Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Γραφικό Εργαλείο Σχεδίασης Σεναρίων Εξαγωγής–Μετασχηματισμού–Φόρτωσης Δεδομένων σε Περιβάλλον Αποθηκών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Ε. Φοινίκεττος

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2004

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Γραφικό Εργαλείο Σχεδίασης Σεναρίων Εξαγωγής–Μετασχηματισμού–Φόρτωσης Δεδομένων σε Περιβάλλον Αποθηκών Δεδομένων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Μάριος Ε. Φοινίκεττος

Επιβλέπων: Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Δεκεμβρίου 2004.

.....	.....	.....
Τ. Σελλής	Ι. Βασιλείου	Ν. Κοζύρης
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2004

.....

Μάριος Ε. Φοινίκεττος

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

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων (ΕΒΓΔ) της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Ευχαριστώ τον καθηγητή κ. Τίμο Σελλή, τον υποψήφιο διδάκτορα και επιβλέποντα της διπλωματικής κ. Άλκη Σιμιτσή και τον υποψήφιο διδάκτορα κ. Πάνο Γεωργαντά για την πολύτιμη βοήθεια που μου πρόσφεραν κατά την εκπόνηση της διπλωματικής αυτής εργασίας, καθώς και όλους τους υπεύθυνους (και μη) του εργαστηρίου για τις πολύτιμες συμβουλές, πάνω σε τεχνικά θέματα, που μου προσέφεραν.

Δεκέμβριος 2004

Μάριος Ε. Φοινίκεττος

Περίληψη

Αντικείμενο αυτής της διπλωματικής είναι η μελέτη και υλοποίηση ενός εργαλείου γραφικής αναπαράστασης σεναρίων ΕΜΦ (Εξαγωγής–Μετασχηματισμού–Φόρτωσης). Με τον όρο *σενάριο ΕΜΦ* εννοούμε το σύνολο των διεργασιών που είναι υπεύθυνες για την εξαγωγή των δεδομένων από διάφορες πηγές, τον καθαρισμό, την προσαρμογή και την εισαγωγή τους σε μία Αποθήκη Δεδομένων. Η διπλωματική βασίζεται στο εννοιολογικό μοντέλο για την αρχική φάση ενός έργου σχεδιασμού και ανάπτυξης μίας Αποθήκης Δεδομένων, που προτείνεται από τους κ.κ. Π. Βασιλειάδη, Α. Σιμιτσή και Σ. Σκιαδόπουλο. Το εργαλείο που δημιουργήθηκε ικανοποιεί μέσω ενός εύχρηστου γραφικού περιβάλλοντος τις απαιτήσεις του μοντέλου σε απλότητα, γενικότητα και επεκτασιμότητα. Μεταξύ άλλων, το εργαλείο παρέχει στο χρήστη δυνατότητα να σχεδιάσει νέα σενάρια, να τα αποθηκεύσει σε ένα repository και να τα φορτώσει για επίδειξη ή περαιτέρω επεξεργασία.

Λέξεις Κλειδιά

Αποθήκη Δεδομένων, Εξαγωγή–Μετασχηματισμός–Φόρτωση, Εννοιολογική Μοντελοποίηση.

Abstract

The object of this Diploma Thesis is the implementation of a tool for the graphical representation of ETL (Extraction–Transformation–Loading) Scenaria. The term *ETL scenario* stands for the set of the processes responsible for the extraction of data from several sources, its cleaning, customization and insertion into a data warehouse. The diploma thesis is based on a conceptual model for the initial phase of a project for the design and development of a Data Warehouse, proposed by P. Vassiliadis, A. Simitsis and S. Skiadopoulos. The created tool satisfies the requirements for simplicity, generity and extensibility, through an easy to use, graphical inteface. Among others, the created tool allows the designing of a new scenaria, the insertion of the scenaria into a repository and the loading of scenaria from the repository to be displayed or further processed.

Keywords

Data Warehouse, Extract–Transform–Load, Conceptual Modeling.

Περιεχόμενα

1 Εισαγωγή **1**

1.1 Αποθήκες Δεδομένων 1

1.2 Διεργασίες Εξαγωγής–Μετασχηματισμού–Φόρτωσης Δεδομένων 3

1.3 Στόχος 7

1.4 Δομή Διπλωματικής 7

I Μοντελοποίηση Διεργασιών ΕΜΦ **9**

2 Ένα Εννοιολογικό Μοντέλο Διεργασιών ΕΜΦ **11**

2.1 Εννοιολογικό Μοντέλο 11

2.2 Μεθοδολογία Χρήσης του Εννοιολογικού Μοντέλου 20

2.3 Μετα–Μοντέλο 25

2.4 Συμπεράσματα 28

3 Εισαγωγή στο Λογικό Μοντέλο **29**

3.1 Εισαγωγή 29

3.2 Σύντομη Αναφορά στο Λογικό Μοντέλο 30

3.3 Εργαλείο Λογικής Σχεδίασης 36

4 Μετάβαση από το Εννοιολογικό στο Λογικό Μοντέλο **39**

4.1 Εισαγωγή 39

4.2 Απεικονίσεις 41

4.3 Σειρά Εκτέλεσης στη Λογική Ροή Έργου 52

4.4 Μεθοδολογία Μετάβασης από Εννοιολογικό σε Λογικό Μοντέλο 64

5 Σχετική Εργασία **71**

II Σχεδίαση και Υλοποίηση Εργαλείου Σχεδίασης Διεργασιών ΕΜΦ **77**

6 Ανάλυση και Σχεδίαση **79**

6.1 Περιγραφή Αρχιτεκτονικής 79

6.2 Περιγραφή Λειτουργιών 81

7 Υλοποίηση **89**

7.1 Πλατφόρμες και Προγραμματιστικά Εργαλεία 89

7.2 Λεπτομέριες Υλοποίησης 90

7.3 Περιγραφή Κώδικα 93

7.4 Περιγραφή Σχήματος Repository 119

8 Χρήση Εργαλείου **125**

8.1 Εγκατάσταση της Εφαρμογής 125

8.2 Εκκίνηση της Εφαρμογής 125

8.3 Κεντρική Οθόνη Προγράμματος 127

8.4 Εισαγωγή Κόμβων και Ακμών στο Σενάριο 132

9 Επίλογος **139**

9.1 Σύνοψη — Συμπεράσματα 139

9.2 Μελλοντικές Επεκτάσεις 140

Κατάλογος Σχημάτων

1.1 Διαδικασία μεταφοράς δεδομένων 2

1.2 Το περιβάλλον των διεργασιών ΕΜΦ 4

1.3 Ο κύκλος ζωής της Αποθήκης Δεδομένων και των διεργασιών ΕΜΦ αυτής 5

1.4 Γενική Αρχιτεκτονική του *Άρκτος II* [39] 6

1.5 Βασικά modules του *Άρκτος II* [39] 6

2.1 Γραφικοί συμβολισμοί για το εννοιολογικό μοντέλο των διεργασιών ΕΜΦ 12

2.2 Το προτεινόμενο μετα-μοντέλο ως διάγραμμα UML 12

2.3 Διάγραμμα του εννοιολογικού μοντέλου για το παράδειγμα 14

2.4 Αναγνώριση των κατάλληλων πηγών δεδομένων 21

2.5 Υποψήφιοι και ενεργοί υποψήφιοι για τις εμπλεκόμενες πηγές δεδομένων 22

2.6 Απλοποίηση του διαγράμματος: (α) αγνοώντας υποψηφίους, (β) θεωρώντας τον ενεργό υποψήφιο 22

2.7 Αντιστοιχίσεις γνωρισμάτων για τη φόρτωση της έννοιας DW.PARTSUPP 24

2.8 Εμπλουτισμός του Σχήματος 2.5 με περιορισμούς εκτέλεσης 25

2.9 Το πλαίσιο μοντελοποίησης διεργασιών ΕΜΦ 26

2.10 Πρότυπα μετασχηματισμών και οι συμβολισμοί τους ανά κατηγορίες 27

3.1 Γραφικοί Συμβολισμοί για το λογικό μοντέλο διεργασιών ΕΜΦ 36

3.2 Παράδειγμα Σεναρίου στο Εργαλείο Λογικής Σχεδίασης 37

3.3 Λεπτομέρια του σεναρίου του Σχήματος 3.2, όπως φαίνεται σε επίπεδο γνωρίσματος . . 38

4.1 Μετάβαση της έννοιας **DW.PARTS** σε ένα σύνολο εγγραφών, μαζί με τα γνωρίσματά της . 41

4.2 Μετάβαση μίας απλής σχέσης παροχής 42

4.3 Πλήρωση της έννοιας $\mathbf{S}_2$ από την έννοια $\mathbf{S}_1$ μέσω του μετασχηματισμού **T** 44

4.4 Πλήρωση συνόλου εγγραφών $\mathbf{S}_2$ από το σύνολο $\mathbf{S}_1$ μέσω της διεργασίας **A** 45

4.5 Παράδειγμα απεικόνισης ενός μετασχηματισμού T σε μία διεργασία A 46

4.6 Σειριακή σύνθεση των μετασχηματισμών $\mathbf{T}_1$ και $\mathbf{T}_2$ 47

4.7 Το μέρος του Παραδείγματος 1 που αφορά μόνο το μετασχηματισμό $\mathbf{f}_3$ 48

4.8 Παραγωγή της λειτουργικής σημασιολογίας της διεργασίας $\mathbf{f}_3$ 49

4.9 Περιορισμοί ΕΜΦ 50

4.10 Αποβολή ενός γνωρίσματος 51

4.11 Σύγκλιση δύο ροών 52

4.12 Τμήμα του παραδείγματος 1 που αφορά την πλήρωση της **DW.PARTS** από τη $\mathbf{S}_2$**.PARTS** . . 53

4.13 Επίπεδα μετασχηματισμών 54

4.14 Ένα παράδειγμα (α) κακού και (β) καλού σχεδιασμού 56

4.15 Ο αλγόριθμος **FS** 57

4.16 Ένα παράδειγμα εκτέλεσης του **FS** για το εννοιολογικό διάγραμμα του Σχήματος 4.13 . 58

4.17 Η περίπτωση του δυαδικού μετασχηματισμού (α) στο εννοιολογικό και (β) στο λογικό μοντέλο 59

4.18 Προβληματικό Εννοιολογικό Διάγραμμα 61

4.19 Ο αλγόριθμος **FSC** 62

4.20 Ο αλγόριθμος **EOLW** 64

4.21 Το εννοιολογικό διάγραμμα του Παραδείγματος 1, μετά το πρώτο βήμα 67

4.22 Το αποτέλεσμα του δεύτερου βήματος 68

4.23 Γραμμικές λογικές ροές έργου 69

4.24 Το λογικό διάγραμμα του Παραδείγματος 1 69

4.25 Το τελικό λογικό διάγραμμα του Παραδείγματος 1 69

6.1 Πλήρες διάγραμμα λειτουργίας εργαλείου 80

6.2 Τα τμήματα (packages) στα οποία χωρίζεται η εφαρμογή. 80

6.3 Σύνδεση στο Repository 82

6.4 Αναγνώριση Πηγών Δεδομένων 82

6.5 Επιλογή γνωρισμάτων για την έννοια **DW.Suppliers** 83

6.6 Στιγμιότυπο της επιφάνειας εργασίας αμέσως μετά την προσθήκη τριών εννοιών 83

6.7 Προσθήκη Υποψηφίων στο Σενάριο 84

6.8 Στιγμιότυπο του σεναρίου, μετά την προσθήκη υποψηφίων. 85

6.9 Άμεση αντιστοίχηση γνωρισμάτων 85

6.10 Εισαγωγή νέου μετασχηματισμού στο σενάριο 86

6.11 Εμφάνιση κρυμμένης πληροφορίας για το μετασχηματισμό **f3** που ορίζεται στο Σχήμα 6.10 87

6.12 Τμήμα του Παραδείγματος 1, σχεδιασμένο στο εργαλείο που κατασκευάσαμε 88

7.1 Οι κλάσεις **ETL.Base.Scenario** και **ETL.Conceptual.Scenario** 91

7.2 Οι κλάσεις **ETL.Base** και **ETL.Conceptual** 92

7.3 Σχήμα Repository 120

1

Εισαγωγή

Στο κεφάλαιο αυτό, δίνεται αρχικά (1.1) μία σύντομη περιγραφή των Αποθηκών Δεδομένων και στη συνέχεια (1.2) γίνεται μία εισαγωγή στις διεργασίες Εξαγωγής–Μετασχηματισμού–Φόρτωσης (ΕΜΦ) δεδομένων. Στην ενότητα 1.3 δίνεται ο στόχος της διπλωματικής αυτής εργασίας, και τέλος, στην ενότητα 1.4 παρουσιάζεται η οργάνωση του τόμου αυτού.

1.1 Αποθήκες Δεδομένων

Από τα μέσα της δεκαετίας του ΄70, η αλματώδης παραγωγή ισχυρών συστημάτων διαχείρισης βάσεων δεδομένων βοήθησε στην ανάπτυξη πληροφοριακών συστημάτων που καλύπτουν τις λειτουργικές ανάγκες οργανισμών και επιχειρήσεων. Τα μεγαλύτερα και ισχυρότερα συστήματα αναπτύχθηκαν με σκοπό τον αυτοματισμό βασικών αναγκών των οργανισμών όπως η διεκπεραίωση τραπεζικών εργασιών και τα λογιστικά συστήματα. Η λειτουργία αυτών των πληροφοριακών συστημάτων είναι πλέον κρίσιμη και πολύτιμη για τη ζωή των οργανισμών στους οποίους έχουν εγκατασταθεί, η δε βάση δεδομένων ενός τέτοιου συστήματος αποτελεί τον πυρήνα τους. Η ορθή σχεδίαση, ανάπτυξη και λειτουργία της βάσης είναι ο σημαντικότερος παράγοντας για την επιτυχία ενός πληροφοριακού συστήματος. Τα συστήματα αυτά παρέχουν τη δυνατότητα επεξεργασίας μεγάλου αριθμού δοσοληψιών που διαχειρίζονται τα δεδομένα του οργανισμού. Ένα άλλο είδος πληροφοριακών συστημάτων που αναπτύσσονται στους οργανισμούς είναι τα συστήματα στήριξης αποφάσεων, που σκοπό έχουν να βοηθήσουν τα στελέχη των οργανισμών να σχεδιάσουν τις δραστηριότητές τους. Η επιτυχία των συστημάτων αυτών είναι επίσης

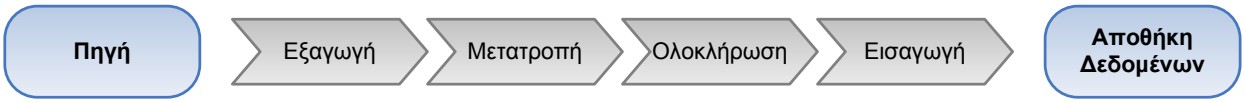
βασικός παράγοντας επιτυχίας του οργανισμού. Μία βασική απαίτηση των συστημάτων στήριξης αποφάσεων είναι η αποδοτική πρόσβαση στα δεδομένα των συστημάτων αυτοματισμού. Το πρόβλημα που προκύπτει, όμως, είναι ότι τα συστήματα αυτοματισμού έχουν ήδη πολύ σοβαρό υπολογιστικό φορτίο από μόνα τους, και επιπλέον, είναι σχεδιασμένα για την εκτέλεση διαφορετικών λειτουργιών.

Είναι σαφές ότι είναι εξαιρετικά δυσχερής η χρήση των βάσεων δεδομένων των πληροφοριακών συστημάτων των οργανισμών από τα συστήματα στήριξης αποφάσεων. Όμως, η αποδοτική χρήση των συστημάτων στήριξης αποφάσεων απαιτεί, όπως προαναφέρθηκε, πρόσβαση στα δεδομένα αυτά. Η εισαγωγή των "Αποθηκών Δεδομένων" είναι η λύση στο κρίσιμο αυτό πρόβλημα.

Με τον όρο *Αποθήκες Δεδομένων (Data Warehouses)* χαρακτηρίζουμε ένα σύνολο τεχνολογιών που επιτρέπει στους αναλυτές ενός οργανισμού τη σχεδίαση της πολιτικής του, έχοντας αποδοτική πρόσβαση στα δεδομένα του οργανισμού. Μία Αποθήκη Δεδομένων διατηρεί δεδομένα που αντλεί από τις βάσεις δεδομένων των πληροφοριακών συστημάτων του οργανισμού, αλλά και άλλες πηγές δεδομένων, όπως αρχεία του οργανισμού ή δεδομένα που προέρχονται από εξωτερικές πηγές. Αυτά τα δεδομένα οργανώνονται στην Αποθήκη Δεδομένων σε δομές κατάλληλες να απαντήσουν τις απαιτήσεις των αναλυτών—χρηστών των συστημάτων στήριξης αποφάσεων. Τα συστήματα στήριξης αποφάσεων αποκτούν πρόσβαση στα δεδομένα λειτουργίας του οργανισμού χωρίς την παρουσία των προαναφερθέντων προβλημάτων. Οι Αποθήκες Δεδομένων παρέχουν τη δυνατότητα για *Συνεχή Αναλυτική Επεξεργασία (On-line Analytical Processing—OLAP)* των δεδομένων, περιέχοντας συνήθως ιστορικά και συγκεντρωτικά δεδομένα που συνήθως αποδεικνύονται χρήσιμα για υποστήριξη αποφάσεων. Επίσης, παρέχουν μία ολοκληρωμένη εικόνα του σχήματος των δεδομένων του οργανισμού. Η σχεδίαση των Αποθηκών Δεδομένων έχει σαν στόχο την αποδοτική απάντηση των πολύπλοκων ερωτήσεων που θέτονται κατά την αναλυτική επεξεργασία δεδομένων από τις εφαρμογές στρατηγικού σχεδιασμού.

Η δημιουργία και η συντήρηση μίας Αποθήκης Δεδομένων είναι μία πολύπλοκη διαδικασία, καθώς πολλές διαφορετικές προσεγγίσεις είναι εφικτές. Αρκετοί οργανισμοί επιδιώκουν να δημιουργήσουν μία Αποθήκη Δεδομένων που θα περιέχει αναλυτικά δεδομένα από όλες τις δραστηριότητες του οργανισμού. Πρόκειται για ένα πολύπλοκο εγχείρημα που απαιτεί μεγάλο κόστος για να επιτύχει [23].

Βασικός παράγοντας για την επιτυχία των Αποθηκών Δεδομένων είναι η ορθή τροφοδοσία της Αποθήκης Δεδομένων από τις πηγές. Η διαδικασία μεταφοράς δεδομένων από τις πηγές στην Αποθήκη Δεδομένων είναι αρκετά πολύπλοκη, καθώς πολλά προβλήματα πρέπει να αντιμετωπισθούν. Τα βήματα που ακολουθούνται κατά τη μεταφορά των δεδομένων, παρουσιάζονται στο Σχήμα 1.1.



Σχήμα 1.1: Διαδικασία μεταφοράς δεδομένων

Για κάθε πηγή που χρησιμοποιούμε στο σύστημα, εγκαθιστούμε λογισμικό που αντλεί τα δεδομένα από την πηγή, τα "καθαρίζει", κρατώντας μόνο αυτά που είναι πραγματικά χρήσιμα και τα μετασχηματίζει με βάση ένα καθορισμένο πρότυπο. Οι μετατροπές που γίνονται στα δεδομένα αφορούν

τόσο τη δομή, όσο και την τιμή τους. Για παράδειγμα, το πεδίο **Ημερομηνία** ενός πίνακα μπορεί να μετασχηματιστεί στα πεδία **Χρόνος**, **Μήνας** και **Ημέρα**, ενώ οι τιμές του πεδίου **Χαρακτηρισμός** είναι πιθανόν να μετατραπούν από **A**, **B** κ.λ.π. σε **1**, **2** κ.λ.π. αντίστοιχα. Αυτό το λογισμικό υλοποιείται με βάση τα ιδιαίτερα χαρακτηριστικά κάθε πηγής και εγκαθίσταται σε υπολογιστές με άμεση πρόσβαση στα δεδομένα της πηγής. Οι Αποθήκες Δεδομένων χρησιμοποιούν ποικίλα εργαλεία για εξαγωγή. Η εξαγωγή δεδομένων από τις απομακρυσμένες πηγές συχνά υλοποιείται μέσω πυλών (gateways) και καθιερωμένων προτύπων διασύνδεσης εφαρμογών (όπως ODBC, Oracle Open Connect, Information Builders EDA/SQL, κ.λ.π.).

Εξωτερικά εργαλεία που εγκαθίστανται για κάθε διαφορετική πηγή δεδομένων αναλαμβάνουν την εξαγωγή των δεδομένων από τις πηγές. Παράλληλα, εκτελούν και μία πρώτη επεξεργασία των δεδομένων αυτών. Καθώς οι Αποθήκες Δεδομένων χρησιμοποιούνται για στρατηγικές αποφάσεις, επιβάλλεται να περιέχουν σωστά δεδομένα. Στις διάφορες πηγές όπου υπάρχει μεγάλος όγκος δεδομένων, είναι πολύ πιθανόν να υπάρχουν λάθη ή ανωμαλίες. Διάφορα εργαλεία βοηθούν στη διάγνωση των ανωμαλιών των δεδομένων και στη διόρθωσή τους, όπου αυτό είναι εφικτό. Ως περιπτώσεις όπου ο καθαρισμός των δεδομένων είναι σημαντικός, αναφέρονται: ασυνέπειες στο μήκος των πεδίων διαφορετικών πηγών, ασυνέπειες σχετικά με την περιγραφή των δεδομένων, ασυνεπείς τιμές δεδομένων, απουσίες εγγραφών και παραβίαση περιορισμών ακεραιότητας.

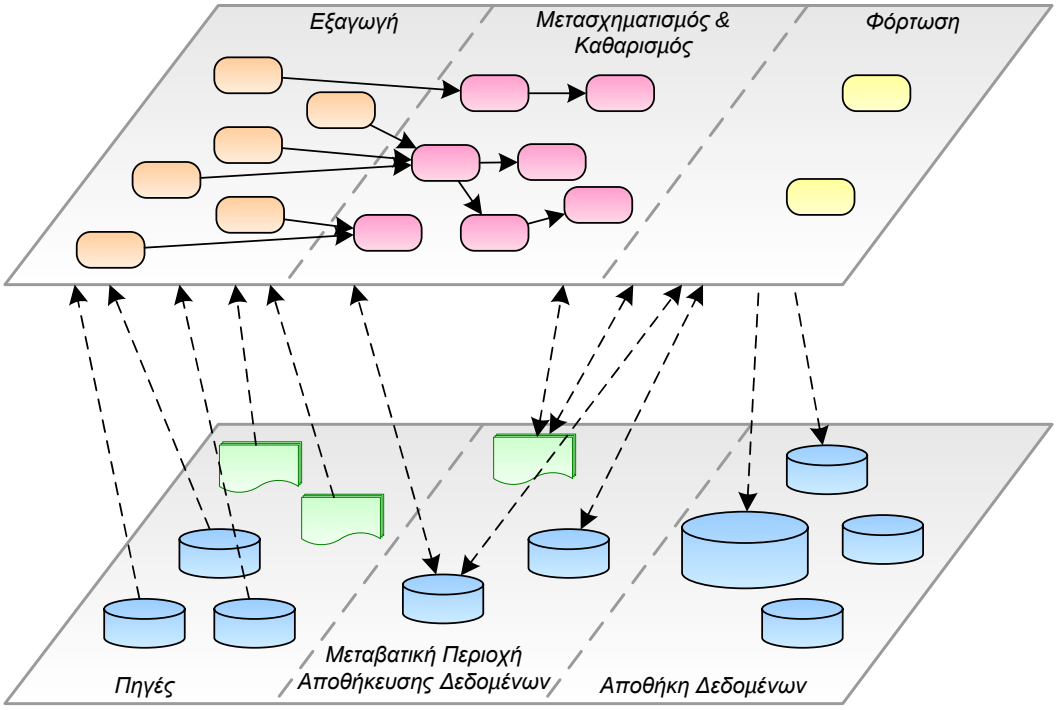
1.2 Διεργασίες Εξαγωγής–Μετασχηματισμού–Φόρτωσης Δεδομένων

Για ένα μεγάλο διάστημα στο παρελθόν, η έρευνα αντιμετώπιζε τις Αποθήκες Δεδομένων ως συλλογές υλοποιημένων όψεων. Αν και αυτή η θεώρηση είναι κομψή και πιθανώς επαρκής για το σκοπό της εξέτασης εναλλακτικών στρατηγικών για τη διατήρηση όψεων, κρίνεται ανεπαρκής για την περιγραφή της δομής και των περιεχομένων μίας Αποθήκης Δεδομένων. Στην αγορά κυκλοφορούν εξειδικευμένα εργαλεία, υπό το γενικό χαρακτηρισμό εργαλεία *Εξαγωγής-Μετασχηματισμού-Φόρτωσης (ΕΜΦ) (Extraction-Transformation-Loading – ETL)*, με σκοπό να διευκολύνουν και να χειριστούν τις λειτουργικές διαδικασίες των Αποθηκών Δεδομένων. Τα εργαλεία ΕΜΦ αναλαμβάνουν το έργο της διατήρησης της ομοιογένειας, του καθαρισμού και της φόρτωσης μίας Αποθήκης Δεδομένων. Το κόστος αυτών των εργασιών υπολογίζεται ότι ανέρχεται στο ένα τρίτο της συνολικής εργασίας και των εξόδων στον προϋπολογισμό μίας Αποθήκης Δεδομένων, ενώ ο χρόνος ανάπτυξής τους μπορεί να φτάσει ως και το 80% του συνολικού χρόνου ανάπτυξης μίας Αποθήκης Δεδομένων [46]. Παρόλα αυτά, κυρίως λόγω της πολυπλοκότητας, του κόστους και της δύσκολης προσαρμογής των έτοιμων πακέτων στις ανάγκες κάθε εταιρίας, πολλοί οργανισμοί αναπτύσσουν μόνοι τους εργαλεία για την εκτέλεση εργασιών ΕΜΦ, προσαρμοσμένα στις ανάγκες των εκάστοτε περιστάσεων.

Για να δώσουμε μία γενική ιδέα της λειτουργικότητας αυτών των εργαλείων, αναφέρουμε τις πιο χαρακτηριστικές λειτουργίες τους, που περιλαμβάνουν: (α) τον εντοπισμό της σχετικής πληροφορίας στην πλευρά της πηγής, (β) την εξαγωγή της πληροφορίας αυτής, (γ) την προσαρμογή και την ενοποίηση πληροφορίας προερχόμενης από πολλαπλές πηγές σε μία κοινή μορφή, (δ) τον καθαρισμό του παρα-

γόμενου συνόλου δεδομένων με βάση τους κανόνες της βάσης δεδομένων αλλά και άλλους λογικούς κανόνες, (ε) την προώθηση των δεδομένων στην Αποθήκη Δεδομένων και/ή στις αποθηκευμένες όψεις.

Στη συνέχεια, θα υιοθετήσουμε το όνομα ΕΜΦ τόσο για τις διεργασίες ΕΜΦ, όσο και για αυτές του καθαρισμού δεδομένων.



Σχήμα 1.2: Το περιβάλλον των διεργασιών ΕΜΦ

Στο Σχήμα 1.2 περιγράφεται το γενικό πλαίσιο των διεργασιών ΕΜΦ. Στο κατώτερο επίπεδο απεικονίζονται τα σημεία αποθήκευσης των δεδομένων που εμπλέκονται στην όλη διαδικασία. Στην αριστερή πλευρά, παρατηρούμε τους αρχικούς προμηθευτές—πηγές δεδομένων. Συνήθως, ως πηγές δεδομένων θεωρούνται σχεσιακές βάσεις δεδομένων και αρχεία. Τα δεδομένα από τις πηγές αυτές εξάγονται από ρουτίνες *εξαγωγής* (όπως φαίνεται στο πάνω αριστερά μέρος του Σχήματος 1.2, οι οποίες παρέχουν είτε ολοκληρωμένα στιγμιότυπα των πηγών, είτε τις εκάστοτε διαφορές τους. Στη συνέχεια, τα εξαγόμενα δεδομένα μεταφέρονται στη *Μεταβατική Περιοχή Αποθήκευσης Δεδομένων (ΜΠΑΔ)* όπου μετασχηματίζονται και καθαρίζονται πριν φορτωθούν στην Αποθήκη Δεδομένων. Η Αποθήκη Δεδομένων απεικονίζεται στο δεξί μέρος του κατώτερου επιπέδου και αποτελεί τον τελικό προορισμό αποθήκευσης των δεδομένων. Η *φόρτωση* της Αποθήκης Δεδομένων γίνεται από αντίστοιχες ρουτίνες που απεικονίζονται στο πάνω δεξί μέρος του σχήματος.

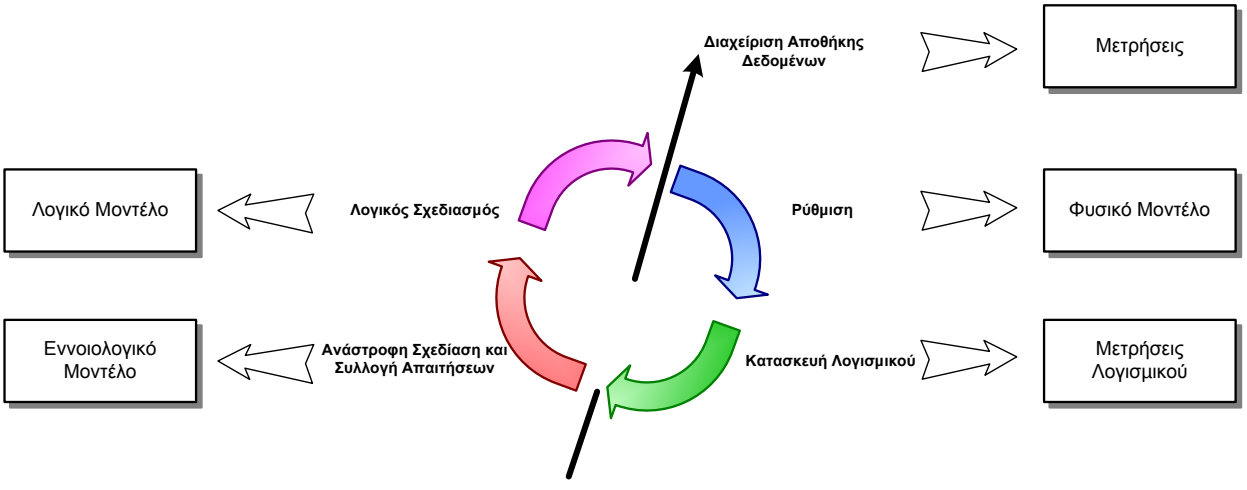
Η σχετική βιβλιογραφία έχει αποδώσει διάφορα χαρακτηριστικά στις διεργασίες ΕΜΦ. Καταρχάς, μπορούμε να τις χαρακτηρίσουμε πολυσύνθετες. Η διαδικασία ανανέωσης μίας Αποθήκης Δεδομένων μπορεί να αποτελείται από πολλές διαφορετικές υπολειτουργίες, όπως ο καθαρισμός των δεδομένων, η αρχειοθέτηση, οι μετασχηματισμοί και η συνένωση, συνδεδεμένα μεταξύ τους με ένα πολυσύνθετο σχέδιο. Επιπλέον, οι διαδικασίες αυτές είναι πολύ σημαντικές για την ορθότητα, την πληρότητα και την ενημερότητα των περιεχομένων της Αποθήκης Δεδομένων, καθώς όχι μόνο διευκολύνουν τη

φόρτωσή της με δεδομένα, αλλά και είναι υπεύθυνες για την ομοιογένεια της δομής τους και την απόρριψη εσφαλμένων και ασυνεπών εγγραφών.

Η διπλωματική αυτή γίνεται στα πλαίσια του προγράμματος *Άρκτος ΙΙ* ([1]). Το Άρκτος ΙΙ είναι ένα εσωτερικό πρόγραμμα του Εργαστηρίου Συστημάτων Βάσεων Γνώσεων και Δεδομένων (ΕΣΒΓΔ) του Ε.Μ.Π, που αποσκοπεί στη μοντελοποίηση και επεξεργασία διεργασιών ΕΜΦ. Στα πλαίσια του προγράμματος αυτού, μελετώνται τα διάφορα στάδια του κύκλου ζωής των Αποθηκών Δεδομένων.

Σκοπός της εργασίας μας είναι ο χειρισμός και η βελτιστοποίηση του σχεδιασμού και της υλοποίησης διεργασιών ΕΜΦ, τόσο κατά το αρχικό στάδιο σχεδιασμού και ανάπτυξης, όσο και κατά τη διάρκεια της συνεχούς εξέλιξης της Αποθήκης Δεδομένων.

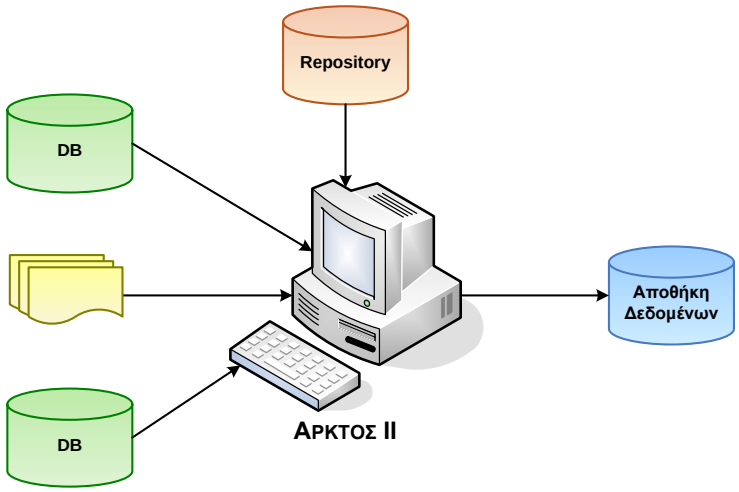
Στη διερεύνηση του παραπάνω προβλήματος, πρέπει να αποσαφηνιστεί η θέση των διεργασιών ΕΜΦ στον *κύκλο ζωής αποθήκης δεδομένων* μίας Αποθήκης Δεδομένων. Όπως μπορούμε να δούμε στο Σχήμα 1.3, ο κύκλος ζωής μίας Αποθήκης Δεδομένων ξεκινά με ένα στάδιο *Ανάστροφης Σχεδίασης και Συλλογής Απαιτήσεων*, κατά την οποία τα δεδομένα αναλύονται ώστε να κατανοηθεί η δομή και το περιεχόμενό τους. Ταυτόχρονα, συλλέγονται οι διάφορες απαιτήσεις από την πλευρά των χρηστών (συνήθως λίγοι διαχειριστές). Το προϊόν αυτού το σταδίου είναι ένα *Εννοιολογικό Μοντέλο* για τις πηγές δεδομένων και τις διεργασίες [47]. Σε δεύτερη φάση, και συγκεκριμένα στο στάδιο *Λογικού Σχεδιασμού*, κατασκευάζεται το λογικό σχήμα της Αποθήκης Δεδομένων και των διεργασιών της [48]. Αποτέλεσμα αυτής της φάσης είναι ένα *Λογικό Μοντέλο* για τις διεργασίες ΕΜΦ. Στο τρίτο στάδιο, το λογικό σχήμα και οι διαδικασίες εμπλουτίζονται με βάση τις επιλογές όσον αφορά στις συγκεκριμένες φυσικές δομές στην Αποθήκη Δεδομένων (π.χ. ευρετήρια) και στις ανάλογες παραμέτρους του περιβάλλοντος εκτέλεσης των λειτουργικών διαδικασιών. Αυτό το στάδιο ονομάζεται *Ρύθμιση*, και το προϊόν του *Φυσικό Μοντέλο* του περιβάλλοντος. Σε ένα τέταρτο στάδιο, αυτό της *Κατασκευής Λογισμικού*, το κατάλληλο λογισμικό δημιουργείται, ελέγχεται και αξιολογείται και κατασκευάζεται μια πρώτη έκδοση της Αποθήκης Δεδομένων. Η όλη διαδικασία καθοδηγείται από συγκεκριμένες *Μετρήσεις Λογισμικού*. Στη συνέχεια, ο κύκλος ξενικά από την αρχή, καθώς οι απαιτήσεις των χρηστών, οι πηγές δεδομένων



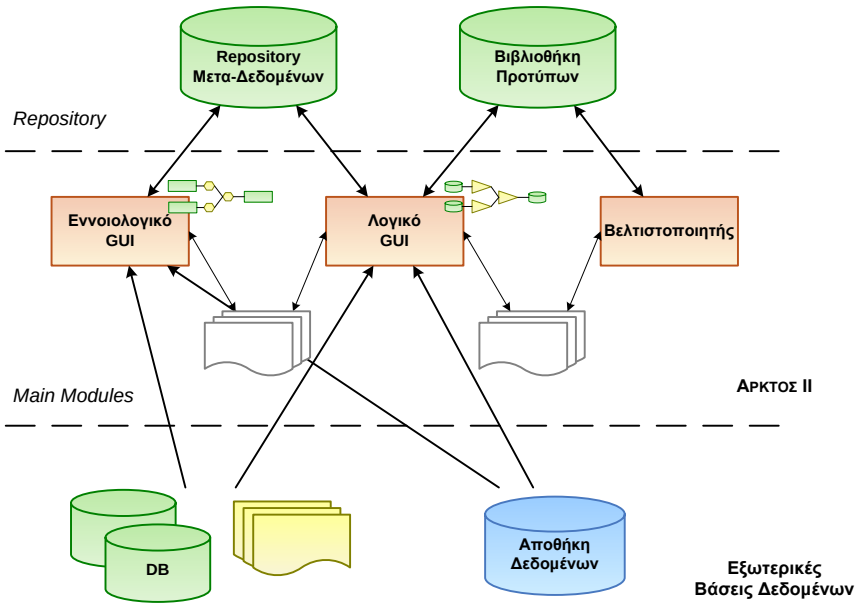
Σχήμα 1.3: Ο κύκλος ζωής της Αποθήκης Δεδομένων και των διεργασιών ΕΜΦ αυτής

και η Αποθήκη Δεδομένων βρίσκονται σε συνεχή εξέλιξη. Ένα επιπλέον στοιχείο που εμφανίζεται στο προσκήνιο μετά την ανάπτυξη της Αποθήκης Δεδομένων, είναι αυτό της *Διαχείρισης* του συστήματος, το οποίο απαιτεί με τη σειρά του ειδικές μετρήσεις για τη συντήρηση και την επίβλεψη της Αποθήκης Δεδομένων.

Μέχρι τώρα έχουν γίνει εργασίες και έχουμε αποκομίσει συμπεράσματα όσον αφορά τη θεωρητική μελέτη των φάσεων εννοιολογικής και λογικής σχεδίασης, καθώς και βελτιστοποίησης διεργασιών ΕΜΦ. Στα πλαίσια αυτά, αναπτύσσεται ένα πρότυπο εργαλείο σχεδίασης και επεξεργασίας διεργασιών ΕΜΦ, που ονομάζεται *Άρκτος ΙΙ*. Στα Σχήματα 1.4 και 1.5, παρουσιάζεται η αρχιτεκτονική του προγράμματος.



Σχήμα 1.4: Γενική Αρχιτεκτονική του *Άρκτος ΙΙ* [39]



Σχήμα 1.5: Βασικά modules του *Άρκτος ΙΙ* [39]

Το λογικό GUI έχει γίνει στο [41], ενώ ο βελτιστοποιητής έχει γίνει στο [40].

1.3 Στόχος

Αυτή η διπλωματική αποσκοπεί στη σχεδίαση και υλοποίηση GUI για το εννοιολογικό μοντέλο διεργασιών Εξαγωγής–Μετασχηματισμού–Φόρτωσης Δεδομένων που προτείνεται στο [46]. Συγκεκριμένα, θέλουμε να κατασκευάσουμε ένα εργαλείο που να προσφέρει ένα εύχρηστο γραφικό περιβάλλον για τη δημιουργία σεναρίων ΕΜΦ κατά το πρώτο στάδιο (στάδιο Ανάστροφης Σχεδίασης και Συλλογής Απαιτήσεων) ανάπτυξης μίας Αποθήκης Δεδομένων.

Με δεδομένο ότι αναφερόμαστε στα αρχικά στάδια σχεδίασης ενός έργου Αποθήκης Δεδομένων, απαίτησή μας είναι, κυρίως, η γρήγορη δημιουργία ενός σεναρίου, και όχι ο πλήρης και αυστηρός ορισμός του· αυτό γίνεται στο στάδιο της λογικής σχεδίασης. Για το σκοπό αυτό, το εργαλείο που θα κατασκευάσουμε θα πρέπει να δένει αρμονικά με το εργαλείο για την ανάπτυξη λογικών σεναρίων ([41]) που υπάρχει ήδη στο πρόγραμμα *Άρκτος II*, στοχεύοντας έτσι στη δημιουργία μίας ολοκληρωμένης πλατφόρμας για τη σχεδίαση και υλοποίηση ενός εργαλείου EMF για έργα Αποθηκών Δεδομένων.

