οι κόμβοι n1 και n2 ανήκουν στο ίδιο μονοπάτι, κάτω από τον κόμβο start.
- `public void findLeaves(TreeNode start, Vector leaves)`
  Επιστρέφει στο Vector leaves όλα τα φύλλα του υποδένδρου με ρίζα το start.

6.2.4.19 public class TreeGraph

Αυτή η κλάση υλοποιεί τους γράφους που αναπαριστούν τις δενδρικές δομές. Χρησιμοποιείται ένα αντικείμενο αυτής της κλάσης για κάθε δένδρο που ζητάει να δει ο χρήστης, και όπως και οι περισσότερες κλάσεις γραφικών, επεκτείνει την κλάση JPanel. Σε αυτή την κλάση υλοποιείται και ένας αλγόριθμος τοποθέτησης των κόμβων, με τρόπο τέτοιο, ώστε να μην υπερκαλύπτονται κόμβοι, σε κανένα σημείο, και να υπάρχει μία ελάχιστη απόσταση μεταξύ τους. Ο αλγόριθμος αυτός δεν περιγράφηκε στην ενότητα 6.2.3 γιατί δεν

είναι από τους βασικούς αλγορίθμους που έχουν να κάνουν με τις δομικές λειτουργίες, αλλά απλά ένας αλγόριθμος αναπαράστασης, που περιγράφεται εδώ.

Η ιδέα είναι να τοποθετούμε σωστά τους κόμβους από τα κάτω επίπεδα προς τα επάνω, και σε κάθε επίπεδο, από αριστερά προς τα δεξιά. Έτσι, τοποθετούνται πρώτα οι κόμβοι του τελευταίου επιπέδου. Αυτοί πάντα δεν υπερκαλύπτονται, γιατί κρατάνε τις ελάχιστες αποστάσεις μεταξύ τους Στο πάνω επίπεδο, βρίσκουμε για κάθε κόμβο το μεσαίο σημείο των παιδιών του, και τον τοποθετούμε σε αυτό το μήκος, στο πάνω επίπεδο φυσικά, αρχίζοντας από τους αριστερούς κόμβους προς τους δεξιότερους. Εδώ υπάρχει η περίπτωση δύο κόμβοι να υπερκαλυφθούν, αν ας πούμε έχουν μόνο ένα κόμβο παιδί και έχουν όνομα αρκετά μεγαλύτερο από αυτό του κόμβου παιδί τους. Τότε, τοποθετούμε τον δεξιότερο από αυτούς τους δύο κόμβους στην ελάχιστη οριζόντια απόσταση που έχουμε ορίσει από τον αριστερό του, υπολογίζουμε την μετατόπιση αυτή και την εφαρμόζουμε στα από κάτω επίπεδα, για όλους τους κόμβους που βρίσκονται κάτω και δεξιά του κόμβου που μετακινήσαμε, δηλαδή για το υποδένδρο με ρίζα τον μετακινούμενο κόμβο και τους κόμβους δεξιά από αυτό. Αυτή η μέθοδος εφαρμόζεται για κάθε κόμβο που υπερκαλύπτει τον αμέσως αριστερό του κόμβο.

Η κλάση έχει ως εξής:

Πεδία

- `public JGraphModelAdapter m_jgAdapter`
  Η μεταβλητή για τον χειρισμό του γράφου.
- `public final Color DEFAULT_BG_COLOR`
  Μία σταθερά που κρατάει την τιμή ενός χρώματος.
- `JGraph jgraph`
  Η μεταβλητή του γράφου.
- `int MIN_DIST`
- `int VDIST`
- `int V_DATA_DIST`
  Κάποιες σταθερές που κρατάνε τις τιμές των αποστάσεων που χρησιμοποιούμε. Με την σειρά, είναι η ελάχιστη απόσταση των κόμβων του ίδιου επιπέδου, η απόσταση των επιπέδων, και η κάθετη απόσταση των κόμβων που δίνουν τις εγγραφές από τους κόμβους φύλλα στους οποίους αναφέρονται.

Μέθοδοι

- `public TreeGraph(String s, int k)`
  Η συνάρτηση κατασκευαστής της κλάσης. Από αυτήν καλείται η επόμενη συνάρτηση, που αρχικοποιεί τον γράφο, και σε αυτήν περνάνε και οι παράμετροι s και k. Το s είναι το όνομα του αρχείου που κρατάει την πληροφορία για την δενδρική δομή, και το k είναι μια παράμετρος για να ξεχωρίζουμε αν πρόκειται για αρχείο διαστάσεων, ή ιεραρχίας.
- `public void init( String s, int k),`
  Αυτή η συνάρτηση αρχικοποιεί τις μεταβλητές του γράφου, φορτώνει την δενδρική δομή που χρειάζεται, και καλεί την συνάρτηση τοποθέτησης των κόμβων treeInGraph.
- `public void positionVertexAt( Object vertex, int x, int y )`
  Αυτή η συνάρτηση τοποθετεί τον κόμβο vertex που παίρνει σαν είσοδο, στο σημείο x,y.
- `public int calcWidth(String s)`
  Υπολογίζει το μήκος του ορθογωνίου που αναπαριστά τους κόμβους, με βάση το μήκος του ονόματός του s.
- `public Vector handleNodes(TreeNode start)`
  Αρχικοποιεί κάποιες βοηθητικές δομές, κατασκευάζει την ρίζα του γραφήματος, κλάσης GraphNode που είδαμε παραπάνω, από την είσοδο start, και καλεί κάποιες άλλες συναρτήσεις. Επίσης, καλεί την συνάρτηση που υπολογίζει το σημείο τοποθέτησης των

κόμβων, σύμφωνα με τον αλγόριθμο που περιγράψαμε. Τέλος, επιστρέφει ένα Vector που περιέχει Vectors. Κάθε ένα από τα εσωτερικά Vectors αναπαριστά ένα επίπεδο του γράφου, και περιέχει τους κατάλληλους κόμβους, κλάσης GraphNode, με την σωστή σειρά.

- `public void treeInGraph(ListenableGraph g, Vector levels)`
  Η είσοδος levels είναι στην ουσία το Vector που προκύπτει από την προηγούμενη συνάρτηση, και το g είναι ο γράφος στον οποίο θα τοποθετηθούν οι κόμβοι. Εδώ καλείται αλλεπάλληλα η συνάρτηση positionVertexAt, με τα κατάλληλα δεδομένα, για την τοποθέτηση των κόμβων.
- `public boolean isAllDummies(Vector v)`
  Επειδή ενδεχομένως η δενδρική δομή να μην έχει όλα τα επίπεδα πλήρη, όταν φτάσουμε σε φύλλα, συνεχίζουμε να τοποθετούμε dummy κόμβους παιδιά, που όμως δεν κάνουν τίποτα, και τελικά δεν θα παρουσιαστούν. Αυτό γίνεται για την σωστή τοποθέτηση και εφαρμογή του αλγορίθμου διάταξης που περιγράψαμε στην εισαγωγή. Όταν ένα επίπεδο αποτελείται μόνο από κόμβους dummy, τότε ξέρουμε ότι έχουμε τελειώσει και με το πιο βαθύ επίπεδο του δένδρου και αυτή η συνάρτηση ελέγχει αυτό ακριβώς το πράγμα.
- `public void initNodes(Vector lstart, Vector l, int k)`
- `public void initNodes2(Vector lstart, Vector l, int k)`
  Δύο συναρτήσεις που κάνουν τα ίδια πράγματα, μόνο που η πρώτη είναι για δένδρα ιεραρχιών και η δεύτερη για δένδρα διαστάσεων. Προσθέτουν στο Vector που, περιέχει τα επίπεδα του δένδρου, τα νέα επίπεδα, κατασκευάζουν για κάθε κόμβο τα GraphNode, και υπολογίζουν πότε πρέπει να μπει κόμβος dummy ή DATA (για τις ιεραρχίες).
- `public void calcCoordinates(Vector l)`
  Παίρνει ως είσοδο το Vector που περιέχει Vectors από GraphNodes, και υπολογίζει για κάθε κόμβο το σημείο στο οποίο θα μπει. Οι συντεταγμένες που υπολογίζονται φυλάσσονται στην δομή GraphNode, όπως αναφέραμε στην περιγραφή της κλάσης αυτής. Ουσιαστικά είναι η υλοποίηση του αλγορίθμου που περιγράψαμε, και όταν βρεθεί ότι κάποιος κόμβος μετακινήθηκε, καλείται η συνάρτηση fixChild, που μεταδίδει την μεταφορά στους από κάτω κόμβους.
- `public int[] findPos(Vector v, TreeNode tn)`
  Υπολογίζει τα pixels στα οποία πρέπει να αρχίσει και να τελειώσει το ορθογώνιο που αναπαριστά τον κόμβο, ανάλογα με την θέση στην οποία βρίσκεται ο κόμβος μέσα στο δένδρο, και με τις θέσεις που έχουν τοποθετηθεί οι αριστερότεροι από αυτόν κόμβοι. Παίρνει σαν είσοδο το Vector που κρατάει τους κόμβους του επιπέδου που εξετάζουμε, και τον κόμβο TreeNode που εξετάζουμε. Επιστρέφει τις συντεταγμένες x,y σε έναν πίνακα από int.
- `public void fixChild(Vector l, GraphNode pn, int offset, int level)`
  Η συνάρτηση αυτή καλεί αναδρομικά τον εαυτό της, και μεταφέρει την μετακίνηση στους διάφορους κόμβους, κάτω και δεξιά από τον κόμβο που μετακινήθηκε. Το Vector l είναι το Vector που κρατάει όλα τα επίπεδα του δένδρου, το pn είναι ο κόμβος στον οποίο εφαρμόζεται η μετακίνηση κάθε φορά, το offset είναι το μέγεθος της μετακίνησης, σε pixels, και το level είναι το επίπεδο στο οποίο βρίσκεται ο έλεγχος κάθε φορά.
- `public void fixDataNode(TreeNode temp, GraphNode gn)`
  Συμπληρώνει στον κόμβο DATA temp το πεδίο με το κείμενο του query, και θέτει για πατέρα του το GraphNode gn.

6.2.4.20 public class TreeNode

Αυτή η κλάση είναι που μαζί με την κλάση Tree υλοποιεί τις δενδρικές δομές στην μνήμη. Όπως αναφέραμε και πιο πάνω, έχει πολλές συναρτήσεις για τον χειρισμό των κόμβων, τις οποίες αναφέρουμε αμέσως παρακάτω. Η κλάση αυτή, έχει λοιπόν ως εξής:

Πεδία

- `private Vector children`
  Σε αυτό το Vector μπαίνουν τα παιδιά του κάθε κόμβου, με αυθαίρετη σειρά.
- `private String label`
  Αυτό είναι η ετικέτα του κόμβου, με το τοπικό όνομα.
- `private String pubLabel`
  Αυτό είναι η ετικέτα με το γενικό όνομα.

- `private long id`
  Ένας ακέραιος που χαρακτηρίζει κάθε κόμβο. Δεν χρησιμοποιείται τόσο πολύ.
- `private TreeNode parent`
  Είναι ο κόμβος πατέρας.
- `private String info`
  Ένα πεδίο String που κρατάει την πληροφορία που βρίσκεται ανάμεσα στο άνοιγμα και το κλείσιμο των tags των αρχείων XML. Χρησιμοποιείται για να κρατάει τα queries των φύλλων.
- `static private Hashtable hash`
  Αυτός ο HashTable είναι για την αντιστοίχηση των τοπικών ονομάτων σε γενικά. Δεν έχει την ίδια μορφή με τον HashTable hash που περιγράψαμε στην κλάση DependenceGraph. Εδώ είναι αρκετά πιο απλός. Για κάθε τοπικό όνομα υπάρχει μια αντιστοιχία για το γενικό όνομα.

Μέθοδοι

- `public TreeNode()`
  Απλή συνάρτηση κατασκευαστής.
- `public TreeNode(String l)`
  Συνάρτηση κατασκευαστής που θέτει το label ίσο με το l.
- `public TreeNode(String l, TreeNode p)`
  Συνάρτηση κατασκευαστής που θέτει το label ίσο με το l, και θέτει το parent ίσο με το p.
- `public void addChild(TreeNode n)`
  Προσθέτει το n στα παιδιά του κόμβου. Το n αλλάζει πατέρα.
- `public void addNewChild(TreeNode n)`
  Αντιγράφει το n, και όλη την δομή που το ακολουθεί, και το βάζει στα παιδιά του. Το n δεν επηρεάζεται.
- `public void addChild(TreeNode n, long i)`
  Προσθέτει το n στα παιδιά του, και θέτει το id του n ίσο με i.
- `public TreeNode getChild(int i)`
  Επιστρέφει τον κόμβο παιδί που βρίσκεται στην θέση i.
- `public Vector getChildNodes()`
  Επιστρέφει το Vector με τα παιδιά του κόμβου.
- `public TreeNode getFirstChild()`
- `public TreeNode getLastChild()`
  Επιστρέφουν το πρώτο και το τελευταίο παιδί, αντίστοιχα.
- `public String getInfo()`
- `public void setInfo(String s)`
  Επιστρέφουν και θέτουν το πεδίο info.
- `public long getId()`
  Επιστρέφει το πεδίο id.
- `public TreeNode getNextSibling()`
  Επιστρέφει το επόμενο στην σειρά παιδί του κόμβου πατέρα του, αν αυτό υπάρχει, αλλιώς επιστρέφει null.
- `public String getNodeName()`
- `public String getLabel()`
  Επιστρέφουν το label του κόμβου.
- `public String getPubLabel()`
- `public void setPubLabel(String l)`
  Επιστρέφουν και θέτουν το pubLabel του κόμβου.
- `public TreeNode getParentNode()`
  Επιστρέφει τον κόμβο πατέρα parent.
- `public boolean hasChildNodes()`
  Επιστρέφει TRUE αν έχει παιδιά, αλλιώς επιστρέφει FALSE.
- `public boolean isLeaf()`
  Επιστρέφει TRUE αν δεν έχει παιδιά, αλλιώς FALSE.
- `public int numberOfChilds()`
  Επιστρέφει τον αριθμό των παιδιών.
- `public void setId(long i)`
  Θέτει το πεδίο id.
- `public void setLabel(String l)`
  Θέτει το πεδίο label.
- `public void setParentNode(TreeNode p)`

Θέτει τον πατέρα ίσο με το p.

- `public void removeChild(TreeNode which)`
  Διαγράφει τον κόμβο παιδί which.
- `public Hashtable getHash()`
  Επιστρέφει τον hash.
- `public void estimatePubLabel(Hashtable h)`
  Υπολογίζει το pubLabel, σύμφωνα με το label και το hash.
- `public void prepareHash()`
  Αρχικοποιεί την δομή hash από το αρχείο SimNodes.txt.
- `public Vector findWords(String s)`
  Χρησιμοποιείται για την ανάλυση της κάθε γραμμής σε λέξεις, που τις επιστρέφει σε ένα Vector με την σειρά.
- `public int hasChildNode(String lab)`
  Αν ο κόμβος έχει παιδί με pubLabel ίσο με το lab, επιστρέφει την θέση του παιδιού, αλλιώς επιστρέφει -1.
- `public int hasChildNodeNPub(String lab)`
  Ακριβώς ότι και το προηγούμενο, αλλά εξετάζει τα label.
- `public boolean hasAncestor(String s)`
  Βρίσκει αν ο κόμβος έχει πρόγονο με pubLabel ίσο με το s.

6.2.4.21 public class ViewTrees

Αυτή η κλάση υλοποιεί το γραφικό περιβάλλον για την απεικόνιση των γραφημάτων των ιεραρχιών, των διαστάσεων και του γράφου εξαρτήσεων. Επίσης, εδώ γίνεται και η επιλογή των ενεργών ιεραρχιών, δηλαδή αυτών στις οποίες θα αποτιμηθεί η ερώτηση μονοπάτι του χρήστη. Εδώ εμφανίζονται τα αντικείμενα των κλάσεων TreeGraph και DGGraph. Η κλάση αυτή, επειδή υλοποιεί ένα γραφικό παράθυρο, επεκτείνει την κλάση JPanel, υλοποιεί τα interfaces ActionListener και WindowListener και έχει ως εξής:

Πεδία

- `JPanel pan`
- `JPanel panel1`
- `JPanel panel2`
- `JPanel panel3`
- `JPanel panel4`
- `JPanel pan1`
  Τα panels που θα υποδεχτούν τα γραφικά στοιχεία.
- `JButton show`
- `JButton close`
  Τα δυο κουμπιά αυτά ενεργοποιούν την εμφάνιση ή το κλείσιμο ενός tab, σύμφωνα με την επιλογή που έχει γίνει, αντίστοιχα.
- `JLabel info`
- `JLabel empty`
- `JLabel empty2`
  Κάποιες ετικέτες μηνυμάτων.
- `JComboBox cbd`
- `JComboBox cbh`
  Τα ComboBoxes που περιέχουν τις επιλογές για τις διαστάσεις και τις ιεραρχίες, αντίστοιχα.
- `JRadioButton r1`
- `JRadioButton r2`
- `JRadioButton r3`
- `ButtonGroup gr`
  RadioButtons, για την απλή επιλογή εμφάνισης δένδρου ιεραρχίας, διαστάσεων, ή του γράφου εξαρτήσεων.
- `JTabbedPane tb`
  Ο υποδοχέας των tabs.

- `JButton editDims`
- `JButton editDep`
- `JButton res`
- `JButton gnt`
- `JButton grep`
- `JButton restart`
- `JButton quit`
- `JToolBar tbr`
  Το ToolBar και τα κουμπιά του.
- `JCheckBox chk`
  Ένα CheckBox, για την επιλογή των ενεργών διαστάσεων.
- `JScrollPane scroll`
  Μια μπάρα κύλισης.