Το μεγαλύτερο όφελος που επιθυμούμε να αποκομίσουμε από τη χρήση του μοντέλου γραφικής αναπαράστασης των σεναρίων, και κατ' επέκταση του εργαλείου, είναι η δημιουργία μίας "εποπτικής εικόνας" του σεναρίου, έτσι ώστε να διευκολυνθεί η εξέλιξη του σεναρίου και να διορθωθούν τυχόν σφάλματα σχεδίασης της Αποθήκης Δεδομένων. Τα σενάρια που θα σχεδιάζονται στο εργαλείο θα μπορούν να γίνουν θέματα συζήτησης και έτσι να συνεισφέρουν θετικά στην ανάπτυξη της Αποθήκης.

1.4 Δομή Διπλωματικής

Ο τόμος που κρατάτε στα χέρια σας αποτελείται από 6 κεφάλαια και αναλύει πλήρως την ανάπτυξη της εργασίας.

Το **Κεφάλαιο 2** εστιάζει την προσοχή του στο εννοιολογικό τμήμα του ορισμού των διεργασιών ΕΜΦ, όπως αυτό περιγράφεται στο [47]. Συγκεκριμένα, ασχολείται με τα αρχικά στάδια του σχεδιασμού της Αποθήκης Δεδομένων. Κατά τη διάρκεια αυτής της φάσης, ο σχεδιαστής της Αποθήκης Δεδομένων ενδιαφέρεται για δύο διαδικασίες, οι οποίες εκτελούνται παράλληλα. Η πρώτη διαδικασία περιλαμβάνει τη *συλλογή των απαιτήσεων του χρήστη*. Η δεύτερη διαδικασία, η οποία είναι ίσης σπουδαιότητας για την επιτυχία της ανάπτυξης της Αποθήκης Δεδομένων, περιλαμβάνει την *ανάλυση της δομής των πηγών δεδομένων που υπάρχουν ήδη* και *την τελική αντιστοίχηση τους στο μοντέλο της Αποθήκευσης Δεδομένων*.

Το **Κεφάλαιο 3** δίνεται μία σύντομη περιγραφή του λογικού μοντέλου που περιγράφεται στο [46]. Αυτό γίνεται για να να αποκτήσει ο αναγνώστης τα απαραίτητα στοιχεία για το λογικό μοντέλο, έτσι ώστε να κατανοήσει ευκολότερα τη μετάβαση από το εννοιολογικό στο λογικό μοντέλο που δίνεται στο επόμενο κεφάλαιο.

Στο **Κεφάλαιο 4** παρουσιάζεται μία μεθοδολογία για τη μετάβαση από το εννοιολογικό στο λογικό

μοντέλο. Αρχικά, γίνεται μία εισαγωγή και περιγράφονται οι λόγοι που μας οδήγησαν στην ανάγκη για τη μετάβαση από το εννοιολογικό στο λογικό μοντέλο. Στη συνέχεια, περιγράφεται αναλυτικά η διαδικασία μετάβασης καθώς και προβλήματα που προκύπτουν κατά τη μετάβαση με τη λύση τους.

Το **Κεφάλαιο 5** παρουσιάζονται ερευνητικές προσπάθειες για τη μοντελοποίηση, τόσο σε εννοιολογικό, όσο και σε λογικό επίπεδο, των διεργασιών ΕΜΦ. Παράλληλα, παρουσιάζονται μερικά εμπορικά εργαλεία για το σκοπό αυτό.

Στο **Κεφάλαιο 6** αναλύεται η σχεδίαση και η ανάπτυξη του εργαλείου. Αρχικά, δίνεται μία περιγραφή της αρχιτεκτονικής του συστήματος και προσδιορίζονται τα λογικά τμήματα που το αποτελούν. Στη συνέχεια, αναλύονται και περιγράφονται αναλυτικά οι δυνατότητες που παρέχει στο χρήστη.

Στο **Κεφάλαιο 7** γίνεται μία περιγραφή της υλοποίησης του συστήματος. Αρχικά τεκμηριώνεται η επιλογή της πλατφόρμας και των προγραμματιστικών εργαλείων που επιλέχθηκαν για την ανάπτυξη. Στη συνέχεια, ο κώδικας της εφαρμογής χωρίζεται σε ενότητες και παρατίθεται μία σειρά από διαγράμματα UML που αφορούν τις συσχετίσεις μεταξύ των κλάσεων, των φορμών, των λειτουργικών μονάδων και των συστατικών στοιχείων τους. Τέλος, περιγράφεται το σχήμα του repository που χρησιμοποιείται για την αποθήκευση των σεναρίων.

Στο **Κεφάλαιο 8** περιγράφεται η φιλοσοφία χρήση του εργαλείου. Δίνονται στιγμιότυπα των εργαλειοθηκών και των μενού του εργαλείου και παρουσιάζονται συνοπτικά οι διάφορες οθόνες με τις οποίες θα έρθει αντιμέτωπος ο χρήστης στην προσπάθειά του να υλοποιήσει ένα εννοιολογικό σενάριο.

Το **Κεφάλαιο 9** αποτελεί τον επίλογο της διπλωματικής. Στο κεφάλαιο αυτό παρουσιάζονται τα οφέλη που αποκομίσθηκαν κατά την εκπόνηση της διπλωματικής αυτής καθώς και τα συμπεράσματα στα οποία καταλήξαμε. Τέλος, παρουσιάζουμε εισηγήσεις για μελλοντικές επεκτάσεις.

Μέρος Ι

Μοντελοποίηση Διεργασιών ΕΜΦ

2

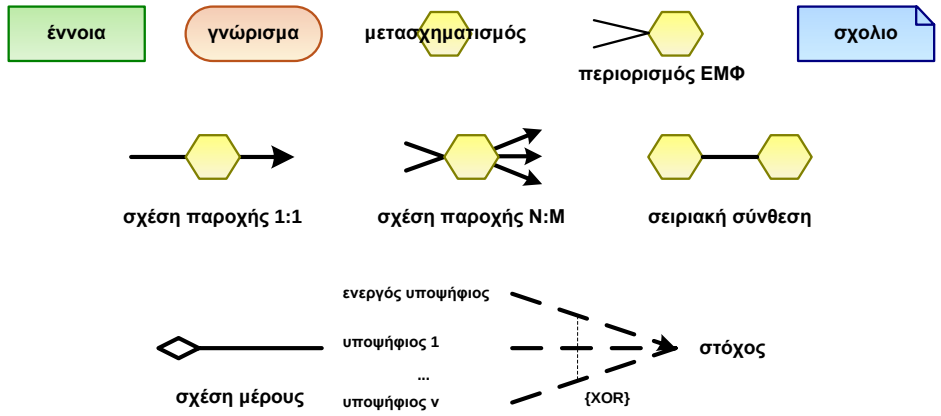
Ένα Εννοιολογικό Μοντέλο Διεργασιών ΕΜΦ

Το κεφάλαιο αυτό οργανώνεται ως εξής: Στην ενότητα 2.1 παρουσιάζεται το εννοιολογικό μοντέλο για διεργασίες ΕΜΦ και στην ενότητα 2.2 παρουσιάζεται μια μεθοδολογία για τη χρήση του. Στην ενότητα 2.3 παρουσιάζεται ένα μετα-μοντέλο με δύο βασικά χαρακτηριστικά: τη γενίκευση και την επεκτασιμότητα. Τέλος, στην ενότητα 2.4 καταγράφονται τα συμπεράσματα της συγκεκριμένης μελέτης, καθώς και κάποια ερευνητικά θέματα που προκύπτουν και αποτελούν αντικείμενα περαιτέρω μελέτης στο μέλλον.

2.1 Εννοιολογικό Μοντέλο

Στην ενότητα αυτή παρουσιάζεται ένα εννοιολογικό μοντέλο για διεργασίες ΕΜΦ. Ο σκοπός είναι να εντοπιστούν σε υψηλό επίπεδο οι οντότητες που χρησιμοποιούνται για τη σύλληψη της σημασιολογίας των διεργασιών ΕΜΦ. Αρχικά, παρουσιάζονται οι γραφικοί συμβολισμοί και το μετα-μοντέλο. Έπειτα, διευκρινίζονται τα δομικά συστατικά του μοντέλου μέσω ενός χαρακτηριστικού παραδείγματος.

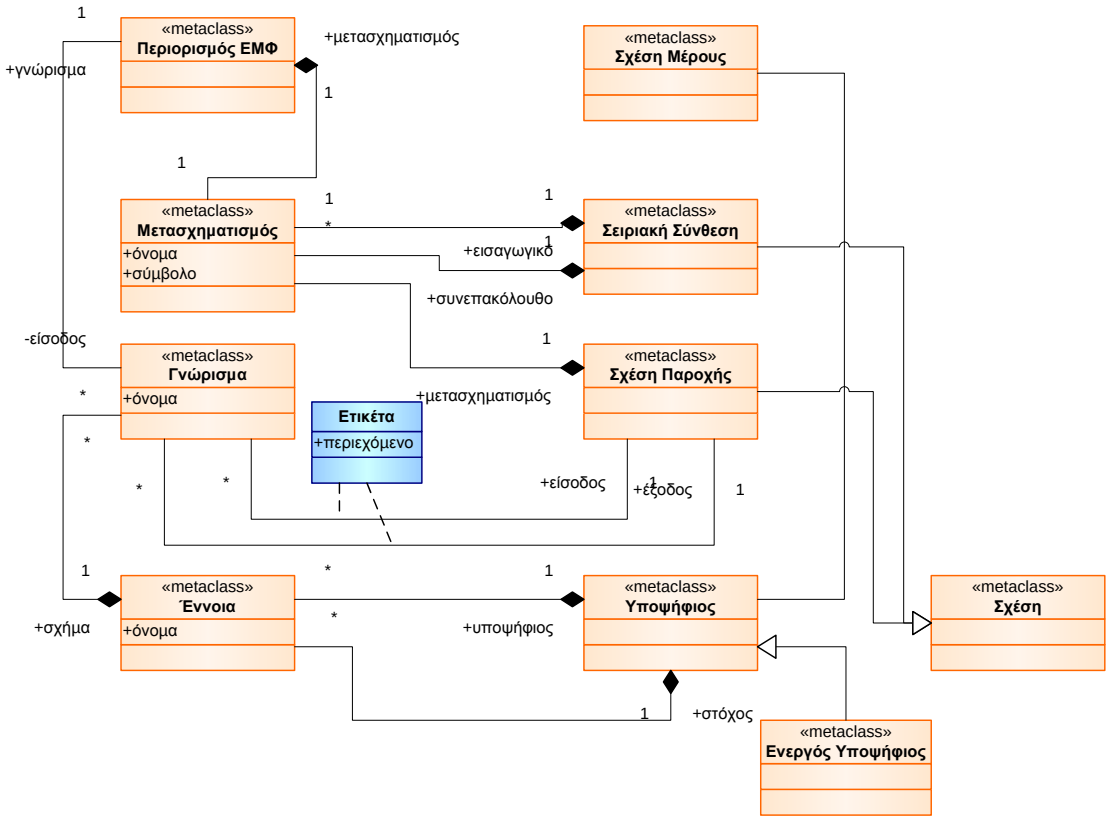
Στο Σχήμα 2.1 απεικονίζονται γραφικά οι διάφορες οντότητες για το μοντέλο που προτείνεται στο [47]. Δεν υιοθετούνται συμβολισμοί UML για τις έννοιες και τα γνωρίσματα, καθώς τα γνωρίσματα στο μοντέλο αυτό, θεωρούνται ως “πολίτες πρώτης τάξης”. Κατά συνέπεια, τα γνωρίσματα δεν περιλαμβάνονται στον ορισμό της οντότητας όπου ανήκουν, όπως για παράδειγμα μία κλάση UML ή ένας σχεσιακός πίνακας. Το μοντέλο είναι ορθόγωνο με τα εννοιολογικά μοντέλα που έχουν προταθεί για Αποθήκες Δεδομένων σχήματος αστέρα. Στην πραγματικότητα, η σχετική εργασία που έχει γίνει για



Σχήμα 2.1: Γραφικοί συμβολισμοί για το εννοιολογικό μοντέλο των διεργασιών ΕΜΦ

το front end των Αποθηκών Δεδομένων, μπορεί να συνδυαστεί με το προτεινόμενο μοντέλο, που είναι σαφώς προσανατολισμένο στο back end της Αποθήκης Δεδομένων.

Στο Σχήμα 2.2 απεικονίζονται οι βασικές οντότητες του προτεινόμενου στο [47] μετα-μοντέλου, ως διάγραμμα UML. Όλα τα συστατικά του εννοιολογικού μοντέλου που εισάγονται στη συνέχεια, αναφέρονται στις οντότητες του Σχήματος 2.2.



Σχήμα 2.2: Το προτεινόμενο μετα-μοντέλο ως διάγραμμα UML

Παράδειγμα. Για τη διευκόλυνση της παρουσίασης του μοντέλου, χρησιμοποιούμε το ακόλουθο παράδειγμα. Θεωρούμε δύο βάσεις δεδομένων, **S1** και **S2**, και μία Αποθήκη Δεδομένων, **DW**, που στο εξής θα αναφέρεται ως στόχος **DW**. Το σενάριο περιλαμβάνει τη διάδοση δεδομένων από την έννοια **PARTS(PKey, Qty, Cost)** της πηγής **S1**, καθώς και από την έννοια **PARTS(PKey, Date, Qty, Cost, Dept)** της πηγής **S2** στο στόχο **DW**. Στην Αποθήκη Δεδομένων, η έννοια **PARTS(PKey, Date, Qty, Cost)** αποθηκεύει καθημερινά (**Date**) πληροφορία για τη διαθέσιμη ποσότητα (**Qty**) και κόστος (**Cost**) των εξαρτημάτων (**PKey**). Υποθέτουμε ότι η πρώτη βάση δεδομένων βρίσκεται στην Ευρώπη, ενώ η δεύτερη στην Αμερική· κατά συνέπεια, τα δεδομένα που προέρχονται από τη δεύτερη πηγή θα πρέπει να προσαρμοστούν στις ευρωπαϊκές μονάδες και τυποποιήσεις. Επίσης, για την πρώτη βάση θεωρούμε ότι απαιτείται ο συνδυασμός πληροφορίας από δύο διαφορετικές έννοιες, στοιχείο που επιτυγχάνεται με μια εξωτερική συνένωση των εννοιών **PS1** και **PS2**. Το Σχήμα 2.3 απεικονίζει το πλήρως ανεπτυγμένο διάγραμμα του εννοιολογικού μοντέλου για αυτό το παράδειγμα. Στο υπόλοιπο αυτής της ενότητας θα εξηγήσουμε κάθε τμήμα του λεπτομερώς.

2.1.1 Συστατικά του μοντέλου

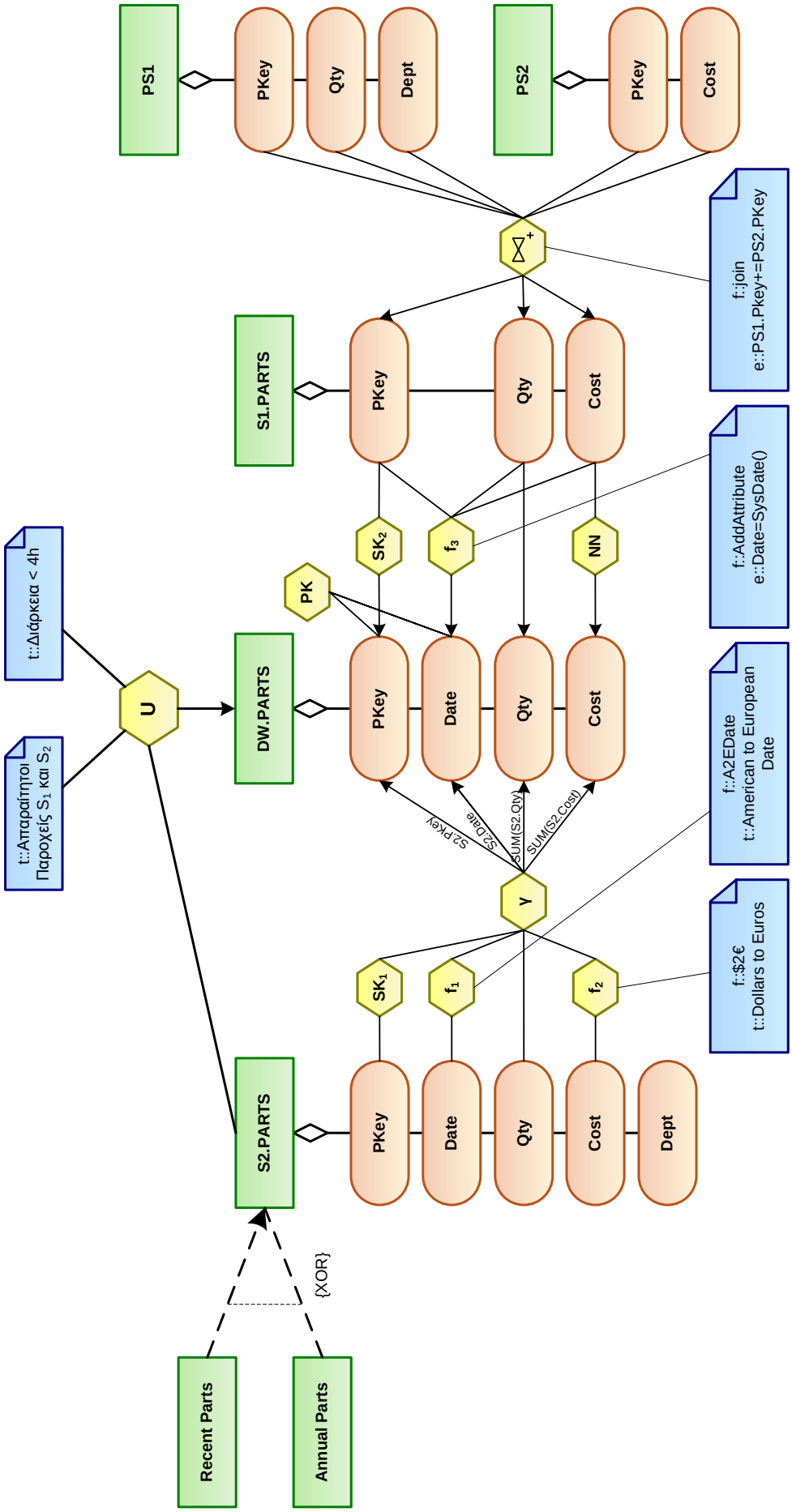
Σε αυτή την ενότητα παρουσιάζονται τα επιμέρους δομικά συστατικά του μοντέλου. Περιγράφονται τα γνωρίσματα, οι έννοιες, οι μετασχηματισμοί, οι περιορισμοί ΕΜΦ και τα σχόλια. Επίσης, περιγράφονται τα διαφορετικά είδη σχέσεων που προβλέπει το εννοιολογικό μοντέλο: σχέση μέρους, σχέση υποψηφίου και σχέση παροχής. Τέλος, παρουσιάζεται η σειριακή σύνθεση μετασχηματισμών που επεκτείνει τη δυνατότητα του μοντέλου να καλύψει πιο σύνθετες περιπτώσεις διεργασιών ΕΜΦ.

Γνωρίσματα. Το *γνώρισμα* αποτελεί τη στοιχειώδη μονάδα πληροφορίας στο μοντέλο. Ο ρόλος των γνωρισμάτων είναι ο ίδιος όπως στα τυποποιημένα διαγράμματα Ο/Σ. Υιοθετούμε τη γραφική αναπαράσταση των γνωρισμάτων του τυποποιημένου προτύπου Ο/Σ και απεικονίζουμε τα γνωρίσματα με οβάλ σχήμα.

Έννοιες. Η *έννοια* αντιπροσωπεύει μία οντότητα στις πηγές ή στην Αποθήκη Δεδομένων. Παραδείγματα εννοιών είναι τα αρχεία δεδομένων στις πηγές, οι πίνακες γεγονότων και διαστάσεων στην Αποθήκη Δεδομένων κ.λ.π.. Η έννοια ορίζεται τυπικά από:

1. ένα όνομα, και
2. ένα σύνολο γνωρισμάτων

ενώ απεικονίζεται γραφικά με ορθογώνιο σχήμα. Με βάση το μοντέλο Ο/Σ, η έννοια αποτελεί γενίκευση οντοτήτων και συσχετίσεων. Ανεξάρτητα όμως από το χρησιμοποιούμενο μοντέλο, είτε πρόκειται για επέκταση του μοντέλου Ο/Σ, είτε για το μοντέλο διαστάσεων, όλες οι οντότητες που συντίθενται από ένα σύνολο γνωρισμάτων, είναι γενικά στιγμιότυπα της κλάσης “έννοια”.



Σχήμα 2.3: Διάγραμμα του εννοιολογικού μοντέλου για το παράδειγμα

Όπως αναφέρεται στο [45], μπορούμε να χρησιμοποιήσουμε διάφορες *φυσικές* δομές αποθήκευσης ως πεπερασμένες λίστες γνωρισμάτων, συμπεριλαμβανομένων των σχεσιακών βάσεων δεδομένων, αρχείων COBOL ή ASCII, πολυδιάστατων κύβων και διαστάσεων. Οι έννοιες είναι απόλυτα ικανές να μοντελοποιήσουν τέτοιου είδους δομές, πιθανότατα μέσω ενός γενικευμένου (IsA) μηχανισμού. Ας θεωρήσουμε για παράδειγμα δομές ΣΑΕΔ. Οι συσχετίσεις των επιπέδων και των τιμών, δε σχετίζονται με την περίπτωση των διεργασιών ΕΜΦ. Η υιοθέτηση των εννοιών είναι αρκετή για το πρόβλημα μοντελοποίησης διεργασιών ΕΜΦ. Εξάλλου, μπορούμε να διασπάσουμε τη γενική δομή *Έννοια* σε υποκλάσεις που θα φέρουν τα χαρακτηριστικά οποιασδήποτε από τις προαναφερθείσες προσεγγίσεις (π.χ. υποκλάσεις *Πίνακας Γεγονότων* και *Πίνακας Διάσταση*), επιτυγχάνοντας έτσι ένα ομοιογενή τρόπο για τη μεταχείριση μοντέλων ΣΑΕΔ και ΕΜΦ.

Επιστρέφοντας στο παράδειγμα του Σχήματος 2.3, παρατηρούμε τις έννοιες **PS1, PS2, S1.PARTS, S2.PARTS** και **DW.PARTS** μαζί με τα γνωρίσματά τους.

Μετασχηματισμοί. Ο *μετασχηματισμός* αντιπροσωπεύει μέρος ή πλήρη ενότητα κώδικα, εκτελώντας μία συγκεκριμένη διαδικασία. Διαχωρίζουμε δύο γενικές κατηγορίες μετασχηματισμών: (α) μετασχηματισμούς που συντηρούν το σχήμα των εισερχόμενων δεδομένων, όπως το φιλτράρισμα ή ο καθαρισμός των δεδομένων (π.χ. έλεγχος για παραβάσεις πρωτεύοντος ή δευτερεύοντος κλειδιού), και (β) μετασχηματισμούς που μεταβάλλουν το σχήμα των εισερχόμενων δεδομένων (π.χ. συνάθροιση). Τυπικά, ένας μετασχηματισμός ορίζεται από (α) ένα πεπερασμένο σύνολο γνωρισμάτων εισόδου, (β) ένα πεπερασμένο σύνολο γνωρισμάτων εξόδου, και (γ) ένα γραφικό σύμβολο που χαρακτηρίζει τη φύση του μετασχηματισμού. Ο μετασχηματισμός απεικονίζεται γραφικά με ένα εξάγωνο σχήμα.

Στο παράδειγμα του Σχήματος 2.3, μπορούμε να δούμε αρκετούς μετασχηματισμούς. Παρατηρήστε, λόγου χάρη, αυτούς που σχετίζονται με τη διάδοση δεδομένων από την έννοια **S1.PARTS** στην **DW.PARTSUPP**. Υπάρχει ένας μετασχηματισμός ανάθεσης υποκατάστατου κλειδιού (κεψωορδΣΚ1), μία συνάρτηση υπολογισμού της ημερομηνίας συστήματος (**f1**) και ένας έλεγχος για κενές τιμές (**NN**) στο γνώρισμα **Cost**.

Οι μετασχηματισμοί δε χρησιμοποιούνται ως αυτόνομες οντότητες στο μοντέλο, αντίθετα αποτελούν μέρος άλλων δομικών συστατικών αυτού. Κατά συνέπεια, περισσότερα για τη λειτουργία τους θα αναφερθούν κατά την περιγραφή των συστατικών που τους χρησιμοποιούν.

Περιορισμοί ΕΜΦ. Σε αρκετές περιπτώσεις, ο σχεδιαστής θέλει να τονίσει το γεγονός ότι τα δεδομένα μίας συγκεκριμένης έννοιας οφείλουν να πληρούν κάποιες απαιτήσεις. Για παράδειγμα, ο σχεδιαστής μπορεί να θέλει να επιβάλει ένα περιορισμό πρωτεύοντος κλειδιού ή ένα περιορισμό μη-μηδενικής τιμής σε ένα σύνολο γνωρισμάτων. Για να καλυφθούν τέτοιες ανάγκες εισάγονται οι *περιορισμοί ΕΜΦ*, που τυπικά ορίζονται ως:

1. ένα πεπερασμένο σύνολο γνωρισμάτων στα οποία εφαρμόζεται ο περιορισμός, και

2. ένας μετασχηματισμός ο οποίος επιβάλλει τον περιορισμό.

Σ' αυτό το σημείο πρέπει να τονιστεί ότι παρ' όλη την ομοιότητα στο όνομα, οι περιορισμοί ΕΜΦ είναι διαφορετικά στοιχεία μοντελοποίησης από τους γνωστούς περιορισμούς UML. Ένας περιορισμός ΕΜΦ απεικονίζεται γραφικά σαν ένα σύνολο από συνεχείς γραμμές που ξεκινούν από τα εμπλεκόμενα γνωρίσματα και καταλήγουν στο μετασχηματισμό υλοποίησης του περιορισμού. Στο παράδειγμα του Σχήματος 2.3, εφαρμόζεται ένας περιορισμός ΕΜΦ πρωτεύοντος κλειδιού (**PK**) στην έννοια **DW.PARTSUPP** που αφορά τα γνωρίσματα **PKey**, **SuppKey**, **Date**.

Σχόλια. Το *σχόλιο*, όπως ακριβώς και στη μοντελοποίηση UML αποτελεί μία άτυπη ετικέτα για να αποτυπωθούν πρόσθετα σχόλια που επιθυμεί να κάνει ο σχεδιαστής στη φάση σχεδιασμού, ή για να εκφράσει περιορισμούς που αφορούν κάποια στοιχεία ή σύνολα στοιχείων [7]. Στο εννοιολογικό μοντέλο που περιγράφουμε, τα σχόλια χρησιμοποιούνται για:

1. επεξηγήσεις στη σημασιολογία των εφαρμοσμένων μετασχηματισμών, και/ή
2. σύντομα σχόλια που εξηγούν σχεδιαστικές αποφάσεις.

Θεωρούμε ότι η πληροφορία που περιέχεται σε ένα σχόλιο της πρώτης κατηγορίας μας δηλώνει, είτε το είδος είτε μία έκφραση/συνθήκη μίας συνάρτησης, και προσαρτάται σε ένα μετασχηματισμό ή σε ένα περιορισμό ΕΜΦ. Η πληροφορία ενός σχολίου της δεύτερης κατηγορίας είναι απλό κείμενο, χωρίς κάποια ειδική σημασιολογία. Αυτού του τύπου σχόλια χρησιμοποιούνται για να καλύψουν της διαφορετικές πτυχές των διεργασιών ΕΜΦ, όπως ο προγραμματισμός με βάση το χρόνο ή κάποιο γεγονός, η επίβλεψη, η καταγραφή συμβάντων, η αντιμετώπιση λαθών, η ανάνηψη από κατάρρευση κ.λ.π.. Όπως και στη UML τα σχόλια απεικονίζονται ως ορθογώνια με την πάνω δεξιά γωνία διπλωμένη. Τυπικά, ένα σχόλιο ορίζεται από:

1. ένα όνομα,
2. κάποιο περιεχόμενο, το οποίο αποτελείται από μία ή περισσότερες προτάσεις της μορφής: <**τύπος**>::<**κείμενο**>.

Τα διάφορα είδη πληροφορίας (προτάσεις) στο περιεχόμενο ενός σχολίου, χωρίζονται με τη χρήση ενός προθέματος πριν γραφεί το περιεχόμενο της πληροφορίας. Το πρόθεμα αυτό μπορεί να είναι (α) **f::** για το είδος μίας συνάρτησης, (β) **e::** για μία έκφραση, ή (γ) **t::** για απλό κείμενο. Για να διατηρηθεί το μοντέλο απλό, ο σχεδιαστής δεν είναι υποχρεωμένος να προσαρτήσει σχόλιο σε κάθε ένα μετασχηματισμό ή περιορισμό ΕΜΦ. Ακόμα, δεν είναι υποχρεωμένος να ορίσει πληροφορία για κάθε ένα από τους τρεις παραπάνω τύπους σε κάθε σχόλιο.

Στο παράδειγμα 1 (Σχήμα 2.3), παρατηρούμε αρκετά σχόλια της πρώτης κατηγορίας, συνημμένα στους μετασχηματισμούς **f1**, **f2**, **f3** και $\bowtie_+$. Για παράδειγμα, το σχόλιο που είναι συνημμένο στο μετασχηματισμό **f3** υποδεικνύει ότι το είδος του προτύπου του μετασχηματισμού είναι **AddAttribute** και η

έκφραση που απαιτείται για τη στιγμιοτυποποίηση είναι **Date=SysDate()**. Η χρήση των τύπων συναρτήσεων και των εκφράσεων γίνεται για την απεικόνιση των μετασχηματισμών του εννοιολογικού μοντέλου σε κάποια πρότυπη διεργασία του λογικού μοντέλου. Το θέμα αυτό εξετάζεται στο Κεφάλαιο 4, ενώ ο αναγνώστης μπορεί να βρει περισσότερες λεπτομέρειες στο [46].

Επιπλέον, τα σχόλια χρησιμοποιούνται σαν περιορισμοί χρόνου εκτέλεσης για την αναγνώριση των ιδιοτήτων που πρέπει να πληρούνται. Για παράδειγμα, στο πάνω μέρος του Σχήματος 2.3, μπορούμε να δούμε ένα περιορισμό χρόνου εκτέλεσης που υποδηλώνει ότι η διάρκεια της φόρτωσης της **DW.PARTS** (που περιλαμβάνει τη φόρτωση των **S1.PARTS** και **S2.PARTS**) δεν μπορεί να ξεπεράσει τις 4 ώρες.

Σχέσεις Υποψηφίου. Στη σχεδίαση Αποθηκών Δεδομένων, είναι πολύ συνηθισμένο, ειδικά στα πρώτα στάδια σχεδιασμού του έργου, να υπάρχουν περισσότερες από μία πιθανές έννοιες (πηγές) για τη φόρτωση μίας έννοιας (στόχου) στην Αποθήκη Δεδομένων. Κάθε μία από τις πιθανές έννοιες ονομάζεται *υποψήφια*. Επομένως, οι *σχέσεις υποψηφίου* χρησιμοποιούνται για να υποδηλώσουν το γεγονός ότι ενδέχεται να υπάρχει ένα σύνολο πηγών ικανών να διαδώσουν δεδομένα σε μία συγκεκριμένη έννοια. Τυπικά, μία σχέση υποψηφίου περιλαμβάνει:

1. ακριβώς μία υποψήφια έννοια, και
2. ακριβώς μία έννοια στόχο.

Οι σχέσεις υποψηφίου απεικονίζονται με έντονες διακεκομμένες γραμμές, μεταξύ των υποψηφίων και της έννοιας–στόχου. Όταν ακριβώς μία από αυτές πρέπει να επιλεγεί, σημειώνουμε το σύνολο των σχέσεων για τη συγκεκριμένη έννοια με ένα περιορισμό UML **{XOR}**.

Σχέσεις Ενεργού Υποψηφίου. Μία *σχέση ενεργού υποψηφίου* εκφράζει το γεγονός ότι, από ένα σύνολο υποψηφίων πηγών για μία έννοια, μόνο ένας υποψήφιος επιλέγεται για να τροφοδοτήσει τη συγκεκριμένη έννοια. Ο υποψήφιος που τελικά επιλέγεται ονομάζεται *ενεργός υποψήφιος*. Κατά συνέπεια, μία σχέση ενεργού υποψηφίου είναι μια εξειδίκευση της σχέσης υποψηφίου με την ίδια δομή, αλλά πιο αυστηρή σημασιολογία. Η σχέση ενεργού υποψηφίου συμβολίζεται γραφικά με ένα έντονο διακεκομμένο βέλος από την έννοια–πηγή στην έννοια–στόχο.

Στο παράδειγμα του Σχήματος 2.3, υποθέτουμε ότι η πηγή **S2** διαθέτει περισσότερα από ένα συστήματα παραγωγής (π.χ. αρχεία COBOL), τα οποία χαρακτηρίζονται ως υποψήφια για το **S2.PARTS**. Έτσι οι διαθέσιμοι υποψήφιοι (που απεικονίζονται στο πάνω αριστερά τμήμα του Σχήματος 2.3) είναι:

- Μία έννοια **Annual Parts** (στην πράξη απεικονίζει ένα αρχείο **F1**), η οποία περιέχει το πλήρες ετήσιο ιστορικό των προμηθευτών εξαρτημάτων. Η κύρια χρήση της αφορά σε αναφορές και περιέχει ένα υπερσύνολο των γνωρισμάτων που χρειάζονται για τις ανάγκες τις Αποθήκης Δεδομένων.

- Μία έννοια **Recent Parts** (στην πράξη απεικονίζει ένα αρχείο **F2**), η οποία περιέχει μόνο τα στοιχεία του τελευταίου μήνα. Χρησιμοποιείται συνεχώς από τους τελικούς χρήστες για την εισαγωγή ή την ανανέωση των δεδομένων, όπως επίσης και από μερικά προγράμματα αναφορών.

Στο Σχήμα 2.3 παρατηρούμε επίσης, ότι η έννοια **Recent Parts** επιλέχθηκε ως ενεργή υποψήφια.

Σχέσεις Παροχής. Μία *σχέση παροχής* απεικονίζει ένα σύνολο γνωρισμάτων εισόδου σε ένα σύνολο γνωρισμάτων εξόδου μέσω ενός μετασχηματισμού. Στην απλή **1:1** περίπτωση, οι σχέσεις παροχής εκφράζουν το γεγονός ότι ένα γνώρισμα εισόδου των πηγών δεδομένων μεταφέρεται σε ένα γνώρισμα εξόδου στην Αποθήκη Δεδομένων. Αν τα γνωρίσματα είναι φυσικά και σημασιολογικά ισοδύναμα, τότε δεν απαιτείται μετασχηματισμός. Σε αντίθετη περίπτωση, η αντιστοίχηση μεταξύ των γνωρισμάτων γίνεται μέσω κατάλληλου μετασχηματισμού (π.χ. μετατροπή ημερομηνίας από ευρωπαϊκή σε αμερικάνικη μορφή, έλεγχος για κενές τιμές, κ.λ.π.).

Γενικά, υπάρχει η περίπτωση αλλαγής του σχήματος των δεδομένων εισόδου. Κατά συνέπεια, οι σχέσεις παροχής τυπικά ορίζονται από:

1. ένα πεπερασμένο σύνολο γνωρισμάτων εισόδου,
2. ένα πεπερασμένο σύνολο γνωρισμάτων εξόδου, και
3. ένα κατάλληλο μετασχηματισμό (τέτοιο ώστε τα γνωρίσματα εισόδου και εξόδου του να απεικονίζονται ένα προς ένα με τα αντίστοιχα γνωρίσματα της σχέσης).

Στην περίπτωση **1:1**, μία σχέση παροχής απεικονίζεται με έντονο βέλος από το γνώρισμα εισόδου στο γνώρισμα εξόδου, ενώ πάνω στο βέλος σημειώνεται ο μετασχηματισμός που χρησιμοποιείται. Στη γενική περίπτωση **N:M**, μία σχέση προμηθευτή απεικονίζεται σαν ένα σύνολο από έντονα βέλη από τα γνωρίσματα εισόδου, προς τα γνωρίσματα εξόδου, μέσω κατάλληλου μετασχηματισμού. Η γραφική αναπαράσταση της σχέσης παροχής **N:M** αποκρύπτει την απεικόνιση μεταξύ των γνωρισμάτων εισόδου και εξόδου. Για να αντισταθμιστεί αυτό το μειονέκτημα, η σύνδεση της σχέσης παροχής με κάθε ένα από τα εμπλεκόμενα γνωρίσματα εξόδου συνοδεύεται από μία ετικέτα, ώστε να μην υπάρχει αμφιβολία για τον προμηθευτή ενός γνωρίσματος εξόδου. Για παράδειγμα, στο Σχήμα 2.3 η σχέση παροχής που αφορά το μετασχηματισμό **γ** είναι **N:M**. Για να μην υπάρχει αμφιβολία για την ένα προς ένα απεικόνιση των γνωρισμάτων εξόδου σε αυτά της εισόδου, η σύνδεση της σχέσης με τα γνωρίσματα εξόδου συνοδεύεται από μία ετικέτα για κάθε γνώρισμα εξόδου που φέρει το όνομα του αντίστοιχου γνωρίσματος εισόδου.

Τέλος, θα πρέπει να σημειωθεί μια συντακτική προσθήκη στο μοντέλο αυτό. Μερικές φορές τυχαίνει μία συγκεκριμένη σχέση παροχής να περιλαμβάνει όλα τα γνωρίσματα ενός συνόλου από έννοιες. Για παράδειγμα, στην περίπτωση της ένωσης, όλα τα γνωρίσματα των εννοιών εισόδου και εξόδου συμμετέχουν στο μετασχηματισμό. Για την αποφυγή της υπερφόρτωσης του σχήματος με υπερβολικά πολλές σχέσεις, εισάγεται ένας συντακτικός συμβολισμός που απεικονίζει τις έννοιες εισόδου

στις έννοιες εξόδου (αντί της απεικόνισης των γνωρισμάτων εισόδου στα γνωρίσματα εξόδου). Αυτό μπορεί να χρησιμοποιηθεί σαν μία λειτουργία σμίκρυνσης—μεγέθυνσης στο διάγραμμα. Στο πρώτο επίπεδο απεικονίζονται μόνο οι έννοιες και δίνεται μία γενική επισκόπηση του σεναρίου. Στο δεύτερο, και πιο λεπτομερές επίπεδο, οι σχέσεις μεταξύ των εννοιών επεκτείνονται στις σχέσεις μεταξύ των εμπλεκομένων γνωρισμάτων, οπότε και παρέχεται το σενάριο ΕΜΦ σε όλη του τη λεπτομέρια.

Επιστρέφοντας και πάλι στο Σχήμα 2.3, εξετάζουμε τις σχέσεις μεταξύ των γνωρισμάτων των εννοιών **S1.PARTS** και **S2.PARTS**. Αρχικά, αγνοούμε τη συνάθροιση **γ** που πραγματοποιείται στα δεδομένα της πηγής **S2** και εξετάζουμε τους άλλους μετασχηματισμούς.

- Το γνώρισμα **PKey** παίρνει στοιχεία απ' ευθείας από το ομώνυμο γνώρισμα των **S1** και **S2**, μέσω ενός μετασχηματισμού ανάθεσης υποκατάστατου κλειδιού (**SK**). Η ανάθεση ενός υποκατάστατου κλειδιού αποτελεί συνηθισμένη τακτική στις Αποθήκες Δεδομένων, που χρησιμοποιείται για την αντικατάσταση των κλειδιών των συστημάτων παραγωγής με ένα ομοιόμορφο κλειδί. Γενικά, οι βασικοί λόγοι για τη χρήση ενός μετασχηματισμού ανάθεσης υποκατάστατου κλειδιού είναι τόσο η επίδοση, όσο και η ομοιογένεια της σήμανσης. Γνωρίσματα τύπου συμβολοακολουθίας, γενικά δεν κρίνονται ως καλές περιπτώσεις κλειδιού για χρήση σε ευρετήριο και συνήθως απαιτείται η αντικατάσταση ενός τέτοιου κλειδιού από κλειδιά αριθμητικού τύπου. Παράλληλα, διαφορετικά συστήματα παραγωγής ενδέχεται να χρησιμοποιούν διαφορετικά κλειδιά για τα ίδια αντικείμενα ή το ίδιο κλειδί για διαφορετικά αντικείμενα, με αποτέλεσμα την ανάγκη για συνολική αλλαγή αυτών των τιμών στην Αποθήκη Δεδομένων. Έστω η περίπτωση όπου το εξάρτημα **Τιμόνι** έχει **PKey=30** στην πηγή **S1** και **PKey=40** στην πηγή **S2**, ενώ στην πηγή **S2** το **PKey=30** να αντιστοιχεί στο εξάρτημα **Πόρτα**. Τέτοιες συγκρούσεις είναι εύκολο να λυθούν με ένα συνολικό μηχανισμό αντικατάστασης, μέσω της αντιστοίχησης ενός ομοιόμορφου υποκατάστατου κλειδιού.