Μέθοδοι

- `public ViewTrees()`
  Συνάρτηση κατασκευαστής της κλάσης. Αρχικοποιεί τα panels και τοποθετεί τα γραφικά στοιχεία στην θέση τους.
- `public void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy, int fl, int anc)`
  Βοηθητική συνάρτηση για την τοποθέτηση των γραφικών στοιχείων.
- `public void initDims()`
  Αρχικοποιεί το ComboBox cbd με τα δένδρα των διαστάσεων.
- `public void initHs()`
  Αρχικοποιεί το ComboBox cbh με τα ονόματα των ιεραρχιών.
- `public void initPanel3()`
  Αρχικοποιεί το panel3, που περιλαμβάνει τα CheckBoxes για τις ενεργές ιεραρχίες.
- `public void actionPerformed(ActionEvent evt)`
  Συνάρτηση χειρισμού των γεγονότων που εγείρονται με το πάτημα των διαφόρων κουμπιών.
- `public void addInTab(String s, int k)`
  Η συνάρτηση που καλείται όταν πατηθεί το κουμπί show. Το s είναι ο τίτλος που θα μπει στο tab, και το k είναι ένας ακέραιος, για να ξεχωρίζουμε αν η επιλογή έγινε για ιεραρχία, για διαστάσεις, ή για τον γράφο εξαρτήσεων. Αν το tab ήδη υπάρχει, τότε ανανεώνεται, γιατί ενδεχομένως να έχουν γίνει αλλαγές.
- `public void windowClosing(WindowEvent evt)`
- `public void windowDeactivated(WindowEvent evt)`
- `public void windowClosed(WindowEvent evt)`
- `public void windowOpened(WindowEvent evt)`
- `public void windowIconified(WindowEvent evt)`
- `public void windowDeiconified(WindowEvent evt)`
  Συναρτήσεις χειρισμού των γεγονότων παραθύρου.

6.2.5 Πακέτα JGraph και JGraphT

Αυτά τα δύο πακέτα μας προσέφεραν την υλοποίηση για την παρουσίαση γράφων. Χρησιμοποιήσαμε έτοιμες συναρτήσεις για την τοποθέτηση των κόμβων σε συγκεκριμένα σημεία, για την τοποθέτηση των ακμών, και για την εμφάνιση της πληροφορίας που εμείς θέλαμε να εμφανιστεί. Τα σημεία που αλλάξαμε αφορούσαν κυρίως την απενεργοποίηση γεγονότων που προέκυπταν από την επιλογή οποιουδήποτε κόμβου, καθώς θέλουμε κάποιοι συγκεκριμένοι κόμβοι να ενεργοποιούν συγκεκριμένες λειτουργίες, όπως πχ. οι κόμβοι που παρουσιάζουν τα δεδομένα που βρίσκονται στα φύλλα των ιεραρχιών, και οι κόμβοι που επιλέγονται κατά την κατασκευή του καθολικού κανονικοποιημένου δένδρου, καθώς πρέπει να επιλέγονται μόνο αυτοί του τελευταίου επιπέδου. Επίσης, αναγκαστήκαμε να αφαιρέσουμε

τις λειτουργίες τροποποίησης του γράφου, όπως την μεταφορά ακμών και κόμβων, και την δημιουργία γωνιών στις ακμές.

Μία αρκετά σημαντική προσθήκη έγινε για τον χειρισμό συγκεκριμένων αντικειμένων. Ένας κόμβος μπορεί να είναι αντικείμενο οποιασδήποτε κλάσης, αλλά εμείς χρειαζόμασταν διαφορετικές λειτουργίες για αντικείμενα συγκεκριμένων κλάσεων, όπως είναι πχ. η εμφάνιση των διαστάσεων στον γράφο και οι κόμβοι των ιεραρχιών. Χρησιμοποιήσαμε πχ. διαφορετικά χρώματα στην εμφάνιση, και οι συναρτήσεις που καθόριζαν την τοποθέτηση των κόμβων χρησιμοποιούσαν πληροφορίες μέσα από την δομή των αντικειμένων, όπως πχ στην κλάση GraphNode.

Συγκεκριμένα, τροποποιήσαμε λίγο τα αρχεία (Δίνουμε και το path): org/jgraph/Jgraph.java, org/jgraph/graph/VertexView.java, org/jgraph/plaf/BasicGrapgUI.java.

Περισσότερες πληροφορίες ως προς την αλλαγή του κώδικα συγκεκριμένων κλάσεων αυτών των πακέτων βρίσκονται στον Τόμο (Β), δηλαδή στο εγχειρίδιο.

7

Έλεγχος

Σε αυτό το κεφάλαιο υλοποιούμε τον έλεγχο του συστήματος που κατασκευάσαμε, προκειμένου να εμφανιστούν λάθη ή παραλείψεις που δεν υπέπεσαν στην αντίληψή μας κατά την σχεδίαση και την υλοποίηση της εφαρμογής.

7.1 Μεθοδολογία Ελέγχου

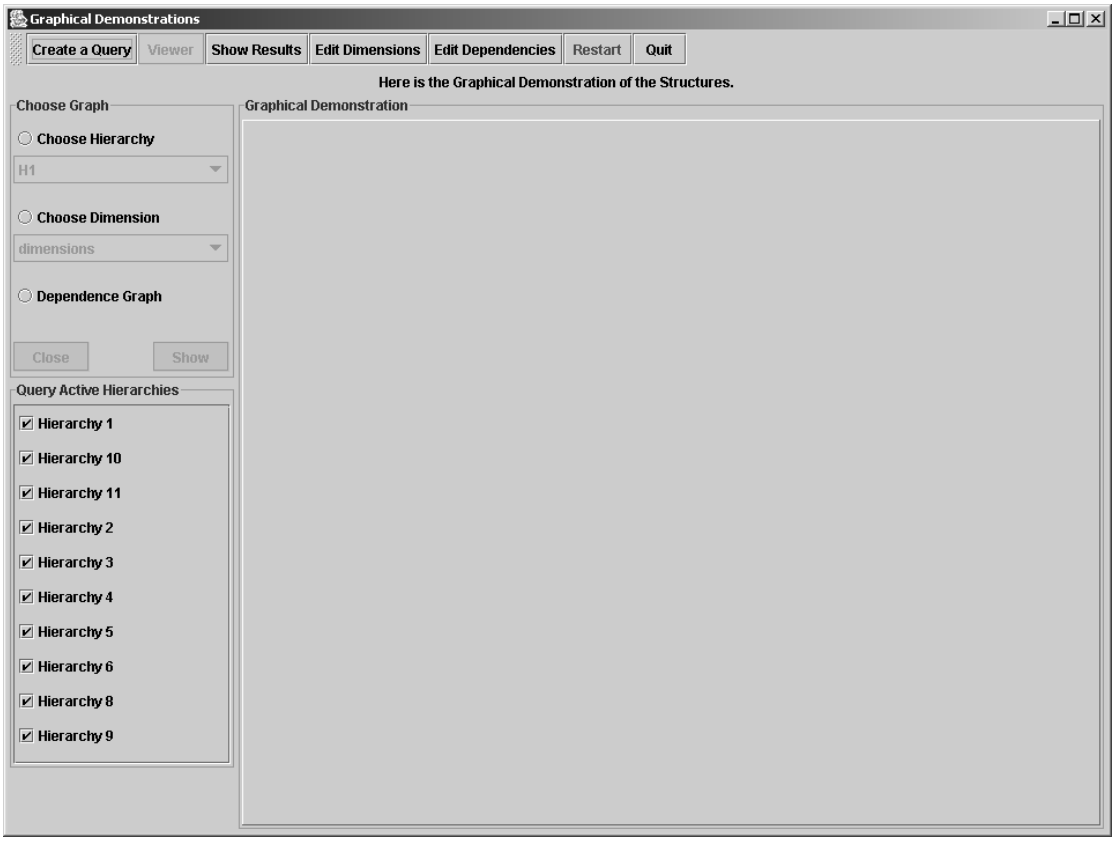
Το σωστό θα ήταν να υποβάλλουμε το σύστημα σε εξαντλητικούς ελέγχους σεναρίων, προκειμένου να είμαστε σίγουροι για την σωστή λειτουργία όλων των περιπτώσεων. Αυτό όμως δεν είναι δυνατόν, οπότε ο έλεγχος έγκειται στην εφαρμογή κάποιων χαρακτηριστικών σεναρίων, που χρησιμοποιούν όλες τις λειτουργίες του συστήματος τουλάχιστον μια φορά. Επειδή έχουμε την δυνατότητα να ελέγχουμε τα αρχεία που παριστάνουν τις δομές και την Βάση Δεδομένων, είμαστε σε θέση να ξέρουμε ποια αποτελέσματα να περιμένουμε, ύστερα από κάθε λειτουργία, και σύμφωνα με τα αναμενόμενα αποτελέσματα κρίνουμε και την σωστή λειτουργία του συστήματος.

7.2 Αναλυτική Παρουσίαση Ελέγχου

Ένα τέτοιο χαρακτηριστικό σενάριο λειτουργίας παρουσιάζεται σε αυτή την ενότητα. Σημειώνουμε ότι παρουσιάζονται όλες οι λειτουργίες, όχι όμως κάτω από όλες τις δυνατές συνθήκες και ακραίες περιπτώσεις, γιατί αυτό δεν είναι δυνατόν να γίνει μέσα στα πλαίσια του γενικού ελέγχου.

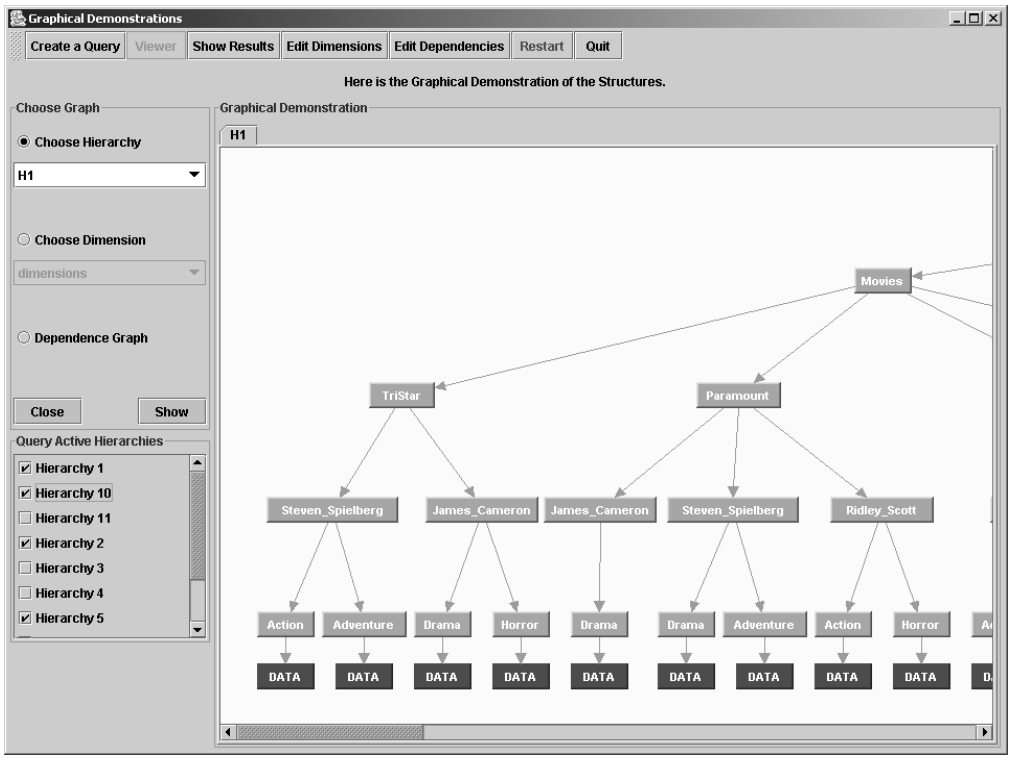
7.2.1 Παρουσίαση Δομών

Ξεκινώντας το σύστημα, βλέπουμε την εξής οθόνη:



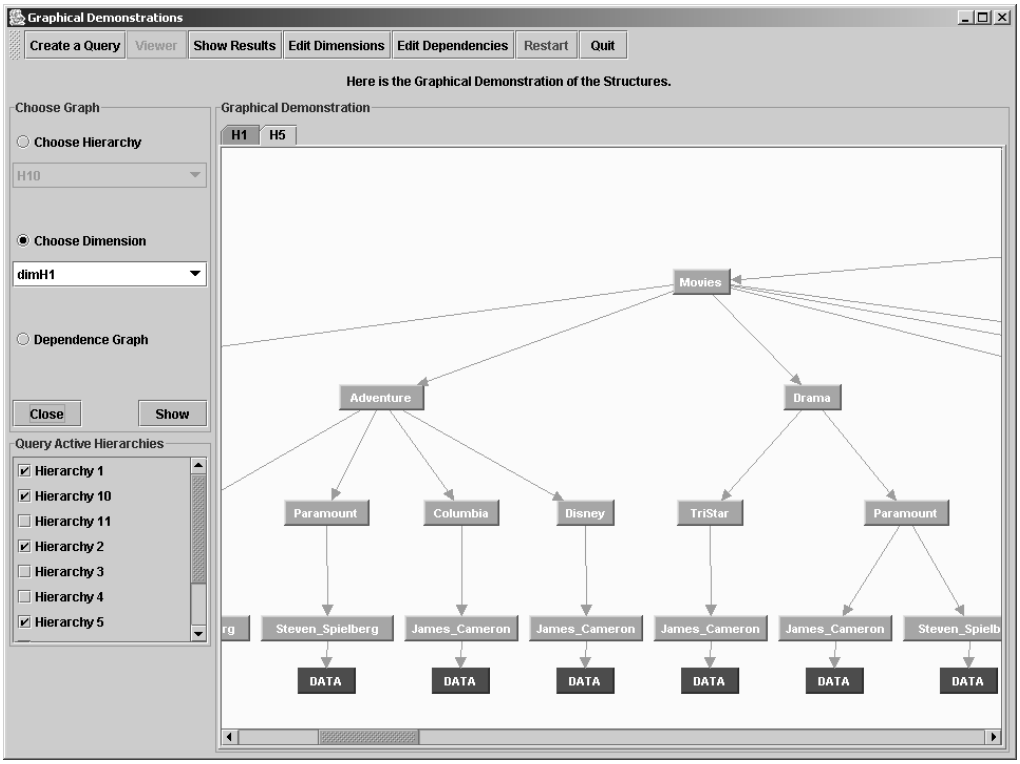
Εικόνα 7.1

Σε αυτή την οθόνη ο χρήστης μπορεί να επιλέξει να δει την γραφική αναπαράσταση των ιεραρχιών, των διαστάσεων, ή του γράφου εξαρτήσεων, επιλέγοντας αριστερά το κατάλληλο RadioButton, και, στην περίπτωση που επιλέγει να δει ιεραρχία ή διαστάσεις, επιλέγοντας την συγκεκριμένη ιεραρχία ή διάσταση από το ComboBox που ενεργοποιείται. Κάτω από αυτά, υπάρχουν με CheckBoxes όλες οι ιεραρχίες. Εδώ ο χρήστης επιλέγει ποιες θέλει να είναι ενεργές, δηλαδή σε ποιές από αυτές θα εφαρμοστεί το Μονοπάτι που θα κατασκευάσει. Για παράδειγμα, η ιεραρχία H1 εμφανίζεται με το πάτημα του κουμπιού “Show”, αφού επιλεχθεί κατάλληλα, και οι ενεργές ιεραρχίες θα είναι η H1, H2, H5, H6 και H10. Αυτά φαίνονται στην Εικόνα 7.2.

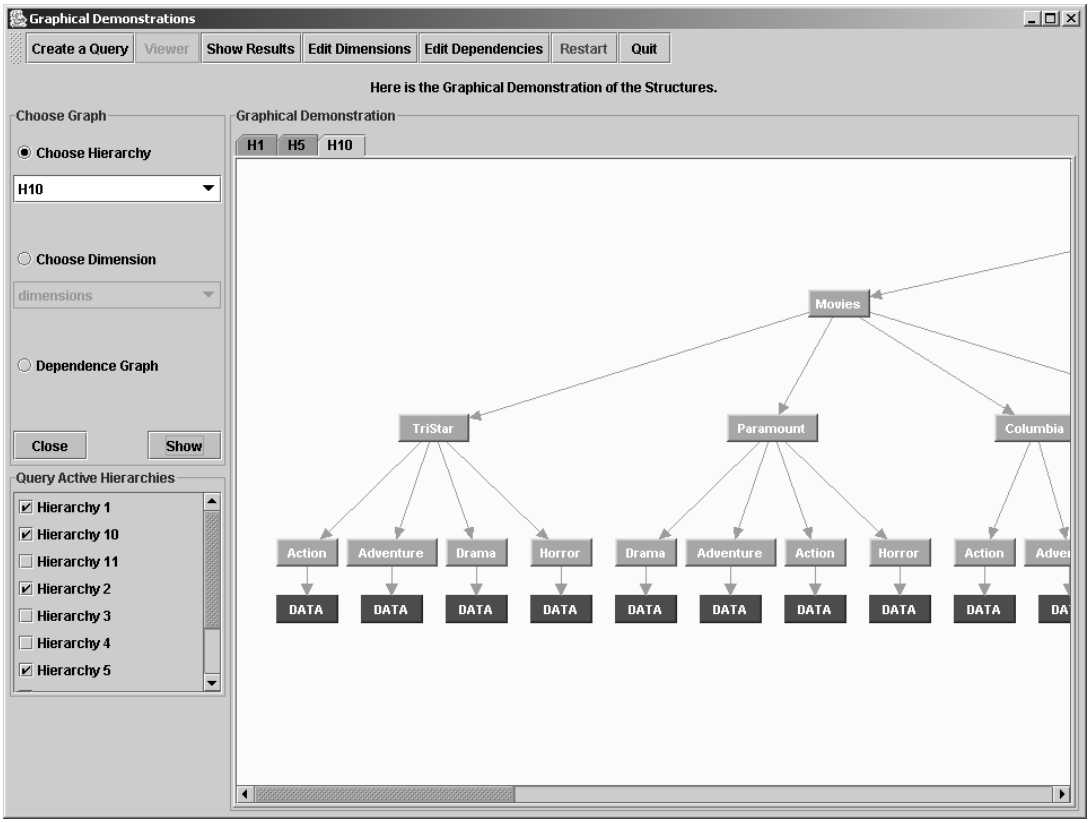


Εικόνα 7.2

Για να συγκρίνουμε την διαφορά στις ιεραρχίες, βλέπουμε και την ιεραρχία H5 στην Εικόνα 7.3, και την H10 στην Εικόνα 7.4. Παρατηρούμε την διαφορά στην δομή των ιεραρχιών, και ότι στην H10 δεν υπάρχουν κόμβοι που να χαρακτηρίζουν σκηνοθέτες. Στην Εικόνα 7.3 δείχνουμε ένα συγκεκριμένο κομμάτι της ιεραρχίας H5, που θα χρησιμοποιηθεί αργότερα.

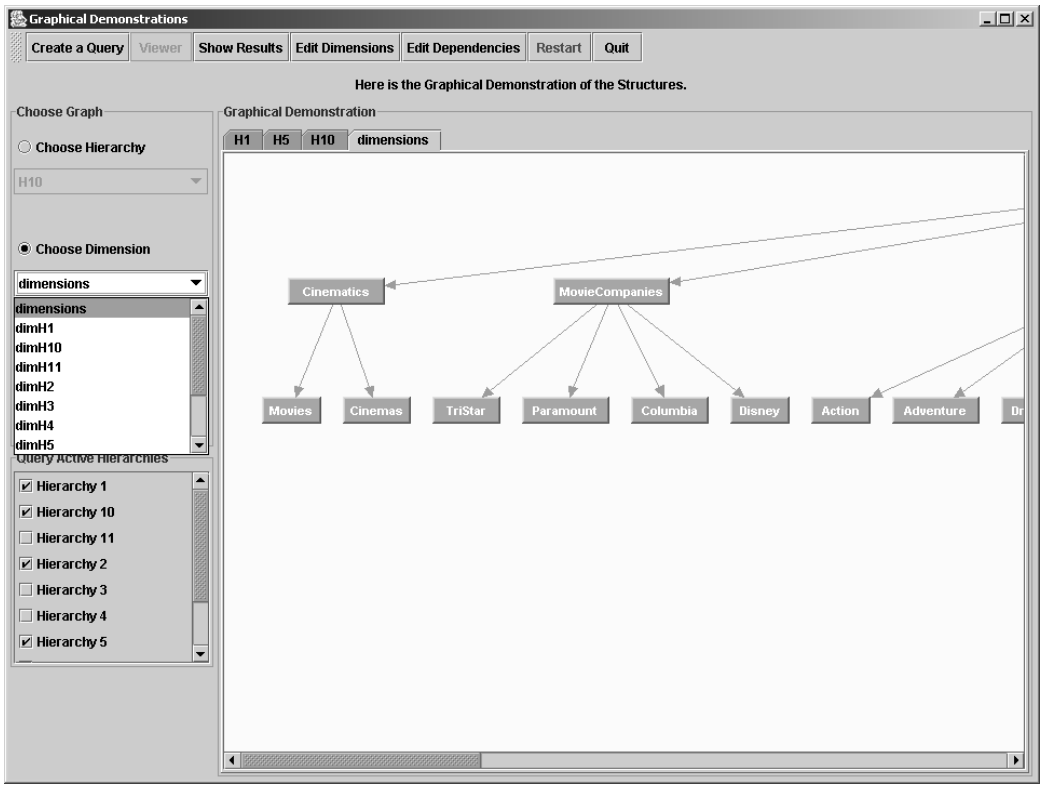


Εικόνα 7.3

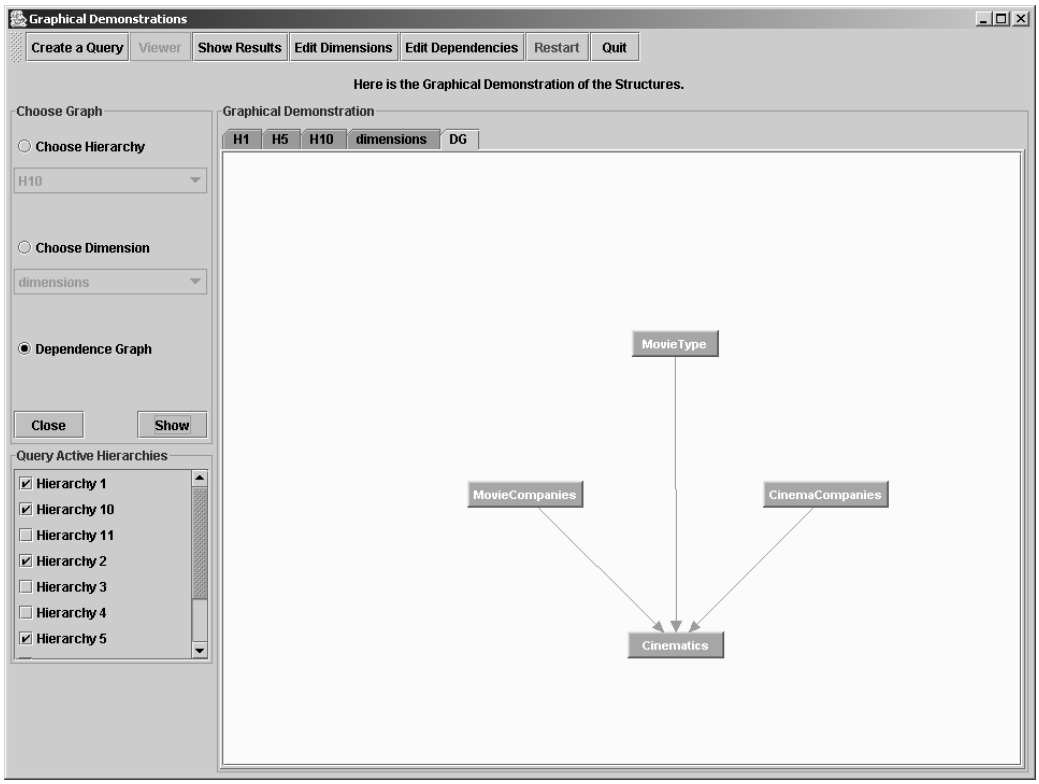


Εικόνα 7.4

Επίσης, μπορούμε να δούμε πως εμφανίζεται ένα δένδρο διαστάσεων, πχ των συγχωνευμένων διαστάσεων, και τον υπάρχοντα γράφο εξαρτήσεων.



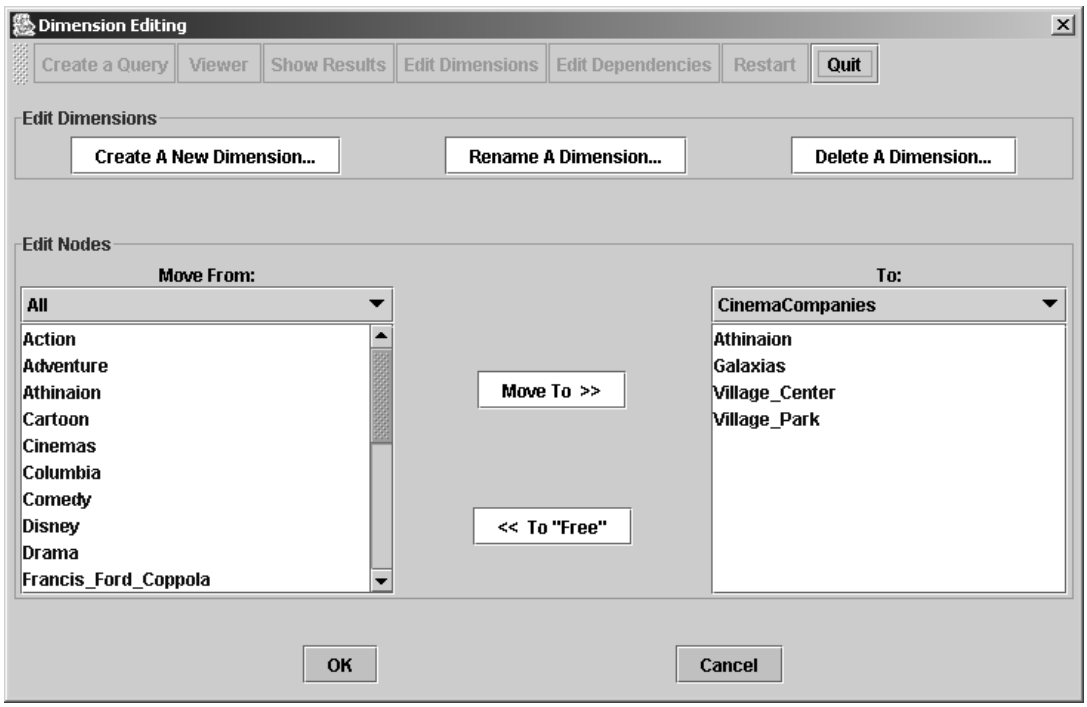
Εικόνα 7.5



Εικόνα 7.6

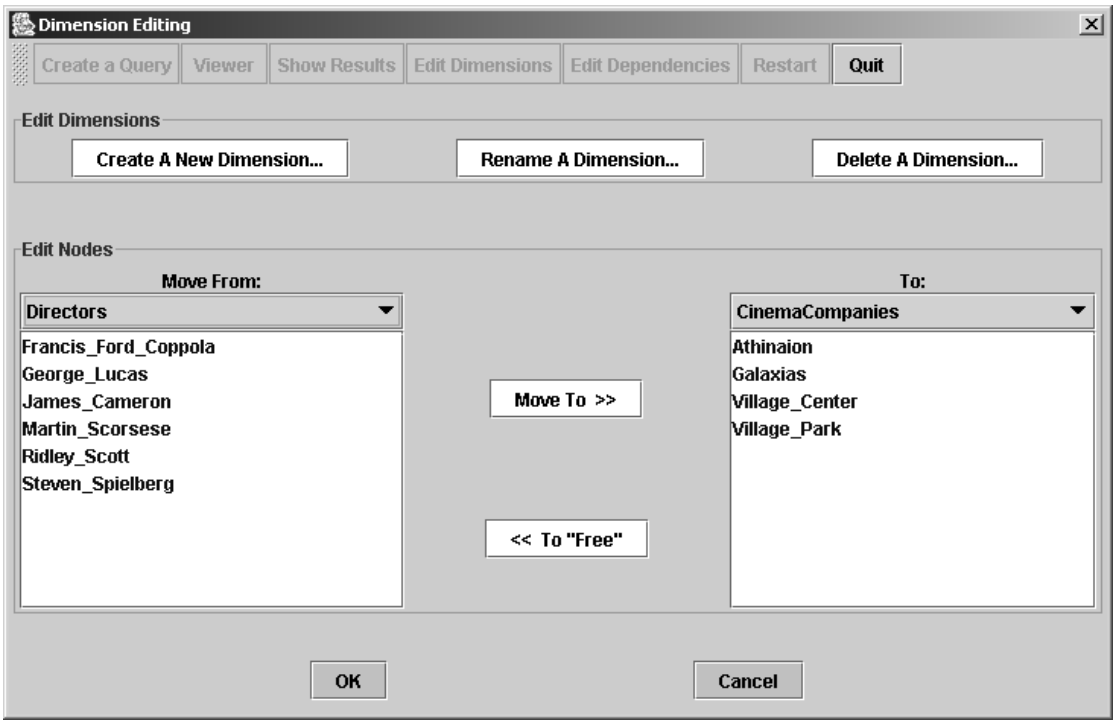
7.2.2 Διαχείριση Διαστάσεων

Εάν θέλουμε να αλλάξουμε τις διαστάσεις, επιλέγουμε το κουμπί με όνομα “Edit Dimensions”, που βρίσκεται πάνω στο ToolBar, και που ανοίγει το εξής παράθυρο:



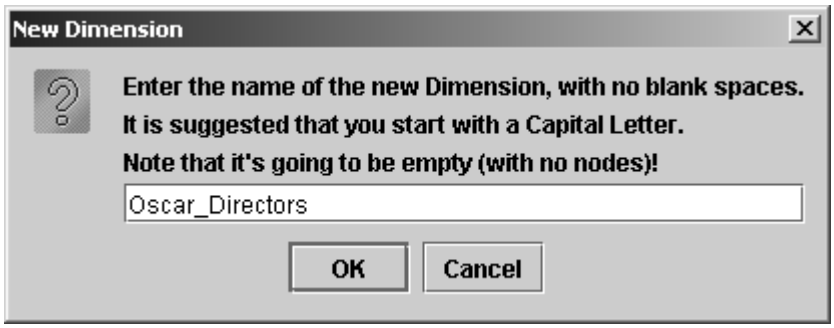
Εικόνα 7.7

Αριστερά και δεξιά βλέπουμε τους κόμβους που ανήκουν στην επιλεγόμενη, από το ComboBox, διάσταση. Αριστερά υπάρχει η επιλογή να δούμε όλους τους κόμβους, ανεξαρτήτως διαστάσεως, καθώς και τους 'ορφανούς' (χωρίς να ανήκουν σε κάποια διάσταση) κόμβους. Εάν θέλουμε να μετακινήσουμε κόμβο από μια διάσταση σε μια άλλη, αριστερά επιλέγουμε τον κόμβο, από την διάσταση στην οποία βρίσκεται, και δεξιά επιλέγουμε την διάσταση προορισμό, και πατάμε το κουμπί "Move To >>". Αν θέλουμε να διαγράψουμε κάποιον κόμβο, επιλέγουμε δεξιά τον κόμβο στην διάσταση που θέλουμε, και πατάμε το "<< To Free", και ο κόμβος θα γίνει 'ορφανός'. Εμείς, για έλεγχο, θα χωρίσουμε την διάσταση με όνομα Directors σε δύο υποκατηγορίες: Στην διάσταση Oscar_Directors και Non_Oscar_Directors. Καταρχήν δείχνουμε αριστερά τους κόμβους που ανήκουν στην διάσταση Directors:



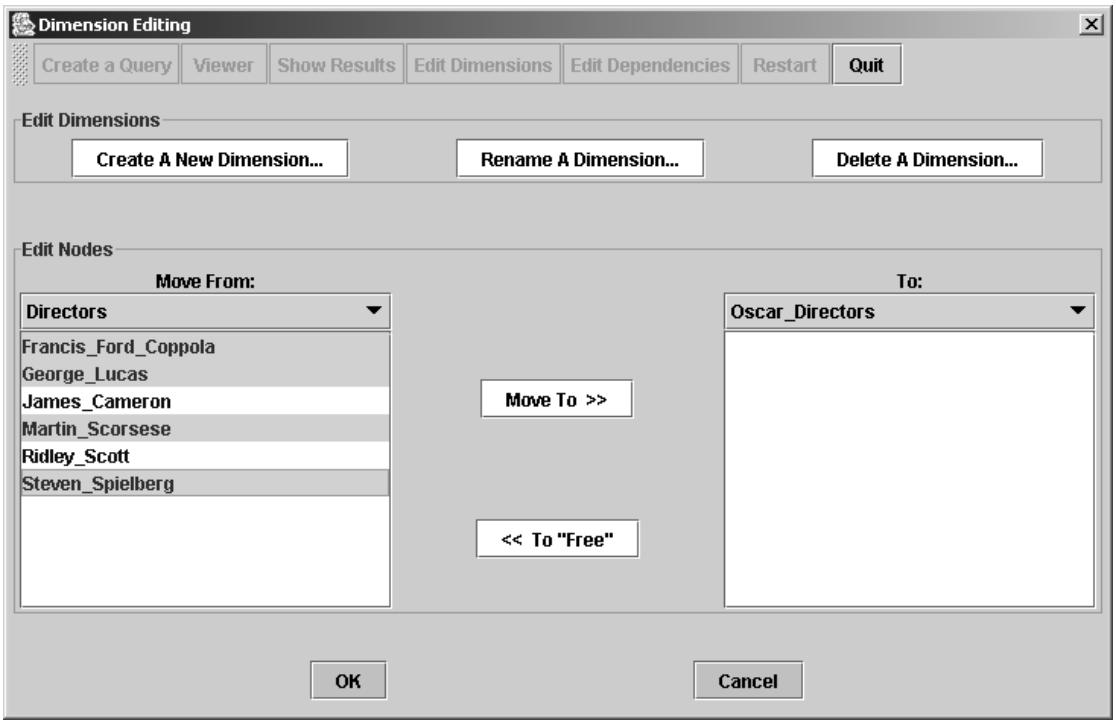
Εικόνα 7.8

Μετά, κατασκευάζουμε μια νέα διάσταση με όνομα Oscar_Directors, στην οποία θα μεταφέρουμε (τυχαία) 4 κόμβους. Η κατασκευή γίνεται πατώντας το κουμπί "Create A New Dimension…", και μας δίνει το εξής παράθυρο, στο οποίο συμπληρώνουμε το όνομα της νέας διάστασης:



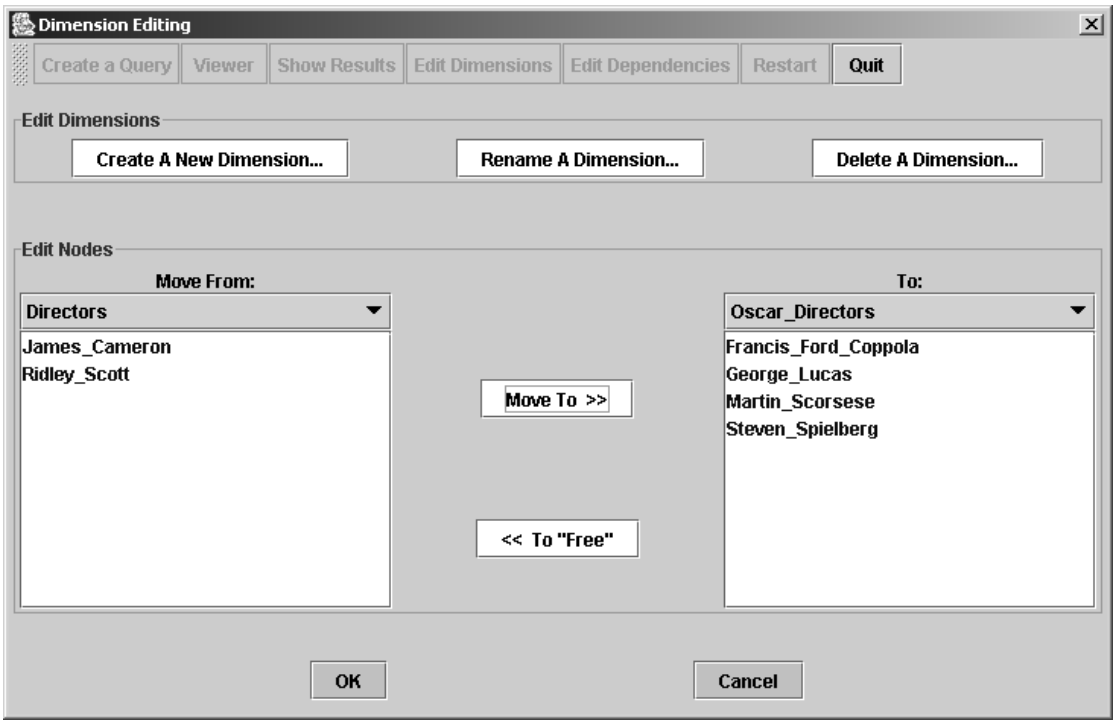
Εικόνα 7.9

Πατάμε το OK, και προετοιμάζουμε τις επιλογές για την μεταφορά των κόμβων. Οι επόμενες δύο εικόνες δείχνουν το πως γίνεται αυτό.