- Το γνώρισμα **Date** δέχεται δεδομένα από το ομώνυμο γνώρισμα της **S2**, μέσω ενός μετασχηματισμού **AmericanToEuropeanDate**. Παράλληλα, η ημερομηνία των εγγραφών που προέρχονται από την **S1** καθορίζεται από την εφαρμογή της συνάρτησης **SysDate()** (διότι η έννοια **S1.PARTS** δεν περιέχει το γνώρισμα αυτό). Παρατηρήστε τη λειτουργία που εφαρμόζεται για τις εγγραφές που προέρχονται από την πηγή **S1**: δέχονται ως είσοδο όλα τα γνωρίσματα της **S1.PARTS** (για να διαπιστωθεί ότι η παραγόμενη τιμή είναι ένα νέο γνώρισμα), περνούν από ένα μετασχηματισμό τύπου συνάρτησης που υπολογίζει την ημερομηνία συστήματος, και από εκεί καταλήγουν στο γνώρισμα **DW.Date**.

- Το γνώρισμα **Qty** παίρνει τις τιμές του απ' ευθείας από τα ομώνυμα γνωρίσματα των δύο πηγών χωρίς την ανάγκη κάποιου μετασχηματισμού.

- Το γνώρισμα **Cost** δέχεται δεδομένα από τα ομώνυμα γνωρίσματα των δύο πηγών. Όσον αφορά στην πηγή **S2**, εφαρμόζεται ένας μετασχηματισμός $\$ \rightarrow €$ για τη μετατροπή του κόστους των εξαρτημάτων σε ευρωπαϊκές τιμές. Όσον αφορά στην πηγή **S1**, εφαρμόζεται ένας μετασχηματισμός μη-κενής τιμής (**NN**), για να αποφευχθεί η φόρτωση στην Αποθήκη Δεδομένων εγγραφών που δεν περιλαμβάνουν κόστος εξαρτημάτων.

Σημειώνουμε πως ενδέχεται να υπάρχουν γνωρίσματα εισόδου τα οποία αγνοούνται κατά τη διάρκεια των μετασχηματισμών ΕΜΦ, όπως για παράδειγμα το **S2.Parts.Dept**.

Σειριακή Σύνθεση Μετασχηματισμών. Συνήθως οι διεργασίες ΕΜΦ περιλαμβάνουν περισσότερους από έναν μετασχηματισμούς για κάθε γνώρισμα. Επομένως υπάρχει ανάγκη περισσοτέρων του ενός μετασχηματισμού σε μία σχέση παροχής. Για παράδειγμα, θα μπορούσαμε να ομαδοποιήσουμε τα δεδομένα εισόδου ως προς κάποιο σύνολο γνωρισμάτων, έχοντας εξασφαλίσει ταυτόχρονα, ότι δεν εμπλέκονται κενές τιμές στη διαδικασία αυτή. Σε μια τέτοια περίπτωση χρειάζεται να εφαρμόσουμε ένα μετασχηματισμό μη-κενής τιμής σε κάθε ένα από τα γνωρίσματα και στη συνέχεια να μεταφέρουμε μόνο τις σωστές σχέσεις στην ομαδοποίηση. Για την επίτευξη αυτού χρειάζεται η σειριακή εφαρμογή των εν λόγω μετασχηματισμών. Ένα πρόβλημα που διαφαίνεται αφορά την απαίτηση, σύμφωνα με τον ορισμό, ένας μετασχηματισμός έχει ένα σύνολο γνωρισμάτων ως είσοδο και ένα σύνολο γνωρισμάτων ως έξοδο. Έτσι, η απλή σύνδεση δύο μετασχηματισμών φαίνεται ασυνεπής. Για να ξεπεραστεί αυτό, εισάγεται η *σειριακή σύνθεση των μετασχηματισμών*. Τυπικά, η σειριακή σύνθεση μετασχηματισμών περιλαμβάνει:

1. ένα μοναδικό μετασχηματισμό έναρξης, και
2. ένα μοναδικό μετασχηματισμό συνέχειας.

Η σειριακή σύνθεση μετασχηματισμών απεικονίζεται γραφικά με έντονες συνεχείς γραμμές που ενώνουν τους εμπλεκόμενους μετασχηματισμούς.

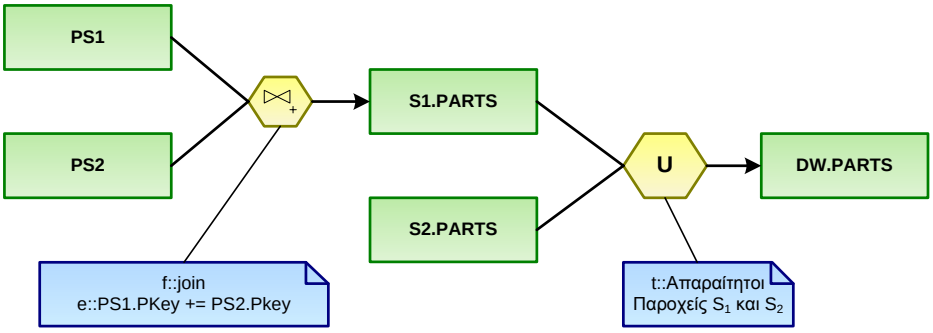
Ένα πιο σύνθετο μέρος του παραδείγματος του Σχήματος 2.3, είναι η συνάθροιση που εφαρμόζεται στα δεδομένα από την πηγή **S2**. Στο παράδειγμα, η πηγή **S2** κρατά πληροφορία για τους προμηθευτές εξαρτημάτων ανάλογα με το τμήμα στο οποίο ανήκουν. Η φόρτωση δεδομένων στην Αποθήκη Δεδομένων, η οποία αγνοεί αυτή τη λεπτομέρεια, απαιτεί την ομαδοποίηση των δεδομένων ανά **PKey** και **Date**, και τη συνάθροιση των **Cost** και **Qty**. Η συγκεκριμένη λειτουργία επιτυγχάνεται από το μετασχηματισμό **γ**. Παράλληλα, οι προαναφερθέντες μετασχηματισμοί δεν αγνοούνται, αλλά ο καθένας μαζί με το μετασχηματισμό συνάθροισης **γ** αποτελούν σειριακή σύνθεση μετασχηματισμών. Σημειώστε επίσης τις ετικέτες στην έξοδο του μετασχηματισμού συνάθροισης που επισημαίνουν τους προμηθευτές δεδομένων στα αντίστοιχα γνωρίσματα εξόδου (π.χ. **S2.PARTS.PKey** για το **DW.PARTS.PKey** και **SUM(S2.PARTS.Qty)** για το **DW.PARTS.Qty**).

2.2 Μεθοδολογία Χρήσης του Εννοιολογικού Μοντέλου

Στην ενότητα αυτή, παρατίθεται η σειρά βημάτων που ένας σχεδιαστής ακολουθεί κατά τη διάρκεια της κατασκευής της Αποθήκης Δεδομένων. Κάθε βήμα αυτής της μεθοδολογίας θα παρουσιαστεί με αναφορά στο παράδειγμα του Σχήματος 2.3. Όπως έχει ήδη διευκρινιστεί, ο τελικός στόχος της

διαδικασίας αυτής είναι η δημιουργία των απεικονίσεων μεταξύ των γνωρισμάτων, ο προσδιορισμός των απαραίτητων μετασχηματισμών, καθώς και η συλλογή οποιασδήποτε βοηθητικής πληροφορίας.

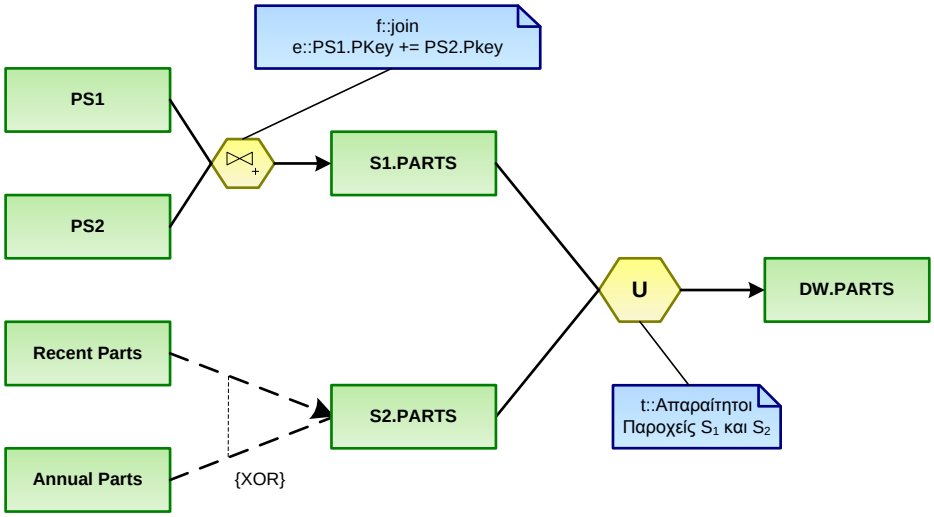
Βήμα 1ο: Αναγνώριση των κατάλληλων πηγών δεδομένων. Το πρώτο πράγμα που θα αντιμετωπίσει ένας σχεδιαστής κατά την περίοδο ανάλυσης και συλλογής των απαιτήσεων της Αποθήκης Δεδομένων, είναι η αναγνώριση των σχετικών πηγών δεδομένων. Υποθέστε ότι για ένα υποσύνολο της Αποθήκης Δεδομένων, έχουμε προσδιορίσει την έννοια **DW.PARTS** ως τον πίνακα συμβάντων για το πως διανέμονται τα εξαρτήματα ανάλογα με τον προμηθευτή τους. Υποθέστε επίσης, ότι έχουμε αποφασίσει για λόγους πληρότητας, πως πρέπει να γεμίζουμε τον πίνακα του αποτελέσματος με δεδομένα και από τις δύο πηγές **S1** και **S2**, δηλαδή χρειαζόμαστε την ένωση των δεδομένων από τις δύο αυτές πηγές. Επιπλέον, για να φορτώσουμε δεδομένα στην έννοια **S1.PARTS** απαιτείται μία εξωτερική συνένωση με της έννοιες **PS1** και **PS2**. Με βάση τις προηγούμενες παρατηρήσεις, το διάγραμμα των πηγών που επιλέξαμε και των σχέσεων μεταξύ αυτών, φαίνεται στο Σχήμα 2.4.



Σχήμα 2.4: Αναγνώριση των κατάλληλων πηγών δεδομένων

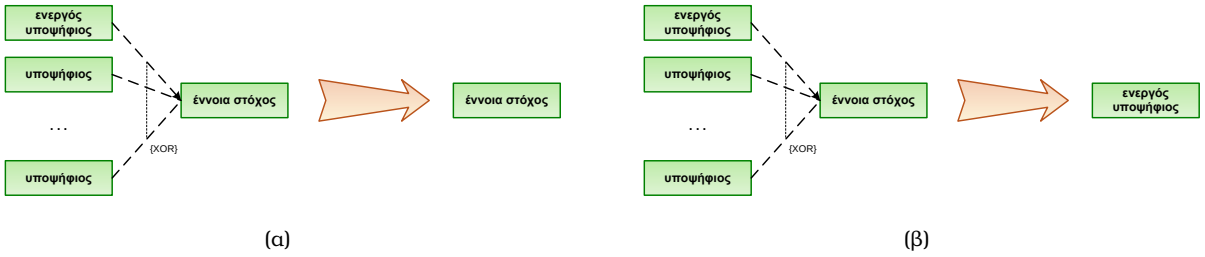
Βήμα 2ο: Υποψήφιοι και Ενεργοί Υποψήφιοι των εμπλεκομένων πηγών δεδομένων. Όπως ήδη έχουμε αναφέρει, κατά τη διάρκεια της ανάλυσης απαιτήσεων, ο σχεδιαστής ενδέχεται να ανακαλύψει ότι περισσότερες από μία πηγές δεδομένων μπορούν να είναι υποψήφιες για να συμπληρώσουν κάποια συγκεκριμένη έννοια. Ο τρόπος με τον οποίο λαμβάνεται μία τέτοια απόφαση είναι πέραν του σκοπού αυτής της εργασίας. Επιγραμματικά, όμως, αναφέρουμε ότι σημαντικό ρόλο σε τέτοιες αποφάσεις παίζουν το πλήθος, η ποσότητα των δεδομένων, αλλά και η διαθεσιμότητα των πηγών. Στο παράδειγμα του Σχήματος 2.3 υποθέσαμε ότι η πηγή **S2** έχει περισσότερα από ένα συστήματα παραγωγής (όπως, για παράδειγμα, αρχεία COBOL), τα οποία είναι υποψήφια για την έννοια **S2.PARTS**. Όπως είδαμε, οι διαθέσιμες πηγές είναι: η έννοια **Annual Parts** (που περιέχει το πλήρες, ετήσιο ιστορικό των προμηθευτών εξαρτημάτων) και η έννοια **Recent Parts** (που περιέχει μόνο τα στοιχεία του τελευταίου μήνα). Οι υποψήφιες έννοιες για την έννοια **S2.PARTS** και η (με βάση το σενάριο του παραδείγματος) ενεργή υποψήφια **Recent Parts**, καθώς και η διαδικασία για την επιλογή αυτή, απεικονίζονται στο Σχήμα 2.5.

Προφανώς, όταν ληφθεί η απόφαση για τον ενεργό υποψήφιο, μπορεί να δημιουργηθεί ένα απλοποιημένο, "λειτουργικό αντίγραφο" του σεναρίου στο οποίο να αποβάλλονται όλοι οι άλλοι υποψήφιοι.



Σχήμα 2.5: Υποψήφιοι και ενεργοί υποψήφιοι για τις εμπλεκόμενες πηγές δεδομένων

Όπως φαίνεται στο Σχήμα 2.6, υπάρχουν δύο τρόποι να επιτευχθεί αυτό (α) αγνοώντας όλες τις πληροφορίες που αφορούν τους υποψηφίους για την έννοια–στόχο, και (β) αντικαθιστώντας την έννοια–στόχο με τον ενεργό υποψήφιο. Με άλλα λόγια, αντί ο σχεδιαστής να φορτώνει το διάγραμμα με πληροφορία όχι άμεσα χρήσιμη, μπορεί να χρησιμοποιήσει μία απλουστευμένη μορφή του. Οι κρυμμένοι υποψήφιοι δε σβήνονται, αντιθέτως παραμένουν μέρη του εννοιολογικού μοντέλου της εξεταζόμενης Αποθήκης Δεδομένων, ώστε να είναι δυνατό να χρησιμοποιηθούν σε μετέπειτα στάδια της λειτουργίας της Αποθήκης Δεδομένων. Ως παράδειγμα, αναφέρουμε την περίπτωση όπου πιθανές αλλαγές στον ενεργό υποψήφιο ενδέχεται να οδηγήσουν σε αναθεώρηση της επιλογής του ως ενεργού υποψηφίου. Το μόνο που αλλάζει είναι ότι απλά οι κρυμμένοι υποψήφιοι δεν φαίνονται στο σχεδιαστή.



Σχήμα 2.6: Απλοποίηση του διαγράμματος: (α) αγνοώντας υποψηφίους, (β) θεωρώντας τον ενεργό υποψήφιο

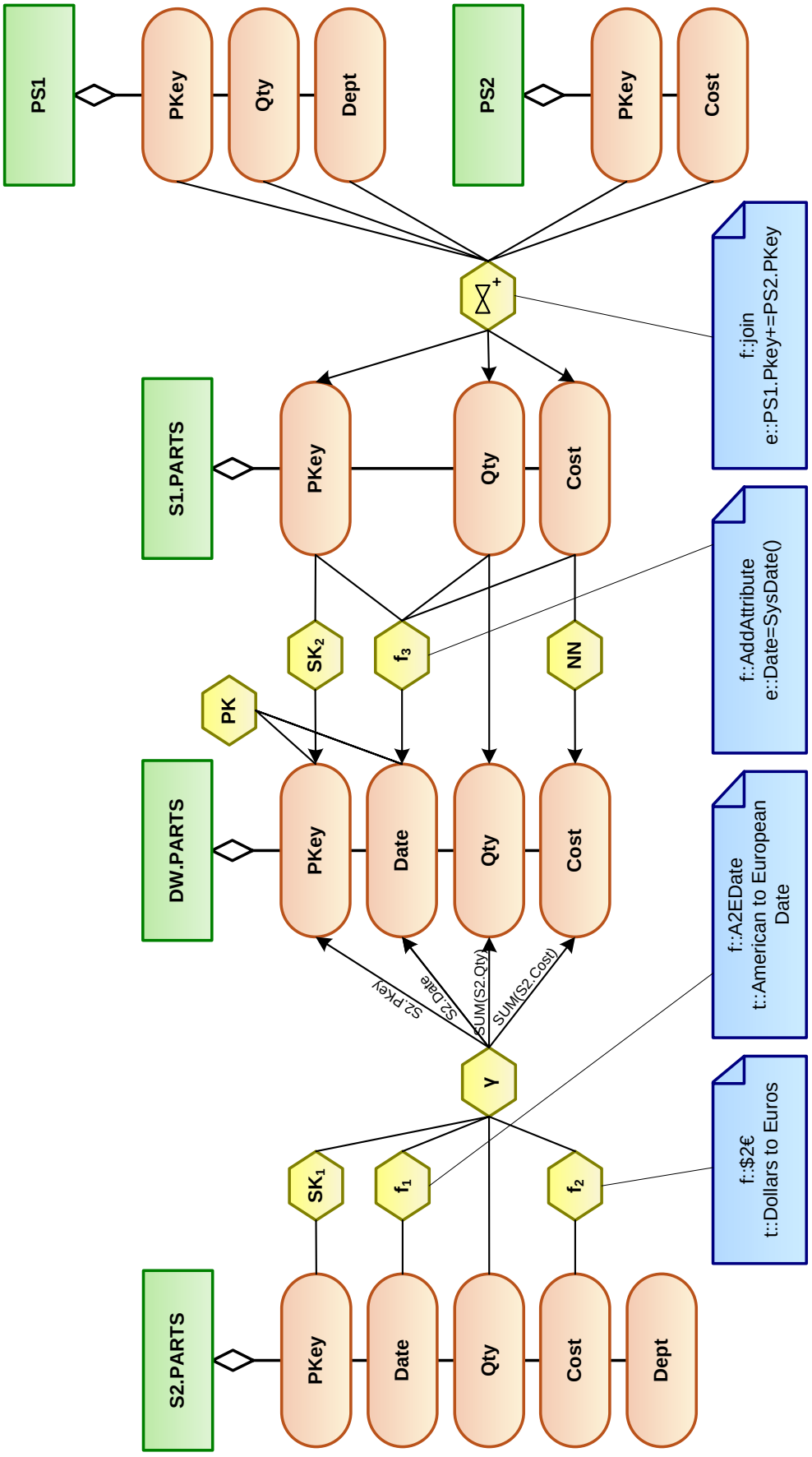
Βήμα 3ο: Αντιστοίχηση γνωρισμάτων μεταξύ πηγών και στόχων. Η πιο δύσκολη διαδικασία για το σχεδιαστή της Αποθήκης Δεδομένων είναι να καθορίσει την αντιστοίχηση των γνωρισμάτων των πηγών με αυτών της Αποθήκης Δεδομένων. Η διαδικασία αυτή περιλαμβάνει αρκετές συζητήσεις με τους διαχειριστές των πηγών δεδομένων ώστε να αποσαφηνιστούν σημεία όπως ο κώδικας, οι κανόνες ή οι τιμές που είναι κρυμμένα στα δεδομένα ή στα προγράμματα των πηγών. Επιπλέον, περιλαμβάνει αρκετές προσπάθειες "προεπισκόπησης δεδομένων" (με τη μορφή δειγματοληψίας ή με απλές ερωτήσεις καταμέτρησης) για να διαπιστωθούν πιθανά προβλήματα των δεδομένων που

παρέχονται. Όπως είναι φυσικό, αυτή η διαδικασία είναι αλληλεπιδραστική και επιρρεπής σε λάθη. Η υποστήριξη που μπορεί να δοθεί στο χρήστη από ένα εργαλείο, εναπόκειται κυρίως στην αντιστοίχηση μεταξύ των εμπλεκομένων γνωρισμάτων, με το ενδιαφέρον να επικεντρώνεται στην επισήμανση των μετασχηματισμών και των εργασιών καθαρισμού που ενδέχεται να περικλείει αυτή η αντιστοίχηση.

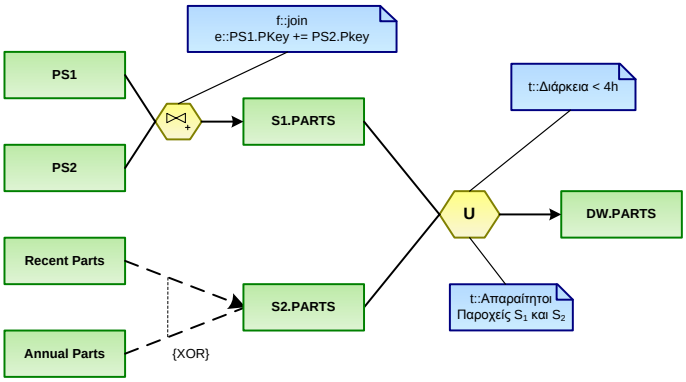
Για κάθε γνώρισμα στόχο πρέπει να οριστεί ένα σύνολο από σχέσεις παροχής. Σε απλές περιπτώσεις, η σχέση παροχής ορίζεται κατευθείαν μεταξύ αρχικού και τελικού γνωρίσματος. Στις περιπτώσεις όπου απαιτείται η χρήση μετασχηματισμού, ο μετασχηματισμός αυτός παρεμβάλλεται μεταξύ του αρχικού και του τελικού γνωρίσματος. Στις περιπτώσεις που απαιτούνται περισσότεροι από ένας μετασχηματισμοί, μία ακολουθία μετασχηματισμών, με τη μορφή σειριακής σύνθεσης παρεμβάλλεται μεταξύ των εμπλεκόμενων γνωρισμάτων. Επίσης, σε αυτό το βήμα σχεδιασμού καθορίζονται και οι περιορισμοί ΕΜΦ. Ένα παράδειγμα των σχέσεων μεταξύ των γνωρισμάτων των εννοιών **PS1**, **PS2**, **S1.PARTS**, **S2.PARTS** και **DW.PARTS** απεικονίζεται στο Σχήμα 2.7.

Βήμα 4ο: Εμπλουτισμός του διαγράμματος με περιορισμούς εκτέλεσης των διεργασιών. Εκτός από τον καθορισμό των εργασιών σε ένα σενάριο ΕΜΦ που καθορίζει την απεικόνιση των πηγών στην Αποθήκη Δεδομένων μαζί με τους κατάλληλους μετασχηματισμούς, υπάρχουν κι άλλες παράμετροι που πιθανόν να χρειάζεται να διευκρινιστούν για το περιβάλλον εκτέλεσης. Αυτού του είδους οι *περιορισμοί εκτέλεσης* περιλαμβάνουν:

- *Προγραμματισμός με βάση το χρόνο ή κάποιο γεγονός.* Ο σχεδιαστής χρειάζεται να αποφασίσει τη συχνότητα με την οποία θα εκτελείτε η διεργασία ΕΜΦ, έτσι ώστε τα δεδομένα να είναι πάντα ανανεωμένα και όλη η διεργασία να εκτελείται στο διαθέσιμο χρόνο ανανέωσης.
- *Επίβλεψη.* Είναι απαραίτητη η συνεχής ροή πληροφοριών για την πρόοδο και την κατάσταση της διεργασίας, έτσι ώστε ο διαχειριστής να είναι ενήμερος για το στάδιο που βρίσκεται το φόρτωμα, το χρόνο έναρξής του, τη διάρκεια, κ.λ.π.. Για το σκοπό αυτό, χρησιμοποιούνται μεταξύ άλλων: εγγραφές σε αρχεία, μηνύματα ενημέρωσης στην οθόνη ή με ηλεκτρονικό ταχυδρομείο, εκτυπώσεις ή γραφικές παραστάσεις, κ.α..
- *Καταγραφή Ιστορικού.* Καταγραφή πληροφοριών, οι οποίες παρουσιάζονται στο τέλος της εκτέλεσης του σεναρίου. Όλες οι σχετικές πληροφορίες (όπως τα προηγούμενα στοιχεία ανάλυσης) παρουσιάζονται χρησιμοποιώντας τις τεχνικές που προαναφέρθηκαν.
- *Χειρισμός Εξαιρέσεων.* Η αντιμετώπιση της παραβίασης των κανόνων της βάσης δεδομένων ή των επιχειρησιακών κανόνων κατά εγγραφή είναι αναγκαίες για τη σωστή εκτέλεση των διεργασιών ΕΜΦ. Χρήσιμες για το σκοπό αυτό θεωρούνται πληροφορίες, όπως το ποιες εγγραφές είναι προβληματικές ή πόσες εγγραφές είναι αποδεκτές (ποσοστό επιτυχίας).
- *Χειρισμός Σφαλμάτων.* Ανάληψη από κατάρρευση και ικανότητες έναρξης και τερματισμού (αποθήκευση των συναλλαγών κάθε μερικές εγγραφές) είναι απολύτως απαραίτητες για τη στιβαρότητα και την αποδοτικότητα της διεργασίας.



Σχήμα 2.7: Αντιστοιχίσεις γνωρισμάτων για τη φόρτωση της έννοιας DW.PARTSUPP



Σχήμα 2.8: Εμπλουτισμός του Σχήματος 2.5 με περιορισμούς εκτέλεσης

Για την καταγραφή όλων των πληροφοριών, υιοθετούνται σημειώσεις τοποθετημένες στις κατάλληλες έννοιες, μετασχηματισμούς ή σχέσεις. Στο Σχήμα 2.8 απεικονίζουμε το διάγραμμα εκτέλεσης του Σχήματος 2.5, σημειώνοντας ένα περιορισμό χρόνου εκτέλεσης δηλώνοντας ότι ο χρόνος εκτέλεσης για το φόρτωμα της **DW.PARTS** (που περιλαμβάνει το φόρτωμα των **S1.PARTS** και **S2.PARTS**) δεν μπορεί να υπερβεί τις 4 ώρες.

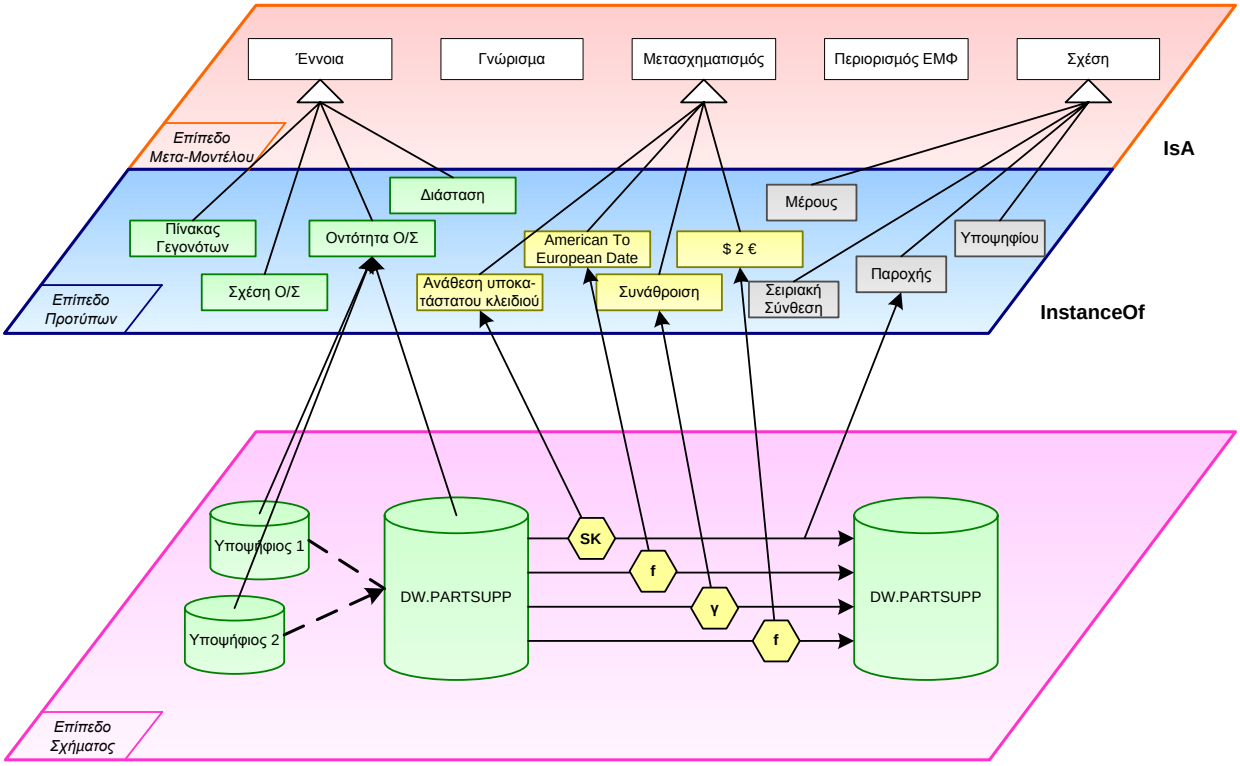
2.3 Μετα–Μοντέλο

Στο [47] υποστηρίζεται ότι το σημείο κλειδί στην εννοιολογική απεικόνιση διεργασιών ΕΜΦ στηρίζεται (α) στην αναγνώριση ενός μικρού συνόλου από από *γενικούς κατασκευαστές*, ικανών να καλύψουν όλες τις περιπτώσεις (γενικότητα), και (β) στον προσδιορισμό ενός μηχανισμού που θα επιτρέπει τη δημιουργία μίας "παλέτας" από *συχνά χρησιμοποιούμενους τύπους*, όπως μετασχηματισμούς, χώρους αποθήκευσης δεδομένων, κ.λ.π (επεκτασιμότητα).

Στο Σχήμα 2.9 απεικονίζεται το πλαίσιο μετα-μοντελοποίησης που εισάγεται με το [47]. Το κατώτατο επίπεδο του Σχήματος 2.9, που ονομάζεται *Επίπεδο Σχήματος*, περιλαμβάνει ένα σενάριο ΕΜΦ. Όλες οι οντότητες που υπάρχουν στο Επίπεδο Σχήματος αποτελούν στιγμιότυπα των κλάσεων *Έννοια*, *Γνώρισμα*, *Μετασχηματισμός*, *Περιορισμός ΕΜΦ* και *Σχέση*. Κατά συνέπεια, όπως φαίνεται στο πάνω μέρος του Σχήματος 2.9, εισάγεται ένα επίπεδο μετα-κλάσης, που ονομάζεται *Επίπεδο Μετα-Μοντέλου*, το οποίο περιλαμβάνει τις προαναφερθείσες κλάσεις. Η σύνδεση μεταξύ των δύο επιπέδων επιτυγχάνεται μέσω *Σχέσεων Στιγμιοτύπων* (InstanceOf). Με την εισαγωγή του Επιπέδου Μετα-Μοντέλου, ικανοποιείται η συνθήκη γενικότητας: οι πέντε κλάσεις που περιλαμβάνονται στο Επίπεδο Μετα-Μοντέλου είναι αρκετά γενικές για να μοντελοποιήσουμε οποιοδήποτε σενάριο ΕΜΦ, μέσω της δημιουργίας κατάλληλων στιγμιοτύπων.

Ωστόσο, το μετα-μοντέλο μπορεί να βελτιωθεί για να καλύψει και τη συνθήκη επεκτασιμότητας. Προκειμένου να γίνει το μοντέλο πραγματικά χρήσιμο για πρακτικές περιπτώσεις διεργασιών ΕΜΦ, εμπλουτίζεται με ένα σύνολο εξειδικευμένων κατασκευαστών ΕΜΦ και εισάγεται ένα τρίτο επίπεδο: το *Επίπεδο Προτύπων*. Οι κατασκευαστές στο Επίπεδο Προτύπων είναι επίσης μετα-κλάσεις, αρκετά πα-

ραμετροποιήσιμες και αποτελούν υποσύνολο των μετα-κλάσεων του Επιπέδου Μετα-Μοντέλου. Κατά συνέπεια, οι κλάσεις του Επιπέδου Προτύπων αποτελούν εξειδικεύσεις (υποκλάσεις) των γενικών κλάσεων του επιπέδου Μετα-Μοντέλου, κάτι που φαίνεται σαν σχέση IsA στο Σχήμα 2.9. Μέσω αυτού του μηχανισμού προσαρμογής, ο σχεδιαστής επιλέγει τα στιγμιότυπα του Επιπέδου Σχήματος από μία πλουσιότερη "παλέτα" κατασκευαστών. Με αυτή τη ρύθμιση, οι οντότητες στο Επίπεδο Σχήματος είναι στιγμιότυπα, όχι μόνο των αντιστοίχων κλάσεων του Επιπέδου Μετα-Μοντέλου, αλλά και των υποκλάσεων του Επιπέδου Προτύπων.



Σχήμα 2.9: Το πλαίσιο μοντελοποίησης διεργασιών ΕΜΦ

Στο παράδειγμα του Σχήματος 2.9, η έννοια **DW.PARTS** πρέπει να φορτωθεί από μία συγκεκριμένη πηγή **S2**. Αρκετοί μετασχηματισμοί λαμβάνουν χώρα κατά τη διάρκεια αυτής της μετάδοσης δεδομένων. Για παράδειγμα, σε αυτό το σενάριο ΕΜΦ χρησιμοποιούνται εκτός των άλλων, μία ανάθεση υποκατάστατου κλειδιού και μία συνάθροιση. Από αυτούς τους δύο, μόνο ένας (**Υποψήφιος 2**) επιλέγεται τελικά για τη διαδικασία αυτή. Όπως κάποιος μπορεί να παρατηρήσει, οι έννοιες που λαμβάνουν μέρος σε αυτό το σενάριο, είναι στιγμιότυπα της κλάσης **΄Εννοια** (που ανήκει στο επίπεδο Μετα-Μοντέλου) και συγκεκριμένα της υποκλάσης **Οντότητα Ο/Σ** (υποθέτοντας ότι υιοθετείται μία επέκταση του μοντέλου Ο/Σ). Στιγμιότυπα και επικαλυπτόμενες κλάσεις σχετίζονται μέσω συνδέσμων τύπου **InstanceOf**. Ο ίδιος μηχανισμός εφαρμόζεται σε όλους τους μετασχηματισμούς του σεναρίου αυτού, οι οποίοι είναι (α) στιγμιότυπα της κλάσης **Μετασχηματισμός** και (β) στιγμιότυπα κάποιας από τις υποκλάσεις που απεικονίζονται στο Επίπεδο Προτύπων του Σχήματος 2.9. Ούτε οι σχέσεις ξεφεύγουν από αυτό τον κανόνα. Για παράδειγμα, παρατηρείστε πως οι συνδέσεις παροχής από την έννοια **S2.PARTS** προς την έννοια **DW.PARTS** σχετίζονται με την κλάση **Σχέση Παροχής** μέσω των κατάλληλων συνδέσεων **InstanceOf**.

Προφανώς για λόγους απλότητας στο Σχήμα 2.9 δεν απεικονίζονται όλες οι δυνατές συσχετίσεις μεταξύ των τριών επιπέδων μοντελοποίησης.

Όσον αφορά την κλάση **΄Εννοια** στο Επίπεδο Προτύπων, μπορούμε να την εξειδικεύσουμε σε αρκετές υποκλάσεις, ανάλογα με το χρησιμοποιούμενο μοντέλο. Στην περίπτωση του μοντέλου Ο/Σ, έχουμε τις υποκλάσεις **Οντότητα Ο/Σ** και **Σχέση Ο/Σ**, ενώ στην περίπτωση του μοντέλου διαστάσεων, έχουμε υποκλάσεις όπως **Πίνακας Γεγονότων** και **Πίνακας Διαστάσεων**.

Ακολουθώντας το ίδιο πλαίσιο, η κλάση **Μετασχηματισμός** εξειδικεύεται περισσότερο σε ένα σύνολο από επαναχρησιμοποιούμενες διεργασίες ΕΜΦ, που απεικονίζεται στο Σχήμα 2.10. Στο Σχήμα 2.6 τα πρότυπα διεργασιών ομαδοποιούνται σε έξι κύριες, λογικές κατηγορίες. Στο Σχήμα 2.5 δεν απεικονίζεται αυτή η ομαδοποίηση για την αποφυγή της υπερφόρτωσης του σχήματος. Αντίθετα, στο εν λόγω σχήμα απεικονίζονται μόνο τέσσερις υποκλάσεις, των οποίων τα στιγμιότυπα παρουσιάζονται στο σενάριο του Επιπέδου Σχήματος.

Φίλτρα	**Μοναδιαίοι Μετασχηματισμοί**	**Δυαδικοί Μετασχηματισμοί**
Selection (σ)	Push	Union (U)
Not Null (NN)	Aggregation (γ)	Join ($\bowtie$)
Primary Key Violation (PK)	Projection (π)	Diff (Δ)
Foreign Key Violation (FK)	Function Application (f)	Update Detection (Δ_{UPD})
Unique Value (UN)	Surrogate Key Assignment (SK)	
Domain Mismatch (DM)	Tuple Normalization (N)	**Σύνθετοι Μετασχηματισμοί**
	Tuple Denormalization (DN)	Slowly Changing Dimension
Λειτουργίες Μεταφοράς		(Type 1,2,3) (SDC-1/2/3)
FTP (FTP)	**Λειτουργίες Επεξεργασίας Αρχείων**	Format Mismatch (FM)
Compress/Decompress (Z/dZ)	EBCDIC to ASCII Conversion (EB2AS)	Data Type Conversion (DTC)
Encrypt/Decrypt (Cr/dCr)	Sort File (Sort)	Switch (σ^*)
		Extended Union (U)

Σχήμα 2.10: Πρότυπα μετασχηματισμών και οι συμβολισμοί τους ανά κατηγορίες

Η πρώτη ομάδα ονομάζεται *Φίλτρα* και περιλαμβάνει ελέγχους ως προς κάποια συγκεκριμένη συνθήκη. Οι σημασίες αυτών των φίλτρων είναι προφανείς: ξεκινώντας από ένα γενικό φίλτρο *επιλογής* (σ) και συνεχίζοντας με ελέγχους *μη-κενής τιμής* (NN), παραβίασης *πρωτεύοντος* (PK) ή *ξένου κλειδιού* (FK) κ.λ.π.. Η δεύτερη ομάδα πρότυπων μετασχηματισμών, ονομάζεται *Μοναδιαίοι Μετασχηματισμοί* και εκτός από τη γενική δραστηριότητα *προώθησης δεδομένων* (push) (η οποία απλά μεταφέρει δεδομένα από τον προμηθευτή στον καταναλωτή), περιλαμβάνει επίσης τις: *προβολή* (π), *συνάθροιση* (γ) και *συνάρτηση* (f) καθώς και τρεις εξειδικευμένους μετασχηματισμούς για Αποθήκες Δεδομένων: *ανάθεση υποκατάστατου κλειδιού* (SK), *κανονικοποίηση* (N) και *αποκανονικοποίηση* (DN). Η τρίτη ομάδα αποτελεί τους *Δυαδικούς Μετασχηματισμούς* και περιλαμβάνει πρότυπα μετασχηματισμών όπως *ένωση* (U), *συνένωση* ($\bowtie$) και *διαφορά* (Δ), όπως και μία ειδική περίπτωση για τη διαφορά που περιλαμβάνει την *αναγνώριση των ενημερώσεων* (Δ_{UPD}). Επιπλέον, υπάρχει ένα σύνολο από *Σύνθετους Μετασχηματισμούς* που περιλαμβάνει πρότυπα μετασχηματιστών κατασκευασμένων από τη σύνθεση άλλων απλούστερων. Για παράδειγμα, αναφέρουμε μετασχηματισμούς όπως: ο *χειρισμός αργά μεταβαλλόμενων διαστάσεων* (SCD) που είναι μία τεχνική για τη φόρτωση διαστάσεων με καινούρια ή αλλαγμένα δεδο-

μένα, η *ασυμβατότητα μορφής* (FM) που είναι μία ειδική περίπτωση συνάρτησης, η *εναλλαγή* ($\sigma^\star$) που είναι ο συνδυασμός αρκετών επιλογών, και η *εκτεταμένη ένωση* (U) που είναι ένας συνδυασμός από δυαδικές ενώσεις για την παραγωγή του αντίστοιχου ν-αδικού τελεστή. Εκτός από τα προαναφερθέντα πρότυπα μετασχηματισμών, που αναφέρονται κυρίως σε λογικούς μετασχηματισμούς, μπορούμε να αναλογιστούμε την περίπτωση προτύπων που αναφέρονται στην εφαρμογή φυσικών μετασχηματισμών σε ολόκληρα αρχεία ή πίνακες. Κυρίως εννοούμε φυσικές πράξεις μεταξύ εννοιών, όπως *Λειτουργίες Μεταφοράς* (FTP, Z/dZ, Cr/dCr) και *Λειτουργίες Επεξεργασίας Αρχείων* (EB2AS, Sort).