Εικόνα 7.10

Πατώντας το κουμπί “Move To >>” γίνεται το εξής:



Εικόνα 7.11

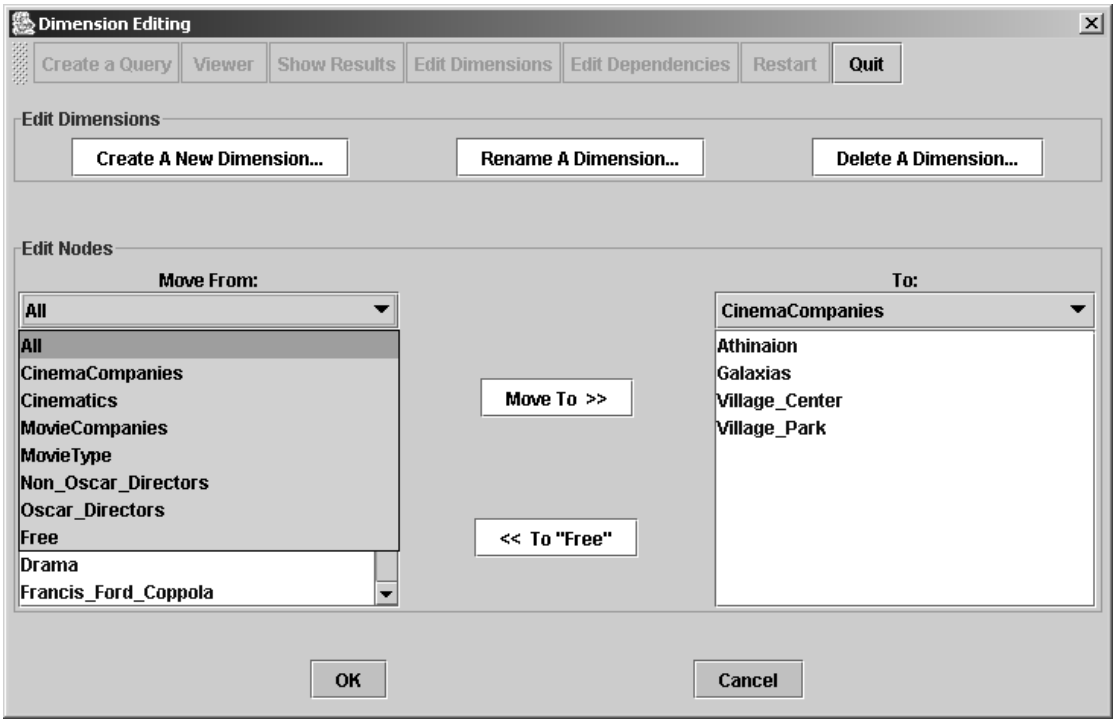
Τώρα, απομένει να μετονομάσουμε την διάσταση Directors σε Non_Oscar_Directors. Πατάμε το κουμπί “Rename A Dimension…” και παίρνουμε το εξής παράθυρο:

Εικόνα 7.12

Επιλέγουμε την διάσταση Directors, βάζουμε το νέο όνομα Non_Oscar_Directors, πατάμε το OK, και βγαίνει ένα παράθυρο επιβεβαίωσης:

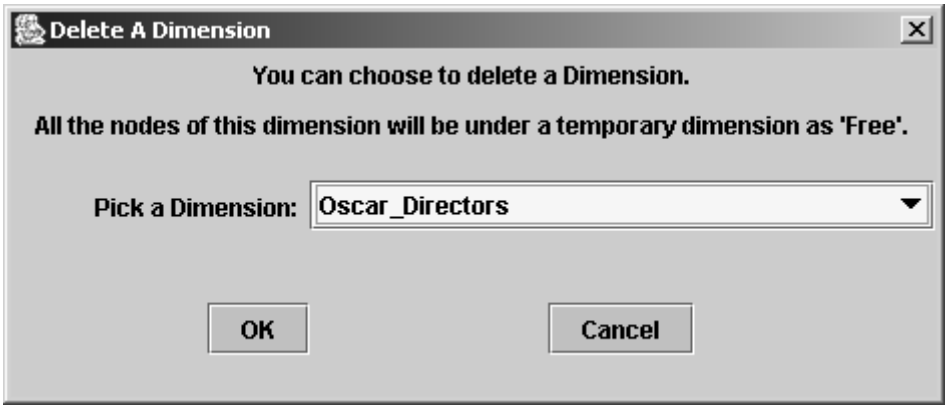
Εικόνα 7.13

Πατάμε το “Yes”, και η αλλαγή έγινε. Μπορούμε να επιβεβαιώσουμε την αλλαγή βλέποντας το παράθυρο:



Εικόνα 7.14

Για να δούμε πως γίνεται και η διαγραφή μιας διάστασης, θα σβήσουμε την διάσταση Oscar_Directors, οι κόμβοι θα μεταφερθούν στην βοηθητική διάσταση Free, τα οποία βήματα θα δείξουμε, και για την επαναφορά του συστήματος σε λογικά πλαίσια, θα ξανα-ορίσουμε την νέα διάσταση Oscar_Directors, και θα μεταφέρουμε τους κόμβους πάλι σε αυτήν. Η διαγραφή γίνεται με το πάτημα του κουμπιού “Delete A Dimension…”, που βγάζει το εξής παράθυρο, στο οποίο επιλέγουμε την διάσταση Oscar_Directors”:

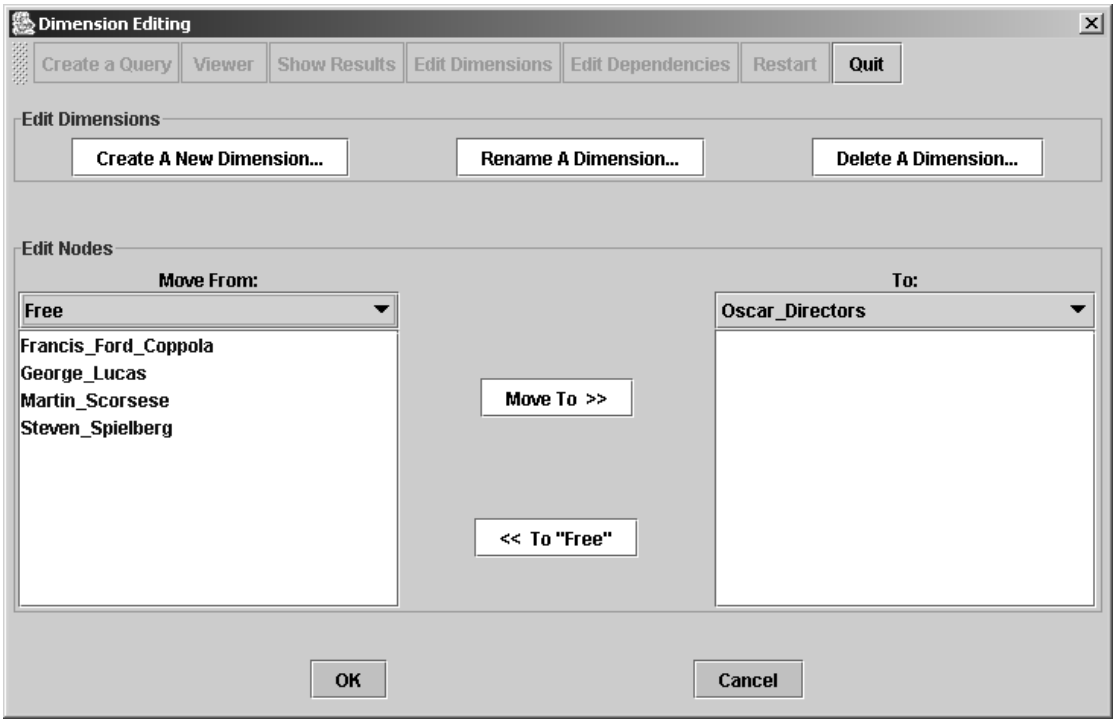


Εικόνα 7.15

Εμφανίζεται ένα παράθυρο επιβεβαίωσης, στο οποίο πατάμε το “Yes”.

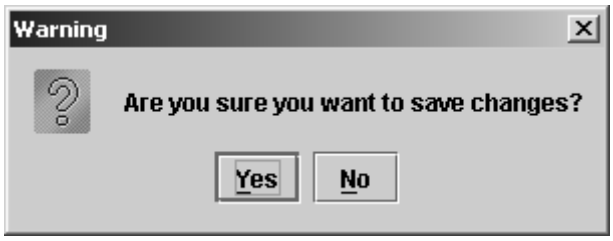
Εικόνα 7.16

Αφού κατασκευάσουμε πάλι την διάσταση Oscar_Directors, μεταφέρουμε τους κόμβους που πήγαν στην βοηθητική διάσταση Free, πίσω στην Oscar_Directors. Για την μεταφορά αυτή, χρησιμοποιούμε το παρακάτω παράθυρο, και κάνουμε την μεταφορά όπως περιγράψαμε προηγουμένως:



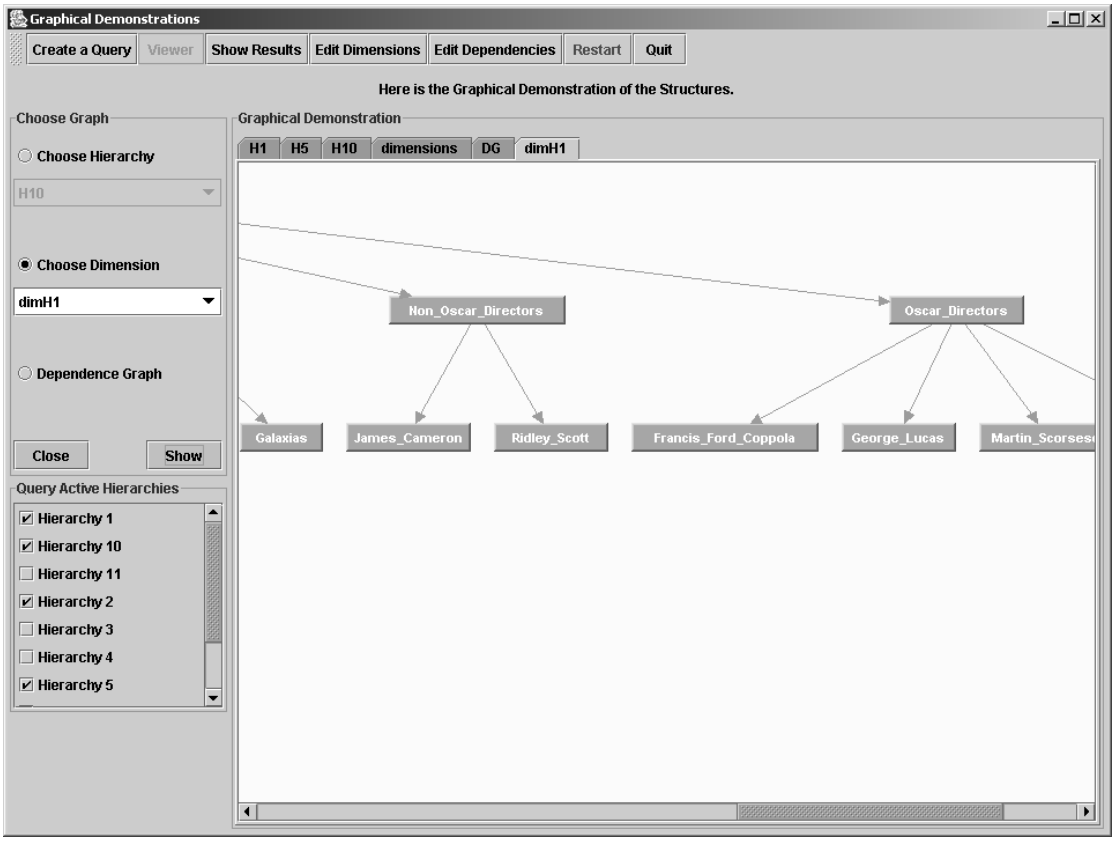
Εικόνα 7.17

Πατάμε το “OK” κάτω αριστερά για να μονιμοποιήσουμε όλες τις αλλαγές που κάναμε, οι οποίες περνάνε και στα αρχεία. Εμφανίζεται το παράθυρο επιβεβαίωσης, όπου πατάμε “Yes”:



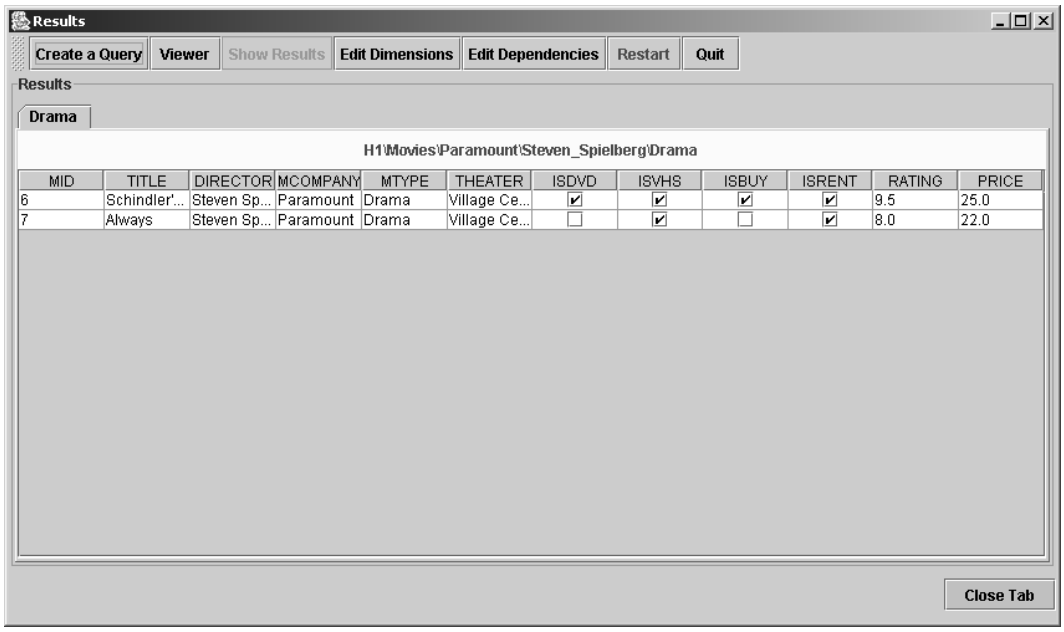
Εικόνα 7.18

Για να επιβεβαιώσουμε τις αλλαγές, μπορούμε να ζητήσουμε να δούμε πως έχει διαμορφωθεί το δένδρο διαστάσεων, πχ. της ιεραρχίας H1. Επιλέγουμε για εμφάνιση την dimH1, και μετακινούμε την μπάρα κύλισης κατάλληλα, για να πάμε στο συγκεκριμένο σημείο. Όντως, βλέπουμε ότι οι αλλαγές έχουν περάσει και στις τοπικές δομές των διαστάσεων.



Εικόνα 7.19

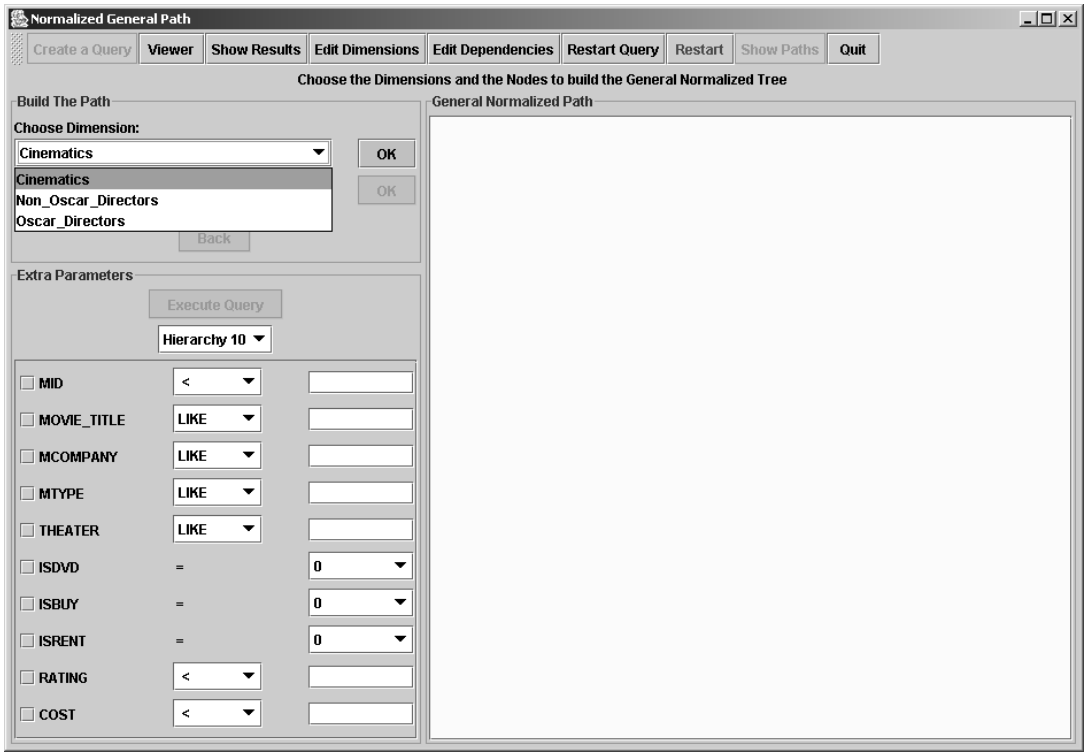
Εάν θέλουμε να δούμε τις εγγραφές που αντιστοιχούν στα φύλλα των ιεραρχιών, τότε πρέπει να κάνουμε 'κλικ' στο κόκκινο κουμπί με το όνομα "DATA" που βρίσκεται κάτω από το φύλλο της επιλογής μας. Έστω ότι επιλέγουμε από την ιεραρχία H1 το φύλλο του μονοπατιού Movies/Paramount/Steven_Spielberg/Drama. Θα εμφανιστεί το παράθυρο με τα αποτελέσματα. Στην προκειμένη περίπτωση υπάρχουν μόνο 2 έγγραφες σε αυτό το μονοπάτι. Στο tab του αποτελέσματος μπορούμε να δούμε με πάνω πάνω το μονοπάτι στο οποίο αντιστοιχεί, το οποίο ξεκινάει από το όνομα της ιεραρχίας. Το tab έχει σαν όνομα την ετικέτα του φύλλου, του οποίο τις εγγραφές επιλέξαμε να δούμε. Να σημειώσουμε ότι το συγκεκριμένο παράδειγμα θα χρησιμοποιηθεί για σύγκριση παρακάτω.



Εικόνα 7.20

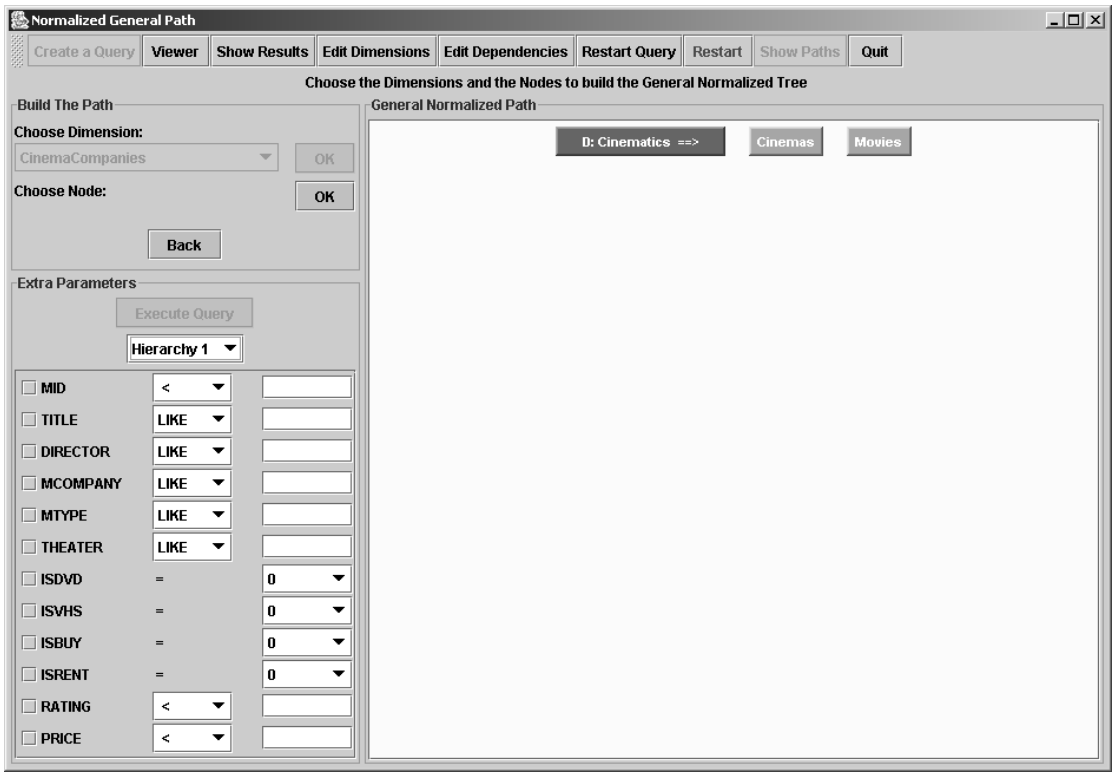
7.2.3 Κατασκευή Ερώτησης

Μπορούμε τώρα να κατασκευάσουμε ένα Μονοπάτι. Θα πατήσουμε το κουμπί “Create Query”, το οποίο θα μας εμφανίσει το παράθυρο, με το οποίο συμπληρώνουμε τις επιλογές μας για την κατασκευή του Μονοπατιού. Σημειώνουμε ότι πρέπει να έχουμε υπ’όψιν μας τον γράφο εξαρτήσεων, όπως αυτός εμφανίζεται στην Εικόνα 7.6.



Εικόνα 7.21

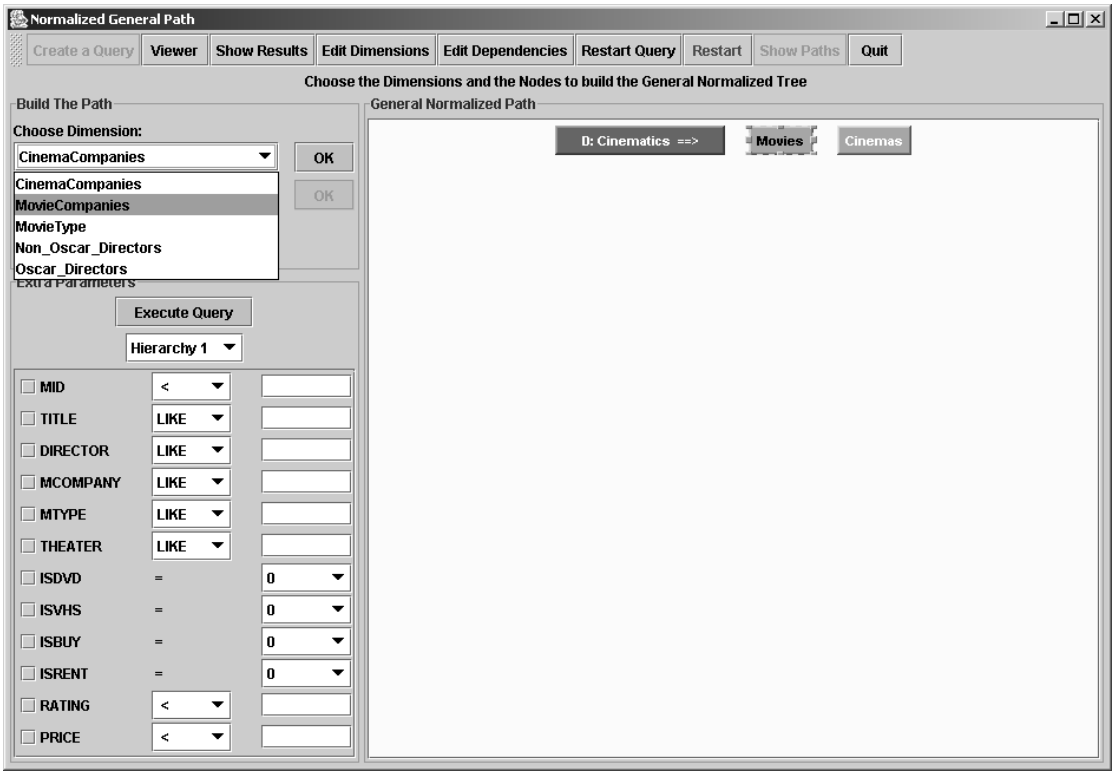
Σε αυτό το παράθυρο επιλέγουμε βήμα βήμα την κατασκευή του Μονοπατιού. Πρώτα απ'όλα παρατηρούμε στο αριστερό κομμάτι του παραθύρου ότι εμφανίζονται τα πεδία του πίνακα της Βάσης Δεδομένων, που αντιστοιχεί στην εκάστοτε ιεραρχία. Έχουμε επιλέξει να δείξουμε τα πεδία του πίνακα της ιεραρχίας H10, γιατί είναι διαφορετικά από αυτά των υπολοίπων ιεραρχιών, τα οποία θα δούμε σε επόμενες εικόνες, τόσο σε αριθμό, όσο και σε όνομα.
Πάνω αριστερά υπάρχει ενεργοποιημένη η επιλογή για τις διαστάσεις. Αυτό το ComboBox περιέχει κάθε φορά τις διαστάσεις που μπορούν να επιλεχθούν, σύμφωνα με το μέχρι εκείνη τη στιγμή κατασκευασμένο Μονοπάτι, και τον γράφο εξαρτήσεων. Οι ανεξάρτητες διαστάσεις, όπως φαίνονται στην Εικόνα 7.21, είναι οι Cinematics, η Non_Oscar_Directors και η Oscar_Directors, με αλφαβητική σειρά. Επιλέγουμε την διάσταση Cinematics, και πατάμε το OK που βρίσκεται δίπλα. Βλέπουμε την απόκριση του συστήματος στην επιλογή μας στην επόμενη εικόνα:



Εικόνα 7.22

Ο πρώτος κόμβος σε κάθε γραμμή είναι το όνομα της διάστασης που επιλέξαμε, και δίπλα παρουσιάζονται αλφαβητικά οι διαθέσιμοι κόμβοι προς επιλογή. Πρέπει να επιλέξουμε έναν από αυτούς, και να πατήσουμε το OK που είναι ενεργοποιημένο αριστερά. Ταυτόχρονα, μπορούμε να παρατηρήσουμε ότι τα πεδία του πίνακα που έχουμε επιλέξει να βλέπουμε, είναι αυτά της ιεραρχίας H1, και μπορούμε να παρατηρήσουμε τις διαφορές με την προηγούμενη

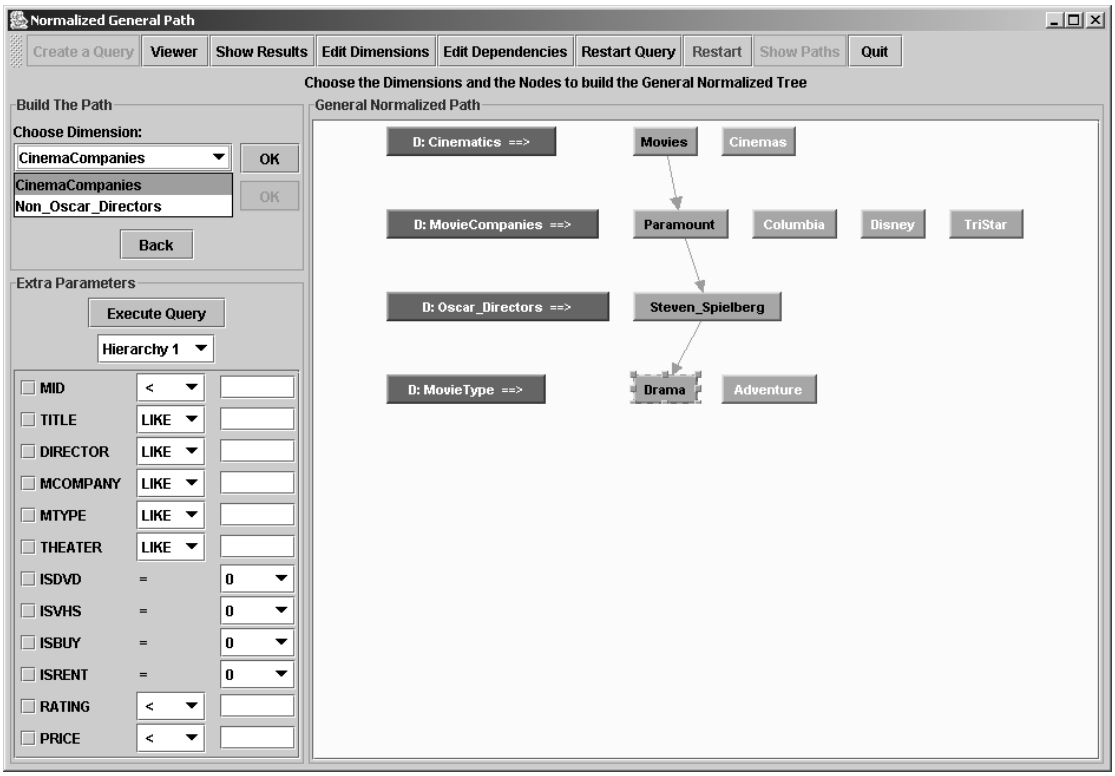
εικόνα. Βλέπουμε λοιπόν την επιλογή για τον πρώτο κόμβο του Μονοπατιού, και δείχνουμε και τις νέες διαθέσιμες διαστάσεις στην επόμενη εικόνα:



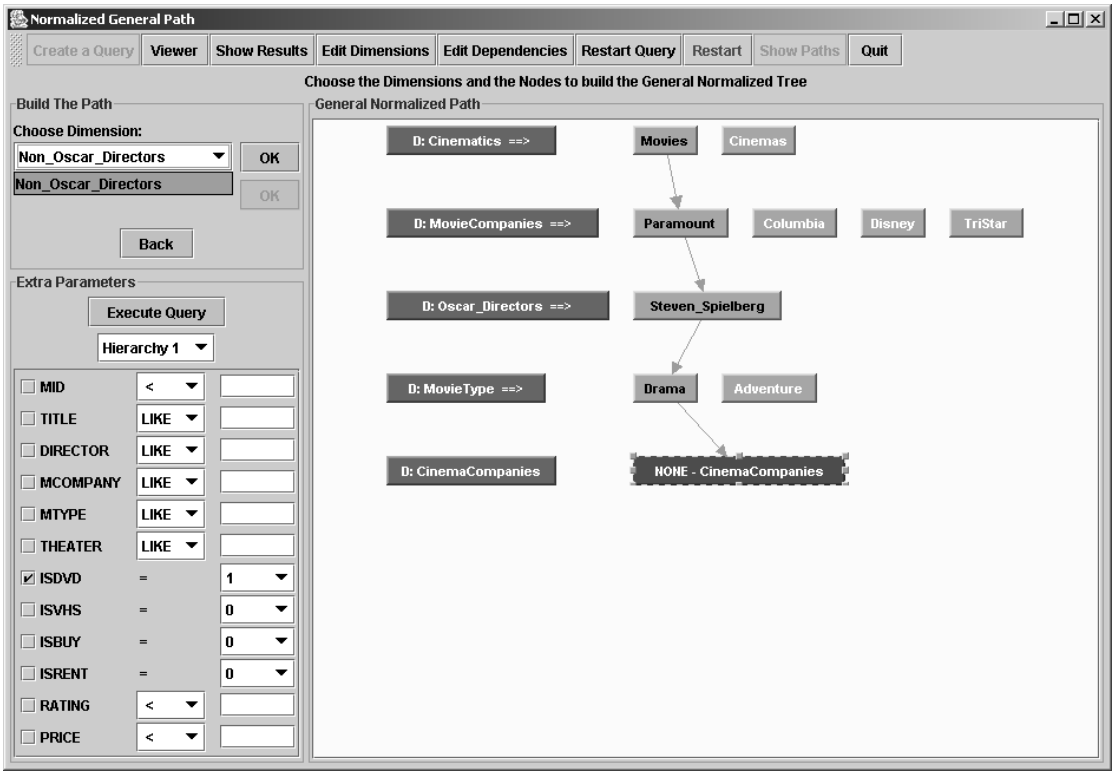
Εικόνα 7.23

Βήμα-βήμα καταλήγουμε στο Μονοπάτι που βλέπουμε στην Εικόνα 7.24. Έχουμε κατασκευάσει το Μονοπάτι Movies/Paramount/Steven_Spielberg/Drama. Παρατηρούμε ότι οι κόμβοι που επιλέγονται έρχονται στην αρχή, και ενώνονται με ακμές με τον πρώτο κόμβο του προηγούμενου επιπέδου. Επίσης σημειώνουμε ότι σε κάθε επίπεδο εμφανίζονται οι κόμβοι που ανήκουν στην επιλεγμένη διάσταση, και οι οποίοι είναι δυνατόν να σχηματίσουν μονοπάτι με τους προηγουμένως επιλεγμένους κόμβους. Αυτό φαίνεται και από το γεγονός ότι δεν εμφανίζονται στην διάσταση Oscar_Directors οι κόμβοι με τα ονόματα των υπολοίπων σκηνοθετών που ανήκουν σε αυτή την διάσταση.

Στις μέχρι τώρα επιλογές μας δεν έτυχε να χρειαστεί να μπει κόμβος Δ, όπως τον ορίσαμε στην Θεωρητική Μελέτη, στο Κεφάλαιο 3. Για να δείξουμε πως παρουσιάζεται αυτό, επιλέγουμε τώρα να γίνει επέκταση ως προς την διάσταση Cinema_Companies (η οποία ξέρουμε ότι θα προκαλέσει κόμβο Δ, γιατί δεν υπάρχει σε καμιά ιεραρχία μονοπάτι που να περιέχει τις μέχρι τώρα επιλογές μας, και κάποιον από τους κόμβους που ανήκουν σε αυτή την διάσταση, δηλαδή έναν από τους Village_Center, Athinaion, Galaxias ή Village_Park). Η Εικόνα 7.25 δείχνει την εισαγωγή ενός κόκκινου κόμβου, που αναπαριστά τον κόμβο Δ .



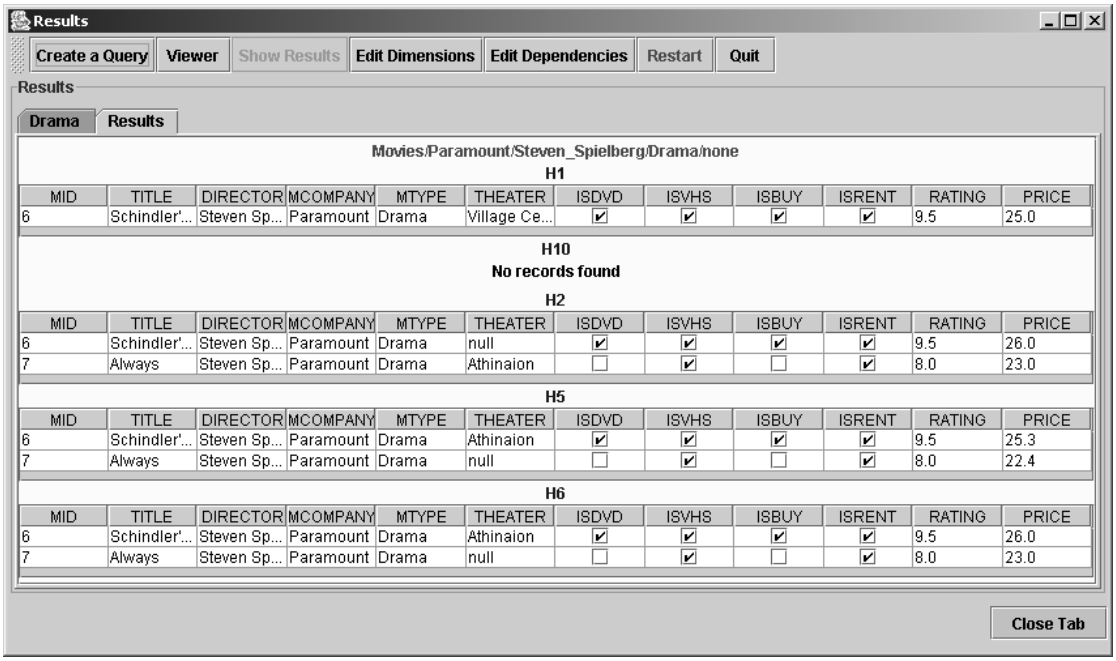
Εικόνα 7.24



Εικόνα 7.25

Στην Εικόνα 7.25, επίσης, έχουμε επιλέξει τον κόκκινο κόμβο, για να μπορούμε να επιλέξουμε περαιτέρω διαστάσεις, ή να πατήσουμε το κουμπί “Execute Query”, που θα