Συνοψίζοντας, το Επίπεδο Μετα-Μοντέλου είναι ένα σύνολο από γενικές οντότητες, ικανό να αναπαραστήσει οποιοδήποτε σενάριο ΕΜΦ. Παράλληλα, η γενικότητα του Επιπέδου Μετα-Μοντέλου συμπληρώνεται με την επεκτασιμότητα του Επιπέδου Προτύπων, που είναι ένα σύνολο από προκατασκευασμένα εξειδικευμένα πρότυπα μετασχηματισμών, ειδικά σχεδιασμένα για να ανταποκρίνονται στους πιο συχνά χρησιμοποιούμενους μετασχηματισμούς των διεργασιών ΕΜΦ.

Πρέπει να τονιστεί, όμως, ότι η "παλέτα" μετασχηματισμών του Σχήματος 2.10 σε καμία περίπτωση δεν περιλαμβάνει το σύνολο τον μετασχηματισμών που ενδέχεται να απαιτηθούν σε ένα σενάριο ΕΜΦ. Η ιδέα γύρω από την έννοια της επεκτασιμότητας και του τρόπου σύλληψης του μοντέλου είναι ότι, αν ο σχεδιαστής το επιθυμεί, μπορεί να σχεδιάσει τα δικά του πρότυπα, που δε συμπεριλαμβάνονται στο Σχήμα 2.10, δίχως να επηρεάζεται η λειτουργικότητα ή ο τρόπος χρήσης του μοντέλου.

2.4 Συμπεράσματα

Τα εργαλεία Εξαγωγής–Μετασχηματισμού–Φόρτωσης (ΕΜΦ) είναι προγράμματα υπεύθυνα για την εξαγωγή των δεδομένων από τις διάφορες πηγές, τον καθαρισμό, την προσαρμογή και την εισαγωγή τους σε μία Αποθήκη Δεδομένων. Το [47] εστιάζεται στο πρόβλημα του ορισμού των διεργασιών ΕΜΦ και παρουσιάζει ένα νέο εννοιολογικό μοντέλο προσανατολισμένο στην ταυτοποίηση των σχέσεων μεταξύ των γνωρισμάτων και των εννοιών, αλλά και στην αναγνώριση των κατάλληλων διεργασιών ΕΜΦ στα πρώτα στάδια ενός έργου σχεδίασης και ανάπτυξης μίας Αποθήκης Δεδομένων. Το μοντέλο αυτό κατασκευάστηκε με τρόπο προσαρμόσιμο και επεκτάσιμο, ώστε να μπορεί ο σχεδιαστής να το ενισχύσει με δικές του, επαναχρησιμοποιούμενες διεργασίες ΕΜΦ, όπως η ανάθεση υποκατάστατων κλειδιών, ο έλεγχος για παραβιάσεις αναφορικής ακεραιότητας, κ.λ.π..

3

Εισαγωγή στο Λογικό Μοντέλο

Στο κεφάλαιο αυτό επικεντρωνόμαστε στο λογικό σχεδιασμό ενός σεναρίου ΕΜΦ μίας Αποθήκης Δεδομένων και πιο συγκεκριμένα στον ορισμό ενός τυπικού λογικού μοντέλου διεργασιών ΕΜΦ. Γίνεται τυπική εισαγωγή στους Χώρους Αποθήκευσης Δεδομένων, στις Διεργασίες και στα συστατικά τους μέρη. Ένα σενάριο ΕΜΦ αποτελείται από ένα συνδυασμό Πηγών Δεδομένων και Διεργασιών. Στη συνέχεια δείχνουμε πως αυτό το μοντέλο αποσυντίθεται σε ένα γράφο, τον οποίο ονομάζουμε Γράφο Αρχιτεκτονικής. Μοντελοποιούνται όλες οι προαναφερθείσες οντότητες σαν κόμβοι και τέσσερα διαφορετικά είδη σχέσεων (στιγμιοτύπου, μέρους, ρύθμισης και παροχής) σαν ακμές.

3.1 Εισαγωγή

Το ιδεατό μοντέλο των διεργασιών ΕΜΦ, καθώς και μία μεθοδολογία για την παραγωγή του, περιγράφεται στο [48]. Στο [46], η προσοχή επικεντρώνεται στο λογικό σχεδιασμό του σεναρίου ΕΜΦ μίας Αποθήκης Δεδομένων με τα ακόλουθα χαρακτηριστικά:

- *Ορισμός ενός τυπικού λογικού μοντέλου ως μία λογική αφαίρεση των διεργασιών ΕΜΦ.* Οι Χώροι Αποθήκευσης Δεδομένων, οι Διεργασίες και τα συστατικά τους μέρη ορίζονται αυστηρά. Ως Διεργασία ορίζεται μία οντότητα με ένα (ή περισσότερα) σχήμα(τα) εισόδου, ένα σχήμα εξόδου, ένα σχήμα απόρριψης για τις εγγραφές που δεν ικανοποιούν τα κριτήριά της και ένα σχήμα παραμέτρων έτσι ώστε η διεργασία να δέχεται κάθε φορά τις κατάλληλες τιμές παραμέτρων. Η ροή των δεδομένων από τους παραγωγούς προς τους καταναλωτές επιτυγχάνεται με τη χρήση

των Σχέσεων Παροχής, οι οποίες απεικονίζουν τα γνωρίσματα (πεδία) του πρώτου σχήματος στα αντίστοιχα του δεύτερου. Ένας σειριοποιήσιμος συνδυασμός από Διεργασίες ΕΜΦ, Σχέσεις Παροχής και Χώρους Αποθήκευσης Δεδομένων συνιστά ένα *Σενάριο ΕΜΦ*.

- *Δείχνεται πως αυτό το μοντέλο μπορεί να αναχθεί σε ένα γράφο*, ο οποίος αποκαλείται *Γράφος Αρχιτεκτονικής*. Μοντελοποιούνται όλες οι προαναφερθείσες οντότητες ως κόμβοι και τέσσερα διαφορετικά είδη σχέσεων ως ακμές. Αυτές οι σχέσεις αντιστοιχούν σε (α) πληροφορίες ελέγχου τύπου (*Σχέσεις Στιγμιότυπου*), (β) *Σχέσεις Μέρους*, π.χ. σε ποια Διεργασία ανήκει κάποιο γνώρισμα, (γ) *Σχέσεις Ρύθμισης*, οι οποίες καλύπτουν το θέμα της παροχής δεδομένων στις παραμέτρους των διεργασιών από γνωρίσματα ή σταθερές, και (δ) *Σχέσεις Παροχής* που προσδιορίζουν τη ροή των δεδομένων από τα γνωρίσματα πηγές στα γνωρίσματα καταναλωτές.

- Τέλος, μετριέται η σημασία καθώς και η ευπάθεια των κόμβων μέσω κάποιων μετρικών και ποιο συγκεκριμένα αυτών της *εξάρτησης* και της *υπευθυνότητας*. Η εξάρτηση δείχνει το βαθμό κατά τον οποίο συνδέεται με άλλες οντότητες που του παρέχουν δεδομένα και η υπευθυνότητα το βαθμό κατά τον οποίο άλλες οντότητες του γράφου εξαρτώνται από τον υπό θεώρηση κόμβο. Η εξάρτηση και η υπευθυνότητα είναι κρίσιμες μετρήσεις για την κατασκευή και την εξέλιξη του περιβάλλοντος ΕΜΦ.

3.2 Σύντομη Αναφορά στο Λογικό Μοντέλο

Το μοντέλο αυτό ξεφεύγει από τις τεχνικές της επίβλεψης, του προγραμματισμού και των αρχείων καταγραφής συμβάντων, καθώς επικεντρώνεται κυρίως στη ροή των δεδομένων από τις πηγές προς την Αποθήκη Δεδομένων μέσω ενός συνδυασμού από Διεργασίες και χώρους αποθήκευσης δεδομένων.

3.2.1 Βασικές Αρχές

Σε αυτό το τμήμα, γίνεται εισαγωγή της τυπικής μοντελοποίησης των Τύπων Δεδομένων, των χώρων αποθήκευσης δεδομένων και των συναρτήσεων, πριν προχωρήσουμε στην περιγραφή της μοντελοποίησης των διεργασιών ΕΜΦ, όπως αυτά αναφέρονται στο [48].

Βασικές Οντότητες. Υποθέτουμε την ύπαρξη ενός αριθμήσιμου συνόλου από *Τύπους Δεδομένων*. Κάθε τύπος $\mathbf{T}$ χαρακτηρίζεται από ένα όνομα και ένα τομέα, δηλαδή ένα αριθμήσιμο σύνολο τιμών. Οι τιμές των τομέων αναφέρονται ως *Σταθερές*. Επίσης υποθέτουμε την ύπαρξη ενός πεπερασμένου συνόλου από *Γνωρίσματα*. Τα γνωρίσματα χαρακτηρίζονται από το όνομά τους και τον τύπο δεδομένων τους και αποτελούν το πιο θεμελιώδες στοιχείο της δομής του όλου συστήματος. Ο τομέας ενός γνωρίσματος είναι ένα υποσύνολο του τομέα του τύπου του. Τα γνωρίσματα και οι σταθερές αναφέρονται συνολικά ως *όροι*.

Σχήμα. Είναι μία πεπερασμένη *λίστα* από γνωρίσματα. Κάθε οντότητα η οποία χαρακτηρίζεται από ένα ή περισσότερα σχήματα, αποκαλείται *Δομημένη Οντότητα*. Επιπλέον, υιοθετούμε την ύπαρξη μίας ειδικής "οικογένειας" από σχήματα, υπό το γενικό όνομα NULL σχήμα, που χρησιμοποιείται ως χώρος κράτησης των δεδομένων που δεν αποθηκεύονται μόνιμα σε κάποιο χώρο αποθήκευσης δεδομένων. Η αναφορά σε μία οικογένεια και όχι σε ένα μόνο NULL σχήμα οφείλεται σε μία λεπτομέρια που έχει να κάνει με τον αριθμό των γνωρισμάτων σε ένα τέτοιο σχήμα.

Σύνολα Εγγραφών. Ορίζεται ως εγγραφή το στιγμιότυπο ενός σχήματος σε μία λίστα από τιμές οι οποίες ανήκουν στους τομείς των αντίστοιχων γνωρισμάτων του σχήματος. Όπως αναφέρεται στο [45] μπορούμε να αντιμετωπίσουμε οποιαδήποτε δομή δεδομένων σαν ένα "σύνολο εγγραφών" με την προϋπόθεση ότι είναι δυνατό να την αναδομήσουμε λογικά σε ένα ένα επίπεδο τυπικό σχήμα εγγραφών. Αρκετές φυσικές δομές αποθήκευσης εντάσσονται στον κανόνα αυτό, όπως οι σχεσιακές βάσεις δεδομένων, αρχεία COBOL ή ASCII, πολυδιάστατοι κύβοι κ.λ.π..

Οι δύο ποιο σημαντικοί τύπο Συνόλων Εγγραφών είναι οι σχεσιακοί πίνακες και τα αρχεία εγγραφών, και σε αυτούς θα επικεντρώσουμε την προσοχή μας. Μία *Βάση Δεδομένων* ορίζεται σαν ένα πεπερασμένο σύνολο από σχεσιακούς πίνακες.

Συναρτήσεις. Υποθέτουμε την ύπαρξη ενός αριθμήσιμου συνόλου από ενσωματωμένους στο σύστημα τύπους συναρτήσεων. Ένας τύπος συνάρτησης περιλαμβάνει ένα όνομα, μία πεπερασμένη λίστα από τύπους δεδομένων των παραμέτρων του και ένα μόνο τύπο αποτελέσματος. Μία *συνάρτηση* είναι ένα στιγμιότυπο ενός τύπου συνάρτησης. Συνεπώς χαρακτηρίζεται κι αυτή από ένα όνομα, μία λίστα από παραμέτρους εισόδου και μία παράμετρο για την τιμή που επιστρέφεται. Οι τύποι δεδομένων του γεννήτορα τύπου συνάρτησης ορίζουν: (α) τους τύπους των παραμέτρων της συνάρτησης καθώς και (β) τους δεκτούς υποψήφιους των τελευταίων (δηλαδή γνωρίσματα ή σταθερές κατάλληλου τύπου δεδομένων).

3.2.2 Διεργασίες και Σχέσεις

Όπως αναφέρεται στο [48] οι διεργασίες είναι η ραχοκοκαλιά της δομής κάθε πληροφοριακού συστήματος. Υιοθετώντας την ορολογία WfMC για τις εργασίες θα τις αποκαλούμε στη συνέχεια *Διεργασίες*. Μία Διεργασία είναι το σύνολο της "εργασίας που διεκπεραιώνεται μέσω ενός από πόρους και εφαρμογές υπολογιστών" [12]. Στο πλαίσιο στο οποίο κινηθήκαμε, οι Διεργασίες είναι λογικές αφαιρέσεις που αναπαριστούν μέρη ή και ολόκληρες ενότητες κώδικα.

Η εκτέλεση μίας Διεργασίας πραγματοποιείται από ένα συγκεκριμένο πρόγραμμα. Συνήθως οι Διεργασίες ΕΜΦ εκτελούνται με την τεχνική του μαύρου κουτιού από ένα ειδικό εργαλείο ή εκφράζονται σε μία scripting γλώσσα (όπως π.χ. PL/SQL, Perl, C). Εντούτοις, στο δικό μας πλαίσιο ασχολούμαστε με τη γενική περίπτωση των Διεργασιών ΕΜΦ· συνεπώς τοποθετείται μία αφαίρεση του πηγαίου

κώδικα με τη μορφή μίας SQL πρότασης, με σκοπό να αποφύγουμε την ενασχόληση με τις ιδιαιτερότητες κάθε προγραμματιστικής γλώσσας. Πρέπει ωστόσο να γίνει σαφές ότι αυτή η SQL πρόταση δεν απεικονίζει τον τρόπο με τον οποίο υλοποιούνται οι Διεργασίες, αλλά τη σημασιολογία της κάθε μίας.

Μία *Στοιχειώδης Διεργασία* τυπικά περιγράφεται από τα παρακάτω στοιχεία:

- *Όνομα.* Ένα μοναδικό αναγνωριστικό για κάθε Διεργασία.
- *Σχήματα Εισόδου.* Ένα πεπερασμένο σύνολο από ένα ή περισσότερα σχήματα εισόδου, τα οποία δέχονται δεδομένα από τις πηγές δεδομένων της Διεργασίας.
- *Σχήμα Εξόδου.* Το σχήμα περιγράφει τον προσωρινό χώρο αποθήκευσης για τις σειρές που ικανοποιούν τον πραγματοποιούμενο από τη Διεργασία έλεγχο.
- *Σχήμα Απόρριψης.* Το σχήμα που περιγράφει τον προσωρινό χώρο αποθήκευσης για τις σειρές που δεν ικανοποιούν τον πραγματοποιούμενο από τη Διεργασία έλεγχο ή οι τιμές τους δεν είναι κατάλληλες για τον εκτελούμενο μετασχηματισμό.
- *Λίστα Παραμέτρων.* Ένα σύνολο από ζεύγη που λειτουργούν ως ρυθμιστές για τη λειτουργικότητα της Διεργασίας (π.χ. το γνώρισμα-στόχος ενός ελέγχου για εξωτερικό κλειδί). Το πρώτο συστατικό του ζεύγους είναι ένα όνομα και το δεύτερο ένα σχήμα, ένα γνώρισμα, μία συνάρτηση ή μία σταθερά.
- *Λειτουργική Σημασιολογία Εξόδου.* Μία SQL πρόταση που περιγράφει το περιεχόμενο που περνιέται στην έξοδο της λειτουργίας, αναφορικά με είσοδο αυτής. Αυτή η πρόταση SQL ορίζει (α) τη λειτουργία που επιτελείται στις σειρές που περνούν από τη Διεργασία και (β) μία αυστηρή απεικόνιση μεταξύ των γνωρισμάτων εισόδου και των αντίστοιχων γνωρισμάτων του σχήματος εξόδου.
- *Λειτουργική Σημασιολογία Απόρριψης.* Μία SQL πρόταση που περιγράφει τις απορριφθείσες εγγραφές, κατά κάποια έννοια παρόμοια με τη Λειτουργική Σημασιολογία Εξόδου. Αυτή η πρόταση θεωρείται εξ ορισμού να είναι η άρνηση της Λειτουργικής Σημασιολογίας Εξόδου, εκτός και αν σαφώς οριστεί διαφορετικά.
- *Σημασιολογία Πηγής/Καταναλωτή Δεδομένων.* Ορίζει (α) αν τα δεδομένα απλώς διαβάζονται (πρότυπη συμπεριφορά) ή διαβάζονται και στη συνέχεια διαγράφονται από την πηγή δεδομένων και (β) αν τα δεδομένα προστίθενται στον καταναλωτή (πρότυπη συμπεριφορά) ή αν τα περιεχόμενα του καταναλωτή επικαλύπτονται από τις εξόδους της λειτουργίας. Συνεπώς, ο τομέας της Σημασιολογίας Πηγής Δεδομένων είναι **{SELECT, DELETE}** και ο τομέας της Σημασιολογίας Καταναλωτή Δεδομένων είναι **{APPEND, OVERWRITE}**.

Πρέπει να σημειωθεί ότι:

- Η Σημασιολογία Πηγής/Καταναλωτή Δεδομένων είναι μία διευκόλυνση που χρησιμοποιείται με σκοπό την των περιπτώσεων Διεργασιών που διαγράφουν ή ενημερώνουν πίνακες στην Αποθήκη Δεδομένων. Στη συνέχεια θα αναφερόμαστε μόνο σε Διεργασίες που απλά διαβάζουν δεδομένα από τις πηγές και προσθέτουν στους στόχους τους. Παρόλα αυτά, θεωρούμε ότι ακόμα και αυτή η περίπτωση δεν αλλοιώνει ιδιαίτερα τα όσα αναφέρονται παρακάτω.

- Όποτε δεν αναφέρεται σαφώς ένας καταναλωτής δεδομένων για το σχήμα εξόδου ή απόρριψης, συνεπάγεται η ύπαρξη του αντίστοιχου NULL σχήματος (περιλαμβάνοντας το σωστό αριθμό γνωρισμάτων). Αυτό στην ουσία σημαίνει ότι τα δεδομένα που παράγονται ούτε προωθούνται σε μία άλλη Διεργασία, ούτε αποθηκεύονται κάπου, δηλαδή απλώς αγνοούνται.

Στο σημείο αυτό, πριν προχωρήσουμε, θα πρέπει να αναφερθούμε στο [40] για να καταλάβουμε πλήρως τις έννοιες που παρουσιάζονται σε αυτή την υποενότητα. Μέχρι στιγμής, ισχυριζόμαστε ότι μία διεργασία έχει σχήματα εισόδου, εξόδου και απόρριψης. Στην πραγματικότητα, όπως θα δούμε στη συνέχεια, ο ορισμός μίας διεργασίας επεκτείνεται και περιέχει άλλα τρία σχήματα: σχήματα λειτουργικότητας (ή παραμέτρων), παραγομένων και αποβλημένων. Τα κίνητρα για την επέκταση αυτή μαζί με την επίσημη παρουσίαση των σχημάτων αυτών δίνονται στο [40]. Προς το παρόν, είναι αρκετό να αναφέρουμε ότι το σχήμα *λειτουργικότητας* αποτελείται από τα γνωρίσματα που λαμβάνουν μέρος στον υπολογισμό που εκτελείται από τη διεργασία (στην πραγματικότητα, αποτελούν τις παραμέτρους της διεργασίας), ένα σχήμα *παραγομένων* περιλαμβάνει όλα τα γνωρίσματα εξόδου τα οποία παράγονται από τις διεργασίες και ένα σχήμα *απόρριψης* αποτελείται από μία λίστα γνωρισμάτων, τα οποία ενώ ανήκουν στο σχήμα εισόδου, δεν διαδίδονται αλλά ούτε και χρησιμοποιούνται (για την παραγωγή κάποιου άλλου γνωρίσματος εξόδου) από τη διεργασία.

Η ροή των δεδομένων από τις πηγές προς την Αποθήκη Δεδομένων επιτυγχάνεται με το συνδυασμό των Διεργασιών σε ένα μεγαλύτερο σενάριο. Η είσοδος μίας Διεργασίας μπορεί να είναι είτε ένας μόνιμος Χώρος Αποθήκευσης Δεδομένων είτε η έξοδος μίας άλλης Διεργασίας, δηλαδή κάθε δομημένη οντότητα υπό κάποιο συγκεκριμένο σχήμα. Το ίδιο ισχύει και για την έξοδο της Διεργασίας. Η απεικόνιση του περάσματος των δεδομένων από της πηγές στους καταναλωτές γίνεται με μία *σχέση Παροχής* ανάμεσα στα γνωρίσματα των εμπλεκομένων σχημάτων.

Τυπικά, μία *Σχέση Παροχής* ορίζεται από τα εξής:

- *Όνομα.* Ένα μοναδικό αναγνωριστικό για τη σχέση.

- *Αντιστοίχηση.* Ένα διατεταγμένο ζεύγος. Το πρώτο μέρος του ζεύγους είναι ένα γνώρισμα το οποίο λειτουργεί ως πηγή και το δεύτερο ένα γνώρισμα που λειτουργεί ως καταναλωτής.

Η αντιστοίχηση δεν είναι αναγκαίο να είναι 1:1, καθώς ένα γνώρισμα εισόδου μπορεί να παρέχει δεδομένα για περισσότερα γνωρίσματα. Παρόλα αυτά, το αντίστροφο δεν ισχύει. Επίσης πρέπει να σημειωθεί ότι ένα γνώρισμα μπορεί να δεχτεί δεδομένα από μια σταθερά σε ορισμένες περιπτώσεις.

Προκειμένου να επιτευχθεί η ροή των δεδομένων από τις πηγές μίας Διεργασίας προς τους καταναλωτές τους, είναι απαραίτητες τρεις ομάδες σχέσεων Παροχής:

1. Μία αντιστοίχηση ανάμεσα στο σχήμα εξόδου των πηγών και το σχήμα εισόδου της Διεργασίας. Αυτό σημαίνει ότι για κάθε γνώρισμα του σχήματος εισόδου θα πρέπει να υπάρχει ένα γνώρισμα της πηγής δεδομένων που να αντιστοιχεί στο προαναφερθέν γνώρισμα.
2. Μία αντιστοίχηση ανάμεσα στα γνωρίσματα του σχήματος εισόδου και σε αυτά του σχήματος εξόδου ή απόρριψης.
3. Μία αντιστοίχηση ανάμεσα στο σχήμα εξόδου της διεργασίας και στο σχήμα εισόδου του καταναλωτή των δεδομένων της.

Οι αντιστοιχίσεις της δεύτερης ομάδας είναι εσωτερικές σε κάθε Διεργασία. Μπορούν να παραχθούν από την SQL πρόταση του σχήματος εξόδου/απόρριψης. Οι υπόλοιπες θα πρέπει να καθοριστούν κατά την κατασκευή του σεναρίου από το χρήστη.

3.2.3 Σενάρια

Ένα *Σενάριο* είναι μία απαρίθμηση από Διεργασίες μαζί με τα σύνολα εγγραφών πηγές-στόχους και τις αντίστοιχες σχέσεις Παροχής για κάθε Διεργασία. Τυπικά, ένα σενάριο αποτελείται από τα εξής:

- *Όνομα.* Ένα μοναδικό αναγνωριστικό.
- *Διεργασίες.* Μία πεπερασμένη λίστα από Διεργασίες. Να σημειωθεί ότι χρησιμοποιώντας λίστα (αντί συνόλου) επιβάλλουμε μία συνολική διάταξη των Διεργασιών στην εκτέλεση του σεναρίου.
- *Σύνολα Εγγραφών.* Ένα πεπερασμένο σύνολο από Σύνολα Εγγραφών.
- *Στόχοι.* Ένα ειδικού σκοπού υποσύνολο των Συνόλων Εγγραφών του σεναρίου, το οποίο περιλαμβάνει τους τελικούς προορισμούς της όλης διαδικασίας (στην ουσία τους πίνακες της Αποθήκης Δεδομένων που πρέπει να "γεμίσουν" από τις Διεργασίες του Σεναρίου).
- *Σχέσεις Παροχής.* Μία πεπερασμένη λίστα από Σχέσεις Παροχής μεταξύ των διεργασιών και των Συνόλων Εγγραφών του Σεναρίου.

Διαισθητικά μπορούμε να πούμε ότι ένα Σενάριο είναι ένα σύνολο από Διεργασίες αναπτυγμένες κατά μήκους ενός γράφου, σε μία σειρά εκτέλεσης η οποία είναι γραμμικά σειριοποιήσιμη.

Επιπλέον, θεωρούμε τους παρακάτω *Περιορισμούς Ακεραιότητας* για ένα Σενάριο:

Στατικοί Περιορισμοί.

- Όλες Οι αδύναμες οντότητες ενός σεναρίου (γνωρίσματα και παράμετροι) πρέπει να ενυπάρχουν σε μία Σχέση Μέρους (δηλαδή θα πρέπει να έχουν ένα περιέχον αντικείμενο).
- Όλες οι αντιστοιχίσεις στις Σχέσεις Παροχής θα πρέπει να ορίζονται μεταξύ γνωρισμάτων του ιδίου τύπου δεδομένων.

Περιορισμοί Ροής Δεδομένων.

- Όλα τα γνωρίσματα του σχήματος εισόδου πρέπει να έχουν μία πηγή δεδομένων
- Ως αποτέλεσμα της προηγούμενης απαίτησης, εάν ένα γνώρισμα χρησιμοποιείται σε μία Διεργασία Α, τότε το περιέχον αντικείμενο του πρέπει να προηγείται της Α στη διάταξη εκτέλεσης του σεναρίου.
- Όλα τα γνωρίσματα των Σχημάτων των Συνόλων Εγγραφών στόχων πρέπει να έχουν μία πηγή δεδομένων.

3.2.4 Ο Γράφος Αρχιτεκτονικής ενός Σεναρίου ΕΜΦ

Στις προηγούμενες παραγράφους δώσαμε τον τυπικό ορισμό των Διεργασιών, των Συνόλων Εγγραφών και των άλλων συνθετικών ενός λογικού σεναρίου ΕΜΦ. Το συνολικό πλάνο ενός σεναρίου ΕΜΦ, συμπεριλαμβάνοντας Διεργασίες, Σύνολα Εγγραφών και συναρτήσεις μπορεί να μοντελοποιηθεί ως γράφος, που ονομάζεται *Γράφος Αρχιτεκτονικής*. Ο γράφος αρχιτεκτονικής περιέχει όλες τις Διεργασίες και τους Αποθηκευτικούς Χώρους Δεδομένων ενός σεναρίου, καθώς και τα συστατικά τους. Επίσης, απεικονίζει τη ροή των δεδομένων μέσα από ένα περιβάλλον ΕΜΦ. Τέλος, καλύπτει τον καθορισμό τύπου για όλες τις εμπλεκόμενες οντότητες, καθώς και τη ρύθμιση του σεναρίου από συγκεκριμένες παραμέτρους.

Όντας γράφος, ο Γράφος Αρχιτεκτονικής ενός σεναρίου ΕΜΦ περιλαμβάνει κόμβους και ακμές. Οι εμπλεκόμενοι τύποι δεδομένων, τύποι συναρτήσεων, σταθερές, γνωρίσματα, διεργασίες, σύνολα εγγραφών και συναρτήσεις αποτελούν τους κόμβους του γράφου. Για την πλήρη κάλυψη των χαρακτηριστικών και των αλληλεπιδράσεων των οντοτήτων όπως αυτά αναφέρθηκαν παραπάνω, μοντελοποιούμε τα διάφορα είδη των σχέσεων ως ακμές του γράφου. Παρακάτω παρατίθενται αυτοί οι τύποι σχέσεων:

Σχέσεις Μέρους. Αυτές οι σχέσεις εμπλέκουν γνωρίσματα και παραμέτρους, και τα συνδέουν με την αντίστοιχη Διεργασία, συνάρτηση ή Σύνολο Εγγραφών στο οποίο ανήκουν.

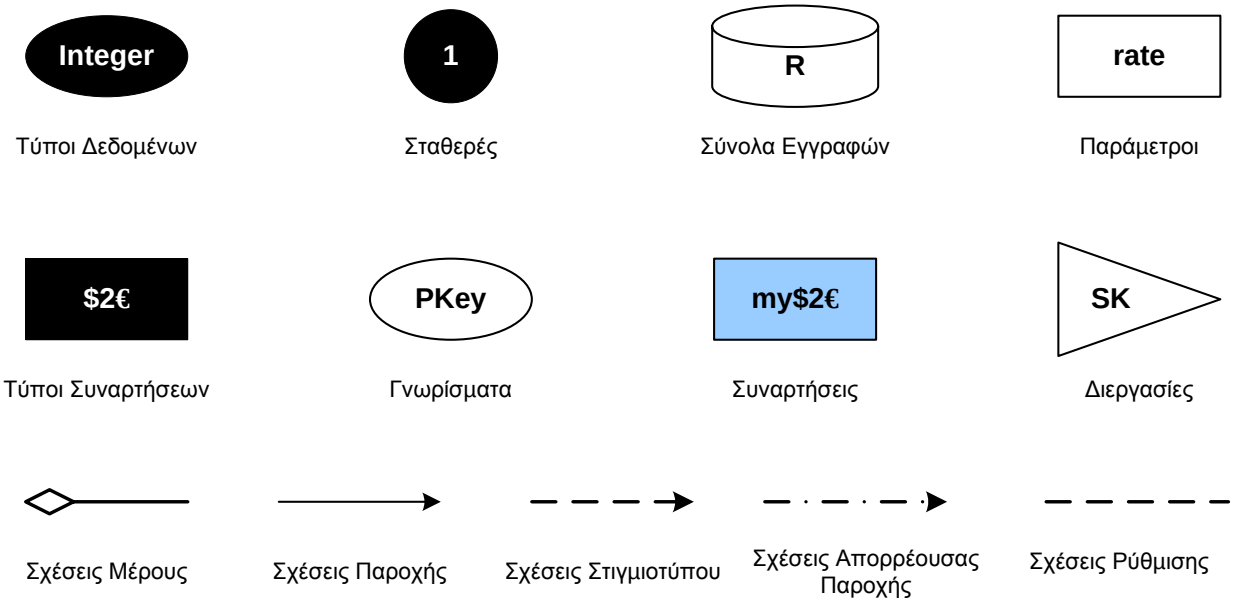
Σχέσεις Στιγμιοτύπου. Τέτοιες σχέσεις ορίζονται μεταξύ ενός τύπου δεδομένων/συνάρτησης και του στιγμιοτύπου του.

Σχέσεις Παροχής. Αυτού του τύπου οι σχέσεις συνδέουν τα γνωρίσματα με μία σχέση πηγής–στόχου.

Σχέσεις Ρύθμισης. Ορίζονται μεταξύ των παραμέτρων των Διεργασιών και των όρων που τις γεμίζουν.

Σχέσεις Απορρέουσας Παροχής. Είναι μία ειδική περίπτωση των σχέσεων Παροχής, η οποία μπορεί να παραχθεί από τη σύνθεση σχέσεων ρύθμισης. Οι σχέσεις Απορρέουσας Παροχής μπορούν να προκύψουν από ένα απλό κανόνα και δεν αποτελούν συστατικό μέρος του σεναρίου.

Η γραφική αναπαράσταση του γράφου αρχιτεκτονικής απεικονίζεται στο Σχήμα 3.1.

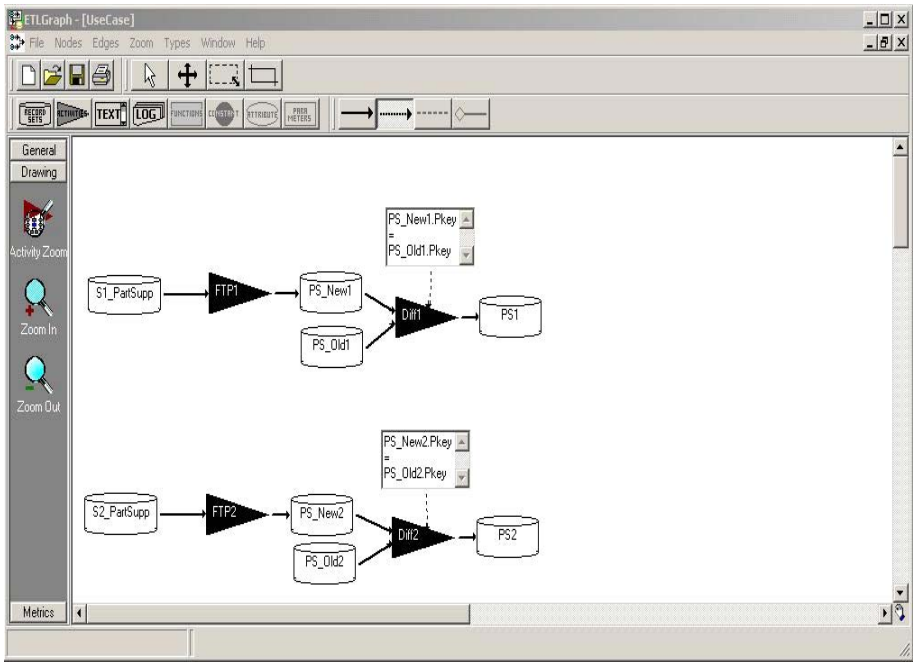


Σχήμα 3.1: Γραφικοί Συμβολισμοί για το λογικό μοντέλο διεργασιών ΕΜΦ

3.3 Εργαλείο Λογικής Σχεδίασης

Στα πλαίσια του προγράμματος *Άρκτος II*, έχει αναπτυχθεί ένα γραφικό εργαλείο σχεδιασμού λογικών σεναρίων ΕΜΦ. Μέσω του εργαλείου αυτού, ο χρήστης επιλέγει τις πηγές και την Αποθήκη Δεδομένων, τις διεργασίες που εμπλέκονται στη διαδικασία αυτή και ορίζει τη ροή των δεδομένων μέσα στο σενάριο. Αυτές οι εργασίες στηρίζονται σε (α) ένα φιλικό περιβάλλον εργασίας και (β) ένα σύνολο προτύπων που μπορούν να ξαναχρησιμοποιηθούν.

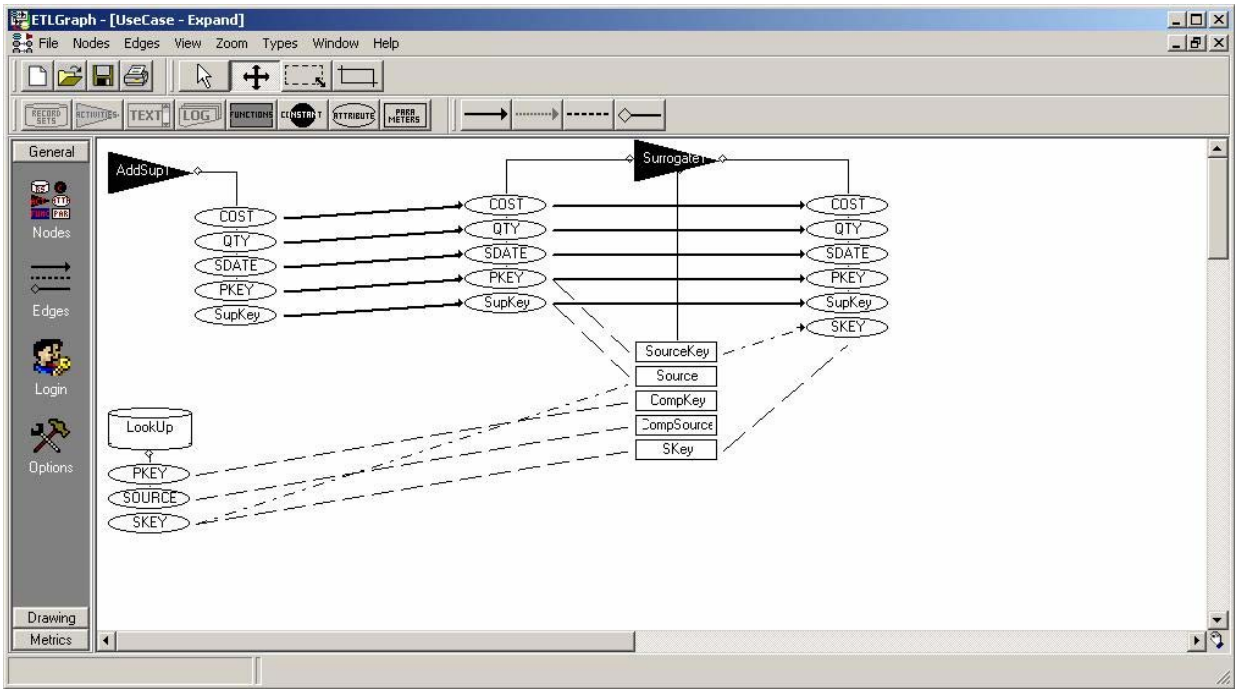
Όλες οι λεπτομέρειες για τον ορισμό μίας διεργασίας μπορούν να εισαχθούν μέσω φορμών και/ή απλών επιλογών. Συγκεκριμένα, ο χρήστης μπορεί να ελέγξει τις πηγές δεδομένων και τις διεργασίες που έχουν ήδη οριστεί στο σενάριο, όπως και τα σχήματά τους (εισόδου, εξόδου και παραμέτρων). Τα γνωρίσματα που ανήκουν στο σχήμα εξόδου μίας διεργασίας ή ενός συνόλου εγγραφών μπορούν εύκολα να εισαχθούν στο σχήμα εισόδου μίας επόμενης διεργασίας ή ενός συνόλου εγγραφών, έτσι ώστε μα οριστεί η ισοδύναμη ροή δεδομένων στο σενάριο. Με παρόμοιο τρόπο, κάποιος μπορεί να ορίσει τις παραμέτρους μίας διεργασίας. Εξ ορισμού, το σχήμα εξόδου μίας διεργασίας αρχικοποιείται σαν αντίγραφο του σχήματος εισόδου. Στη συνέχεια, ο χρήστης μπορεί να αλλάξει αυτή την επιλογή ανάλογα με τις ανάγκες του· για παράδειγμα, διαγράφοντας ή μετονομάζοντας κατάλληλα γνωρίσματα. Σαν σχήμα απόρριψης μίας διεργασίας θεωρείται ένα αντίγραφο του σχήματος εισόδου της αντίστοιχης



Σχήμα 3.2: Παράδειγμα Σεναρίου στο Εργαλείο Λογικής Σχεδίασης

διεργασίας, και ο χρήστης επιλέγει τη φυσική του θέση· για παράδειγμα τη φυσική θέσει του αρχείου συμβάντων που περιέχει τις σειρές που έχουν απορριφθεί από τη συγκεκριμένη διεργασία. Εκτός από αυτές τις δυνατότητες, ο χρήστης μπορεί (α) να σχεδιάσει τα επιθυμητά γνωρίσματα ή παραμέτρους, (β) να ορίσει τα ονόματα και τους τύπους δεδομένων τους, (γ) να τα συνδέσει στα σχήματά τους, (δ) να δημιουργήσει σχέσεις παροχής και σχέσεις ρύθμισης μεταξύ τους, και (ε) να σχεδιάσει τις κατάλληλες ακμές μεταξύ των κόμβων του γράφου αρχιτεκτονικής. Το εργαλείο εξασφαλίζει την συνέπεια ενός σεναρίου, επιτρέποντας στο χρήστη να σχεδιάζει μόνο σχέσεις που ικανοποιούν τους περιορισμούς του μοντέλου. Όσον αφορά τις σχέσεις παροχής και στιγμιοτύπου, αυτές υπολογίζονται αυτόματα και εμφανίζονται ή κρύβονται μέσω μίας επιλογής στο μενού του εργαλείου. Στο Σχήμα 3.2 παρουσιάζεται ένα παράδειγμα σεναρίου στο Εργαλείο Λογικής Σχεδίασης.