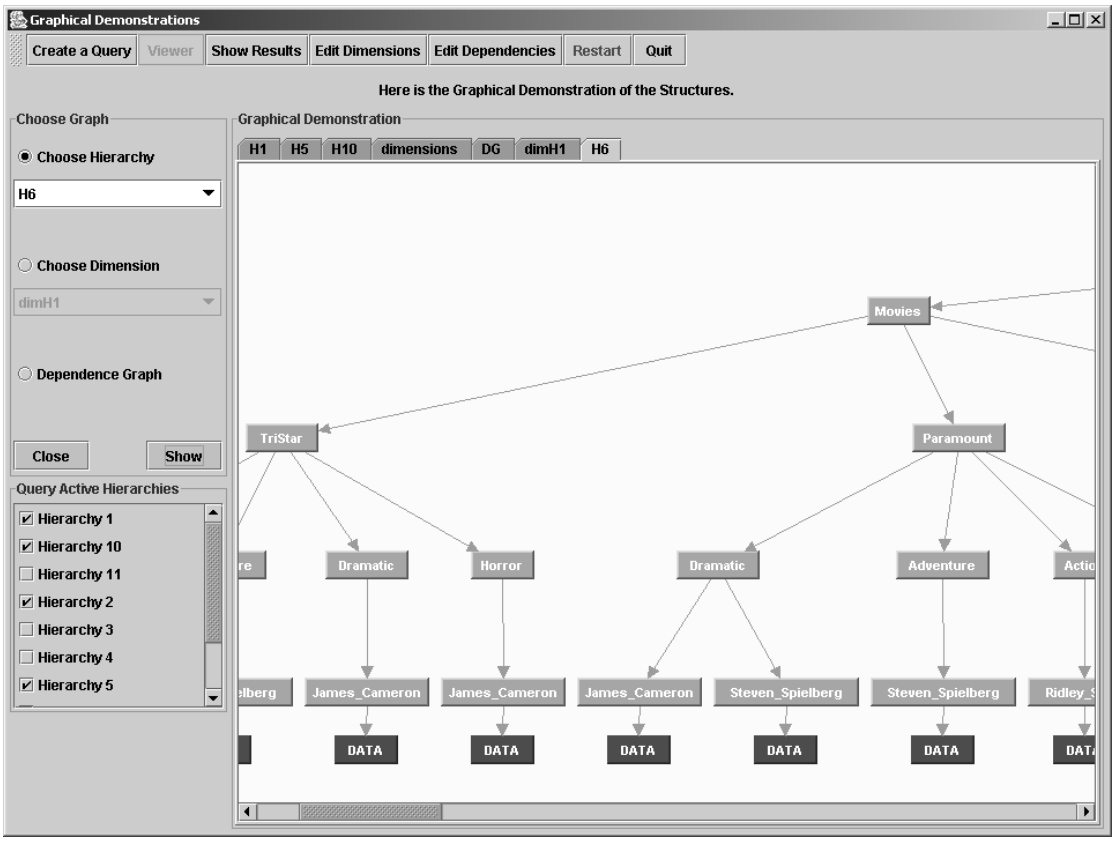
εκτελέσει το query. Να σημειώσουμε ότι στο panel που φαίνονται τα πεδία του πίνακα της ιεραρχίας H1 έχουμε επιλέξει να επιστρέψουν οι εγγραφές που έχουν το πεδίο 'ISDVD' ίσο με 1. Αυτό, για να συγκρίνουμε τα αποτελέσματα με αυτά που επιστρέφουν για την ιεραρχία H1 στην Εικόνα 7.20, όπου στην ουσία ζητάμε εγγραφές από το ίδιο μονοπάτι της H1. Τα αποτελέσματα θα δείχνουν ως εξής:



Εικόνα 7.26

Πάνω πάνω βλέπουμε την σειρά της επιλογής των κόμβων που έγινε για την κατασκευή του Μονοπατιού. Βλέπουμε ότι ο τελευταίος κόμβος έχει το όνομα 'none', και αντιπροσωπεύει τον κόμβο Δ στο Μονοπάτι. Παρατηρούμε ότι όντως τα αποτελέσματα της ιεραρχίας H1 δεν είναι τα ίδια, και ότι αν και στις ιεραρχίες H5 και H6 η δομή τους δεν ταιριάζει με αυτήν της ιεραρχίας H1 (σύμφωνα με την οποία κατασκευάστηκε και το Μονοπάτι), η ερώτηση αποτιμήθηκε σωστά. Η δομή της ιεραρχίας H6 φαίνεται στην επόμενη εικόνα, και είναι διαφορετική τόσο από την H1, όσο και από την H5.

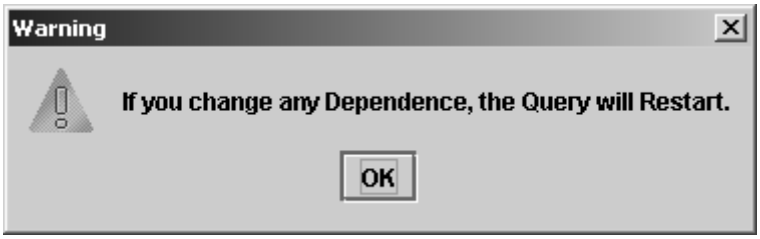
Η ιεραρχία H10 δεν επέστρεψε εγγραφές, γιατί δεν έχει κόμβο με το όνομα Steven_Spielberg, και δεν κατάφερε να ταυτιστεί με το Μονοπάτι κανένα από τα μονοπάτια αυτής της ιεραρχίας. Αντίθετα, η ιεραρχία H2 επέστρεψε αποτελέσματα. Αν και δεν την δείξαμε, η H2 έχει την ίδια δομή με την H1, και περιττεύει η παρουσίασή της.



Εικόνα 7.27

7.2.4 Διαχείριση του Γράφου Εξαρτήσεων

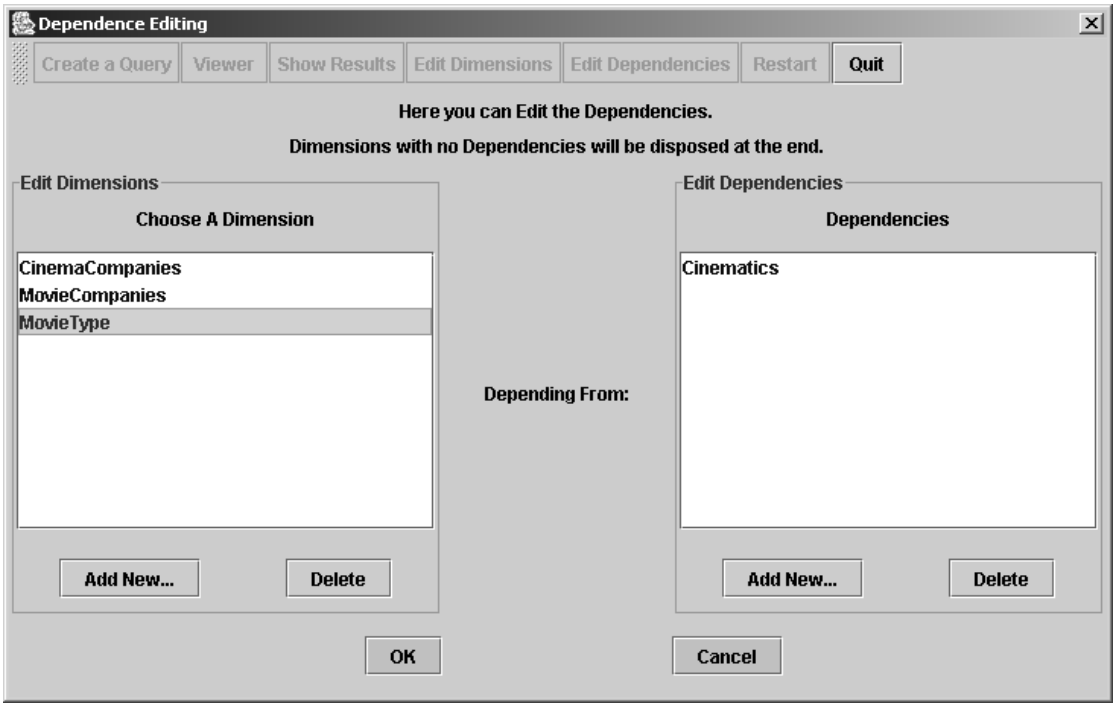
Προκειμένου να κάνουμε σαφή την λειτουργία του γράφου εξαρτήσεων, θα προσθέσουμε μια (συγκεκριμένη) εξάρτηση και θα εφαρμόσουμε το ίδιο Μονοπάτι για τις ίδιες ιεραρχίες, και θα συγκρίνουμε τα αποτελέσματα. Καταρχήν μεταφερόμαστε στο παράθυρο που χειριζόμαστε τον γράφο εξαρτήσεων πατώντας το κουμπί “Edit Dependencies” από οποιοδήποτε παράθυρο. Εάν το πατήσουμε από το παράθυρο που κατασκευάζουμε το Μονοπάτι, θα πάρουμε το εξής μήνυμα, το οποίο μας προειδοποιεί ότι αν αλλάξουμε εξαρτήσεις, θα σβηστεί το υπάρχον query:



Εικόνα 7.28

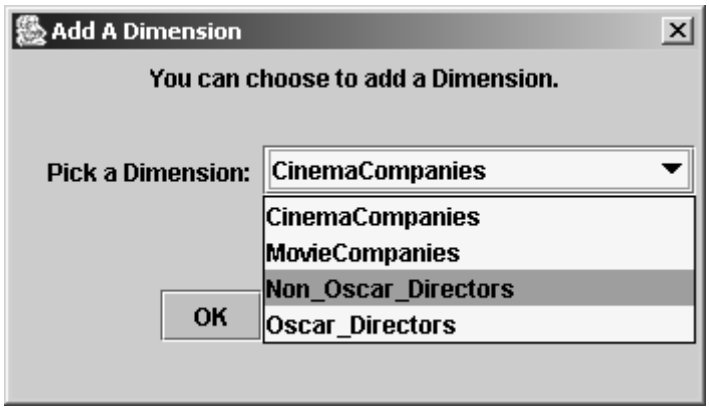
Εμφανίζεται το παράθυρο που φαίνεται στην Εικόνα 7.29. Αριστερά βρίσκονται τα ονόματα των διαστάσεων που έχουν εξάρτηση, και αν επιλέξουμε κάποια από αυτές, θα δούμε δεξιά

τις διαστάσεις από τις οποίες εξαρτούνται. Έχουμε επιλέξει την διάσταση ‘Movie_Type’, η οποία εξαρτάται από την διάσταση ‘Cinematics’.



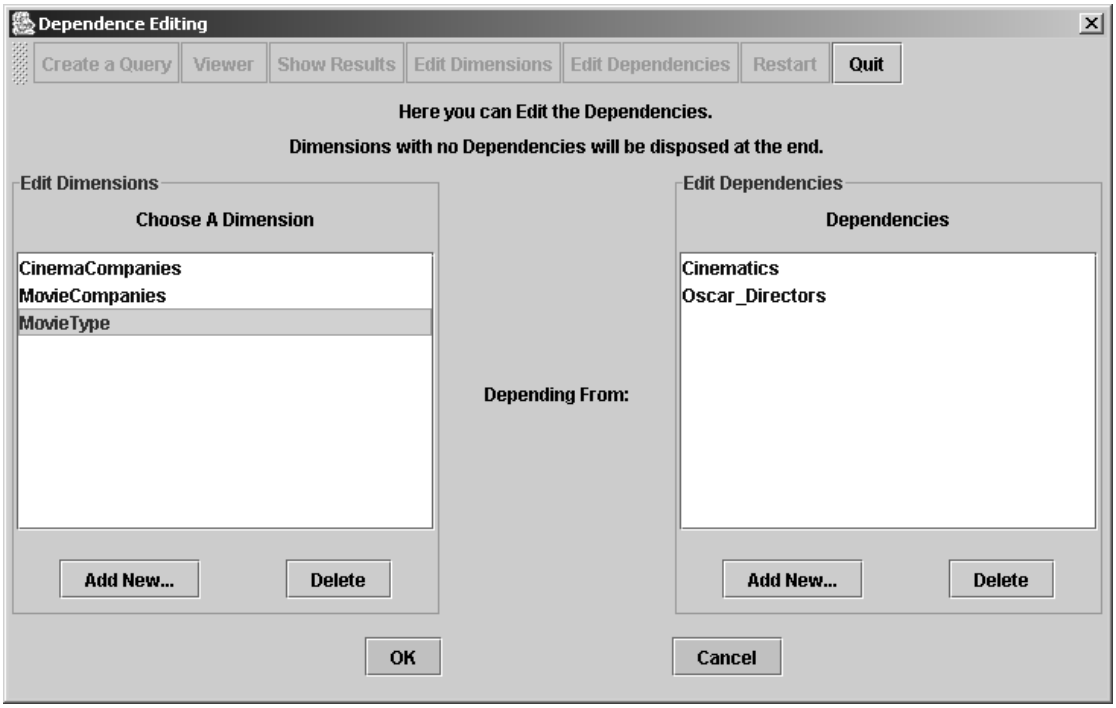
Εικόνα 7.29

Εάν θέλουμε να προσθέσουμε μια εξάρτηση ενός ανεξάρτητου προς το παρόν κόμβου, πατάμε το κουμπί “Add New…” που βρίσκεται στο αριστερό πλαίσιο της οθόνης, κάτι που θα δούμε παρακάτω. Η αλλαγή που θα κάνουμε στον γράφο εξαρτήσεων δεν είναι τέτοια, αλλά απλά θα προσθέσουμε δύο επιπλέον εξαρτήσεις στην διάσταση ‘Movie_Type’. Επιλέγουμε αριστερά την διάσταση ‘Movie_Type’ και πατάμε στο δεξί πλαίσιο το κουμπί “Add New…”, και εμφανίζεται το εξής παράθυρο:



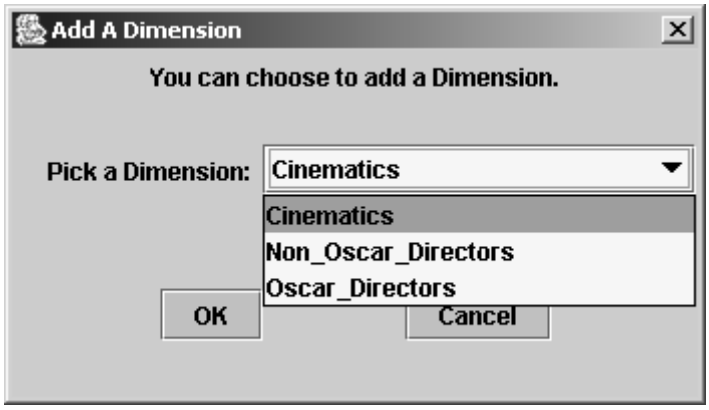
Εικόνα 7.30

Σε αυτό το παράθυρο επιλέγουμε μια από τις διαθέσιμες διαστάσεις για δημιουργία εξάρτησης από αυτές. Δεν εμφανίζονται οι διαστάσεις από τις οποίες υπάρχει άμεση εξάρτηση, όπως στην προκειμένη περίπτωση η διάσταση 'Cinematics'. Έτσι, βάζουμε εξάρτηση από την διάσταση 'Oscar_Directors', και το βλέπουμε στο παράθυρο ως εξής:



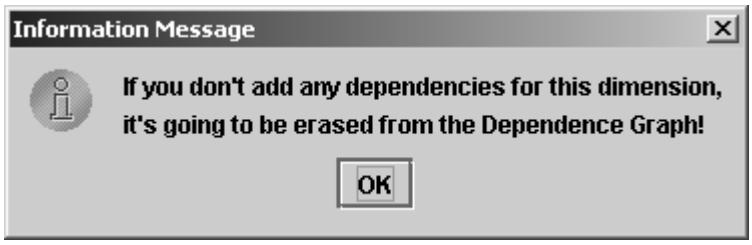
Εικόνα 7.31

Εάν προσπαθήσουμε να δημιουργήσουμε στον γράφο εξαρτήσεων κύκλο, θα εμφανιστεί ένα μήνυμα αποτυχίας, γιατί κάτι τέτοιο δεν επιτρέπεται. Για δοκιμή, θα δοκιμάσουμε να βάλουμε μια εξάρτηση της διάστασης 'Cinematics' από την 'Movie_Type', που δημιουργεί κύκλο. Πατώντας αριστερά το "Add New…" προσθέτουμε αριστερά την διάσταση 'Cinematics', όπως φαίνεται από την Εικόνα 7.32:



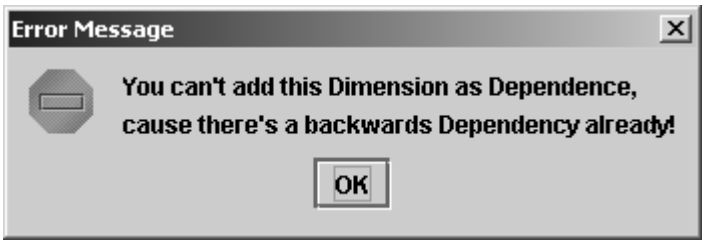
Εικόνα 7.32

Έτσι εμφανίζεται το εξής παράθυρο, που μας προειδοποιεί ότι αν δεν προσθέσουμε εξαρτήσεις για αυτή την διάσταση, τότε θα σβηστεί από τον γράφο εξαρτήσεων.