Εκτός από τα παραπάνω, το εργαλείο λογικής σχεδίασης προσφέρει τη δυνατότητα μεγέθυνσης/σμίκρυνσης, μία πολύ χρήσιμη δυνατότητα κατά την κατασκευή της ροής δεδομένων στο σενάριο, μέσω των απεικονίσεων "παροχής", μεταξύ των γνωρισμάτων. Ο σχεδιαστής μπορεί να δουλεύει στο σενάριο σε δύο επίπεδα λεπτομέρειας: (α) στο *επίπεδο οντότητας* με ορατά μόνο τα σύνολα εγγραφών και οι διεργασίες που εμπλέκονται στη διαδικασία, όπου οι σχέσεις παροχής τους εμφανίζονται μεταξύ των αντίστοιχων οντοτήτων, και (β) στο *επίπεδο γνωρίσματος*, όπου ο χρήστης μπορεί να δει και να δουλέψει στα συστατικά μέρη μίας διεργασίας καθώς και τις αντίστοιχες σχέσεις παροχής σε επίπεδο γνωρισμάτων. Στο Σχήμα 3.3 παρουσιάζεται τμήμα του σεναρίου του Σχήματος 3.2. Παρατηρείστε (α) πως οι σχέσεις παροχής εμφανίζονται και συνδέουν γνωρίσματα με τις αντίστοιχες οντότητες· (β) πως οι σχέσεις παροχής συνδέουν τα γνωρίσματα μεταξύ του· (γ) πως οι σχέσεις ρύθμισης τροφοδοτούν τις παραμέτρους των διεργασιών και (δ) πως οι σχέσεις στιγμιοτύπου συσχετίζουν τα γνωρίσματα με τους αντίστοιχους τύπους δεδομένων που απεικονίζονται στο κάτω δεξιά τμήμα του σχήματος.



Σχήμα 3.3: Λεπτομέρια του σεναρίου του Σχήματος 3.2, όπως φαίνεται σε επίπεδο γνωρίσματος

Στο εργαλείο λογικής σχεδίασης, το στοιχείο της *προσαρμογής* υποστηρίζεται μέσω επαναχρησιμοποιήσιμων προτύπων. Η έννοια των προτύπων αποτελούν βασικό στοιχείο του εργαλείου αυτού. Υπάρχουν πρότυπα πρακτικά για κάθε πτυχή του μοντέλο: τύπους δεδομένων, συναρτήσεις και διεργασίες. Τα πρότυπα είναι επεκτάσιμα, έτσι παρέχουν στο χρήστη τη δυνατότητα να προσαρμόσει το περιβάλλον ανάλογα με τις ανάγκες του. Ειδικά για τις διεργασίες, που αποτελούν τον πυρήνα του λογικού μοντέλου, παρέχεται ένα συγκεκριμένο μενού με ένα σύνολο από συχνά χρησιμοποιούμενες Διεργασίες ΕΜΦ. Το σύστημα έχει ένα ενσωματωμένο μηχανισμό υπεύθυνο για την αρχικοποίηση των προτύπων LDL, που υποστηρίζονται μέσω μίας γραφικής φόρμας που βοηθά το χρήστη να ορίσει τις παραμέτρους του προτύπου επιλέγοντας τις τιμές τους από τα κατάλληλα αντικείμενα του σεναρίου. Επιπλέον αυτών, μία ξεχωριστή δυνατότητα του εργαλείου, είναι η ικανότητά του να υπολογίζει τη σχεδιαστική ποιότητα του σεναρίου, χρησιμοποιώντας ένα σύνολο μετρικών, είτε σε όλο το σενάριο είτε σε κάθε διεργασία αυτού.

4

Μετάβαση από το Εννοιολογικό στο Λογικό Μοντέλο

Στο κεφάλαιο αυτό περιγράφεται μία ημιαυτόματη μετάβαση από το εννοιολογικό στο λογικό μοντέλο για διεργασίες ΕΜΦ ([39]). Στην ενότητα 4.2 παρουσιάζονται οι διάφορες απεικονίσεις των δομικών συστατικών του εννοιολογικού στο λογικό μοντέλο. Στην ενότητα 4.3 παρουσιάζεται μία μεθοδολογία για την εύρεση της σειράς εκτέλεσης των διεργασιών στο λογικό μοντέλο, ενώ στην ενότητα 4.4 παρουσιάζεται μέσω μίας σειράς βημάτων, η μεθοδολογία δημιουργίας του λογικού μοντέλου από το εννοιολογικό.

4.1 Εισαγωγή

Με τη συσχέτιση ενός λογικού με ένα εννοιολογικό μοντέλο, ενοποιούμε τα πλεονεκτήματα και των δύο μοντέλων. Αφ΄ ενός έχουμε ένα απλό μοντέλο αποτελεσματικό για τα πρώτα στάδια του σχεδιασμού ενός έργου Αποθήκης Δεδομένων, και αφ΄ ετέρου, ένα λογικό μοντέλο το οποίο μας παρέχει τυπικές και σημασιολογικά θεμελιωμένες έννοιες για τη σύλληψη των ιδιαιτεροτήτων μίας διεργασίας ΕΜΦ. Η μεθοδολογία για τη μετάβαση από το ένα μοντέλο στο άλλο, παρέχει μία συνεκτική τεχνική για τη μοντελοποίηση και τη διαχείριση διεργασιών ΕΜΦ.

Κατά τη μετάβαση έχουμε να αντιμετωπίσουμε διάφορα προβλήματα. Αρχικά, πρέπει να αναγνωρίσουμε την αντιστοιχία μεταξύ των δύο μοντέλων. Επειδή το εννοιολογικό μοντέλο είναι κατασκευασμένο με τρόπο πολύ γενικό, κάθε εννοιολογική οντότητα απεικονίζεται σε μία λογική οντότητα. Αντίθετα, η αντίστροφη απεικόνιση δεν είναι δυνατή. Κατ΄ επέκταση, για κάθε εννοιολογική οντότητα, ορίζουμε

την αντίστοιχη λογική οντότητα και περιγράφουμε μία μέθοδο για την αυτόματη μετάβαση από την πρώτη στη δεύτερη.

Παρ' όλα αυτά, μπορούμε να κάνουμε κάτι ακόμη καλύτερο. Μπορούμε να προχωρήσουμε πέραν της ένα-προς-ένα απεικόνισης και να ενώσουμε πληροφορίες για περισσότερες από μία εννοιολογικές οντότητες, επιτυγχάνοντας έτσι τον καλύτερο ορισμό μίας λογικής οντότητας. Για παράδειγμα, η εννοιολογική οντότητα *μετασχηματισμός* απεικονίζεται σε μία λογική *διεργασία*. Όμως, δεν είναι προφανές πώς η διεργασία θα περιγραφεί πλήρως από την πληροφορία που παρέχεται από τον αντίστοιχο μετασχηματισμό. Χρησιμοποιώντας τα εννοιολογικά εισόδου και εξόδου του μετασχηματισμού και της παρεχόμενης πηγής, μπορούμε εύκολα να αναγνωρίσουμε τα σχήματα της αντίστοιχης διεργασίας είτε άμεσα (σχήματα εισόδου, εξόδου και λειτουργικότητας), είτε έμμεσα (σχήματα παραγομένων και αποβλημένων). Αυτό, όμως, δεν είναι αρκετό διότι δεν παρέχεται καθόλου πληροφορία για την επεξεργασία της κατάλληλης πρότυπης διεργασίας. Όπως θα δούμε στη συνέχεια, το πρόβλημα αυτό μπορεί να λυθεί με τη χρήση πρόσθετης πληροφορίας προσαρμοσμένης σε κάποιο σχόλιο προσαρτημένο στον εννοιολογικό μετασχηματισμό.

Επιπλέον, έχουμε να αντιμετωπίσουμε δύο προβλήματα σημειογραφίας: την περίπτωση των γνωρισμάτων που αποβάλλονται από τη ροή των δεδομένων, και την σύγκλιση δύο διαφορετικών ροών δεδομένων στην ίδια πηγή δεδομένων. Στο εννοιολογικό μοντέλο, όταν τα δεδομένα κάποιου γνωρίσματος, σταματήσουν σε κάποιο σημείο να διαδίδονται, δεν τοποθετούμε την ακμή που συμβολίζει την παροχή από το γνώρισμα αυτό σε κάποιο άλλο. Επίσης, όταν επιθυμούμε τη σύγκλιση δύο ροών προς κάποια συγκεκριμένη έννοια, τη συνδέουμε απλά και με τις δύο αυτές ροές. Στο λογικό μοντέλο, ενεργούμε διαφορετικά σε κάθε περίπτωση. Κατά συνέπεια, θα πρέπει να διακρίνουμε τέτοιες συνθήκες στο εννοιολογικό σχεδιάγραμμα και τότε, όπως θα δούμε και στη συνέχεια, να χρησιμοποιούμε κάποια επιπλέον διεργασία για να εκφράσουμε τις περιπτώσεις αυτές στο λογικό διάγραμμα.

Στο Κεφάλαιο 2 είδαμε ότι το εννοιολογικό μοντέλο δεν αποτελεί ροή έργου. Αντίθετα, απλά αναγνωρίζει τους μετασχηματισμούς που χρειάζονται για σε μία διεργασία ΕΜΦ. Η τοποθέτηση των μετασχηματισμών στο εννοιολογικό διάγραμμα, δεν υποδηλώνει καθαρά τη σειρά εκτέλεσής τους. Το λογικό μοντέλο όμως αποτελεί μία ροή έργου και κατ' επέκταση είναι πολύ σημαντικό να καθορίζεται η σειρά εκτέλεσης των διεργασιών. Έτσι, στη συνέχεια, παρουσιάζουμε μία μεθοδολογία για την ημιαυτόματη αναγνώριση της ορθής σειράς εκτέλεσης των διεργασιών στο λογικό επίπεδο, όπου αυτό είναι δυνατό, ομαδοποιώντας τους μετασχηματισμούς του εννοιολογικού μοντέλου σε στάδια *ισοδύναμης-σειράς* μετασχηματισμών.

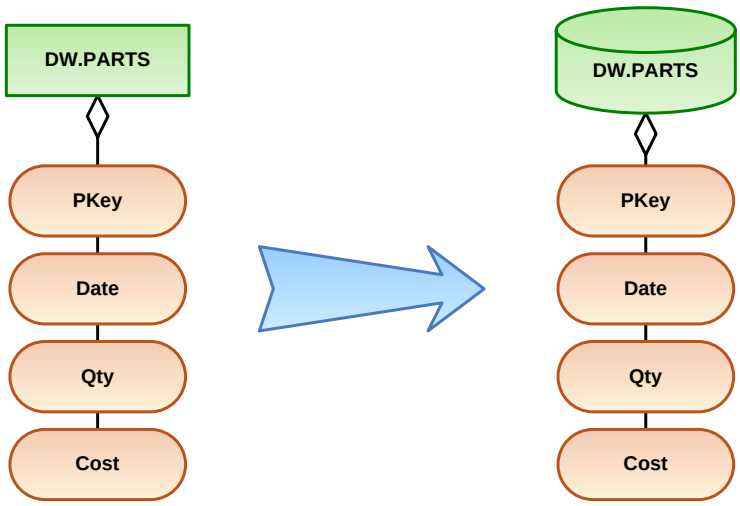
Με στόχο την τυποποίηση της μετάβασης μεταξύ των δύο μοντέλων και την αντιμετώπιση των προαναφερθέντων προβλημάτων, παρουσιάζεται στη συνέχεια ένα σύνολο βημάτων, το οποίο αποτελεί τη μεθοδολογία για τη μετάβαση από το εννοιολογικό στο λογικό μοντέλο. Η παρουσίαση της μεθοδολογίας αυτής, για σκοπούς ευκολίας, θα στηριχθεί στο Παράδειγμα 1 (Σχήμα 2.3) που παρουσιάστηκε στο Κεφάλαιο 2.

4.2 Απεικονίσεις

Στην ενότητα αυτή θα εξετάσουμε ένα προς ένα τα συστατικά του εννοιολογικού μοντέλου, θα αναγνωρίσουμε τις αντίστοιχες λογικές οντότητες, και, θα περιγράψουμε πως κάθε εννοιολογική οντότητα απεικονίζεται στην αντίστοιχη λογική. Οι έννοιες και τα γνωρίσματα απεικονίζονται στα σύνολα εγγραφών και στα γνωρίσματα, αντίστοιχα. Οι μετασχηματισμοί και οι περιορισμοί ΕΜΦ απεικονίζονται στις διεργασίες. Τα σχόλια χρησιμοποιούνται για την αναγνώριση και την επεξεργασία του κατάλληλου προτύπου διεργασίας. Επίσης, καταπιανόμαστε με δύο ειδικά προβλήματα σχεδιασμού: την περίπτωση των γνωρισμάτων που αποβάλλονται από μία ροή δεδομένων και τη σύγκλιση δύο διαφορετικών ροών σε μία πηγή δεδομένων.

4.2.1 Έννοιες και Γνωρίσματα

Όπως έχει ήδη αναφερθεί, ένας από τους βασικούς στόχους του εννοιολογικού μοντέλου είναι η αναγνώριση όλων των πηγών δεδομένων, μαζί με τα γνωρίσματά τους, που λαμβάνουν μέρος στην όλη διεργασία ΕΜΦ. Για κάθε έννοια στο εννοιολογικό μοντέλο, ορίζεται ένα σύνολο εγγραφών στο λογικό. Το όνομα και η λίστα των γνωρισμάτων του συνόλου εγγραφών είναι ίδια με αυτά της έννοιας. Υπάρχει αντιστοιχία ένα-προς-ένα σε κάθε ένα από τα γνωρίσματα του εννοιολογικού μοντέλου σε ένα αντίστοιχο στο λογικό μοντέλο. Δηλαδή, το όνομα και ο τύπος δεδομένων του παραμένει ο ίδιος. Το Σχήμα 4.1 απεικονίζει τη μετάβαση από την έννοια **DW.PARTS** στο σύνολο εγγραφών **DW.PARTS** μαζί με τα γνωρίσματά του.



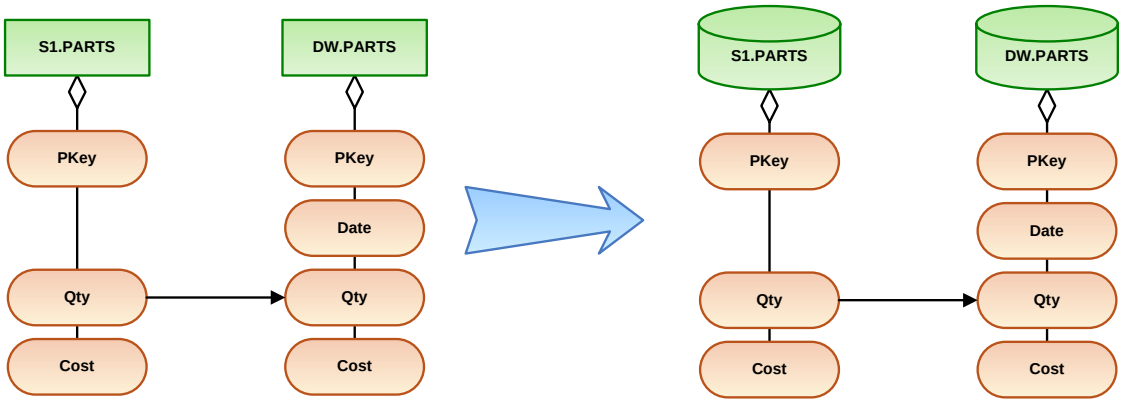
Σχήμα 4.1: Μετάβαση της έννοιας **DW.PARTS** σε ένα σύνολο εγγραφών, μαζί με τα γνωρίσματά της

4.2.2 Σχέσεις

Το εννοιολογικό μοντέλο περιλαμβάνει τέσσερις τύπους σχέσεων: μέρους, υποψηφίου, ενεργού υποψηφίου και σχέσεις παροχής.

Οι σχέσεις μέρους χρησιμοποιούνται για να επισημαίνουν το γεγονός ότι μία συγκεκριμένη έννοια αποτελείται από ένα σύνολο γνωρισμάτων, δηλαδή, στο μοντέλο τα γνωρίσματα χρησιμοποιούνται σαν "πολίτες πρώτης κατηγορίας". Τα χαρακτηριστικά αυτά διατηρούνται και στο λογικό μοντέλο. Έτσι, οι εννοιολογικές σχέσεις μέρους απεικονίζονται σε λογικές σχέσεις μέρους, με ακριβώς την ίδια σημασία και τα ίδια χαρακτηριστικά. Η χρήση των σχέσεων μέρους φαίνεται στο Σχήμα 4.1.

Για τις σχέσεις υποψηφίου και ενεργού υποψηφίου, δεν υπάρχει άμεση απεικόνιση στο λογικό μοντέλο. Η εισαγωγή τους στο εννοιολογικό μοντέλο καλύπτει την περίπτωση όπου στα πρώιμα στάδια του σχεδιασμού μίας Αποθήκης Δεδομένων μπορεί να υπάρχουν περισσότερες από μία έννοιες (πηγές δεδομένων) για την πλήρωση μίας συγκεκριμένης έννοιας. Όταν προχωρούμε στο λογικό μοντέλο, τα προβλήματα αυτά έχουν ήδη λυθεί. Οι κατάλληλες αποφάσεις έχουν παρθεί και για κάθε σύνολο εγγραφών (πηγή δεδομένων) πρέπει να ξέρουμε επακριβώς το σύνολο εγγραφών από το οποίο προέρχεται. Κατά συνέπεια, οι υποψήφιες πηγές αποβάλλονται από το σχεδιάγραμμα και ο ενεργός υποψήφιος (αυτός που έχει τελικά επιλεγεί) μετασχηματίζεται σε μία εννοιολογική έννοια (δείτε επίσης Ενότητα 2.2) και στη συνέχεια σε ένα λογικό σύνολο εγγραφών.



Σχήμα 4.2: Μετάβαση μίας απλής σχέσης παροχής

Οι σχέσεις παροχής αναπαριστούν τη ροή των δεδομένων κατά τη διάρκεια μίας διεργασίας ΕΜΦ. Κατά συνέπεια, μία εννοιολογική σχέση παροχής από το γνώρισμα-πηγή σε ένα γνώρισμα-στόχο, περιλαμβάνει όλους τους κατάλληλους μετασχηματισμούς που απαιτούνται να εφαρμοστούν από τις σχεδιαστικές απαιτήσεις. Όταν δεν παρεμβάλλεται κάποιος μετασχηματισμός μεταξύ του γνωρίσματος-πηγής και του γνωρίσματος-στόχου, υπάρχει αντιστοιχία μεταξύ της εννοιολογικής και της λογικής σχέσης παροχής (Σχήμα 4.2).

Οι πιο πολύπλοκες περιπτώσεις όπου απαιτείται η παρουσία ενός ή και περισσοτέρων μετασχηματισμών μεταξύ των γνωρισμάτων-πηγών και των γνωρισμάτων-στόχων, καλύπτονται στην επόμενη υποενότητα, μαζί με τους εννοιολογικούς μετασχηματισμούς.

4.2.3 Εννοιολογικοί Μετασχηματισμοί

Στο εννοιολογικό μοντέλο, οι μετασχηματισμοί χρησιμοποιούνται για την αναπαράσταση ενεργειών για:

1. Τη συντήρηση του σχήματος των δεδομένων (π.χ. καθαρισμός, φιλτράρισμα κ.λ.π.). Γενικά, τους μετασχηματισμούς αυτού του είδους, τους ονομάζουμε *φίλτρα*.
2. Την αλλαγή του σχήματος των δεδομένων (π.χ. συνάθροιση). Τους μετασχηματισμούς αυτούς τους ονομάζουμε *μετασχηματιστές*.

Σε λογικό επίπεδο, χρησιμοποιούμε διεργασίες για να αναπαραστήσουμε τις ενέργειες αυτές. Έτσι, η μετάβαση από το εννοιολογικό στο λογικό περιλαμβάνει την απεικόνιση των εννοιολογικών μετασχηματισμών σε λογικές διεργασίες.

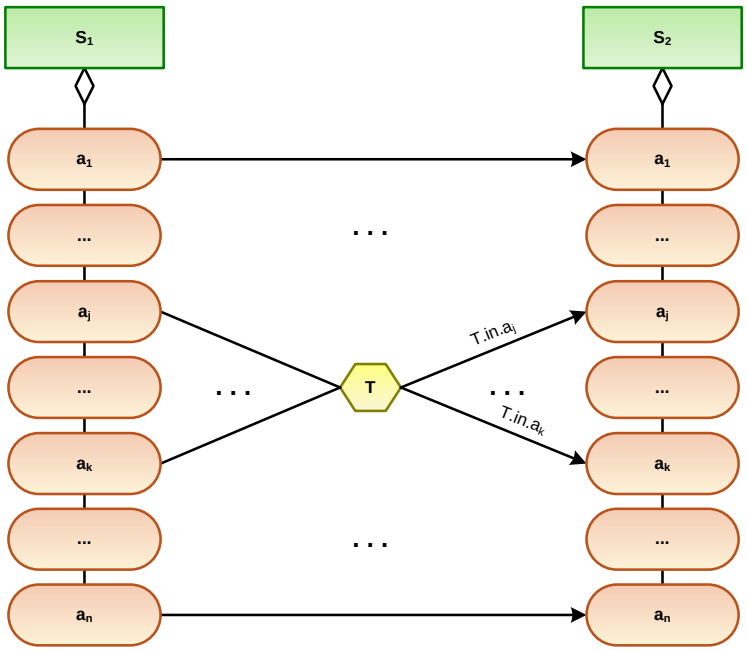
Για το σκοπό αυτό, έχουμε να αντιμετωπίσουμε τρία βασικά προβλήματα:

1. Τον ορισμό των *ιδιοτήτων* μίας διεργασίας (π.χ. όνομα, σχήματα, σημασιολογία).
2. Τη *σειριακή σύνθεση* των εννοιολογικών μετασχηματισμών
3. Την εύρεση κατάλληλης *σειράς εκτέλεσης* των διεργασιών στο λογικό μοντέλο.

Σε αυτή την υποενότητα, αντιμετωπίζουμε τα πρώτα δύο προβλήματα. Το τελευταίο αντιμετωπίζεται στην επόμενη υποενότητα. Κατά συνέπεια, εξετάζουμε την απεικόνιση των φίλτρων σε διεργασίες και στη συνέχεια επεξεργαζόμαστε την απεικόνιση των μετασχηματισμών σε διεργασίες.

Ιδιότητες μίας διεργασίας

Η απεικόνιση των *φίλτρων σε διεργασίες* είναι άμεση. Με δεδομένο το ότι δεν επηρεάζουν το σχήμα των δεδομένων, μπορούμε να εργαστούμε ως ακολούθως. Στο παράδειγμα του Σχήματος 4.3 παρουσιάζεται η πλήρωση της έννοιας $\mathbf{S}_2$ από την έννοια $\mathbf{S}_1$ μέσω του μετασχηματισμού **T**, ο οποίος επιδρά στα γνωρίσματα $\{\mathbf{a}_j, \ldots, \mathbf{a}_k\}$. Έτσι, ο μετασχηματισμός **T** έχει ως σχήμα εισόδου $\mathbf{T.in}=\{\mathbf{S}_1.\mathbf{a}_j, \ldots, \mathbf{S}_1.\mathbf{a}_k\}$ και ως σχήμα εξόδου $\mathbf{T.out}=\{\mathbf{S}_2.\mathbf{a}_j, \ldots, \mathbf{S}_2.\mathbf{a}_k\}$. Τα γνωρίσματα της έννοιας-πηγής $\mathbf{S}_1$ που συνδέονται με το μετασχηματισμό **T** θεωρούνται παράμετροι του **T**. Έτσι, στο λογικό επίπεδο, τα γνωρίσματα αυτά ανήκουν στο σχήμα λειτουργικότητας της διεργασίας **T**. Τα υπόλοιπα γνωρίσματα της $\mathbf{S}_1$ απλά διαδίδουν δεδομένα στα αντίστοιχα γνωρίσματα της $\mathbf{S}_2$. Λόγω της μη αλλαγής των σχημάτων εισόδου και εξόδου, τα φίλτρα έχουν κενά σχήματα παραγομένων και μη-διαδεδομένων. Τέλος, παρατηρείστε στο Σχήμα 4.3 ότι κάθε ακμή εξόδου του εννοιολογικού μετασχηματισμού προς κάποιο γνώρισμα στόχο, αναγράφεται το όνομα του γνωρίσματος-πηγής που είναι υπεύθυνο για την πλήρωση του συγκεκριμένου γνωρίσματος στόχου. Ως αποτέλεσμα αυτού γνωρίζουμε επακριβώς τον παροχέα για κάθε στόχο και έτσι έχουμε τη ροή δεδομένων υπό έλεγχο.

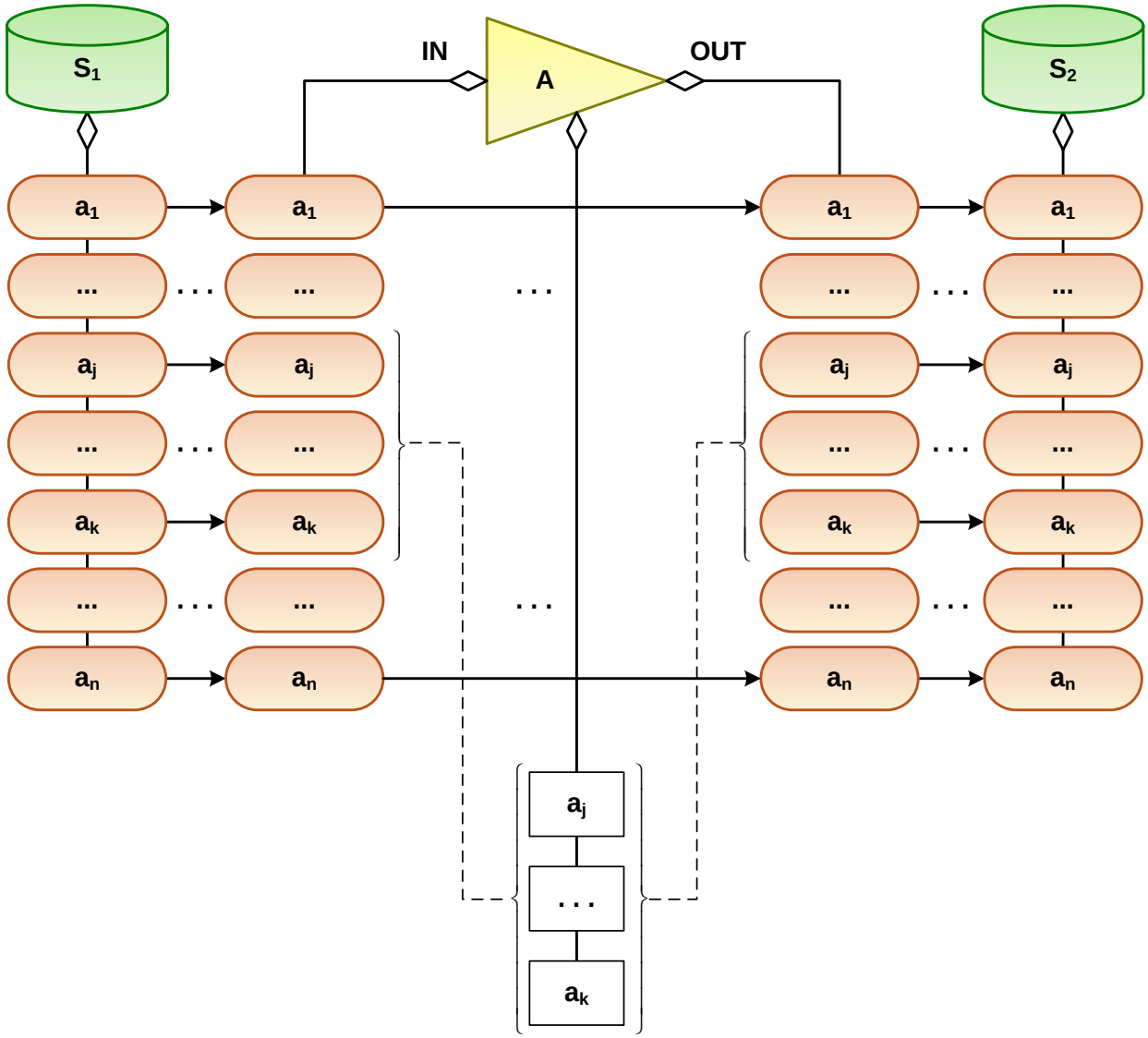


Σχήμα 4.3: Πλήρωση της έννοιας $\mathbf{S}_2$ από την έννοια $\mathbf{S}_1$ μέσω του μετασχηματισμού **T**

Με βάση αυτά, το εννοιολογικό σχεδιάγραμμα του Σχήματος 4.3 αντιστοιχεί στο λογικό σχεδιάγραμμα του Σχήματος 4.4. Στο Σχήμα 4.4 απεικονίζεται μία διεργασία **A** μαζί με τα σχήματα εισόδου, εξόδου και λειτουργικότητας. Τα δεδομένα από το σύνολο εγγραφών $\mathbf{S}_1$ διαδίδονται στο σχήμα εισόδου της διεργασίας **A**. Μερικά από τα γνωρίσματα εισόδου απλά τροφοδοτούν τα αντίστοιχα γνωρίσματα εξόδου της διεργασίας **A**, ενώ κάποια άλλα γνωρίσματα εξόδου $\{a_j, \ldots, a_k\}$ τροφοδοτούνται με χρήση του κατάλληλου σχήματος λειτουργικότητας μέσω σχέσεων ρύθμισης (βλέπε υπονότητα 3.2.4).

Η απεικόνιση *μετασχηματιστών σε διεργασίες* διαφέρει στο ότι το σχήμα παραγωγής και/ή το σχήμα απόρριψης της αντίστοιχης διεργασίας δεν είναι κενό. Υπενθυμίζουμε ότι ένας μετασχηματιστής αλλάζει το σχήμα των δεδομένων. Έτσι, πρέπει να εμπλουτίσουμε την τεχνική που παρουσιάστηκε για την απεικόνιση των φίλτρων με μία μέθοδο για τον προσδιορισμό των άλλων δύο σχημάτων. Προφανώς, αυτό δεν μπορεί να θεωρηθεί σαν πρόβλημα, αν κάποιος συγκρίνει τα σχήματα εισόδου και εξόδου του εννοιολογικού μετασχηματισμού. Τα γνωρίσματα που αποβάλλονται είναι τα γνωρίσματα εισόδου που δεν χρησιμοποιούνται από κάποιο γνώρισμα εξόδου. Αντίστοιχα, τα παραγόμενα γνωρίσματα είναι γνωρίσματα εξόδου που δεν προέρχονται από κάποιο γνώρισμα εισόδου του σχήματος εισόδου του εννοιολογικού μετασχηματισμού.

Τυπικά, ένας μετασχηματισμός **T** σε εννοιολογικό επίπεδο (είτε είναι φίλτρο, είτε είναι μετασχηματιστής) αντιστοιχεί σε μία διεργασία **A** στο λογικό επίπεδο. Τα γνωρίσματα εισόδου του **T** αποτελούν το σχήμα λειτουργικότητας της αντίστοιχης διεργασίας **A** (**A.fun**). Αν υπάρχουν γνωρίσματα που ανήκουν στο σχήμα εξόδου του **T** τα οποία δεν έχουν κάποιο παροχέα στο σχήμα εισόδου του **T**, τότε αποτελούν το σχήμα παραγωγής της διεργασίας **A** (**A.gen**). Με τον ίδιο τρόπο, αν υπάρχουν γνωρίσματα στο σχήμα εισόδου του **T** τα οποία δεν χρησιμοποιούνται από κάποιο γνώρισμα εξόδου στο σχήμα εξόδου του **T**, τότε αυτά τα γνωρίσματα αποτελούν το σχήμα απόρριψης της διεργασίας **A** (**A.pro**). Στην απλή

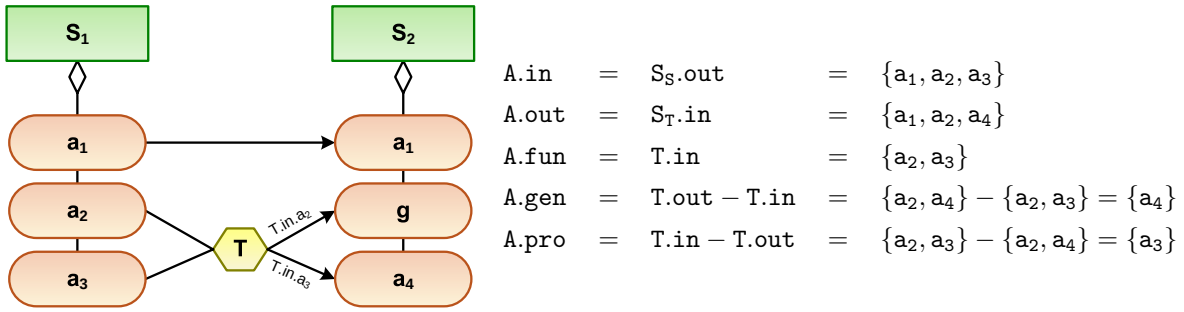


Σχήμα 4.4: Πλήρωση συνόλου εγγραφών $\mathbf{S}_2$ από το σύνολο $\mathbf{S}_1$ μέσω της διεργασίας **A**

περίπτωση μίας διεργασίας ΕΜΦ που περιλαμβάνει μόνο ένα μετασχηματισμό, τα γνωρίσματα της έννοιας-πηγής (έστω $\mathbf{S}_S$) του **T** αποτελούν το σχήμα εισόδου μίας διεργασίας **A** (**A.in**) και τα γνωρίσματα του σχήματος-στόχου (έστω $\mathbf{S}_T$) αποτελούν το σχήμα εξόδου της διεργασίας **A** (**A.out**). Έτσι, ισχύει η ακόλουθη διατύπωση:

$$\begin{aligned} \texttt{A.in} &= \texttt{S}_\texttt{S}\texttt{.out} \\ \texttt{A.out} &= \texttt{S}_\texttt{T}\texttt{.in} \\ \texttt{A.fun} &= \texttt{T.in} \\ \texttt{A.gen} &= \texttt{T.out} - \texttt{T.in} \\ \texttt{A.pro} &= \texttt{T.in} - \texttt{T.out} \end{aligned}$$

Προφανώς, για φίλτρα **A.gen** = $\emptyset$ και **A.pro** = $\emptyset$. Ένα παράδειγμα για τον καθορισμό των σχημάτων μιας διεργασίας υπάρχει στο Σχήμα 4.5.

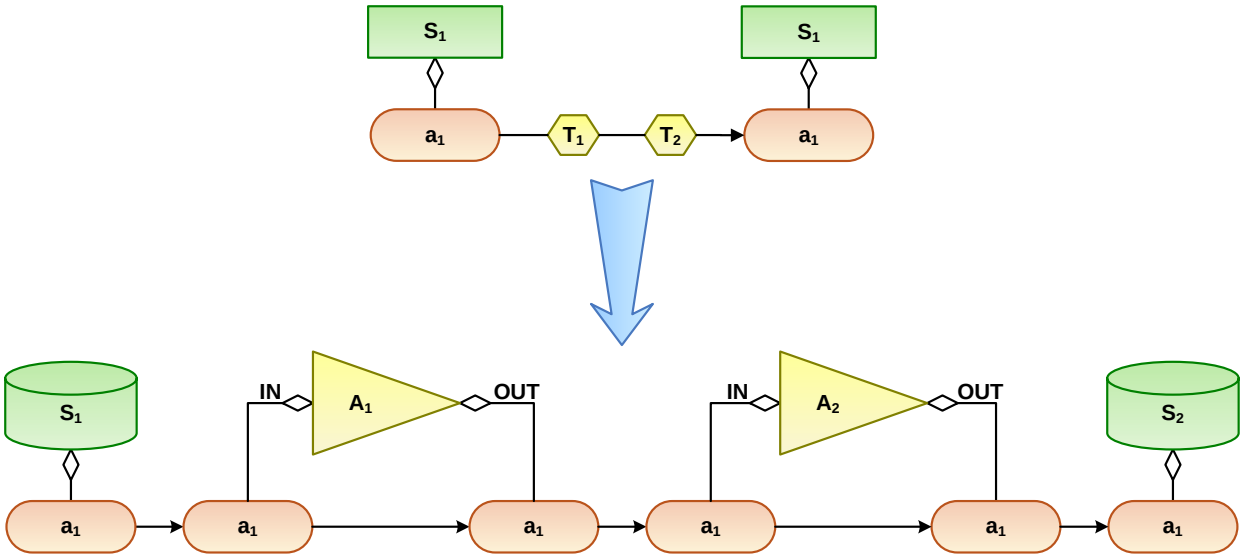


Σχήμα 4.5: Παράδειγμα απεικόνισης ενός μετασχηματισμού T σε μία διεργασία A

Σημειώστε ότι για πιο σύνθετες περιπτώσεις, τα σχήματα εισόδου και εξόδου δεν υπολογίζονται τόσο εύκολα. Για παράδειγμα, υποθέστε ότι το Σχήμα 4.5 εμπλουτίζεται με ένα δεύτερο μετασχηματισμό T′ που επιδρά μόνο στο γνώρισμα-πηγή $\mathbf{S}_S.\mathbf{a}_1$ και το μετασχηματίζει στο γνώρισμα-στόχο $\mathbf{S}_T.\mathbf{a}_5$. Τότε, τα σχήματα εισόδου και εξόδου της διεργασίας **A** δεν είναι αυτά που απεικονίζονται στο Σχήμα 4.5. Αν η διεργασία **A** εκτελείται πριν από από την **A'** (τη διεργασία που αντιστοιχεί στο μετασχηματισμό **T'**, τότε τα σχήματα εισόδου και εξόδου θα είναι τα **{**$\mathbf{a}_1$**,**$\mathbf{a}_2$**,**$\mathbf{a}_3$**}** και **{**$\mathbf{a}_1$**,**$\mathbf{a}_2$**,**$\mathbf{a}_4$**}** αντίστοιχα. Στην αντίθετη περίπτωση (όπου η διεργασία **A'** εκτελείται μετά την **A**, τότε τα σχήματα εισόδου και εξόδου (της **A**) θα είναι $\mathbf{a}_5$,$\mathbf{a}_2$,$\mathbf{a}_3$ και $\mathbf{a}_5$,$\mathbf{a}_2$,$\mathbf{a}_4$, αντίστοιχα. Στη συνέχεια θα ασχοληθούμε λεπτομερώς στον τρόπο αντιμετώπισης τέτοιων περιπτώσεων, ενώ το [40] παρουσιάζει τον αλγόριθμο **Schema Generator** που αυτοματοποιεί τη δημιουργία όλων των σχημάτων εισόδου και εξόδου που περιλαμβάνονται στην όλη διεργασία ΕΜΦ.

Σειριακή Σύνθεση

Στο εννοιολογικό μοντέλο, όταν χρειαζόμαστε να συνδυάσουμε πολλούς μετασχηματισμούς σε μία σχέση παροχής (όπως το συνδυασμό των $\mathbf{SK}_1$ και **γ** στο Παράδειγμα 1), εφαρμόζουμε μία σειριακή σύνθεση. Η απεικόνιση του σειριακού μετασχηματισμού στο λογικό μοντέλο είναι άμεση. Στο Σχήμα 4.6 παρατηρούμε μία σειριακή σύνθεση μεταξύ των δύο μετασχηματισμών $\mathbf{T}_1$ και $\mathbf{T}_2$. Αμφότεροι οι μετασχηματισμοί $\mathbf{T}_1$ και $\mathbf{T}_2$ απεικονίζονται στις διεργασίες $\mathbf{A}_1$ και $\mathbf{A}_2$, αντίστοιχα. Η σειριακή σύνθεση των δύο μετασχηματισμών αντιστοιχεί σε μία ακολουθία δύο διεργασιών $\mathbf{A}_1$ και $\mathbf{A}_2$ στο λογικό επίπεδο. Η σειρά εκτέλεσης των δύο διεργασιών εξαρτάται από τη σειρά των αντίστοιχων μετασχηματισμών στη σειριακή σύνθεση. Τα σχήματα των δύο διεργασιών ορίζονται όπως έχουμε δει σε προηγούμενη ενότητα. Η μόνη διαφορά που υπάρχει, είναι ότι το σχήμα εξόδου της αρχικής διεργασίας $\mathbf{A}_1$ θα τροφοδοτήσει το σχήμα εισόδου της επόμενης διεργασίας $\mathbf{A}_2$, δηλαδή: $\mathbf{A}_2$**.in=**$\mathbf{A}_1$**.out**.