Εικόνα 7.33

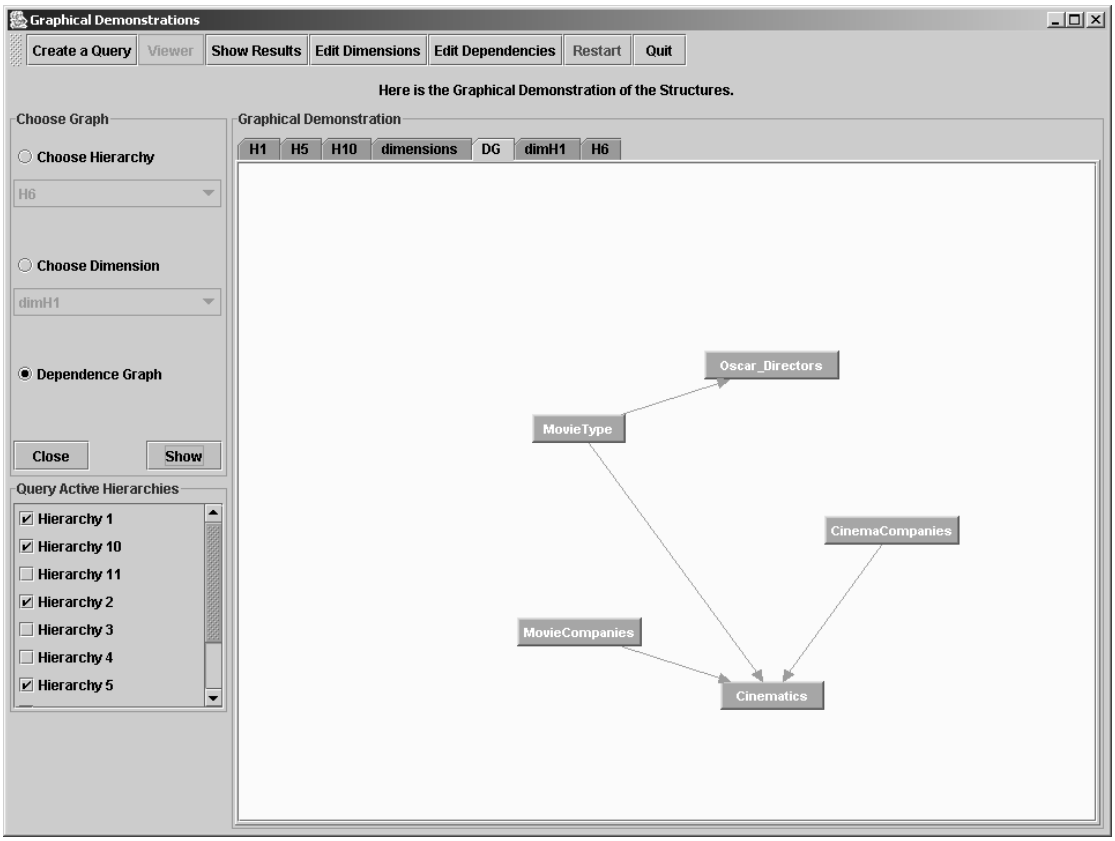
Επιλέγουμε OK, και στην Εικόνα 7.32 βλέπουμε την νέα διάσταση αριστερά. Την επιλέγουμε, και ξεκινάμε την διαδικασία για την προσθήκη εξάρτησης, όπως περιγράψαμε παραπάνω. Στο επόμενο παράθυρο (παρόμοιο με την Εικόνας 7.30) επιλέγουμε την διάσταση 'Movie_Type', που δημιουργεί κύκλο, και παίρνουμε το εξής μήνυμα:



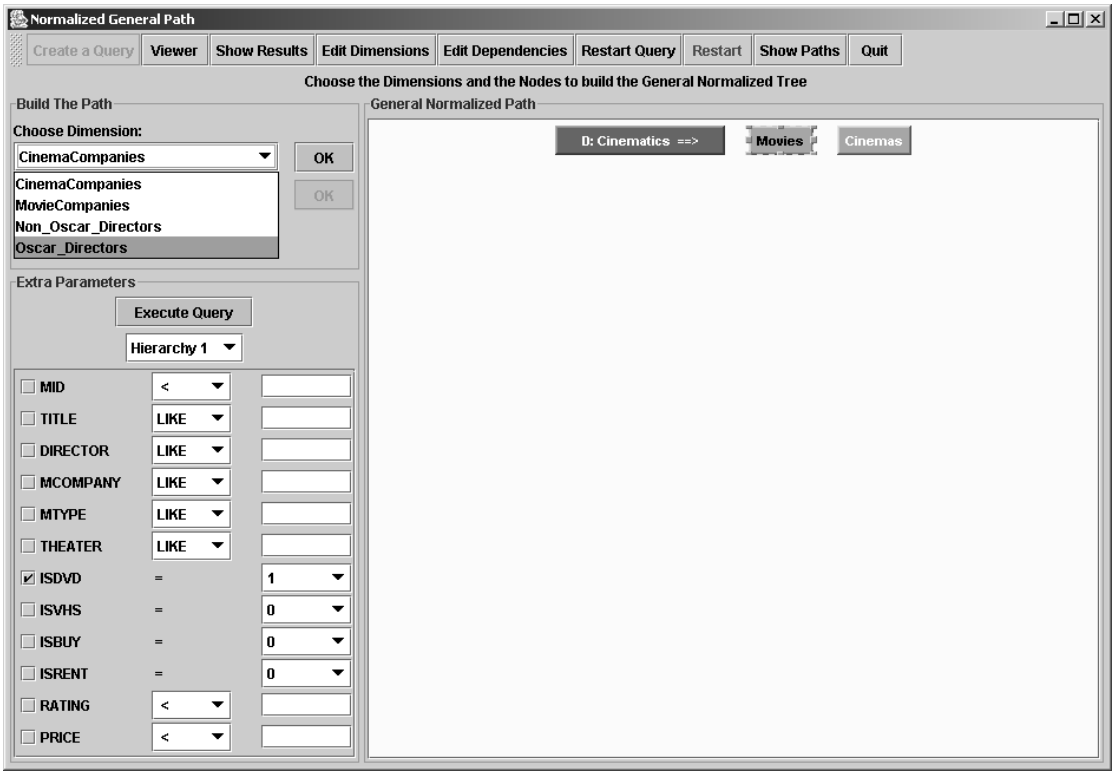
Εικόνα 7.34

Αυτό συμβαίνει και στην περίπτωση που ο κύκλος προκύπτει από έμμεσες εξαρτήσεις. Πατάμε τώρα το OK στο κεντρικό παράθυρο, ώστε να περάσουν οι αλλαγές και στα αρχεία, και να γίνουν μόνιμες. Μπορούμε τώρα να δούμε το νέο σχήμα του γράφου εξαρτήσεων, όπως φαίνεται στην Εικόνα 7.35. Βλέπουμε την προσθήκη του νέου κόμβου 'Oscar_Directors' και την εξάρτηση της διάστασης 'Movie_Type' από αυτόν με το νέο βέλος.

Μπορούμε τώρα να κατασκευάσουμε το ίδιο Μονοπάτι όπως πριν, και να συγκρίνουμε τα αποτελέσματα. Κατά την κατασκευή, παρατηρούμε ότι η διάσταση 'Movie_Type' δεν προτείνεται για επιλογή, πριν επιλεχθούν και οι δύο διαστάσεις από τις οποίες εξαρτάται. Βλέπουμε ένα παράδειγμα κατά την επιλογή διάστασης στο δεύτερο βήμα, στην Εικόνα 7.36, όπου δεν υπάρχει προς επιλογή η διάσταση 'Movie_Type'.

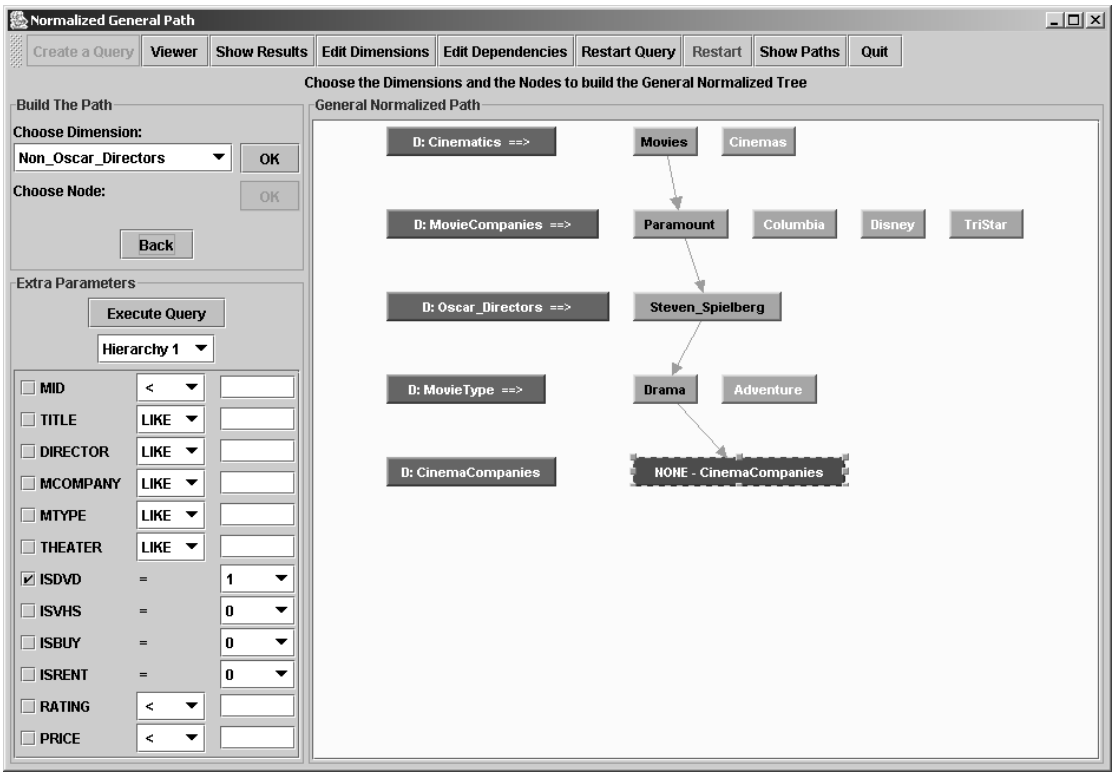


Εικόνα 7.35



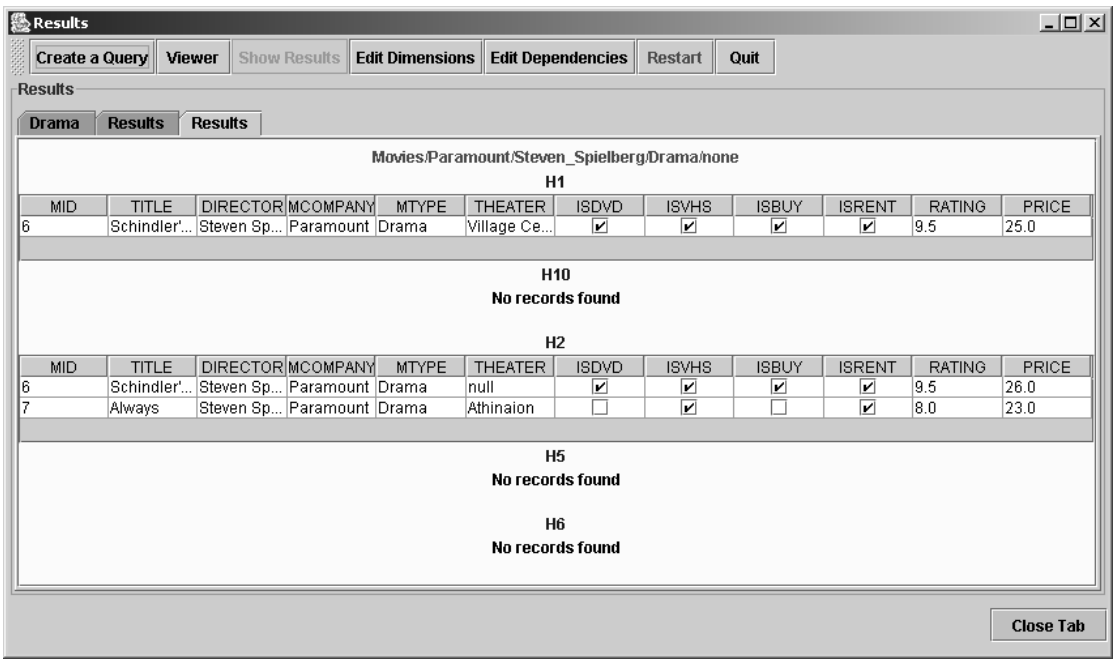
Εικόνα 7.36

Συνεχίζουμε και κατασκευάζουμε το ίδιο Μονοπάτι, κάτι που φαίνεται στην Εικόνα 7.37:



Εικόνα 7.37

Πρόκειται για ακριβώς το ίδιο Μονοπάτι. Πατώντας το κουμπί για την εκτέλεσή του, βλέπουμε τα εξής αποτελέσματα:

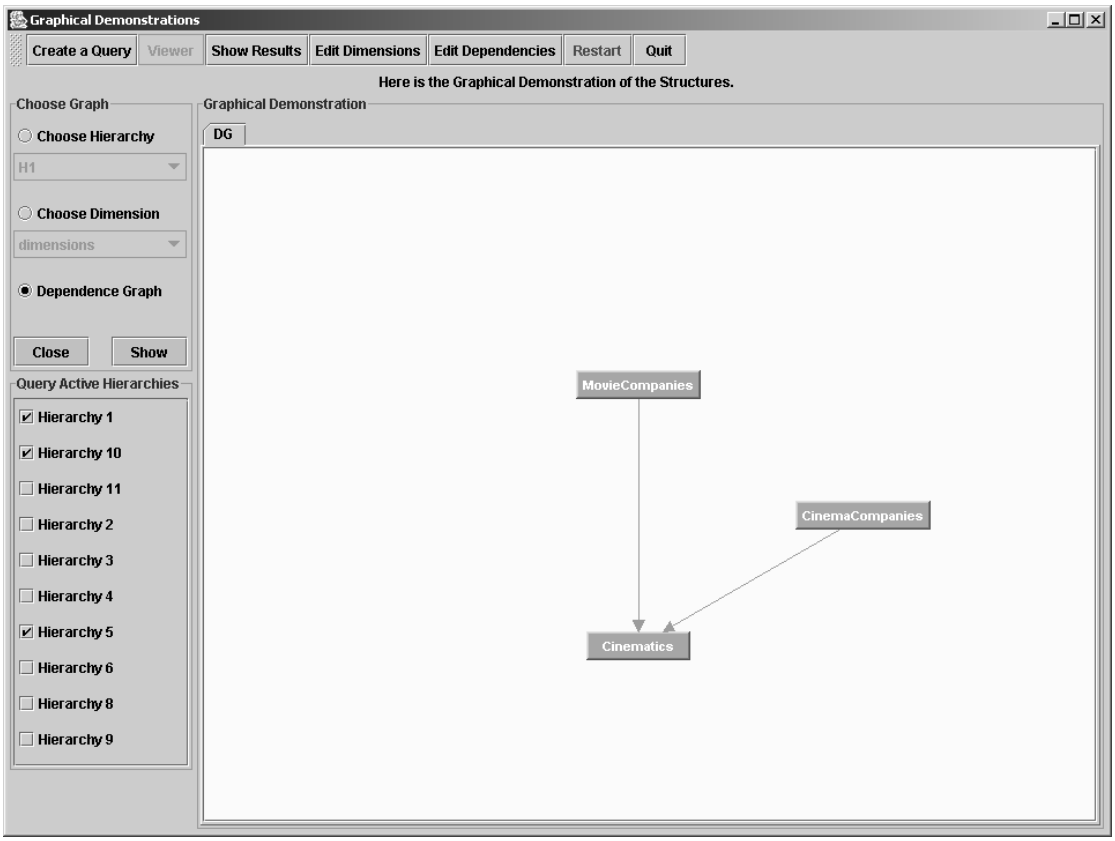


Εικόνα 7.38

Παρατηρούμε ότι εκτός από την ιεραρχία H10, για την οποία εξηγήσαμε παραπάνω γιατί δεν επιστρέφει αποτελέσματα, ούτε η H5, ούτε η H6 επιστρέφουν αποτελέσματα. Αυτό, γιατί ο αλγόριθμος που αναπροσαρμόζει την σειρά επιλογής των διαστάσεων του Μονοπατιού, προκειμένου να υπολογίσει τα ομόλογα Μονοπάτια που θα αποτιμηθούν στις ιεραρχίες, δεν μπορεί να φέρει την επιλογή του κόμβου 'Drama' πριν την επιλογή του κόμβου 'Steven_Spielberg', γιατί δεν το επιτρέπει ο γράφος εξαρτήσεων. Όμως, αν δούμε από στις Εικόνες 7.3 και 7.27, και στις δύο περιπτώσεις ο κόμβος με το όνομα 'Drama' βρίσκεται πάνω από τον κόμβο 'Steven Spielberg'. Έτσι, δεν μπόρεσε να κατασκευαστεί ένα Μονοπάτι που να ταιριάζει με την δομή αυτών των δύο ιεραρχιών. Με αυτό το παράδειγμα γίνεται σαφής η λειτουργία του γράφου εξαρτήσεων.

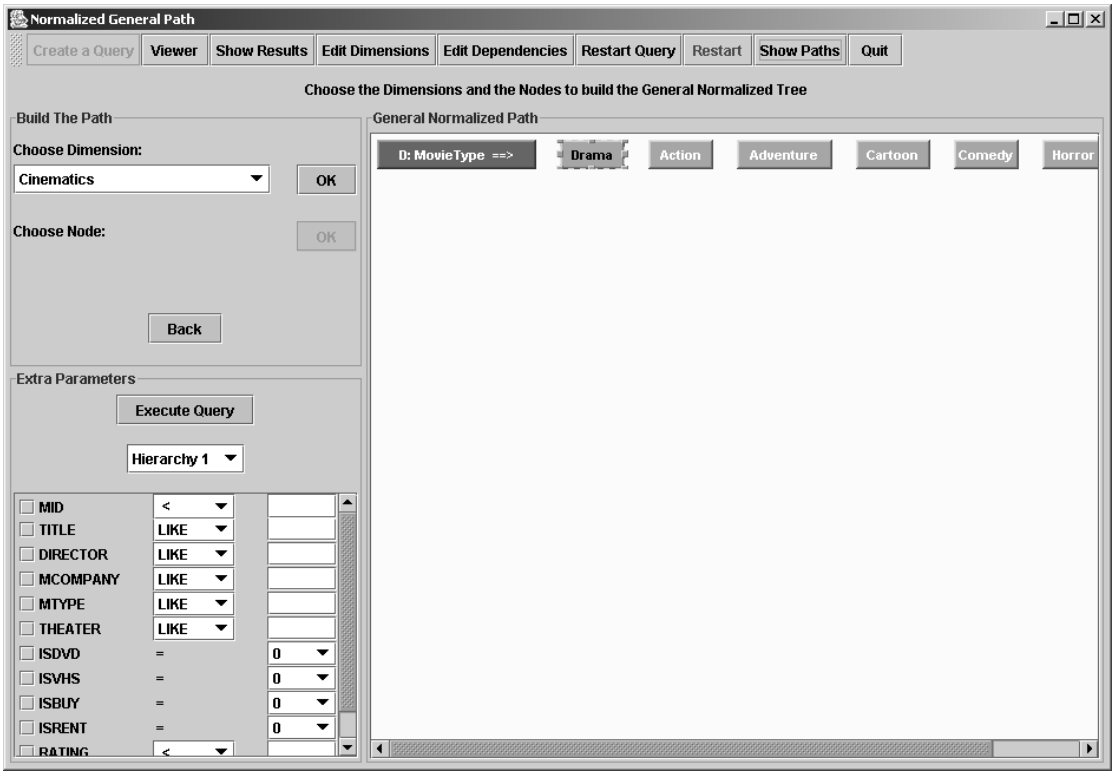
7.2.5 Αναζήτηση Μονοπατιών

Τέλος, θα δείξουμε πως το σύστημα αναζητά μονοπάτια στις ιεραρχίες, όταν το Μονοπάτι του χρήστη δεν έχει δοθεί πλήρως, αλλά έχει επιλεγεί μόνο 1 κόμβος. Καταρχήν, σβήνουμε την εξάρτηση της διάστασης 'Movie_Type' από τις διαστάσεις 'Cinematics' και 'Oscar_Directors'. Έτσι ο γράφος εξαρτήσεων θα έχει την μορφή της εικόνας 7.39:



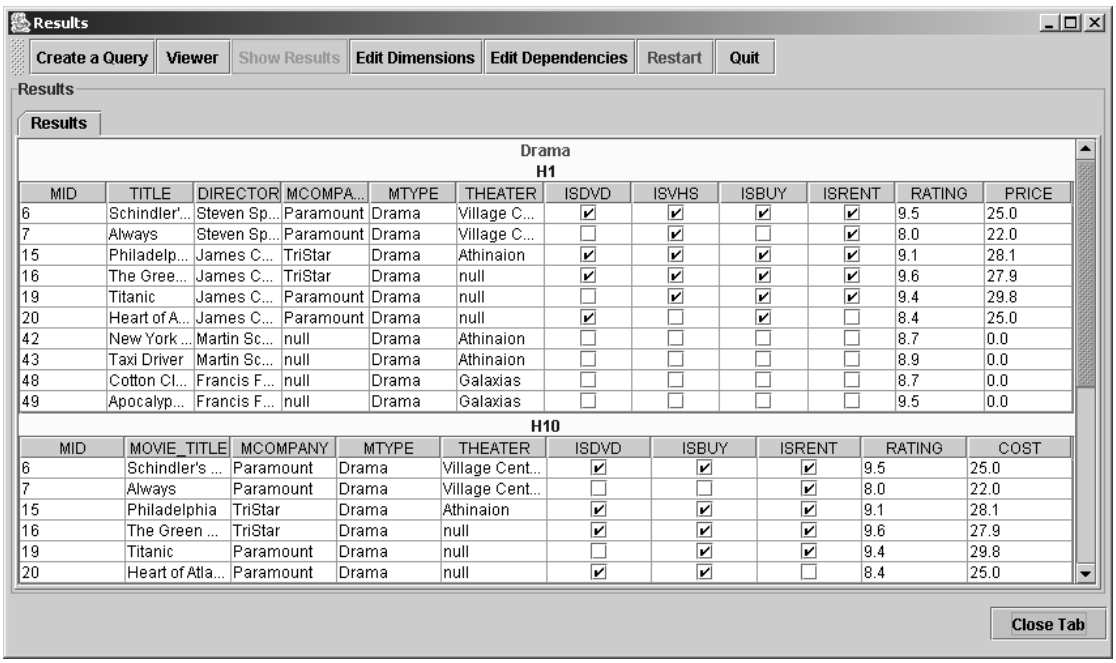
Εικόνα 7.39

Επίσης, θέτουμε ως ενεργές ιεραρχίες τις H1, H5 και H10, όπως φαίνεται και στη Εικόνα 7.39. Μετά κατασκευάζουμε το Μονοπάτι, δηλαδή επιλέγουμε μόνο μια διάσταση και έναν κόμβο, τον 'Drama', ως εξής:



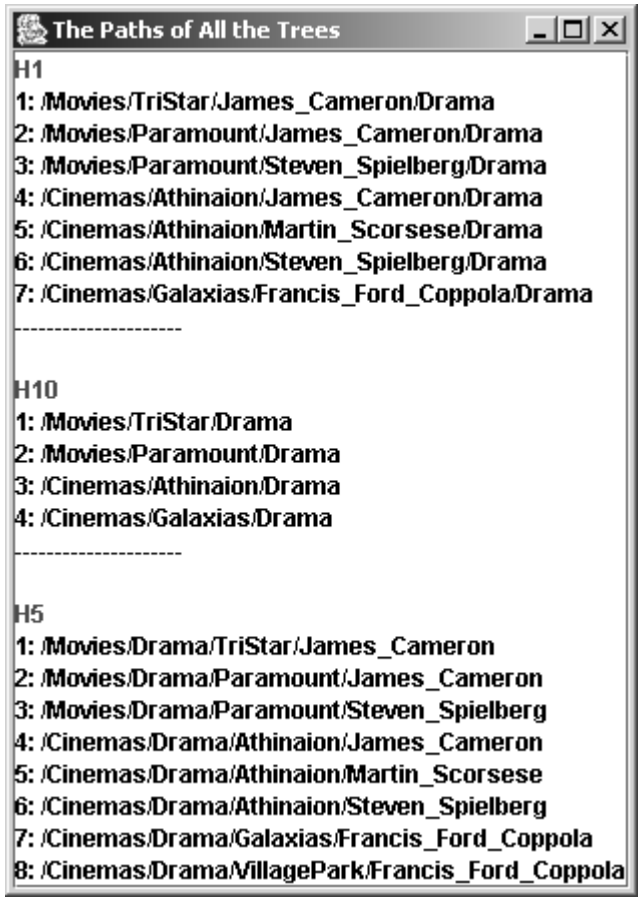
Εικόνα 7.40

Πατάμε το "Execute Query" και παίρνουμε τα ακόλουθα αποτελέσματα:



Εικόνα 7.41

Να σημειώσουμε ότι παρακάτω στην εικόνα, αν και δεν φαίνεται, υπάρχουν και τα αποτελέσματα για την ιεραρχία H5. Τώρα μπορούμε να πατήσουμε το κουμπί “Show Paths” που βρίσκεται στο γραφικό παράθυρο, στο οποίο κατασκευάζουμε το Μονοπάτι. Με αυτό το κουμπί θα δούμε τα μονοπάτια των ιεραρχιών που έδωσαν αποτελέσματα. Στην συγκεκριμένη περίπτωση, θα δούμε όλα τα μονοπάτια των ιεραρχιών H1, H5 και H10 που περιέχουν κόμβο με την ετικέτα ‘Drama’. Αυτά φαίνονται στη επόμενη εικόνα:



Εικόνα 7.42

Με αυτό το τελευταίο παράδειγμα τελειώσαμε την όλη διαδικασία ελέγχου, και γίνεται σαφής η λειτουργία του συστήματος. Καταλήγουμε έτσι στο συμπέρασμα ότι το σύστημα δουλεύει σωστά, σύμφωνα και με τα αναμενόμενα αποτελέσματα.

8

Επίλογος

Με το κεφάλαιο αυτό ολοκληρώνουμε την παρούσα διπλωματική εργασία. Ακολουθεί μια σύντομη ανασκόπηση, και αναφέρονται παρατηρήσεις και προτάσεις για πιθανές επεκτάσεις και βελτιώσεις πάνω στην εργασία αυτή.

8.1 Σύνοψη και Συμπεράσματα

Το διαδίκτυο χρησιμοποιείται ευρέως ως τόπος εύρεσης πληροφορίας, γύρω από ποικίλα θέματα. Γεγονός είναι ότι δεν έχει αναπτυχθεί ένας ενιαίος τρόπος οργάνωσης του μεγάλου όγκου πληροφορίας που υπάρχει διαθέσιμη στο διαδίκτυο. Το πρόβλημα έρχεται στην επιφάνεια, κυρίως όταν ένας χρήστης θελήσει να αναζητήσει πληροφορίες για ένα συγκεκριμένο τομέα γνώσης, χρησιμοποιώντας πολλές διαφορετικές πηγές. Επειδή η κάθε πηγή είναι ανεξάρτητη και αυτόνομη, και επειδή οι ανάγκες λειτουργίας διαφέρουν από πηγή σε πηγή, είναι σύνηθες η κάθε πηγή να ταξινομεί την πληροφορία που παρέχει με διαφορετικό τρόπο από άλλες πηγές.

Θα ήταν πολύ χρήσιμο αν ένας χρήστης μπορούσε να αναζητήσει τέτοιες πληροφορίες, από διάφορες πηγές, χωρίς αυτή η διαφορετική οργάνωση στην κάθε πηγή να είναι ανασταλτικός παράγοντας. Για το λόγο αυτό, έχουν αναπτυχθεί διάφορες τεχνικές που προσπαθούν να

επιλύσουν το πρόβλημα αυτό, συνήθως συγχωνεύοντας τους καταλόγους από τις διάφορες πηγές σε ένα καθολικό σχήμα, μέσα από το οποίο ο χρήστης κατασκευάζει τις ερωτήσεις του, και οι οποίες, σύμφωνα με κάποιους κανόνες αντιστοίχισης, αποτιμώνται στις διάφορες πηγές.

Στην διπλωματική αυτή εργασία, προκειμένου να λύσουμε το προαναφερθέν πρόβλημα, **προσθέσαμε σημασιολογική γνώση** πάνω στις ιεραρχίες των πηγών. Ορίσαμε τις έννοιες των διαστάσεων και του γράφου εξαρτήσεων, που αποτελούν την σημασιολογική γνώση που προστίθεται στις ιεραρχίες, και το καθολικό κανονικοποιημένο δένδρο, το οποίο αποτελεί τον μηχανισμό, με τον οποίο ο χρήστης κατασκευάζει την ερώτηση που θέλει να απευθύνει στις ιεραρχίες των διαφόρων πηγών, αποκρύπτοντάς του την οργάνωση της πληροφορίας στην κάθε ιεραρχία. Το θεωρητικό υπόβαθρο στο οποίο βασιστήκαμε, παρουσιάστηκε στο Κεφάλαιο 3.

Το σύστημα που υλοποιήσαμε, του οποίου πληροφορίες για τις απαιτήσεις και τις λειτουργίες μπορούμε να βρούμε στο Κεφάλαιο 4, σκοπό έχει να υλοποιήσει και να εφαρμόσει τις δομές που αναφέραμε θεωρητικά, για να προσφέρει στον χρήστη την δυνατότητα να κατασκευάζει ερωτήσεις, χωρίς να έχει ιδιαίτερη επίγνωση της δομής της κάθε ιεραρχίας. Το επόμενο βήμα ήταν η υλοποίηση ενός μηχανισμού, ο οποίος να καθιστά τις ερωτήσεις του χρήστη ικανές να απαντηθούν σε όλες τις ιεραρχίες, και να εφαρμόζει την αποτίμησή τους, επιστρέφοντας τα αποτελέσματα. Ο χρήστης δεν χρειάζεται να γνωρίζει λεπτομέρειες για τον μηχανισμό αυτό.

Στο Κεφάλαιο 5 μπορούμε να βρούμε λεπτομέρειες για την σχεδίαση του συστήματος, για τα μέρη που το απαρτίζουν, και τον τρόπο υλοποίησης αυτών στην πράξη, κυρίως όσον αφορά τις δομές στην μνήμη, με τις οποίες χειριζόμαστε τα θεωρητικά μοντέλα που αναφέραμε στο Κεφάλαιο 3.

Στο Κεφάλαιο 6 μπορούμε να βρούμε όλες τις λεπτομέρειες που αφορούν το προγραμματιστικό κομμάτι της υλοποίησης του συστήματος. Περιγράφονται οι βασικοί αλγόριθμοι, η λειτουργία της Βάσης Δεδομένων, και κυρίως αναφέρονται όλες οι κλάσεις που υλοποιήθηκαν, με τα απαραίτητα σχόλια για τις διάφορες λειτουργίες των μεθόδων τους, και επεξηγήσεις για την χρήση των πεδίων τους.

Τέλος, στο Κεφάλαιο 7 μπορούμε να δούμε πως λειτουργεί το σύστημα, όσον αφορά το γραφικό περιβάλλον διεπαφής χρήστη, και να δοκιμάσουμε την λειτουργία του, βάζοντας ελεγχόμενη είσοδο, και συγκρίνοντας τα αποτελέσματα με τα αναμενόμενα. Στην διάρκεια του ελέγχου αυτού, μπορούμε να διερευνήσουμε και να δούμε όλες τις λειτουργίες που προσφέρει το σύστημα, καθώς και την επίδραση των σημασιολογικών εννοιών, που προσδώσαμε στις ιεραρχίες, στα τελικά αποτελέσματα.

8.2 Μελλοντικές Επεκτάσεις

Στην ενότητα αυτή θα αναφερθούμε σε διάφορες παρατηρήσεις που κάναμε κατά την εκπόνηση αυτής της διπλωματικής εργασίας, και σε προτάσεις για δυνατές βελτιώσεις και επεκτάσεις πάνω σε αυτή. Ακολουθεί μια λίστα με αυτές:

- Επέκταση των ιεραρχιών, και κατάλληλη τροποποίηση του συστήματος, ώστε να μπορούμε να χειρισθούμε πέραν από XML ιεραρχίες, και RDF σχήματα, τα οποία χρησιμοποιούνται πλέον ευρέως.
- Επέκταση σε πιο σύνθετα σχήματα XML, με την χρήση των attributes και άλλων στοιχείων, ή με συγκεκριμένα DTDs για τις ιεραρχίες.
- Εισαγωγή και διαγραφή ιεραρχιών μέσα από το σύστημα.
- Εξειδίκευση του γράφου εξαρτήσεων στις τοπικές ιεραρχίες, δηλαδή η συμπλήρωση του Μονοπατιού να γίνεται λαμβάνοντας υπ'όψιν τις εξαρτήσεις μόνον των τοπικών διαστάσεων που υπάρχουν στην κάθε ιεραρχία. Για παράδειγμα, έστω ότι σε μια ιεραρχία υπάρχουν οι διαστάσεις Α, Β και Γ με την σειρά Α/Β/Γ, και οι Β και Γ εξαρτώνται από μια διάσταση Δ που δεν υπάρχει στην συγκεκριμένη ιεραρχία ({Β,Γ}→Δ). Ένα ημιτελές Μονοπάτι, που αποτελείται μόνο από έναν κόμβο της διάστασης Α, δεν μπορεί να αποτιμηθεί σε αυτή την ιεραρχία, γιατί τα Β και Γ δεν μπορούν να επιλεχθούν πριν από το Δ, που όμως δεν υπάρχει σε αυτή την ιεραρχία. Άρα, μονοπάτια σαν το Α/Β/Γ, δεν μπορούν να κατασκευαστούν αυτόματα ξεκινώντας από το Α, γιατί πριν το Β και το Γ πρέπει να υπάρχει το Δ, κάτι που δεν μπορεί να γίνει για την συγκεκριμένη ιεραρχία. Αυτό λύνεται αν δεν λάβουμε υπ'όψιν τις εξαρτήσεις από διαστάσεις που δεν υπάρχουν στην εξεταζόμενη ιεραρχία, όπως εδώ τις εξαρτήσεις {Β, Γ} → Δ.
- Βελτίωση των αλγορίθμων, καθώς γίνεται αρκετά συχνά εξαντλητική διάσχιση στις αρχικές ιεραρχίες, προκειμένου να ελέγξουμε αν υπάρχουν κόμβοι που να ανήκουν σε μια Α διάσταση, κλπ. Αυτό θα βοηθήσει στην διαχείριση μεγαλύτερων σχημάτων ιεραρχιών, σε μικρότερο χρόνο.

9

Βιβλιογραφία

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu, Data on the web. From relations to semistructured data and XML, Morgan Kaufmann Publishers, San Francisco, 2002.

[2] Bernd Amann, Catriel Beeri, Irini Fundulaki and Michel Scholl, Ontology-based integration of XML Web Resources, 2002

[3] S. Boag, D. Chamberlin, M.F. Fernandez, D. Florescu, J. Robie, J. Simon, and M. Stefanescu (eds.). XQuery 1.0: An XML Query Language. W3CWorking Draft, December 2001. URL: http://www.w3.org/TR/xquery.

[4] J. Clark and S. DeRose (eds.). XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. http://www.w3c.org/TR/xpath.

[5] S. Cluet, P. Veltri, and D. Vodislav. Views in a Large Scale XML Repository. In *Proceedings VLBD*, Rome, Italy, September 2001.

[6] T. Dalamagas, A. Meliou, T. Sellis, Modelling and manipulating the structure of portal catalogs, Tech. report, KDBS Lab, Dept of Electrical and Computer Eng., NTU Athens, Jan 2003

[7] M. Erdmann and R. Studer. How to structure and access XML documents with ontologies. *Data Knowledge Engineering*, 36(3):317–335, 2001.

[8] Jihye Jung, Dongkyu Kim, Sang-goo Lee, Chisu Wu, Kapsoo Kim. EE-Cat: Extended Electronic Catalog for Dynamic and Flexible Electronic Commerce. Database Lab. Dept. of Computer Science, Seoul National University.

[9] Dongkyu Kim, Jaebum Kim, Sang-goo Lee. Catalog Integration for Electronic Commerce through Category-Hierarchy Merging Tecnique. 12th Int.l Wrkshp on Research Issues in Data Engineering, 1066-1395/02, 2002.

[10] A.Y. Levy. Answering queries using views: a survey. *VLDB Journal*, 2001

[11] G. Lausen and P. J. Marron. Adaptive evaluation techniques for querying XML-based E-catalogs. In Proceedings of the 12th International Workshop on Research Issues on Data Engineering. IEEE Press, February 2002.

[12] B. Ludascher, Y. Papakonstantinou, and P. Velikhov. A Framework for Navigation-Driven Lazy Mediators. In *Proc. of WebDB*. Philadelphia, USA, 1999.

[13] I. Manolescu, D. Florescu, and D. Kossmann. Answering XML Queries over Heterogeneous Data Sources. In *Proceedings of VLDB*, Rome, Italy, September 2001.

[14] Pedro Jose Marron, Georg Lausen, Martin Weber. Catalog Integration Made Easy. ICDE'03, 1063-6382/03. 2003.

[15] E. Rahm and P. A. Bernstein, A survey of approaches to automatic schema matching, VLDB Journal 10 (2001), no. 4, 334-350.

[16] Rogers Cadenhead, Laura Lemay, Teach Yourself Java in 21 days, Sams Publishing, 2003

[17] Θεόδωρος Δαλαμάγκας, Διαχείριση Ιεραρχικών Σχημάτων στο Σημασιολογικό Ιστό, Εργαστήριο ΣΒΓΔ, Ε.Μ.Π, Φεβρουάριος 2004, Διδακτορική Διατριβή.

[18] Αλεξάνδρα Μέλιου, Μοντελοποίηση και Διερεύνηση των Αλγεβρικών Ιδιοτήτων Δενδρικών Ιεραρχικών Δομών, Διπλωματική Εργασία, 2003

[19] http://xfml.org/

[20] http://www.nimble.com/.

[21] http://www.semanticweb.org

[22] http://www.topicmaps.org/

[23] http://www.w3c.org/XML

[24] http://www.w3c.org/XML/Schema