Σχήμα 4.6: Σειριακή σύνθεση των μετασχηματισμών $\mathbf{T}_1$ και $\mathbf{T}_2$

4.2.4 Μετασχηματισμος Σχολίων

Μέχρι τώρα, έχουμε περιγράψει πως μπορούν να καθοριστούν τα σχήματα μίας διεργασίας και οι κατάλληλες αντιστοιχίσεις μεταξύ των γνωρισμάτων. Ένα άλλο πρόβλημα είναι η αναγνώριση της λειτουργικής σημασιολογίας μίας διεργασίας ΕΜΦ, δηλαδή του κατάλληλου προγράμματος LDL++ που περιγράφει τη λειτουργία μίας διεργασίας ΕΜΦ. Λόγω του μηχανισμού προσαρμογής, ο σχεδιαστής δεν είναι υπεύθυνος να γράψει τον απαιτούμενο κώδικα LDL++ για το πρόγραμμα αυτό από το μηδέν. Αντίθετα, πρέπει να διαλέξει μία πρότυπη διεργασία από την παλέτα που παρέχεται από το πλαίσιό μας. Η έκφραση μίας διεργασίας, η οποία βασίζεται σε ένα συγκεκριμένο απλό πρότυπο, παράγεται από ένα σύνολο κανόνων LDL++ της ακόλουθης μορφής, η οποία ονομάζεται *Πρότυπη Μορφή*:

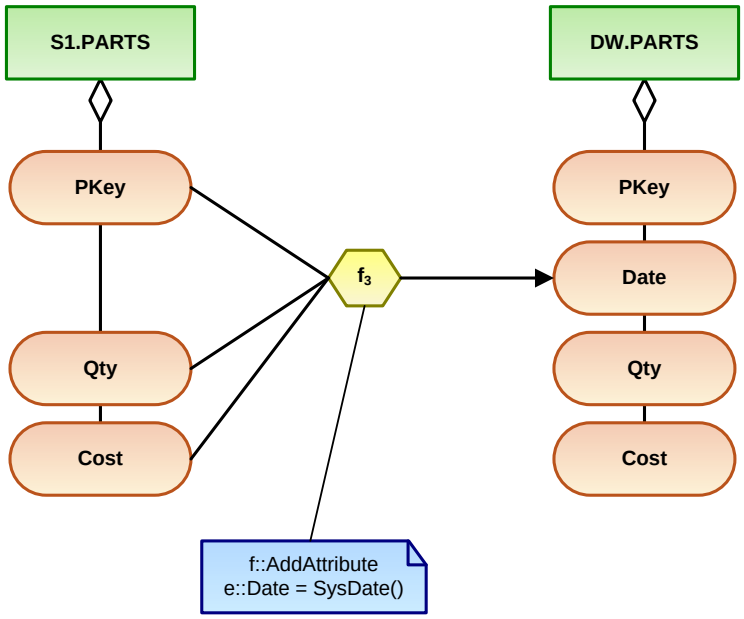
```
OUTPUT() <- INPUT(), EXPRESSION, MAPPING.
```
(*Γενική Μορφή Πρότυπης Διεργασίας*)

Συγκεκριμένα, πρέπει να περιγράψουμε (α) πως ο σχεδιαστής επιλέγει μία πρότυπη διεργασία, και (β) τι είναι το μέρος **EXPRESSION**, με δεδομένο ότι τα μέρη **OUTPUT**, **INPUT** και **MAPPING** καλύφθηκαν κατά την αντιστοίχηση ενός εννοιολογικού μετασχηματισμού σε μία λογική διεργασία.

Στο παράδειγμα 1 (Σχήμα 2.3) υπάρχουν διάφορα σχόλια προσαρτημένα σε κάποιους μετασχηματισμούς (π.χ. στον $\mathbf{f}_3$), με στόχο να κάνουν πιο κατανοητή τη λειτουργία τους. Όπως έχουμε δει στο εννοιολογικό μοντέλο, ένα σχόλιο αντιπροσωπεύει (α) το είδος ενός μετασχηματισμού, και/ή (β) μία έκφραση, και/ή (γ) απλό κείμενο που εξηγεί σχεδιαστικές αποφάσεις ή περιορισμούς. Οι τρεις αυτές κατηγορίες σχολίων χωρίζονται με τη χρήση ενός απλού προθέματος σε κάθε σχόλιο: (α) **f::** πριν από ένα είδος μετασχηματισμού, (β) **e::** πριν από κάποια έκφραση, και (γ) **t::** πριν από απλό κείμενο.

Η πληροφορία που ακολουθεί το πρόθεμα **f::** σε κάποιο κόμβο, καθορίζει την κατάλληλη πρότυπη διεργασία για το μετασχηματισμό στον οποίο είναι προσαρτημένο το σχόλιο αυτό, ενώ η αντίστοιχη

πληροφορία μετά το **e::** υποδηλώνει τις απαιτούμενες εκφράσεις για την πρότυπη διεργασία αυτή. Στο Σχήμα 4.7 απεικονίζεται ένα μέρος του Παραδείγματος 1 που αφορά μόνο το μετασχηματισμό $\mathbf{f}_3$. Ο μετασχηματισμός $\mathbf{f}_3$ αντιπροσωπεύει μία συνάρτηση και συνοδεύεται από ένα σχόλιο το οποίο περιέχει διπλή πληροφορία για: (α) το είδος (το όνομα του προτύπου) της συνάρτησης: **f::addAttribute**, και (β) την απαιτούμενη έκφραση (**EXPRESSION**) για το αντίστοιχο πρότυπο: **e::Date=SysDate()**.



Σχήμα 4.7: Το μέρος του Παραδείγματος 1 που αφορά μόνο το μετασχηματισμό $\mathbf{f}_3$

Στο Σχήμα 4.8 φαίνεται πως χρησιμοποιείται η πληροφορία αυτή για τη παραγωγή της λειτουργικής σημασιολογίας της διεργασίας $\mathbf{f}_3$. Χρησιμοποιούμε την **f::** πληροφορία για να διαλέξουμε το κατάλληλο πρότυπο για τη διεργασία στο λογικό μοντέλο. Έτσι, ο εννοιολογικός μετασχηματισμός $\mathbf{f}_3$ απεικονίζεται στη λογική διεργασία $\mathbf{f}_3$ της οποίας η λειτουργική σημασιολογία καθορίζεται από την πρότυπη διεργασία **addAttribute**. Η πρώτη γραμμή του Σχήματος 4.8 αντιπροσωπεύει το λεπτομερή ορισμό αυτής της πρότυπης διεργασίας. Ο μηχανισμός προτύπων που χρησιμοποιούμε είναι ένας μηχανισμός αντικατάστασης, ο οποίος βασίζεται σε μακροεντολές που διευκολύνουν την αυτόματη παραγωγή κώδικα LDL++. Μετά τη χρήση των μακροεντολών (2η γραμμή) παίρνουμε μια πιο ευκολονόητη έκδοση του προτύπου **addAttribute** (3η γραμμή). Τα τέσσερα τμήματα της Μορφής Προτύπου είναι πιο καθαρά σε αυτή την μορφή του προτύπου. Έτσι, το τμήμα **EXPRESSION** της **addAttribute** είναι **@OUTFIELD=@VALUE**, δηλαδή το νέο γνώρισμα (**OUTFIELD**) λαμβάνει μία αρχική τιμή (**VALUE**). Για το μετασχηματισμό $\mathbf{f}_3$, το τμήμα **EXPRESSION** είναι: **e::Date=SysDate()**. Μετά τη παραγωγή των παραμέτρων (4η γραμμή), παίρνουμε τη λειτουργική σημασιολογία της διεργασίας $\mathbf{f}_3$ (5η γραμμή). Περισσότερες λεπτομέρειες για το μηχανισμό προτύπων υπάρχουν στο [46].

Αν στον ορισμό του προτύπου χρησιμοποιούνται περισσότερες από μία δηλώσεις, δηλαδή το πρότυπο είναι βασισμένο σε κάποιο πρόγραμμα, τότε είναι δυνατό να απαιτούνται περισσότερες από μία εκφράσεις. Επίσης, ακόμα και σε πρότυπο με μία δήλωση, είναι δυνατό να έχουμε περισσότερες από μία εκφράσεις στον ορισμό του. Σε τέτοιες περιπτώσεις, το αντίστοιχο σχόλιο πρέπει παρέχει όλες

Πλήρης ορισμός Προτύπου **addAttribute**	`a_out( [i<arityOf(a_out)+1]{A_OUT_$i$,} @OUTFIELD ) <-`
	`  a_in1( [i<arityOf(a_in1)] {A_IN1_$i$,}`
	`  [i=arityOf(a_in1)] {A_IN1_$i$} ),`
	`  @OUTFIELD = @VALUE,`
	`  [i<arityOf(a_in1)] {A_OUT_$i$=A_IN1_$i$,}`
	`  [i=arityOf(a_in1)] {A_OUT_$i$=A_IN1_$i$}`

Μακροεντολές	`DEFINE INPUT_SCHEMA AS`
	`  [i<arityOf(a_in1)] {A_IN1_$i$,}`
	`  [i=arityOf(a_in1)] {A_IN1_$i$}`
	`DEFINE OUTPUT_SCHEMA AS`
	`  [i<arityOf(a_out)] {A_OUT_$i$,}`
	`  [i=arityOf(a_out)] {A_OUT_$i$}`
	`DEFINE DEFAULT_MAPPING AS`
	`  [i<arityOf(a_in1)] {A_OUT_$i$= A_IN1_$i$,}`
	`  [i=arityOf(a_in1)] {A_OUT_$i$= A_IN1_$i$}`

Πρότυπο	`a_out(OUTPUT_SCHEMA, @OUTFIELD) <-`
	`  a_in1(INPUT_SCHEMA),`
	`  @OUTFIELD = @VALUE,`
	`  DEFAULT_MAPPING.`

Απόδοση τιμών στις Παραμέτρους	`@OUTFIELD = DATE` `@VALUE = SYSDATE`

Λειτουργική Σημασιολογία της διεργασίας f_3	`f3_out(PKEY_out,QTY_out,COST_out,DATE) <-`
	`  f3_in1(PKEY_in,QTY_in,COST_in),`
	`  DATE = SYSDATE,`
	`  PKEY_out=PKEY_in,`
	`  QTY_out=QTY_in,`
	`  COST_out=COST_in.`

Σχήμα 4.8: Παραγωγή της λειτουργικής σημασιολογίας της διεργασίας $\mathbf{f}_3$

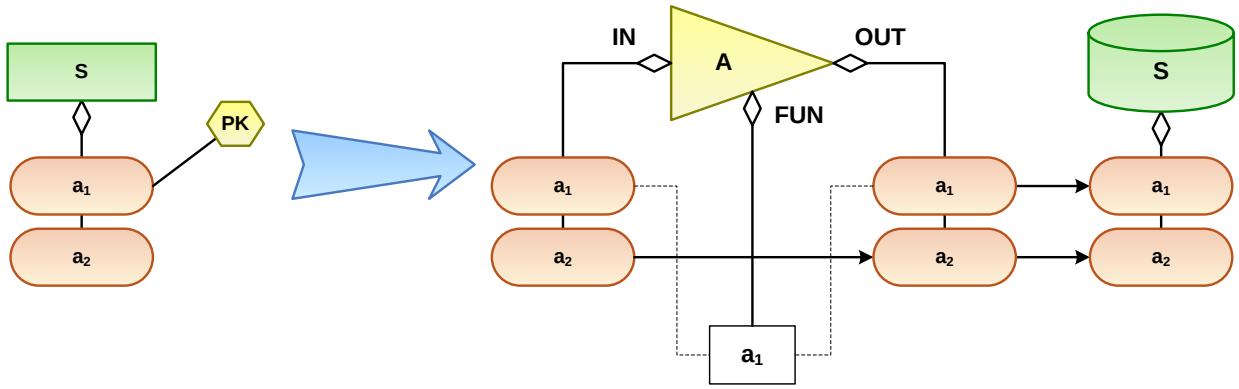
τις απαραίτητες εκφράσεις για να τοποθετηθούν στον ορισμό του προτύπου. Αν κάποιο σχόλιο δεν παρέχει όλες τις απαιτούμενες εκφράσεις, ή αν δεν τις παρέχει με την ορθή σειρά, τότε η πρότυπη διεργασία δεν μπορεί να κατασκευαστεί αυτόματα και απαιτείται παρέμβαση του χρήστη.

Τέλος, η πληροφορία **t::** δεν χρησιμοποιείται άμεσα στο λογικό μοντέλο. Η πληροφορία αυτή αξιοποιείται όσο είναι χρήσιμη στο λογικό σχεδιασμό, για να σχολιάζει τη λογική ροή έργου με ανεπίσημες παρατηρήσεις σε σχέση με διαφορετικές πτυχές της διεργασίας ΕΜΦ, όπως ο προγραμματισμός με βάση το χρόνο ή κάποιο γεγονός, η επίβλεψη, η καταγραφή συμβάντων, η αντιμετώπιση λαθών, η ανάνηψη από κατάρρευση, οι περιορισμοί χρόνου εκτέλεσης κ.λ.π..

4.2.5 Μετασχηματισμός Περιορισμών ΕΜΦ

Στο εννοιολογικό μοντέλο, οι περιορισμοί ΕΜΦ χρησιμοποιούνται για να δηλώσουν ότι τα δεδομένα κάποιας συγκεκριμένης έννοιας πληρούν διάφορες απαιτήσεις. Για παράδειγμα, στο παράδειγμα του Σχήματος 2.3 (Παράδειγμα 1), απαιτούμε ένα περιορισμό **PK** στα γνωρίσματα **PKEY** και **DATE** της έννοιας **DW.PARTS**. Η λειτουργικότητα ενός περιορισμού ΕΜΦ περιγράφεται σημασιολογικά από ένα μετασχηματισμό που υλοποιεί την επιβολή του περιορισμού. Έτσι, στην περίπτωση περιορισμών ΕΜΦ, εργαζόμαστε όπως και στις περιπτώσεις των εννοιολογικών μετασχηματισμών, και τους μετατρέπουμε σε λογικές διεργασίες.

Συγκεκριμένα, κάθε περιορισμός ΕΜΦ εφαρμόζεται σε κάποια γνωρίσματα μίας έννοιας $\mathbf{S}_i$ μετατρέπεται σε μία διεργασία **A** που συμβαίνει στη λογική ροή έργου *ακριβώς πριν* από το αντίστοιχο σύνολο εγγραφών $\mathbf{S}_i$ που αντιστοιχεί στην έννοια $\mathbf{S}_i$. Τα σχήματα της διεργασίας δημιουργούνται με τον ίδιο τρόπο και με τις ίδιες διαδικασίες όπως στην περίπτωση της αντιστοίχησης εννοιολογικών μετασχηματισμών. Το πεπερασμένο σύνολο γνωρισμάτων της $\mathbf{S}_i$, στα οποία εφαρμόζεται ο περιορισμός, αποτελούν το σχήμα λειτουργικότητας της **A**. Τα σχήματα εισόδου και εξόδου της **A** αποτελούνται από όλα τα γνωρίσματα της $\mathbf{S}_i$. Στο Σχήμα 4.9 υπάρχει μία αναπαράσταση της μετατροπής ενός περιορισμού ΕΜΦ σε λογικές διεργασίες, με βάση το παράδειγμα 1.



Σχήμα 4.9: Περιορισμοί ΕΜΦ

4.2.6 Ειδικές Περιπτώσεις

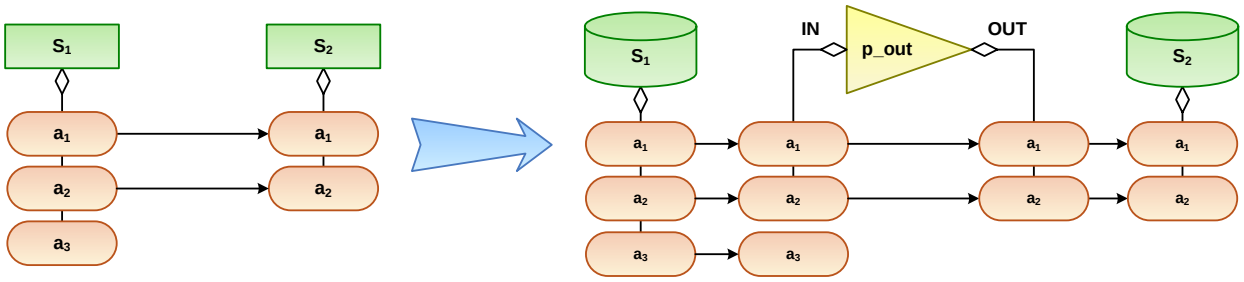
Σε αυτή την υποενότητα θα ασχοληθούμε με δύο σχεδιαστικά προβλήματα που προέκυψαν κατά την αντιστοίχηση του εννοιολογικού μοντέλου στο λογικού μοντέλου. Αρχικά, θα ασχοληθούμε με την περίπτωση των γνωρισμάτων που αποβάλλονται από τη ροή δεδομένων, και στη συνέχεια, θα μοντελοποιήσουμε τη σύγκλιση δύο διαφορετικών ροών δεδομένων σε μία κοινή πηγή δεδομένων.

Αποβολή ενός γνωρίσματος

Κατά τη διάρκεια μίας διεργασίας ΕΜΦ κάποια από τα αρχικά γνωρίσματα μπορεί να αποβληθούν από τη ροή. Ξεχωρίζουμε δύο διαφορετικές περιπτώσεις:

1. *Ένα γνώρισμα ανήκει στο σχήμα κάποιας έννοιας.* Αυτή η περίπτωση εμφανίζεται στο εννοιολογικό μοντέλο σαν ένα γνώρισμα από το οποίο δεν ξεκινά κάποια ακμή παροχής (ένα τέτοιο γνώρισμα είναι το **S_2.PARTS.DEPT**, στο Παράδειγμα 1, όπου δεν διαδίδεται προς το **DW**).

2. *Ένα γνώρισμα που λαμβάνει μέρος σε κάποιο μετασχηματισμό αλλά δεν διαδίδεται περισσότερο.* Αυτή η περίπτωση εμφανίζεται στο εννοιολογικό μοντέλο σαν ένα γνώρισμα που ανήκει στο σχήμα εισόδου κάποιου μετασχηματισμού, αλλά δεν υπάρχει μία ακμή εξόδου με το όνομα του γνωρίσματος που αποβλήθηκε από το μετασχηματισμό προς κάποιο από τα γνωρίσματα εξόδου (ένα τέτοιο γνώρισμα είναι το **PS_1.DEPT** το οποίο συμμετέχει στην εξωτερική συνένωση, αλλά δεν διαδίδεται προς το **S_1.PARTS**).



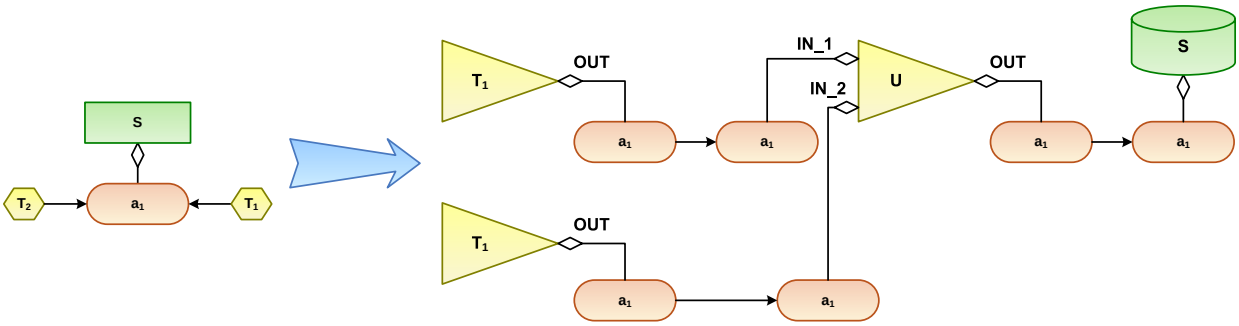
Σχήμα 4.10: Αποβολή ενός γνωρίσματος

Κατά τη μετάβαση από το εννοιολογικό στο λογικό μοντέλο, η πρώτη περίπτωση αντιμετωπίζεται με την προσθήκη μίας επιπλέον διεργασίας, η οποία αποβάλλει τα κατάλληλα γνωρίσματα. Η διεργασία αυτή πρέπει να τοποθετείται αμέσως μετά το αντίστοιχο σύνολο εγγραφών. Η όλη διαδικασία απεικονίζεται στο Σχήμα 4.10. Κάποιος θα μπορούσε να πει ότι η περίπτωση αυτή θα μπορούσε να αντιμετωπιστεί ακριβώς όπως και στο εννοιολογικό επίπεδο, δηλαδή το γνώρισμα αυτό να μη συνδέεται με μία ακμή παροχής προς κάποιο επόμενο γνώρισμα. Το αντεπιχείρημα σε αυτή την περίπτωση είναι ότι η λύση αυτή χρειάζεται λόγω του ότι διευκολύνει την αυτόματη (ανα-)κατασκευή όλων των σχημάτων της ροής έργου της διεργασίας ΕΜΦ στο λογικό μοντέλο (βλέπε [40]).

Σύγκλιση δύο ροών

Στο εννοιολογικό μοντέλο, η πλήρωση μίας έννοιας (π.χ. της Αποθήκης Δεδομένων) από περισσότερες από μία πηγές δηλώνεται αφηρημένα με ακμές παροχής που συνδέονται απλά στην έννοια αυτή. Στο λογικό μοντέλο, η διαδικασία αυτή είναι αυστηρότερη και χρειάζεται μία πιο ακριβή προσέγγιση για να καλυφτεί η πλήρωση ενός συνόλου εγγραφών από περισσότερες από μία πηγές. Μία λύση για το

θέμα αυτό είναι η προσθήκη μίας επιπλέον διεργασίας, η οποία θα ενοποιεί τις διάφορες ροές. Έτσι, η σύγκλιση δύο (ή περισσοτέρων) ροών αποτυπώνεται στο λογικό μοντέλο με τη χρήση μίας διεργασίας συνένωσης **U**.



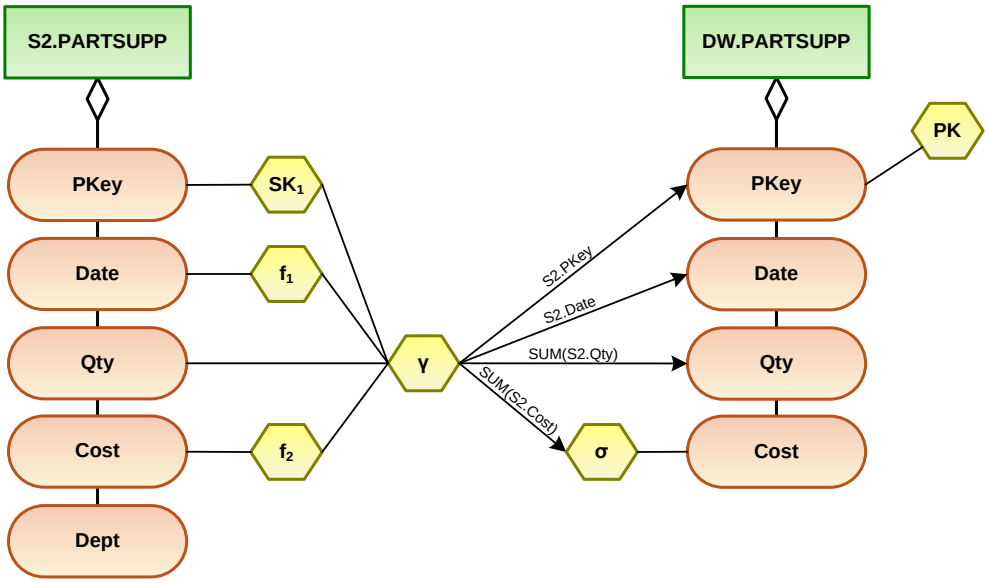
Σχήμα 4.11: Σύγκλιση δύο ροών

Το Σχήμα 4.11 απεικονίζει τη μετάβαση της σύγκλισης δύο (φυσικά, αυτό μπορεί να συμβεί και για περισσότερες) ροών στο λογικό μοντέλο. Το σχήμα εξόδου της τελευταίας διεργασίας κάθε ροής πληρώνει ένα σχήμα εισόδου της διεργασίας συνένωσης, δηλαδή $\mathbf{U.in_1{=}T_1.out}$ και $\mathbf{U.in_2{=}T_2.out}$. Προφανώς, πριν την προσθήκη της διεργασίας συνένωσης, κάθε μία από τις εμπλεκόμενες ροές πληρώνει την ίδια πηγή δεδομένων, δηλαδή το ίδιο σχήμα. Κατά συνέπεια, τα σχήματα εξόδου της διεργασίας σύγκλισης είναι ολόιδια. Με δεδομένο ότι μία διεργασία συνένωσης έχει κενά σχήματα λειτουργικότητας, παραγωγής και αποβολής, το σχήμα εξόδου της είναι επίσης ίδιο με κάθε ένα από τα σχήματα εισόδου και επιπλέον, είναι το ίδιο με το σχήμα του τελικού συνόλου εγγραφών. Για παράδειγμα, στο Σχήμα 4.11 ισχύουν οι ισότητες $\mathbf{U.out{=}T_1.out{=}T_2.out}$.

4.3 Σειρά Εκτέλεσης στη Λογική Ροή Έργου

Μέχρι τώρα, έχουμε ξεκαθαρίσει πως από το εννοιολογικό μοντέλο μίας διεργασίας ΕΜΦ, κάποιος μπορεί να (α) αναγνωρίσει τις εμπλεκόμενες πηγές δεδομένων, (β) περιγράψει τις απεικονίσεις μεταξύ των γνωρισμάτων, και (γ) να ονομάσει τους μετασχηματισμούς που χρειάζεται να λάβουν μέρος στην όλη διαδικασία. Μία ποιο λεπτομερής μελέτη της ροής δεδομένων δημιουργεί μερικά ερωτηματικά όσον αφορά τη σειρά εκτέλεσης των διεργασιών στη λογική ροή έργου. Ας υποθέσουμε το τμήμα του παραδείγματος 1 που απεικονίζεται στο Σχήμα 4.12. Για να σχεδιάσουμε μία λογική ροή έργου για το παράδειγμα αυτό, πρέπει να απαντήσουμε ερωτήματα όπως: "(Ε1:) Ποια από τις διεργασίες $\mathbf{SK_1}$ και $\mathbf{\gamma}$ πρέπει να προηγηθεί;" ή "(Ε2:) Ποια από τις διεργασίες $\mathbf{SK_1}$ και $\mathbf{f_1}$ πρέπει να προηγηθεί;"

Στην ενότητα αυτή παρουσιάζεται μία μέθοδος για τον καθορισμό της σειράς εκτέλεσης των διεργασιών κατά τη λογική ροή έργου. Αρχικά, χωρίς βλάβη της γενικότητας, εξετάζεται την απλή περίπτωση της πλήρωσης μίας έννοιας από μία άλλη. Έχοντας ως απώτερο στόχο την κατηγοριοποίηση των μετασχηματισμών με βάση τη θέση τους στο σχεδιάγραμμα, ορίζονται τα στάδια μετασχηματισμών και παρατίθεται ένας αλγόριθμος για την εύρεσή τους. Επίσης, ασχολούμαστε με θέματα που αφορούν



Σχήμα 4.12: Τμήμα του παραδείγματος 1 που αφορά την πλήρωση της **DW.PARTS** από τη $\mathbf{S_2}$**.PARTS**

τη σειρά των μετασχηματισμών που βρίσκονται στο ίδιο στάδιο. Στη συνέχεια, η μέθοδος αυτή επεκτείνεται για τη σύλληψη πιο πολύπλοκων και ρεαλιστικών περιπτώσεων, που περιλαμβάνουν δύο ή περισσότερες πηγές δεδομένων.

4.3.1 Επίπεδα

Αρχικά, υποθέστε την περίπτωση της απλής ροής "μία πηγή–ένας στόχος", όπως στο παράδειγμα του Σχήματος 4.12, με σκοπό τον ορισμό επιπέδων μετασχηματισμών στο εννοιολογικό σχεδιάγραμμα. Η ερώτηση E1 μπορεί να απαντηθεί με απλή παρατήρηση του Σχήματος 4.12. Με δεδομένο ότι ο μετασχηματισμός **γ** έχει γνωρίσματα εισόδου τα οποία ανήκουν στο σχήμα εξόδου των $\mathbf{SK}_1$, $\mathbf{f}_1$ και $\mathbf{f}_2$, πρέπει να εκτελείται μετά από αυτούς στο λογικό επίπεδο. Ομοίως, η διεργασία **ς** πρέπει να βρίσκεται μετά την **γ**. Επιπλέον, μετά την παρουσίαση των ειδικών περιπτώσεων στην υποενότητα 4.2.6, πρέπει να λάβουμε υπόψη μας δύο ακόμη διεργασίες: (α) τον περιορισμό ΕΜΦ **PK**—θυμίζουμε ότι όταν μετασχηματίζουμε ένα περιορισμό ΕΜΦ σε μία διεργασία, πρέπει να τον τοποθετούμε αμέσως πριν το σύνολο εγγραφών στο οποίο αντιστοιχεί (**DW.PARTS**)—και (β) την "κρυμμένη" διεργασία που αποβάλλει το γνώρισμα **DEPT**—θυμίζουμε ότι αυτή η διεργασία θα πρέπει να τοποθετηθεί αμέσως μετά την αρχική έννοια (**$\mathbf{S_2}$.PARTS**).

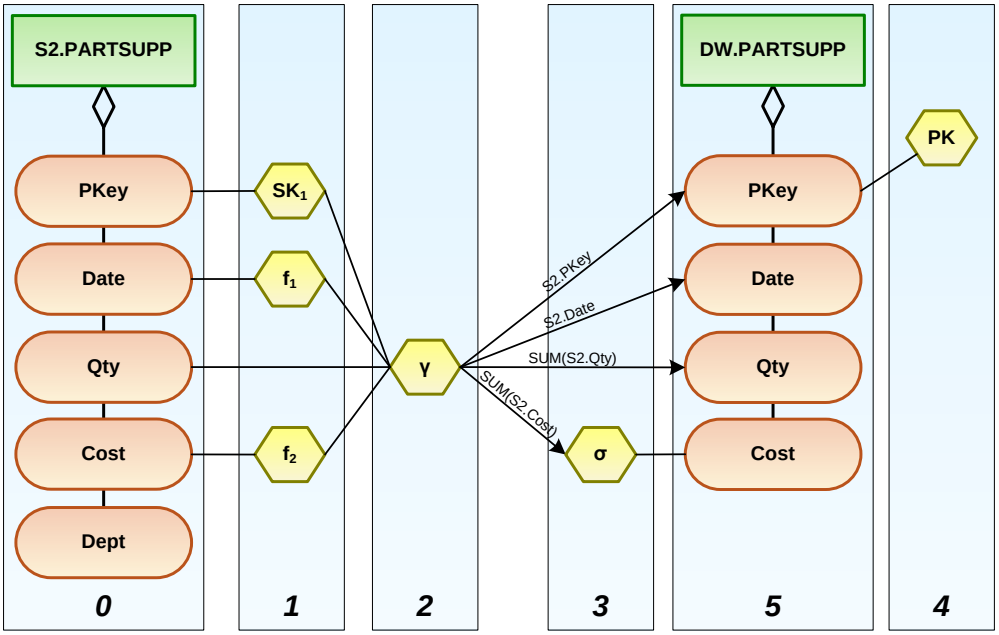
Αμέσως μετά, διαισθητικά, κάποιος μπορεί να χωρίσει το σχεδιάγραμμα σε διάφορα σύνολα μετασχηματισμών (Σχήμα 4.13) βασιζόμενος κυρίως στις εξής παρατηρήσεις:

1. η θέση ορισμένων διεργασιών στη ροή έργου μπορεί εύκολα να καθοριστεί παρατηρώντας απλά τους παροχείς και τους καταναλωτές τους,

2. μερικές διεργασίες μπορούν να τοποθετηθούν εύκολα στη ροή έργου, εφαρμόζοντας απλώς τους

κανόνες απεικόνισης των περιορισμών ΕΜΦ και τις ειδικές περιπτώσεις της υποενότητας 4.2.6.

Οι παραπάνω παρατηρήσεις μας οδηγούν στο χωρισμό του διαγράμματος σε διάφορα *επίπεδα μετασχηματισμών*. Ένα *επίπεδο μετασχηματισμού* (ή *επίπεδο*) είναι μία οπτική περιοχή του εννοιολογικού διαγράμματος που υποδεικνύει τους μετασχηματισμούς που περιέχονται σε αυτή την περιοχή. Ένα επίπεδο εμφανίζεται σαν ένα παραλληλόγραμμο, αριθμημένο στο κάτω μέρος του με ένα μοναδικό χαρακτηριστικό αριθμό: ένα ακέραιο αριθμό, αυτόματα αυξανόμενο, με αρχική τιμή ίση με μηδέν, όπου το επίπεδο μηδέν αποτελείται μόνο από την αρχική έννοια.



Σχήμα 4.13: Επίπεδα μετασχηματισμών

Υποθέστε δύο εννοιολογικούς μετασχηματισμούς $\mathbf{T}_i$ και $\mathbf{T}_j$. Τότε, ο καθορισμός της σειράς εκτέλεσης των αντίστοιχων διεργασιών $\mathbf{A}_i$ και $\mathbf{A}_j$ στο λογικό μοντέλο επιτυγχάνεται ως εξής: Όταν ο $\mathbf{T}_i$ ανήκει σε ένα επίπεδο με μικρότερο αριθμό από ότι το επίπεδο του $\mathbf{T}_j$, τότε η διεργασία $\mathbf{A}_i$ έχει χαμηλότερη προτεραιότητα εκτέλεσης (δηλαδή εκτελείται πριν) από τη διεργασία $\mathbf{A}_j$.

Μέχρι τώρα, έχουμε απαντήσει στο πρώτο ερώτημα (Ε1), αλλά όχι στο δεύτερο (Ε2). Έτσι, εκκρεμεί ακόμη το πρόβλημα του καθορισμού της σειράς εκτέλεσης μεταξύ των μετασχηματισμών που ανήκουν στο ίδιο επίπεδο.

Οι μετασχηματισμοί που ανήκουν στο ίδιο επίπεδο, ονομάζονται *μετασχηματισμοί ισοδύναμου επιπέδου*.

Καταπιανόμαστε με αυτό το πρόβλημα, με την ακόλουθη σκέψη: Αν όλες οι διεργασίες προέρχονται από μετασχηματισμούς ισοδύναμου επιπέδου, μπορούν να αλλάξουν σειρά, τότε δεν υπάρχει πρόβλημα με τη σειρά τους. Κατά συνέπεια, απαντούμε σε ερωτήματα που αφορούν τη σειρά διεργασιών αυτού του είδους, μελετώντας ποιες από αυτές τις διεργασίες μπορούν να αλλάξουν θέση μεταξύ τους. Για τις υπόλοιπες διεργασίες, απαιτείται επιπλέον πληροφορία από το χρήστη.

Στο [40] οι συγγραφείς ασχολούνται με το πότε δύο διεργασίες μπορούν να αλλάξουν θέση στη λογική ροή έργου, δηλαδή πότε μπορούμε να ανταλλάξουμε τις προτεραιότητες εκτέλεσής τους. Στο [40] υπάρχει μία αυστηρή απόδειξη ότι η αντιστροφή δύο διεργασιών $\mathbf{A}_1$ και $\mathbf{A}_2$ επιτρέπεται όταν ισχύουν οι συνθήκες:

1. $\mathbf{A}_1$ και $\mathbf{A}_2$ είναι συνεχόμενες στο γράφο· χωρίς βλάβη της γενικότητας, υποθέστε ότι η $\mathbf{A}_1$ είναι παροχέας της $\mathbf{A}_2$
2. τόσο η $\mathbf{A}_1$ όσο και η $\mathbf{A}_2$ έχουν κοινά σχήματα εισόδου και εξόδου και το σχήμα εξόδου τους περιλαμβάνει μόνο ένα καταναλωτή
3. το σχήμα λειτουργικότητας της $\mathbf{A}_1$ και της $\mathbf{A}_2$ είναι υποσύνολο του σχήματος εισόδου τους, τόσο πριν όσο και μετά την αλλαγή θέσης
4. τα σχήματα εισόδου των $\mathbf{A}_1$ και $\mathbf{A}_2$ δεν αποτελούν υποσύνολα των παροχέων τους, τόσο πριν όσο και μετά την αλλαγή θέσης

Εξ ορισμού, όλοι οι μετασχηματισμοί ισοδύναμου επιπέδου μπορεί να είναι συνεχόμενοι στο γράφο, και αυτό ισχύει επίσης και για τις αντίστοιχες διεργασίες· έτσι η πρώτη συνθήκη ικανοποιείται. Με δεδομένο ότι τα επίπεδα ορίζονται μεταξύ δύο εννοιών, σε ένα συγκεκριμένο επίπεδο, όλοι οι μετασχηματισμοί, όπως και οι αντίστοιχες διεργασίες τους, έχουν ένα σχήμα εισόδου και ένα σχήμα εξόδου· έτσι ικανοποιείται και η δεύτερη συνθήκη. Επομένως, πρέπει να εξετάσουμε αν ικανοποιούνται οι συνθήκες (3) και (4), για να αποφασίσουμε αν δύο μετασχηματισμοί που ανήκουν στο ίδιο επίπεδο μπορούν να αλλάξουν σειρά ή όχι.

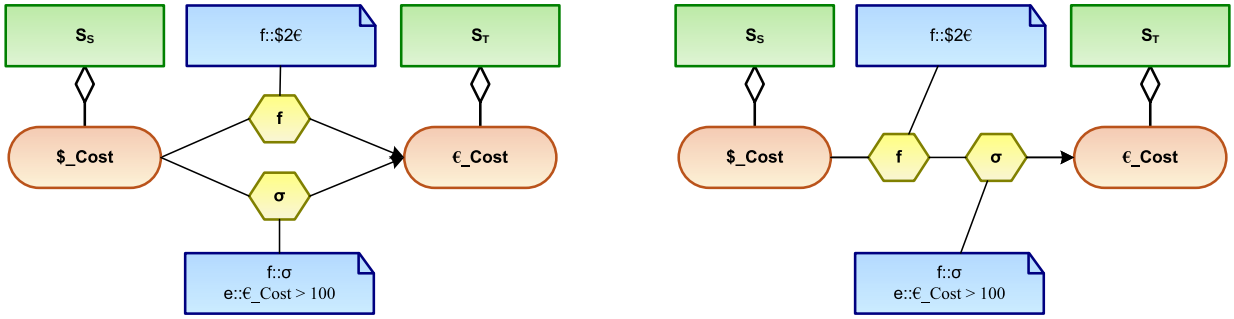
Οι μετασχηματισμοί που ανήκουν στο ίδιο επίπεδο, και των οποίων οι αντίστοιχες διεργασίες στο λογικό διάγραμμα μπορούν να αλλάξουν θέση, ονομάζονται *μετασχηματισμοί ισοδύναμης σειράς*. Για δύο μετασχηματισμοί ισοδύναμης σειράς $\mathbf{A}_i$ και $\mathbf{A}_{i-1}$ ισχύουν οι ακόλουθες σχέσεις:

$$\begin{aligned} \mathtt{A_i.fun} &\subseteq \mathtt{A_i.in} \\ \mathtt{A_i.in} &\subseteq \mathtt{A_{i-1}.out} \end{aligned}$$

Οι παραπάνω σχέσεις αναπαριστούν τις συνθήκες (3) και (4) αντίστοιχα. Αν αυτές οι δύο σχέσεις δεν ισχύουν για δύο μετασχηματισμούς ισοδύναμου επιπέδου, τότε δεν μπορούμε να αποφασίσουμε αυτόματα για τη σειρά τους και χρειαζόμαστε επιπλέον πληροφορία από το σχεδιαστή.

Προφανώς, αυτή η επιπλέον πληροφορία χρειάζεται μόνο σε περιπτώσεις κακού σχεδιασμού. Όπως επισημαίνεται στο [46] (κατ' επέκταση και στο Κεφάλαιο 2), δεν παρέχεται κάποια αυστηρή μέθοδος επαλήθευσης του εννοιολογικού μοντέλου. Αυτή η απόφαση στηρίζεται στην επιλογή των συγγραφέων του [46] για ένα απλό μοντέλο που έχει ως κύριο στόχο την αναγνώριση των πηγών δεδομένων και των μετασχηματισμών που εμπλέκονται στην συνολική διεργασία ΕΜΦ. Το μοντέλο αυτό προορίζεται

όχι μόνο προς διαχειριστές, αλλά και προς διευθυντές και ανθρώπους με χαμηλή εξειδίκευση στις Αποθήκες Δεδομένων, με σκοπό να βοηθήσει αυτές τις ομάδες ανθρώπων να καταλάβουν η μία την άλλη χρησιμοποιώντας μία κοινή σχεδιαστική διάλεκτο. Επομένως, επιτρέπονται περιπτώσεις κακού εννοιολογικού σχεδιασμού, με δεδομένο ότι όταν το παραδοτέο αυτού του σταδίου μεταβεί στο επόμενο επίπεδο (στο λογικό διάγραμμα) θα συμπληρωθεί και θα διορθωθεί όπου αυτό είναι αναγκαίο.



Σχήμα 4.14: Ένα παράδειγμα (α) κακού και (β) καλού σχεδιασμού

Ας δούμε τώρα το παράδειγμα του Σχήματος 4.14. Στο Σχήμα 4.14(α) μπορούμε να παρατηρήσουμε δύο μετασχηματισμούς ισοδύναμου επιπέδου: **f** και σ. Υποθέστε ότι η έννοια $\mathbf{S}_S$ περιέχει τιμές σε δολάρια. Ο πρώτος μετασχηματισμός μετατρέπει τις τιμές από δολάρια σε ευρώ, ενώ το δεύτερο φίλτρο, αφαιρεί τις τιμές που είναι μικρότερες από **100**€. Προφανώς, αν το φίλτρο έχει υψηλότερη προτεραιότητα από τη συνάρτηση, τότε θα είχαμε σημασιολογικό πρόβλημα· το αποτέλεσμα του φιλτραρίσματος δολαρίων με το κατώφλι σε ευρώ δεν είναι εφικτό. Παράλληλα, όπως αναφέρεται αναλυτικά στο [40], η συνθήκη (3) αποτυγχάνει: όταν το φίλτρο εφαρμόζεται πρώτο, τότε σ.**fun={€_COST}**$\neq$ σ.**in={$COST}**. Έτσι, οι δύο αυτοί μετασχηματισμοί δεν είναι ισοδύναμης σειράς και ο σχεδιαστής θα ερωτηθεί για την ορθή σειρά. Αν ο σχεδιαστής δώσει το εννοιολογικό διάγραμμα του Σχήματος 4.14(β), τότε η σειρά είναι ορισμένη επακριβώς, διότι οι δύο μετασχηματισμοί ανήκουν σε διαφορετικά επίπεδα.

4.3.2 Ορισμός των Επιπέδων

Σε αυτή την ενότητα θα παρουσιάσουμε τον αλγόριθμο **FS** (Σχήμα 4.15 που χρησιμοποιείται για τον αυτόματο καθορισμό όλων των σταδίων ενός εννοιολογικού διαγράμματος που περιλαμβάνει μία έννοια-πηγή και μία έννοια-στόχο. Διεκπεραιώνει τις ακόλουθες εργασίες: (α) στην αρχή βρίσκει όλες τις απαραίτητες αποβολές γνωρισμάτων, όπως αυτές ορίζονται στην ενότητα 4.2.6, και (β) στη συνέχεια, χωρίζει το διάγραμμα σε αρκετά στάδια, εκμεταλλευόμενος τις παρατηρήσεις για τους παροχείς και τους καταναλωτές κάθε μετασχηματισμού.

Ως είσοδο, ο αλγόριθμος **FS** παίρνει έναν υπογράφο $\mathbf{G_p} = (\mathbf{V}', \mathbf{E}')$ του πλήρους εννοιολογικού γράφου $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, δηλαδή $\mathbf{V}' \subseteq \mathbf{V}$ και $\mathbf{E}' \subseteq \mathbf{E}$, που αποτελείται από μία έννοια-πηγή $\mathbf{S}_S$, μία έννοια-στόχο $\mathbf{S}_T$, τα γνωρίσματά τους, και όλους τους μετασχηματισμούς μεταξύ των δύο αυτών εννοιών. Στον αλγόριθμο αυτό, θεωρούμε όλους τους μετασχηματισμούς του $\mathbf{G_p}$ σαν ένα σύνολο **AT**, η δημιουργία

του οποίου θεωρείται τετριμμένη. Ως έξοδο, ο αλγόριθμος **FS** επιστρέφει ένα πίνακα από σύνολα **Stage()**. Κάθε στοιχείο **id** του πίνακα **stage(id)** είναι ένα σύνολο που περιέχει τους μετασχηματισμούς που ανήκουν στο επίπεδο με id ίσο με **id**. Σημειώστε όταν τοποθετούμε ένα μετασχηματισμό απόρριψης στο επίπεδο **Stage()**, δεν είναι πραγματικός μετασχηματισμός του εννοιολογικού μοντέλου, αλλά τον χρησιμοποιούμε ως εικονικό για να σημειώσουμε μία τοποθεσία στη ροή δεδομένων.

Όλες οι περιπτώσεις γνωρισμάτων που απορρίπτονται υπολογίζονται στην αρχή (Γρ. 7–10). Μας ενδιαφέρουν μόνο τα γνωρίσματα της έννοιας-πηγής (βλέπε υποενότητα 4.2.6 που δεν έχουν κάποια σχέση παροχής. Όλοι οι μετασχηματισμοί απόρριψης γνωρισμάτων τοποθετούνται στο ίδιο στάδιο (Γρ: 9), καθώς, προφανώς είναι μετασχηματισμοί ισοδύναμης σειράς. Στη συνέχεια, ο αλγόριθμος **FS** υπολογίζει τους υπόλοιπους μετασχηματισμούς (Γρ. 11–17). Αν όλοι οι παροχείς ενός συγκεκριμένου μετασχηματισμού ανήκουν σε προηγούμενα επίπεδα, δηλαδή σε επίπεδα με μικρότερο **id** από αυτό του παρόντος επιπέδου, τότε ο μετασχηματισμός πρέπει να τοποθετηθεί στο παρόν επίπεδο (Γρ. 14–17).

Algorithm Find Stages (FS)

1. **Input:** A graph $G_P=(V',E')$, where V' contains a source concept C_S, a target concept C_T, their attributes, and a set **AT** of all transformations between C_S and C_T.
2. **Output:** An array $Stage_{C_S,C_T}[id]$, $id=0,...,n$, where n is the number of all possible stages between C_S and C_T.
3. **Begin**
4. `id = 0;`
5. `Stage[id] = {`C_S`};` // source concept is placed in stage 0
6. `id++;`
7. **for each** attribute $a_j \in C_S$.out **{**
8. **if** ($\forall x \in (V'-C_S), \neg$ $\exists$edge(a_j,x)) **{**
9. `Stage[id] = {`π-out$_{a_j}$`};` // all project-out's are placed in the same stage
10. **}}**
11. **while** ($AT \neq \emptyset$) **{**
12. `id++;` // a new id for a new stage
13. **for each** transformation $T_i \in AT$ **{**
14. **if** ($\forall a_j \in T_i$.in, $\exists$ node $V \in$ Stage[id-k],k=1,...,id, s.t. $a_j \in V$.out) **{**
15. **AT** = **AT** − {T_i};
16. `Stage[id] = Stage[id]` $\cup$ {T_i}; // add transformation T_i into current stage
17. **}}}**
18. `Stage[++id] = {`C_T`};` // target concept is placed in the last stage
19. `return Stage[];` // return all stages
20. **End.**

Σχήμα 4.15: Ο αλγόριθμος **FS**

Παράδειγμα. Στη συνέχεια θα εφαρμόσουμε τον **FS** σε ένα τμήμα του Παραδείγματος 1, που απεικονίζεται στο Σχήμα 4.13. Ας θεωρήσουμε την τροφοδοσία της Αποθήκης Δεδομένων **DW.PARTS** από την

πηγή δεδομένων **S2.PARTS**. Σε αυτή την περίπτωση, έχουμε $\mathbf{C}_S$**=S2.PARTS** και $\mathbf{S}_T$**=DW.PARTS**. Το σύνολο **AT** όλων των μετασχηματισμών μεταξύ των δύο εννοιών είναι $\mathbf{AT} = \{\mathtt{SK}, \gamma, \mathtt{f_1}, \mathtt{f_2}, \sigma\}$. Θυμηθείτε ότι η χρήση ενός συνόλου καταργεί κάθε λογής σειρά μεταξύ των μετασχηματισμών. Στο Σχήμα 4.16 παρουσιάζεται ένα παράδειγμα εκτέλεσης του αλγορίθμου για την περίπτωση αυτή. Επίσης, σημειώστε ότι στο Σχήμα 4.13 δεν υπάρχουν αποβολές γνωρισμάτων.

	id	**T_i**	**T_i.in**	**a_j**	**Stage[id-k] k = 1, …, id**	**2nd if cond.**	**AT**	**Stage[]**
1	0	-	{}	-	{}	-	{SK,γ,f_1,f_2,σ}	Stage[0]={S2.PARTS}
2	1	*	*	DEPT	{S2.PARTS}	-	{SK,γ,f_1,f_2,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$}
3	2	SK	{PKEY}	PKEY	{S2.PARTS, π-out$_{DEPT}$}	TRUE	{γ,f_1,f_2,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK}
4	2	γ	{SK.out, QTY,f_1.out, f_2.out}	SK.out	{S2.PARTS, π-out$_{DEPT}$}	FALSE	{γ,f_1,f_2,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK}
5	2	f_1	{DATE}	DATE	{S2.PARTS, π-out$_{DEPT}$}	TRUE	{γ,f_2,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1}
6	2	f_2	{COST}	COST	{S2.PARTS, π-out$_{DEPT}$}	TRUE	{γ,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2}
7	2	σ	{γ.out}	γ.out	{S2.PARTS, π-out$_{DEPT}$}	FALSE	{γ,σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2}
8	3	γ	{SK.out, QTY,f_1.out, f_2.out}	**	{S2.PARTS, π-out$_{DEPT}$, SK,f_1,f_2}	TRUE	{σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2} Stage[3]={γ}
9	3	σ	{γ.out}	γ.out	{S2.PARTS, π-out$_{DEPT}$, SK,f_1,f_2}	FALSE	{σ}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2} Stage[3]={γ}
10	4	σ	{γ.out}	γ.out	{S2.PARTS, π-out$_{DEPT}$, SK,f_1,f_2, γ}	TRUE	{}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2} Stage[3]={γ} Stage[4]={σ}
11	5	-	{}	-	{}	-	{}	Stage[0]={S2.PARTS} Stage[1]={π-out$_{DEPT}$} Stage[2]={SK,f_1,f_2} Stage[3]={γ} Stage[4]={σ} Stage[5]={DW.PARTS}

Για οικονομία χώρου, δεχόμαστε ότι:

* Παρουσιάζουμε μόνο το στάδιο που αφορά το γνώρισμα **DEPT** που αποβάλλεται από τη ροή

** Παραλείπουμε τα στάδια για τα γνωρίσματα του **γ**. Είναι προφανές ότι όλα τα γνωρίσματα του **γ.in** ικανοποιούν τη 2η συνθήκη-if (Γρ. 14) του αλγόριθμου **FS**, δηλαδή κάθε γνώρισμα του **γ.in** ανήκει στο σχήμα εξόδου κάποιου μετασχηματισμού που περιλαμβάνεται στο **Stage(id), id=1,2**

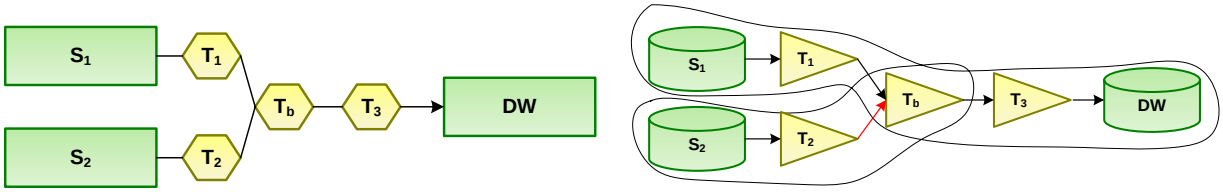
Σχήμα 4.16: Ένα παράδειγμα εκτέλεσης του **FS** για το εννοιολογικό διάγραμμα του Σχήματος 4.13

4.3.3 Στάδια σε πιο Πολύπλοκα Διαγράμματα

Μέχρι τώρα έχουμε ασχοληθεί με απλές διεργασίες "μία πηγή–ένας στόχος". Αυτό αναθεωρείται λαμβάνοντας υπόψη μας τους δυαδικούς μετασχηματισμούς που συνδυάζουν περισσότερες από μία ροές. Σε αυτή την περίπτωση ακολουθούμε μία τριπλή διαδικασία για την εύρεση της σειράς εκτέλεσης των διεργασιών στη γενική ροή έργου: (α) υπολογίζουμε τα στάδια για κάθε μία ανεξάρτητη ροή "μία πηγή–ένας στόχος", (β) κατασκευάζουμε της γραμμικές λογικές ροές έργου, και (γ) τις ενοποιούμε σε μία ροή έργου. Η ένωση των δύο ροών έργου επιτυγχάνεται στον κοινό τους κόμβο, δηλαδή είτε σε κάποιο σύνολο εγγραφών είτε σε κάποιο δυαδικό μετασχηματισμό, που ονομάζεται *σημείο σύνδεσης* των δύο ροών έργου. Όπως έχουμε ήδη συζητήσει στην υποενότητα 4.2.6, αν το σημείο σύνδεσης είναι μία πηγή δεδομένων, τότε για τη σύγκλιση των δύο ροών έργου θα χρειαστούμε μία επιπλέον διεργασία συνένωσης (U) που πρέπει να τοποθετηθεί ακριβώς πριν από το σημείο σύνδεσης, δηλαδή την κοινή πηγή δεδομένων. Αν το σημείο σύνδεσης είναι ένας δυαδικός μετασχηματισμός τότε απλώς ενώνουμε τις δύο ροές σε αυτό τον κόμβο.

Αρχικά δίνουμε μία διαισθητική περιγραφή πως αντιμετωπίζουμε τα πολύπλοκα διαγράμματα που περιλαμβάνουν περισσότερες από μία πηγές, και στη συνέχεια παρουσιάζουμε έναν αλγόριθμο για την αυστηρή εύρεση των σταδίων σε τέτοια πολύπλοκα διαγράμματα.

Χωρίς βλάβη της γενικότητας, μπορούμε να υποθέσουμε την περίπτωση ροής "δύο πηγές–ένας στόχος". Προφανώς, για κάθε δυαδικό μετασχηματισμό $\mathbf{T}_b$ υπάρχουν δύο παροχείς και ένας καταναλωτής. Αυτές οι τρεις οντότητες μπορεί να είναι είτε έννοιες είτε μετασχηματισμοί ανάλογα με τη θέση του $\mathbf{T}_b$ στη ροή έργου. Υποθέστε την περίπτωση της πλήρωσης μίας πηγής δεδομένων **DW** από δύο πηγές δεδομένων $\mathbf{S}_1$ και $\mathbf{S}_2$ (Σχήμα 4.17(α)) μέσω μερικών μοναδιαίων μετασχηματισμών και ενός δυαδικού μετασχηματισμού $\mathbf{T}_b$.



(α) (β)

Σχήμα 4.17: Η περίπτωση του δυαδικού μετασχηματισμού (α) στο εννοιολογικό και (β) στο λογικό μοντέλο

Μπορούμε να χρησιμοποιήσουμε μία αυξητική τεχνική για το σχεδιασμό της λογικής ροής έργου. Αρχικά, υπολογίζουμε τα $\mathbf{Stage}_{S1,DW}$ και $\mathbf{Stage}_{S2,DW}$ που περιέχουν τα επίπεδα των ροών "μία πηγή–ένας στόχος" που ορίζονται από τα δύο ζεύγη εννοιών $\mathbf{S}_1$**-DW** και $\mathbf{S}_2$**-DW**. Για να βρούμε την κατάλληλη σειρά στη λογική ροή έργου όταν περιλαμβάνεται ένας δυαδικός μετασχηματισμός $\mathbf{T}_b$, επιλέγουμε (α) όλους τους μετασχηματισμούς που λαμβάνουν μέρος στη ροή από την πρώτη έννοια $\mathbf{S}_1$ στην έννοια–στόχο **DW**, και (β) το υποσύνολο των μετασχηματισμών της δεύτερης ροής που αποτελείται από όλους τους μετασχηματισμούς από την έννοια $\mathbf{S}_2$ μέχρι και τον δυαδικό μετασχηματισμό $\mathbf{T}_b$. Έτσι, από το

επίπεδο **Stage**$_{S1,DW}$ χρησιμοποιούμε όλα τα στοιχεία του:

$$\texttt{Stage}_{\texttt{S1,DW}}[\texttt{i}],\ \ \texttt{i} = 0, \ldots, \texttt{n} \qquad \text{όπου:} \quad \texttt{Stage}_{\texttt{S1,DW}}[\texttt{n}] = \{\texttt{DW}\}$$

ενώ από το επίπεδο **Stage**$_{S2,DW}$ χρησιμοποιούμε μόνο τα στοιχεία:

$$\texttt{Stage}_{\texttt{S2,DW}}[\texttt{i}],\ \ \texttt{i} = 0, \ldots, \texttt{k} \qquad \text{όπου:} \quad \texttt{Stage}_{\texttt{S2,DW}}[\texttt{n}] = \{\texttt{T}_\texttt{b}\}$$

Για παράδειγμα, στο Σχήμα 4.17 έχουμε:

Για $\texttt{S}_1 - \texttt{DW}$:	Για $\texttt{S}_2 - \texttt{DW}$:
$\texttt{Stage}_{\texttt{S1,DW}}[0] = \{\texttt{S}_1\}$	$\texttt{Stage}_{\texttt{S2,DW}}[0] = \{\texttt{S}_2\}$
$\texttt{Stage}_{\texttt{S1,DW}}[1] = \{\texttt{T}_1\}$	$\texttt{Stage}_{\texttt{S2,DW}}[1] = \{\texttt{T}_2\}$
$\texttt{Stage}_{\texttt{S1,DW}}[2] = \{\texttt{T}_\texttt{b}\}$	$\texttt{Stage}_{\texttt{S2,DW}}[2] = \{\texttt{T}_\texttt{b}\}$
$\texttt{Stage}_{\texttt{S1,DW}}[3] = \{\texttt{T}_3\}$	$\texttt{Stage}_{\texttt{S2,DW}}[3] = \{\texttt{T}_3\}$
$\texttt{Stage}_{\texttt{S1,DW}}[4] = \{\texttt{DW}\}$	$\texttt{Stage}_{\texttt{S2,DW}}[4] = \{\texttt{DW}\}$

Στο παράδειγμα αυτό, **n=4** και **k=2**, κι έτσι, από τη δεύτερη ροή επιλέγουμε μόνο τα πρώτα τρία επίπεδα (0..2). Προφανώς, δεν υπολογίζουμε όλα τα επίπεδα της δεύτερης ροής, αλλά υπολογίζουμε μόνο τα πρώτα **k** επίπεδα (**0..k-1**). Κατασκευάζουμε τη λογική ροή **S**$_1$**-DW** βασιζόμενοι και στα πέντε επίπεδα (0..4) και τη λογική ροή **S**$_2$**-DW** βασιζόμενοι στα πρώτα τρία επίπεδα (0..2). Αμφότερες οι ροές φαίνονται στο Σχήμα 4.17(β). Τέλος, ενοποιούμε τις δύο ροές στο σημείο σύνδεσης τους με τη διεργασία **T**$_b$.

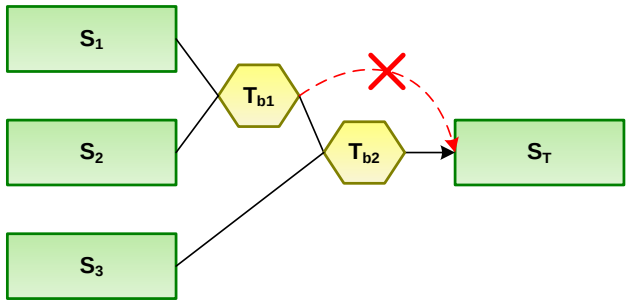
Η βεβαιότητα ότι με αυτή την τεχνική, τίποτα δεν χάνεται, υποστηρίζεται με το ακόλουθο θεώρημα:

Θεώρημα 4.3.1 *Αν δύο ροές,* ***S***$_i$*-****S***$_T$*,* ***i=1,2****, που τροφοδοτούν μία πηγή δεδομένων* ***S***$_T$ *από δύο πηγές δεδομένων* ***S***$_1$ *και* ***S***$_2$ *έχουν ένα δυαδικό μετασχηματισμό* ***T***$_b$ *ως σημείο σύνδεσης, τότε η υπο-ροή* ***T***$_b$*-****S***$_T$ *είναι κοινή στις δύο αρχικές ροές.*

Απόδειξη. *Η απόδειξη είναι προφανής καθώς εξ ορισμού, κάθε μετασχηματισμός/διεργασία έχει ένα και μόνο ένα σχήμα εξόδου.*

Έτσι, η κατάσταση που απεικονίζεται στο Σχήμα 4.18 δεν είναι αποδεκτή, διότι ο μόνος τρόπος να εμφανιστούν διαφορετικές ροές μετά τον δυαδικό μετασχηματισμό είναι να επιτραπεί η ύπαρξη διαφορετικών σχημάτων εξόδου, κάτι το οποίο απαγορεύεται.

Μετά από αυτή τη διαισθητική περιγραφή, παρουσιάζουμε ένα αλγόριθμο για την *Εύρεση Επιπέδων σε Πολύπλοκα Εννοιολογικά Διαγράμματα.* Ο αλγόριθμος **FSC** (Σχήμα 4.19) ορίζει αυστηρά τη σειρά



μη αποδεκτή κατάσταση:

$S_1 - S_T : S_1, \ldots, T_{b1}, \ldots, T_{b2}, \ldots, S_T$

$S_2 - S_T : S_2, \ldots, T_{b1}, \ldots, S_T$

$S_3 - S_T : S_3, \ldots, T_{b2}, \ldots, S_T$

αποδεκτή κατάσταση:

$S_1 - S_T : S_1, \ldots, T_{b1}, \ldots, T_{b2}, \ldots, S_T$

$S_2 - S_T : S_2, \ldots, T_{b1}, \ldots, T_{b2}, \ldots, S_T$

$S_3 - S_T : S_3, \ldots, T_{b2}, \ldots, S_T$

Σχήμα 4.18: Προβληματικό Εννοιολογικό Διάγραμμα

εκτέλεσης των διεργασιών σε ροές έργου που περιέχουν περισσότερες από μία πηγές. Ελέγχει όλες τις ροές από κάθε πηγή προς την Αποθήκη Δεδομένων. Αν η ροή είναι *απλή*, όταν δηλαδή δεν περιέχεται δυαδικός μετασχηματισμός, τότε απλά εφαρμόζει τον αλγόριθμο **FS** στη ροή αυτή. Αν η ροή είναι *σύνθετη*, όταν δηλαδή περιέχεται τουλάχιστον ένας δυαδικός μετασχηματισμός, τότε κάθε φορά καταγράφει τους δυαδικούς μετασχηματισμούς, και στη συνέχεια, εφαρμόζει ξανά τον αλγόριθμο **FS**.

Σε ένα πιο λεπτομερές επίπεδο, ο αλγόριθμος **FSC** παίρνει ως είσοδο ένα κατευθυνόμενο γράφο που αντιπροσωπεύει ένα εννοιολογικό διάγραμμα που πιθανόν να περιλαμβάνει περισσότερες από μία πηγές με δυαδικούς μετασχηματισμούς. Ελέγχει όλες τις πιθανές ροές που μπορούν να προκύψουν από δύο οποιεσδήποτε έννοιες (Γρ. 6). Είναι προφανές από την παραπάνω διαισθητική ανάλυση ότι βασικός στόχος είναι η εύρεση των ορίων κάθε ροής. Προφανώς, ψάχνουμε μόνο για τον στόχο κάθε ροής, με δεδομένο ότι η πηγή είναι πάντα γνωστή. Αρχικά, σε κάθε ροή, πηγή είναι η πρώτη έννοια (Γρ. 7). Αν δεν υπάρχουν δυαδικοί μετασχηματισμοί σε αυτή τη ροή, τότε στόχος είναι η δεύτερη έννοια. Αντιμετωπίζουμε την περίπτωση να υπάρχει κάποιος δυαδικός μετασχηματισμός T_b με διπλό τρόπο (Γρ. 9-14). Την πρώτη φορά που βρίσκουμε το μετασχηματισμός T_b, απλώς τον προσθέτουμε σε ένα σύνολο με το όνομα `visited_`T_b και η σειρά του ορίζεται από τη συνολική ροή. Κατά την επόμενή του εμφάνιση (ή εμφανίσεις αν υποθέσουμε ότι ένας δυαδικός μετασχηματισμός μπορεί να έχει πέραν των δύο σχημάτων εισόδου) του T_b, η δεύτερη έννοια δεν θεωρείται πλέον ως ο στόχος της τρέχουσας ροής· σε αυτή την περίπτωση, χρησιμοποιούμε το δυαδικό μετασχηματισμό T_b σαν στόχο (Γρ. 11). Αυτό συμβαίνει διότι η ροή από το μετασχηματισμό T_b μέχρι τη δεύτερη έννοια έχει ήδη καθοριστεί προηγουμένως, στην πρώτη εμφάνιση του T_b. Το σύνολο `visited_`T_b χρησιμοποιείται για να καταχωρούνται οι δυαδικοί μετασχηματισμοί που έχουν ήδη εμφανιστεί.

Μετά την απόφαση για το στόχο, πρέπει να καθορίσουμε τα όρια της ροής που πρέπει να ελεγχθούν για να αποφασιστεί η ροή εκτέλεσης των λογικών διεργασιών. Για ακόμη μία φορά, αυτό επιτυγχάνεται μέσω της εφαρμογής του αλγορίθμου **FS** στη ροή μεταξύ της αρχικής έννοια και του επιλεγμένου

στόχου (Γρ. 17). Ο αλγόριθμος **FS** επιστρέφει ως έξοδο τον πίνακα Stage[] (βλέπε προηγούμενη υποενότητα) που περιέχει την απαραίτητη πληροφορία για τη σειρά εκτέλεσης της συγκεκριμένης ροής. Όλοι οι πίνακες Stage[] όλων των εννοιολογικών ροών που εξετάζονται τοποθετούνται σε ένα άλλο πίνακα, τον LW[].

Όταν ο αλγόριθμος **FSC** ολοκληρώσει την εκτέλεσή του, επιστρέφει τον πίνακα LW[] που περιέχει όλες τις ανεξάρτητες απλές ροές που πρέπει να χρησιμοποιηθούν για την κατασκευή της λογικής ροής έργου (Γρ. 19). Εδώ πρέπει να σημειωθεί ότι ο όρος "κατασκευή" αναφέρεται στον καθορισμό της τοποθέτησης (με σειρά εκτέλεσης) όλων των πηγών δεδομένων και των μετασχηματισμών που λαμβάνουν μέρος στη διεργασία ΕΜΦ, παρά στην σύνθεση όλων των συστατικών του λογικού μοντέλου.

Algorithm Find Stages in Complex Conceptual Designs (FSC)

1. **Input:** A graph **G**=(**V,E**) that represents the whole conceptual design
2. **Output:** An array LW[id], id=0,...,n, where n is the number of all possible flows among concepts
3. **Begin**
4. visited_T_b = {};
5. id = 0;
6. **for each** pair C_S,C_T **{**
7. Target = C_T;
8. **AT** = all_transformations(C_S,C_T); //find all transformations involved into the flow of C_S and C_T
9. **for each** binary transformation T_b∈**AT {**
10. **if** (T_b∈visited_T_b) **{**
11. Target = T_b; //if we have already counted T_b then we need only the flow from source to T_b
12. **}**
13. **else {**
14. visited_T_b = visited_T_b ∪ {T_b};
15. **}**
16. **}**
17. LW[++id] = FS(**G'**$_{Cs-CT}$(**V',E'**), s.t. **G'**$_{Cs-CT}$ is the simple flow between C_S and Target);
18. **}**
19. return LW[];
20. **End.**

Σχήμα 4.19: Ο αλγόριθμος **FSC**

Σημειώστε ότι όταν αναφερόμαστε στο Stage[] ή στο LW[] χρησιμοποιούμε τους όρους "έννοια", "πηγή δεδομένων", "σύνολο εγγραφών", "μετασχηματισμός" ή "διεργασία" χωρίς καμία διάκριση, καθώς ό,τι τοποθετείται σε αυτές τις δύο δομές δεδομένων, Stage[] ή LW[], είναι απλά μία θέση για την την πραγματική οντότητα στην οποία αναφέρεται· θυμηθείτε ότι στην υποενότητα 4.2, οι έννοιες και οι μετασχηματισμοί απεικονίζονται σε σύνολα εγγραφών και διεργασίες αντίστοιχα, αλλά τα ονόματά τους παραμένουν αναλλοίωτα.

4.3.4 Σειρά Εκτέλεσης Διεργασιών

Μέχρι τώρα, έχουμε εισάγει την έννοια των σταδίων, και έχουμε παρουσιάσει πως αυτά μπορούν να αναγνωριστούν στο εννοιολογικό διάγραμμα. Είμαστε τώρα έτοιμοι να περιγράψουμε πως η σειρά εκτέλεσης των διεργασιών στο λογικό διάγραμμα μπορεί να καθοριστεί.

Συνήθως, οι διεργασίες ΕΜΦ αποτελούνται από περισσότερες από μία ροές "μία πηγή–ένας στόχος". Έτσι, γενικά, πρέπει να να ακολουθούμε την τεχνική που περιγράψαμε στην προηγούμενη υποενότητα για την εύρεση της σειράς εκτέλεσης των διεργασιών που λαμβάνουν μέρος στην όλη διαδικασία. Αρχικά, βρίσκουμε την κατάλληλη σειρά εκτέλεσης για κάθε απλή ροή στο εννοιολογικό διάγραμμα, έτσι ώστε να εξασφαλίσουμε την κατάλληλη τοποθέτηση των διεργασιών σε γραμμικές λογικές ροές έργου. Κάθε μία από αυτές τις ροές έργου αντιστοιχεί σε ένα στοιχείο του πίνακα `LW[]` που παρουσιάζεται στην υποενότητα 4.3.3.

Η σειρά εκτέλεσης των διεργασιών στη λογική ροή έργου αποφασίζεται με χρήση του αλγόριθμου **EOLW**. Ο αλγόριθμος **EOLW** (Σχήμα 4.20) παίρνει ως είσοδο τον πίνακα `LW[]` που υπολογίστηκε στην προηγούμενη υποενότητα. Προφανώς, βασικός του στόχος είναι η κατασκευή του λογικού διαγράμματος, με όρους "γεμίσματος" θέσεων, παρά της δημιουργίας της όλης διεργασίας ΕΜΦ με τη σημασιολογία της· αυτό είναι δουλεία που θα γίνει αργότερα. Έτσι, δημιουργεί και επιστρέφει ένα γράφο **G(V,E)** που περιέχει όλες τις απαραίτητες πηγές δεδομένων και διεργασίες.

Η διαδικασία έχει ως ακολούθως. Για κάθε απλή ροή, δηλαδή για κάθε στοιχείο του `LW[]`, ο αλγόριθμος **EOLW** επεξεργάζεται όλα τα στάδια μετασχηματισμών, δηλαδή τα στάδια του `Stage[]` (Γρ. 7–20). Κάθε φορά που βρίσκει ένα νέο κόμβο, είτε πηγή δεδομένων είτε διεργασία, τον προσθέτει στο γράφο μαζί με μία ακμή που συνδέει τον κόμβο αυτό με τον προηγούμενό του στη ροή (Γρ. 9–13). Αν ένας κόμβος έχει ήδη προστεθεί στο γράφο τότε έχουμε περίπτωση σύγκλισης δύο ροών: ο παρόν κόμβος και ο αυτός που έχει ήδη προστεθεί στο γράφο (Γρ. 14). Ξεχωρίζουμε δύο πιθανές περιπτώσεις: ο κόμβος είτε είναι μία δυαδική διεργασία ή μία πηγή δεδομένων, δηλαδή ανήκει στο **RS** ή στο **A**. Η περίπτωση μίας διεργασίας την οποία έχουμε ήδη επισκεφτεί είναι η απλούστερη· τότε το μόνο που χρειαζόμαστε είναι μία σύνδεση από τον προηγούμενο κόμβο της παρούσας ροής στην δυαδική διεργασία (Γρ. 15). Η δεύτερη περίπτωση απαιτεί τη χρήση μίας επιπλέον διεργασίας Συνένωσης (βλέπε υποενότητα 4.2.6) τοποθετημένης αμέσως πριν από την αντίστοιχη πηγή δεδομένων (Γρ. 16–20). Αρχικά, διαγράφουμε την ακμή μεταξύ της πηγής δεδομένων και του τελευταίου κόμβου της ροής που έχει ήδη συνδεθεί σε αυτή (Γρ. 17). Στη συνέχεια, προσθέτουμε ένα νέο κόμβο που αντιπροσωπεύει την διεργασία Συνένωσης (Γρ. 18) και τον συνδέουμε με τις δύο ροές και την πηγή δεδομένων (Γρ. 19).

Algorithm Execution Order in Logical Workflows (EOLW)

```
1.  Input: An array LW[id], id=0,...,n, where n is the number of all possible flows among concepts
2.  Output: A graph G=(V,E) that represents the whole logical design
3.  Begin
4.    for each LW[i] in LW {
5.      previous = LW[i].Stage[0]; // at first, previous is equal to the source concept of the flow
6.      V = V ∪ {previous};
7.      for each Stage[j] in LW[i], j>0 { // for all stages, but the first one
8.        for each node n∈Stage[j] { // for each node (i.e., data store or activity) in the stage
9.          if (n∉V) { // if this is the first appearance of the node
10.           V = V ∪ {n}; // add it to the design
11.           E = E ∪ {previous,n}; // connect it with the previous one in the flow
12.           previous = n; // now it is the old one
13.         }
14.         if (n∈V) { // if the node has already been added in the design
15.           if (n∈A) { E = E ∪ (previous,n); } // if the node is an activity, then it is a binary one
16.           if (n∈RS) { // if it is a recordset
17.             E = E − {x,n}, s.t. x∈V ∧ ∃(x,n)∈E; // delete the connection to the first flow
18.             V = V ∪ {U}; // add a Union activity
19.             E = E ∪ {previous,U} ∪ {x,U} ∪ {U,n}; // add the new edges to connect the two flows with
                                                      // the target recordset through a Union activity
20.   }}}}
21.   return G(V,E); // return the graph that represents the logical design
22. End.
```

Σχήμα 4.20: Ο αλγόριθμος **EOLW**

4.4 Μεθοδολογία Μετάβασης από Εννοιολογικό σε Λογικό Μοντέλο

Σε αυτή την ενότητα, θα παρουσιαστεί συνοπτικά την ακολουθία βημάτων που ακολουθεί ένας σχεδιαστής, κατά τη μετάβαση από το εννοιολογικό στο λογικό μοντέλο. Όπως έχουμε ήδη αναφέρει, ο απώτερος στόχος αυτής της σχεδιαστικής διαδικασίας είναι η δημιουργία μίας απεικόνισης μεταξύ των δύο μοντέλων, μαζί με κάθε σχετική βοηθητική πληροφορία. Επίσης, κάθε στάδιο αυτής της μεθοδολογίας θα παρουσιαστεί και θα εξηγηθεί με τη βοήθεια του παραδείγματος 1 (Σχήμα 2.3).

Βήμα 1ο: Προετοιμασία. Αρχικά, βελτιώνουμε το εννοιολογικό μοντέλο έτσι ώστε να μην υπάρχουν αμφιβολίες για τίποτα. Πρώτα, οι (ενεργοί) υποψήφιοι δεν πρέπει να απεικονίζονται άμεσα στο λογικό μοντέλο. Αφού αποφασίσουμε ποιος θα είναι ο ενεργός υποψήφιος, δημιουργείται ένα αντίγραφο του σεναρίου που αφαιρεί όλους τους υποψηφίους. Όπως έχουμε δει στην υποενότητα 2.2, υπάρχουν δύο τρόποι για την αφαίρεση ενός ενεργού υποψηφίου από το εννοιολογικό διάγραμμα (α) αγνοώντας όλη

την πληροφορία που αφορά τους υποψηφίους για την έννοια-στόχο και (β) αντικαθιστώντας την έννοια στόχο με τον ενεργό υποψήφιο. Κατά συνέπεια, οι υποψήφιες πηγές δεδομένων αφαιρούνται από το διάγραμμα και ο ενεργός υποψήφιος (αυτός που έχει επιλεγεί τελικά) μετατρέπεται σε εννοιολογική έννοια και στη συνέχεια, σε λογική διεργασία.

Επιπλέον, κάθε επιπρόσθετη πληροφορία που απεικονίζεται με τη μορφή σχολίου, όπως για παράδειγμα, περιορισμοί χρόνου εκτέλεσης, σχόλια κ.λ.π., τα οποία δεν χρειάζονται κατά τη απεικόνιση, αφαιρούνται από το διάγραμμα, αλλά καταγράφονται σε κάποιο αρχείο συμβάντων.

Βήμα 2ο: Έννοιες και Γνωρίσματα. Στη συνέχεια, αρχίζουμε την κατασκευή του λογικού διαγράμματος. Ξεκινούμε προσθέτοντας όλα τα απαραίτητα σύνολα εγγραφών μαζί μετα γνωρίσματά τους. Στην υποενότητα 4.2.1, είδαμε ότι όλες οι έννοιες του εννοιολογικού διαγράμματος απεικονίζονται σε σύνολα εγγραφών στο λογικό διάγραμμα. Επίσης, τα γνωρίσματά τους απεικονίζονται σε γνωρίσματα των αντίστοιχων συνόλων εγγραφών. Έτσι, η σχέσεις μέρους παραμένουν αναλλοίωτες μετά τη μετάβαση από το ένα μοντέλο στο άλλο.

Βήμα 3ο: Μετασχηματισμοί. Αμέσως μετά, είμαστε έτοιμοι να κατασκευάσουμε το βασικό τμήμα μίας διεργασίας ΕΜΦ στο λογικό επίπεδο: τις διαδικασίες καθαρισμού και ολοκλήρωσης των δεδομένων, τις διεργασίες. Αρχικά, καθορίζουμε ποιες είναι οι κατάλληλες διεργασίες και ποια είναι η σειρά εκτέλεσής τους. Το πρώτο είναι ήδη έτοιμο από το εννοιολογικό μοντέλο: όλοι οι μετασχηματισμοί που υπάρχουν στο εννοιολογικό διάγραμμα, απεικονίζονται σε λογικές διεργασίες. Επιπλέον, αναγνωρίζουμε τις διεργασίες εκείνες που δεν είναι φαίνονται στο εννοιολογικό διάγραμμα, όπως την αποβολή κάποιου γνωρίσματος ή τη σύγκλιση δύο ροών (βλέπε υποενότητα 4.2.6).

Στη συνέχεια, βρίσκουμε τη σειρά εκτέλεσης των διεργασιών. Λαμβάνοντας υπόψη ότι οι διεργασίες ΕΜΦ αποτελούνται, συνήθως, από πολλές σύνθετες ροές, ακολουθούμε την τεχνική που περιγράφεται στην υποενότητα 4.3.4. Μετά τη δημιουργία του γράφου που αντιπροσωπεύει το λογικό διάγραμμα, το εμπλουτίζουμε με την κατάλληλη σημασιολογία. Χρησιμοποιώντας την τεχνική που περιγράψαμε στην υποενότητα 4.2.4, εκμεταλλευόμαστε την πληροφορία που μας παρέχεται από τα σχόλια, για να αποτυπώσουμε πλήρως τη σημασιολογία κάθε διεργασίας.

Βήμα 4ο: Περιορισμοί ΕΜΦ. Το επόμενο βήμα ασχολείται με τους περιορισμούς ΕΜΦ που επιβάλλονται στις πηγές δεδομένων. Οι περιορισμοί ΕΜΦ απεικονίζονται σε λογικές διεργασίες με τρόπο όμοιο με αυτό των μετασχηματισμών (βλέπε υποενότητα 4.2.5). Θυμηθείτε ότι όταν μετασχηματίζουμε έναν περιορισμό ΕΜΦ σε μία διεργασία, τότε πρέπει να την τοποθετήσουμε αμέσως μετά το σύνολο εγγραφών στο οποίο εφαρμόζεται. Έτσι, στο βήμα αυτό, εμπλουτίζουμε το λογικό διάγραμμα με επιπλέον διεργασίες που αντιπροσωπεύουν όλους τους περιορισμούς ΕΜΦ του εννοιολογικού διαγράμματος. Η σημασιολογία αυτών των διεργασιών προσδιορίζεται με ίδιο τρόπο με αυτή της απεικόνισης ενός μετασχηματισμού σε μία διεργασία: με τη χρήση επιπλέον πληροφορίας που περιέχεται σε σχόλια προσαρτημένα στους περιορισμούς ΕΜΦ. Σε ότι αφορά τη σειρά εκτέλεσης αυτών

των νέων διεργασιών, δηλώνουμε ότι τα κριτήρια της σειράς εκτέλεσης είναι ίδια με αυτά που ισχύουν σε περίπτωση μετασχηματισμών ισοδύναμου επιπέδου.

Βήμα 5ο: Παραγωγή Σχημάτων. Όσον αφορά τα σχήματα των διεργασιών, αναφέρουμε ότι εκτός από αυτά που αναφέρθηκαν στην ενότητα 4.2, στο [40], παρουσιάζεται ένας αλγόριθμος για την αυτόματη δημιουργία όλων των σχημάτων, όλων των διεργασιών του λογικού διαγράμματος. Συνοπτικά, για δύο συνεχόμενες διεργασίες $\mathtt{A}_1$ και $\mathtt{A}_2$, ισχύει:

$$\begin{aligned} \mathtt{A}_2.\mathtt{in} &= \mathtt{A}_1.\mathtt{out} \\ \mathtt{A}_2.\mathtt{out} &= \mathtt{A}_2.\mathtt{in} \cup \mathtt{A}_2.\mathtt{gen} - \mathtt{A}_2.\mathtt{pro} \end{aligned}$$

Δηλαδή, το σχήμα εισόδου μίας διεργασίας είναι ίδιο με το σχήμα εξόδου του παροχέα της, ενώ το σχήμα εξόδου της, ισούται με το σχήμα εισόδου ένωση με το σχήμα παραγωγής μείον το σχήμα απόρριψης.

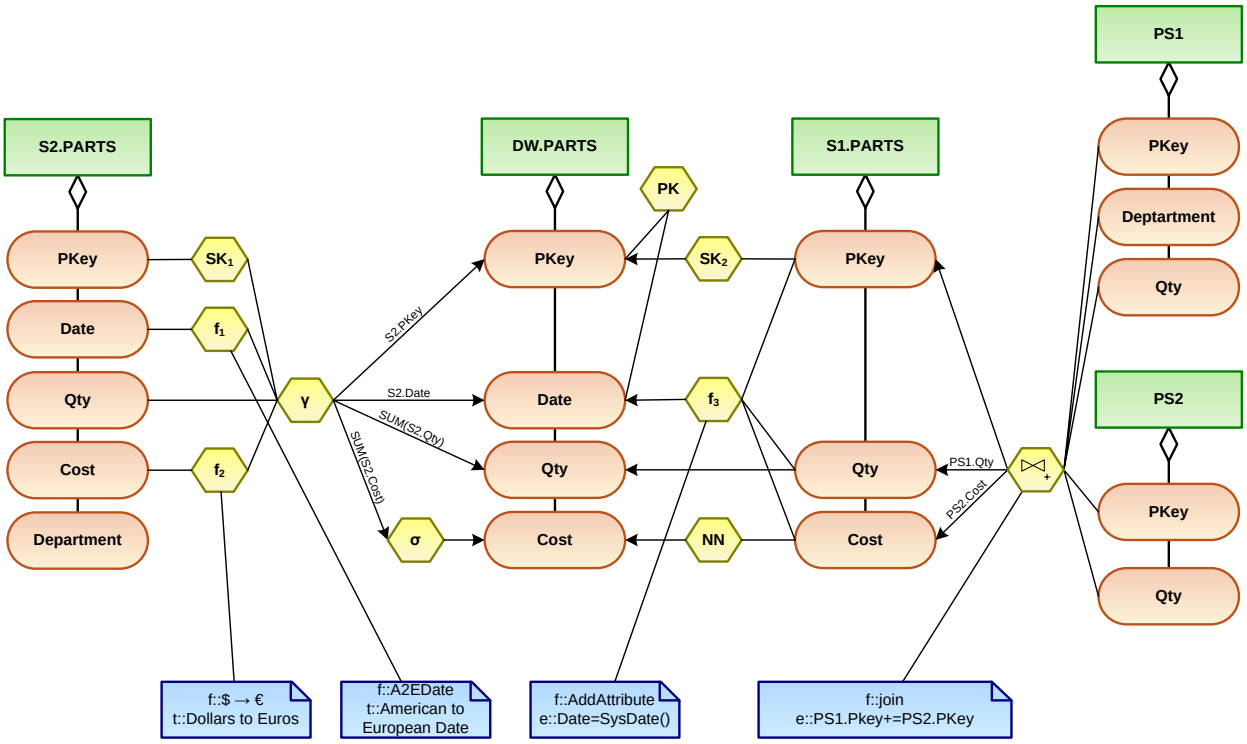
Τέλος, επισημαίνουμε ξανά, ότι λόγω του ότι δεν υπάρχει αυστηρός έλεγχος στο εννοιολογικό μοντέλο, η μεθοδολογία αυτή δεν αποτελεί μία πλήρως αυτοματοποιημένη διαδικασία. Σε μερικές περιπτώσεις, υπάρχει η ανάγκη για επιπλέον πληροφορία από το σχεδιαστή/διαχειριστή. Επίσης, το τελικό παραδοτέο δεν είναι πάντα πλήρες και ακριβές λογικό διάγραμμα. Ο σχεδιαστής/διαχειριστής θα πρέπει να το εξετάσει, να το συμπληρώσει ή να αλλάξει, για να επιτύχει το σκοπό του.

4.4.1 Ένα Παράδειγμα Εφαρμογής

Στη συνέχεια παρουσιάζουμε τη μετάβαση από το εννοιολογικό διάγραμμα του Παραδείγματος 1 (Σχήμα 2.3) στο λογικό μοντέλο, ακολουθώντας τα βήματα της παραπάνω μεθοδολογίας.

Βήμα 1ο: Προετοιμασία. Το πρώτο βήμα απαιτεί την αφαίρεση όλων των υποψηφίων. Έτσι, υποθέτοντας ότι έχει επιλεγεί η έννοια **RecentParts** σαν ενεργός υποψήφιος, τότε αγνοούμε όλη την πληροφορία από τους άλλους υποψηφίους για την έννοια στόχο, **S2.PARTS**) και αφαιρούμε όλους τους υποψηφίους καθώς και τις αντίστοιχες σχέσεις από το διάγραμμα. Επίσης, η πληροφορία που δεν είναι χρήσιμη για τη μετάβαση, αφαιρείται από το διάγραμμα, αλλά καταγράφεται σε ένα αρχείο συμβάντων. Στο Σχήμα 4.21 απεικονίζεται το εννοιολογικό διάγραμμα του Παραδείγματος 1 (Σχήμα 2.3), αμέσως μετά το 1ο Βήμα.

Βήμα 2ο: Έννοιες και Γνωρίσματα. Στη συνέχεια, συνεχίζουμε με την αναγνώριση των εννοιών που εμφανίζονται στο διάγραμμα και ξεκινάμε την κατασκευή του λογικού διαγράμματος. Έτσι, προσθέτουμε τα κατάλληλα σύνολα εγγραφών, μαζί με τα γνωρίσματά τους. Το αποτέλεσμα αυτής της διαδικασίας, παρουσιάζεται στο Σχήμα 4.22.



Σχήμα 4.21: Το εννοιολογικό διάγραμμα του Παραδείγματος 1, μετά το πρώτο βήμα

Βήμα 3ο: Μετασχηματισμοί. Στη συνέχεια, εμπλουτίζουμε το διάγραμμα με πληροφορία που αφορά τους μετασχηματισμούς που περιλαμβάνονται στην όλη διεργασία ΕΜΦ. Αρχικά, αποφασίζουμε ποιες είναι οι κατάλληλες διεργασίες. Όλοι οι μετασχηματισμοί απεικονίζονται σε λογικές διεργασίες. Επίσης, μπορεί να υπάρχουν διεργασίες που δεν εμφανίζονται ως μετασχηματισμοί στο εννοιολογικό διάγραμμα, όπως αυτές που αποβάλλουν κάποια γνωρίσματα ή της σύγκλισης δύο ροών. Στο Παράδειγμα 1, έχουμε αναγνωρίσει τους ακόλουθους μετασχηματισμούς, που θα πρέπει να απεικονιστούν σε λογικές διεργασίες:

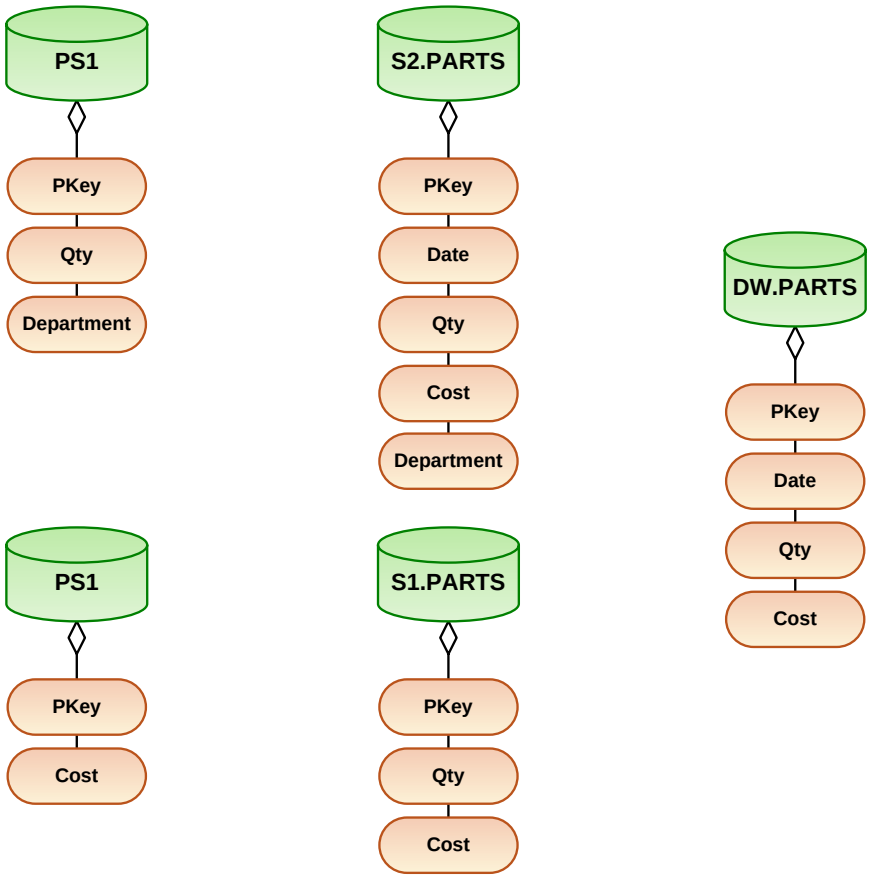
$$\mathtt{SK}_1,\ \mathtt{f}_1,\ \mathtt{f}_2,\ \gamma,\ \sigma,\ \mathtt{SK}_2,\ \mathtt{f}_3,\ \mathtt{NN},\ \bowtie_+$$

Επιπλέον, υπάρχουν άλλες δύο διεργασίες που πρέπει να προστεθούν στο διάγραμμα, μία για την περιγραφή της απόρριψης του γνωρίσματος **Department** από την έννοια **S2.PARTS** και μία για την περιγραφή της σύγκλισης των ροών **S1.PARTS-DW.PARTS** και **S2.PARTS-DW.PARTS**:

$$\pi\_\mathtt{out} \text{ και } \mathtt{U}$$

Αμέσως μετά, θα πρέπει να βρούμε την κατάλληλη θέση γι' αυτές τις διεργασίες. Αυτή η διεργασία ΕΜΦ αποτελείται από τέσσερις ροές: δύο απλές (**S1.PARTS-DW.PARTS** και **S2.PARTS-DW.PARTS**) και δύο σύνθετες (**PS1-S1.PARTS** και **PS2-S1.PARTS**).

Η εφαρμογή του αλγορίθμου **FSCW** μας παρέχει τον πίνακα `LW[]` για το παράδειγμα αυτό:

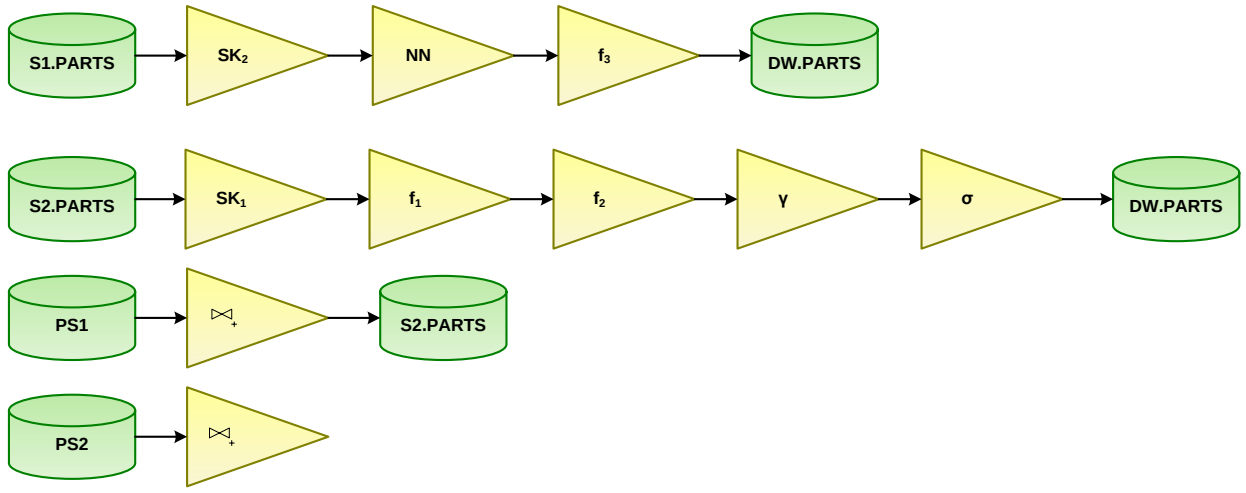


Σχήμα 4.22: Το αποτέλεσμα του δεύτερου βήματος

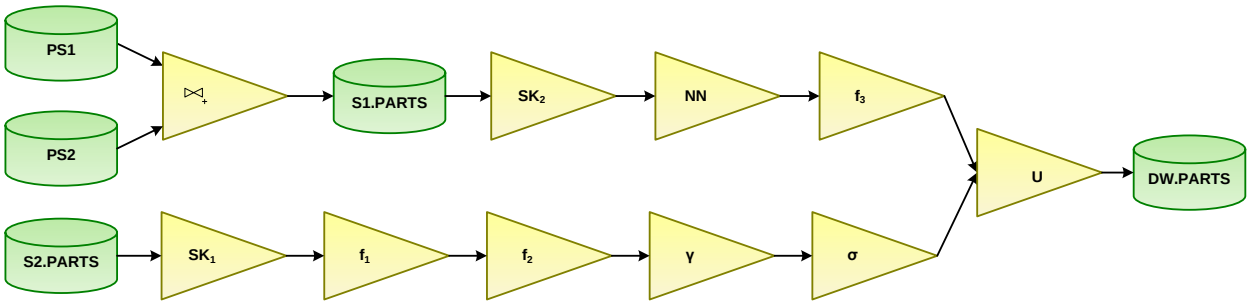
$$
\begin{aligned}
\mathtt{S_1.PARTS - DW.PARTS}: \quad & \mathtt{LW[0]} = [\{\mathtt{S_1.PARTS}\}, \{\mathtt{SK_2}, \mathtt{f_3}, \mathtt{NN}\}, \{\mathtt{DW.PARTS}\}] \\
\mathtt{S_2.PARTS - DW.PARTS}: \quad & \mathtt{LW[1]} = [\{\mathtt{S_2.PARTS}\}, \{\mathtt{SK_1}, \mathtt{f_1}, \mathtt{f_2}\}, \{\gamma\}, \{\sigma\}, \{\mathtt{DW.PARTS}\}] \\
\mathtt{PS_1.PARTS - S_1.PARTS}: \quad & \mathtt{LW[2]} = [\{\mathtt{PS_1}\}, \{\bowtie_+\}, \{\mathtt{S_1.PARTS}\}] \\
\mathtt{PS_2.PARTS - S_1.PARTS}: \quad & \mathtt{LW[3]} = [\{\mathtt{PS_2}\}, \{\bowtie_+\}]
\end{aligned}
$$

Όλοι οι μετασχηματισμοί σε κάθε στάδιο αυτού του παραδείγματος είναι ισοδύναμης σειράς, έτσι, δεν υπάρχει ένας μόνο τρόπος για την τοποθέτησή τους στο λογικό διάγραμμα. Με βάση τον αλγόριθμό **EOLW** κατασκευάζουμε τη λογική ροή έργου. Η γραμμική ροή έργου απεικονίζεται στο Σχήμα 4.23. Για ευκολία στην παρουσίαση, στα επόμενα σχήματα δεν παρουσιάζουμε τα γνωρίσματα των διεργασιών.

Παρατηρούμε ότι υπάρχουν δύο περιπτώσεις κοινών κόμβων (συνδεδεμένα σημεία): τελική πηγή δεδομένων **DW.PARTS** και η διεργασία εξωτερικής συνένωσης. Η πρώτη περίπτωση απαιτεί μία επιπλέον διεργασία Συνένωσης, ενώ στη δεύτερη περίπτωση θα συνδέσουμε απλά τις ροές αυτές στη δυαδική διεργασία. Έτσι, το λογικό διάγραμμα αντιπροσωπεύεται από τον γράφο του Σχήματος 4.24.

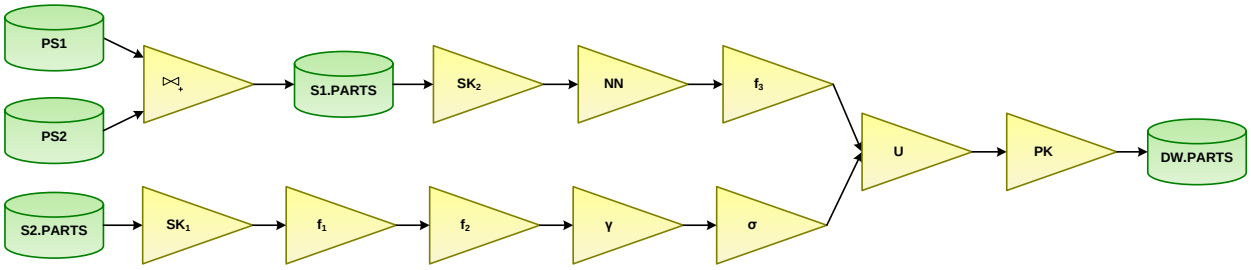


Σχήμα 4.23: Γραμμικές λογικές ροές έργου



Σχήμα 4.24: Το λογικό διάγραμμα του Παραδείγματος 1

Βήμα 4ο: Περιορισμοί ΕΜΦ. Οι τελευταίοι κατασκευαστές που πρέπει να προστεθούν, είναι οι περιορισμοί ΕΜΦ. Ελέγχουμε ποιες πηγές δεδομένων έχουν έχουν προσαρτημένο κάποιο περιορισμό ΕΜΦ, και το αναπαριστούμε με την κατάλληλη διεργασία, τοποθετημένη αμέσως πριν την αντίστοιχη πηγή δεδομένων στο λογικό διάγραμμα. Στο συγκεκριμένο παράδειγμα, μόνο η πηγή **DW.PARTS** έχει κάποιο περιορισμό ΕΜΦ, τύπου **PK** προσαρτημένο. Το τελικό λογικό διάγραμμα παρουσιάζεται στο Σχήμα 4.25.



Σχήμα 4.25: Το τελικό λογικό διάγραμμα του Παραδείγματος 1

Βήμα 5ο: Παραγωγή Σχημάτων. Τέλος, όλα τα σχήματα για όλες τις διεργασίες θα πρέπει να καθοριστούν. Λαμβάνοντας υπόψη ότι τα σχήματα λειτουργικότητας, παραγωγής και απόρριψης

έχουν καθοριστεί στα βήματα 3 και 4, το μόνο που απομένει είναι ο συγχρονισμός των σχημάτων εισόδου και εξόδου. Για το σκοπό αυτό χρησιμοποιούμε τον αλγόριθμο **Schema Generator**, που παρουσιάζεται στο [40].

5

Σχετική Εργασία

Στο κεφάλαιο αυτό θα σας παρουσιάσουμε σχετική εργασία, τόσο με τη μορφή ερευνητικών προσπαθειών όσο και έτοιμων εμπορικών εργαλείων. Αρχικά παρουσιάζονται ερευνητικές προσπάθειες για εννοιολογικά μοντέλα για Αποθήκες Δεδομένων γενικά, και στη συνέχεια εννοιολογικά μοντέλα για διεργασίες Εξαγωγής–Μετασχηματισμού–Φόρτωσης δεδομένων σε Αποθήκες Δεδομένων. Στη συνέχεια παρουσιάζονται προσπάθειες για λογικά μοντέλα Εξαγωγής–Μετασχηματισμού–Φόρτωσης δεδομένων· τεχνικές για τη μετάβαση από εννοιολογικό σε λογικό μοντέλο· και τέλος, αναφέρουμε κάποια εμπορικά εργαλεία για διεργασίες Εξαγωγής–Μετασχηματισμού–Φόρτωσης δεδομένων.

Εννοιολογικά Μοντέλα για Αποθήκες Δεδομένων. Το front end της Αποθήκης Δεδομένων, έχει μονοπωλήσει την έρευνα στο εννοιολογικό κομμάτι για τη μοντελοποίηση των Αποθηκών Δεδομένων. Στην πραγματικότητα, το μεγαλύτερο μέρος της εργασίας για τη μοντελοποίηση του εννοιολογικού μέρους, στο πεδίο των Αποθηκών Δεδομένων, έχει αφιερωθεί στη σύλληψη των εννοιολογικών χαρακτηριστικών του σχήματος αστέρα της Αποθήκης, των επιμέρους συλλογών δεδομένων και των συναθροίσεων (για μία πιο αναλυτική συζήτηση, δες [44]). Οι ερευνητικές προσπάθειες μπορούν να ομαδοποιηθούν σε τέσσερις κατηγορίες: (α) *μοντελοποίηση των διαστάσεων* [24, 25], (β) *επεκτάσεις του τυπικού μοντέλου Ο/Σ* [10, 11, 20, 29, 38, 43], (γ) *μοντελοποίηση UML* [30, 42], και (δ) *ιδιαίτερα μοντέλα* [16, 17, 44], χωρίς όμως ξεκάθαρο νικητή. Οι υποστηρικτές της μοντελοποίησης των διαστάσεων υποστηρίζουν ότι το μοντέλο είναι ελαχιστοποιημένο και κατανοητό (ειδικά από τους τελικούς χρήστες) καθώς και ότι αντιστοιχίζεται άμεσα με λογικές δομές. Οι υποστηρικτές του τυπικού μοντέλου Ο/Σ καθώς και του μοντέλου UML, βασίζουν τις απόψεις τους στη δημοτικότητα των

μοντέλων τους καθώς και στα σημασιολογικά θεμέλια και στην καλή διαμόρφωση των εννοιολογικών σχημάτων των Αποθηκών Δεδομένων. Τα μοντέλα της τελευταίας κατηγορίας, που δεν εμπίπτουν σε μία από τις προηγούμενες γενικές κατηγορίες, στηρίζονται στην καινοτομία και στην ικανότητα να προσαρμόζονται στις ιδιαιτερότητες του περιβάλλοντος ΣΑΕΔ.

Εννοιολογικά Μοντέλα για Διεργασίες ΕΜΦ. Υπάρχουν μερικές προσπάθειες σχετικά με το συγκεκριμένο πρόβλημα, αν και δε γνωρίζουμε κάποια άλλη προσπάθεια που να εξετάζει συγκεκριμένα τις λεπτομέρειες των διεργασιών ΕΜΦ σε εννοιολογικό επίπεδο. Μπορούμε να αναφερθούμε στο [9] ως μία προσπάθεια ξεκάθαρου διαχωρισμού της διαδικασίας ανανέωσης της Αποθήκης Δεδομένων από την παραδοσιακή επεξεργασία ως συντήρηση όψεων ή ως διαδικασία μαζικής φόρτωσης. Όμως, το προτεινόμενο μοντέλο είναι ανεπίσημο και εστιάζεται περισσότερο στην παρουσίαση αποδείξεων για την πολυπλοκότητα της προσπάθειας αυτής παρά στην τυπική μοντελοποίηση των διεργασιών αυτών καθ' αυτών. Στα [10, 11] εισάγεται η έννοια των ισχυρισμών μέσα στο μοντέλο, προκειμένου να συλλάβουν τις αντιστοιχίσεις μεταξύ των πηγών και της Αποθήκης Δεδομένων. Εντούτοις, οποιοσδήποτε μετασχηματισμός θα πρέπει να οριστεί εκ νέου για τη μετάβαση στο λογικό μοντέλο. Εκτός αυτών των ερευνητικών προσεγγίσεων, από το χώρο της αγοράς, το μοντέλο που περιγράφεται στο [25] μπορεί να θεωρηθεί ως μία άτυπη τεκμηρίωση της γενικής διαδικασίας ΕΜΦ.

Λογικά Μοντέλα για Διεργασίες ΕΜΦ. Στο [13] περιγράφεται το AJAX, ένα εργαλείο για καθαρισμό δεδομένων, που ασχολείται με τυπικά προβλήματα καθαρισμού δεδομένων, όπως ταυτοποίηση αντικειμένων, λάθη εξαιτίας εσφαλμένων καταχωρήσεων, καθώς και ασυμφωνίες δεδομένων μεταξύ ισοδύναμων εγγραφών. Το AJAX παρέχει ένα πλαίσιο εργασίας, όπου η λογική ενός προγράμματος καθαρισμού δεδομένων μοντελοποιείται ως κατευθυνόμενος γράφος απεικονίσεων, ταιριασμάτων, ομαδοποιήσεων και συνενώσεων δεδομένων. Το [37] παρουσιάζει το σύστημα καθαρισμού δεδομένων Potter's Wheel, το οποίο προσφέρει τη δυνατότητα εφαρμογής διαφόρων αλγεβρικών λειτουργιών πάνω σε ένα σύνολο δεδομένων με κύρια χαρακτηριστικά την αλληλεπίδραση και την επανάληψη. Στο [36] υπάρχει μία εκτεταμένη παρουσίαση του χώρου του καθαρισμού των δεδομένων, η οποία θίγει κάποια ερευνητικά θέματα, καθώς και μία επισκόπηση των εμπορικών εργαλείων. Το [28] προσαρμόζει κάποιους υπάρχοντες αλγόριθμους στο συγκεκριμένο πεδίο έρευνας. Το [8] επικεντρώνεται σε ένα σχετικό υποπρόβλημα, το θέμα του διαχωρισμού διευθύνσεων σε επιμέρους στοιχεία και προτείνει μία λύση μέσω εκπαίδευσης ενός Hidden Markov μοντέλου. Στα [45, 49] γίνεται μία προσπάθεια να καλυφθούν σχεδιαστικά θέματα ΕΜΦ, προσπαθώντας: (α) να δείξουν τον τρόπο που συνδέονται οι διαδικασίες της Αποθήκης Δεδομένων με μία αποθήκη μεταδεδομένων, (β) να κατασκευάσουν ένα εργαλείο, και (γ) να καλύψουν θέματα ποιότητας για τις διαδικασίες της Αποθήκης Δεδομένων.

Μετάβαση από το Εννοιολογικό στο Λογικό Μοντέλο. Μέχρι τώρα, δεν έχουμε υπόψη μας κάποια άλλη ερευνητική προσέγγιση που αφορά τη μετάβαση από εννοιολογικό σε λογικό μοντέλο για διεργασίες ΕΜΦ. Όμως, σε παράλληλη πορεία έρευνας βρίσκονται αρκετές ερευνητικές προσεγγίσεις

που αφορούν την ημι-αυτοματοποιημένη μετάβαση από εννοιολογικά μοντέλα σε διάφορες φάσεις τις λογικής σχεδίασης Αποθηκών Δεδομένων [4, 5, 16, 19, 29, 33, 34, 35, 50].

Εμπορικά Εργαλεία. Τα εμπορικά εργαλεία συνήθως υλοποιούν τη ροή δεδομένων στο περιβάλλον της Αποθήκης Δεδομένων, η οποία είναι μία (αν και πολύ σημαντική) από τις διεργασίες μίας Αποθήκης Δεδομένων. Τα περισσότερα εργαλεία ΕΜΦ μπορούν να χωριστούν σε δύο κατηγορίες: τα *engine based* και τα *code-generation based*. Αυτά που ανήκουν στην πρώτη κατηγορία, υποθέτουν ότι τα δεδομένα πρέπει να περάσουν από μία μηχανή για να μετασχηματιστούν και να επεξεργαστούν. Αντίθετα, σε αυτά που ανήκουν στη δεύτερη κατηγορία, οι διεργασίες λαμβάνουν χώρα είτε στο σύστημα πηγή, είτε στο σύστημα στόχο.

Μία πρόσφατη μελέτη [15] αναφέρει ότι λόγω της ποικιλίας και των ετερογενών χαρακτηριστικών των πηγών δεδομένων, είναι μάλλον απίθανο να υπάρξει μία ευρεία αγορά εργαλείων ΕΜΦ. Παρόλο που η αγορά εργαλείων ΕΜΦ έφθασε τα $667 εκατομμύρια το 2001, ο ρυθμός ανάπτυξης της βρισκόταν στο 11% (τιμή χαμηλή σε σύγκριση με το 60% του 2000). Αυτό εξηγείται εν μέρει από τη γενική οικονομική ύφεση. Από τεχνολογικής σκοπιάς, το βασικό χαρακτηριστικό του χώρου είναι η ανάμιξη των παραδοσιακών εταιριών που παράγουν και εφοδιάζουν την αγορά με Συστήματα Διαχείρησης Βάσεων Δεδομένων, με την ενσωμάτωση λύσεων ΕΜΦ στα συστήματά τους. Οι τρεις βασικές εταιρίες του χώρου, που εφοδιάζουν την αγορά με λύσεις ΕΜΦ "χωρίς επιπλέον κόστος" είναι η Oracle με το Oracle Warehouse Builder [31], η Microsoft με τα Data Transformation Services [26] και η IBM με το Data Warehouse Center [21]. Παρόλα αυτά, οι βασικές εταιρίες στο χώρο είναι οι Informatica με το Powercenter [22] και η Ascential με το DataStage [2, 3] (με το τελευταίο να εμφανίζεται στις προτεινόμενες λύσεις ΕΜΦ της IBM). Η μελέτη συνεχίζεται με εισηγήσεις για μελλοντικές τεχνολογικές απαιτήσεις και προβλέψεις που περιλαμβάνουν την ενοποίηση των εργαλείων ΕΜΦ με (α) προσαρμογείς XML, (β) εργαλεία Επιχειρησιακής Ολοκλήρωσης Εφαρμογών, (γ) προσαρμοσμένα εργαλεία ποιότητας και (δ) την πορεία προς συστήματα παράλληλης επεξεργασίας ροών έργου ΕΜΦ.

Τα παραπάνω ενισχύονται από μία ακόμα πρόσφατη μελέτη [14], όπου οι συγγραφείς σημειώνουν τη μη αποδοχή του κόστους για καθαρά εργαλεία ΕΜΦ, κυρίως λόγω της κρίσης και της ενσωμάτωσης λύσεων ΕΜΦ από τους βασικούς προμηθευτές της αγοράς σε ΣΔΒΔ. Η μελέτη Gartner ασχολείται με το ρόλο των τριών βασικών εταιριών του χώρου των βάσεων δεδομένων (IBM, Microsoft, Oracle) και σημειώνει ότι σιγά σιγά έχουν αρχίσει να αποκτούν ένα ποσοστό της αγοράς ΕΜΦ μέσω των λύσεων που ενσωματώνουν στα ΣΔΒΔ τους.

Στη συνέχεια, ασχολούμαστε λεπτομερώς με τους βασικούς προμηθευτές της αγοράς εργαλείων αποκλειστικά για ΕΜΦ και βλέπουμε τρία εργαλεία που παρέχονται από εταιρίες παραγωγής ΣΔΒΔ, καθώς και δύο εργαλεία που έχουν υψηλές πωλήσεις. Παρόλα αυτά, δίνουμε έμφαση στο ότι τα πρώτα τρία εργαλεία έχουν το πλεονέκτημα του ελαχιστοποιημένου κόστους, καθώς περιλαμβάνονται στη βάση δεδομένων, ενώ τα τελευταία δύο έχουν το πλεονέκτημα ότι μπορούν να χρησιμοποιηθούν σε πολύπλοκες περιπτώσεις, που δεν προβλέπονται από τα πρώτα τρία εργαλεία.

IBM. Η βάση δεδομένων DB2 περιλαμβάνει το Data Warehouse Center [21], ένα εργαλείο που αυτοματοποιεί την επεξεργασία Αποθηκών Δεδομένων και το DB2 Warehouse Manager που επεκτείνει τις δυνατότητες του Data Warehouse Center με πρόσθετους αντιπροσώπους (agents), μετασχηματισμούς και δυνατότητες μεταδεδομένων. Το Data Warehouse Center χρησιμοποιείται για τον ορισμό διεργασιών μεταφοράς και μετασχηματισμού δεδομένων για την Αποθήκη Δεδομένων. Το Warehouse Manager χρησιμοποιείται για τον προγραμματισμό, τη συντήρηση και την επιτήρηση των διεργασιών αυτών. Στο Data Warehouse Center περιλαμβάνεται ο *σχεδιαστής σχήματος αποθήκης (warehouse schema modeler)*, ο οποίος είναι ένα εξειδικευμένο εργαλείο για τη δημιουργία και αποθήκευση του σχήματος που σχετίζεται με μία Αποθήκη Δεδομένων. Κάθε σχήμα που προέρχεται από μία διεργασία μπορεί να χρησιμοποιηθεί με τη μορφή μεταδεδομένων σε ένα εργαλείο ΣΑΕΔ. Ο *σχεδιαστής διεργασιών (process modeler)* επιτρέπει στο χρήστη να ορίσει γραφικά τα βήματα που απαιτούνται για τη δημιουργία και τη συντήρηση Αποθηκών Δεδομένων και εξαρτημένων data martrs. Το DB2 Warehouse Manager περιλαμβάνει εκτεταμένες συναρτήσεις ΕΜΦ σε σχέση με τις βασικές του DB2 Warehouse Center. Επιπρόσθετα, παρέχει διαχείριση μετα-δεδομένων, λειτουργίες αποθήκευσης, καθώς προβλέπει σύνδεση με άλλα ανεξάρτητα εργαλεία λογισμικού μέσω καταλόγων πληροφορίας.

Microsoft. Το εργαλείο που προσφέρεται από τη Microsoft για την υλοποίηση του Open Information Model εμφανίζεται με την ονομασία *Data Transformation Services* [26, 6]. Τα Data Information Services (DTS) είναι το σύνολο των λειτουργιών διαχείρισης δεδομένων του SQL Server (από την έκδοση 7.0 και μετά), που παρέχουν δυνατότητες εισαγωγής, εξαγωγής και λειτουργίες διαχείρισης δεδομένων μεταξύ πηγών δεδομένων τύπου OLE DB [27], ODBC και ASCII. Τα Data Transformation Services χαρακτηρίζονται από ένα βασικό αντικείμενο, το οποίο ονομάζεται πακέτο, και το οποίο αποθηκεύει πληροφορία για τις παραπάνω λειτουργίες και τη σειρά με την οποία πρέπει να εκτελεστούν. Ένα πακέτο μπορεί να περιέχει μία ή περισσότερες συνδέσεις σε διαφορετικές πηγές δεδομένων, και διαφορετικές λειτουργίες και μετασχηματισμούς που εκτελούνται βηματικά ορίζοντας μία ροή έργου [18]. Τα λογισμικά κομμάτια που υποστηρίζουν το DTS παρέχονται με τον SQL Server. Αυτά τα κομμάτια περιέχουν:

- *DTS Designer:* Ένα γραφικό περιβάλλον που χρησιμοποιείται για το σχεδιασμό και την εκτέλεση πακέτων DTS.
- *DTS Export and Import Wizards:* Βοηθούς που διευκολύνουν τη διαδικασία ορισμού πακέτων DTS για την εισαγωγή, την εξαγωγή και το μετασχηματισμό δεδομένων.
- *DTS Programming Interfaces:* Ένα σύνολο από Αυτοματισμούς OLE και ένα σύνολο από διεπαφές COM για τη δημιουργία προσαρμοσμένων μετασχηματισμών για κάθε σύστημα που υποστηρίζει αυτοματισμούς OLE ή COM.

Oracle. Το Oracle Warehouse Builder [31, 32] είναι ένα εργαλείο για ΕΜΦ και Αποθήκες Δεδομένων που στηρίζεται σε repository. Η βασική του αρχιτεκτονική αποτελείται από δύο τμήματα, το *περιβάλλον σχεδίασης* και το *περιβάλλον εκτέλεσης*. Κάθε ένα από αυτά τα τμήματα ασχολείται με μία διαφορετική πτυχή του συστήματος. Το περιβάλλον σχεδίασης ασχολείται με τα μετα-

δεδομένα, ενώ το περιβάλλον εκτέλεσης με τα φυσικά δεδομένα. Το τμήμα μετα-δεδομένων περιστρέφεται γύρω από το repository μετα-δεδομένων και το εργαλείο σχεδίασης. Το repository βασίζεται στο πρότυπο Common Warehouse Model (CWM) και αποτελείται από ένα σύνολο πινάκων σε μία βάση δεδομένων Oracle. Το front-end του εργαλείου (γραμμένο εξ ολοκλήρου σε Java) περιλαμβάνει βοηθούς και γραφικά εργαλεία για τη σύνδεση με το repository. Το τμήμα των δεδομένων περιστρέφεται γύρω από το περιβάλλον εκτέλεσης και την βάση της Αποθήκης. Το εκτελέσιμο του Warehouse Builder είναι ένα σύνολο από πίνακες, sequences, πακέτα και triggers τα οποία είναι εγκατεστημένα στο σχήμα-στόχο. Ο γεννήτορας κώδικα που στηρίζεται στα μετα-δεδομένα του repository, παράγει τον απαιτούμενο κώδικα για την υλοποίηση της Αποθήκης. Ο Warehouse Builder παράγει scripts (αρχεία ελέγχου SQL *Loader για επίπεδα αρχεία, ABAP για εξαγωγή SAP/R3 και PL/SQL για όλα τα άλλα συστήματα) για τις διεργασίες ΕΜΦ και εντολές SQL DDL για τα αντικείμενα της βάσης δεδομένων. Ο παραγόμενος κώδικας εφαρμόζεται είτε στο σύστημα αρχείων, είτε στη βάση δεδομένων.

Ascential Software. Η σουίτα εφαρμογών DataStage XE από την Ascential Software [2, 3] (πρώην Informix Business Solutions) είναι ένα ολοκληρωμένο σύνολο εργαλείων για την ανάπτυξη Αποθηκών Δεδομένων, το οποίο περιλαμβάνει ένα εργαλειο ΕΜΦ (DataStage), ένα εργαλείο για τον έλεγχο της ποιότητας (Quality Manager) και ένα εργαλείο διαχείρισης μετα-δεδομένων (MetaStage). Το DataStage ETL αποτελείται από τέσσερα εργαλεία σχεδίασης και διαχείρισης: τα *Manager*, *Designer*, *Director* και *Administrator*, όπως επίσης και από ένα *repository μετα-δεδομένων* και ένα *διακομιστή*. Το DataStage Manager είναι το βασικό εργαλείο διαχείρισης μετα-δεδομένων. Στο Designer, οι διεργασίες ΕΜΦ εκτελούνται σε ανεξάρτητα στάδια (αρχικό, τελικό και στάδιο μετασχηματισμών) με σκοπό τη δημιουργία διεργασιών ΕΜΦ. Το Director ασχολείται με τον έλεγχο αξιοπιστίας και τον προγραμματισμό των διεργασιών. Το Administrator ελέγχει τις συναρτήσεις ασφαλείας. Ο διακομιστής, είναι ο μηχανισμός που μεταφέρει τα δεδομένα από την πηγή στο στόχο.

Informatica. Το Informatica PowerCenter [22] αποτελεί την κυρίαρχη (σύμφωνα με πρόσφατες μελέτες [14, 15]) πλατφόρμα ολοκλήρωσης δεδομένων για τη δημιουργία, εγκατάσταση και διαχείριση επιχειρηματικών Αποθηκών Δεδομένων, καθώς και άλλων έργων ολοκλήρωσης δεδομένων. Ο πυρήνας του Informatica PowerCenter είναι ένας μηχανισμός ολοκλήρωσης δεδομένων που εκτελεί όλες τις συναρτήσεις εξαγωγής δεδομένων, μετασχηματισμού, μετατροπής και φόρτωσης στη μνήμη, χωρίς τη δημιουργία κώδικα και χωρίς να απαιτείται από τους προγραμματιστές να προγραμματίσουν αυτές τις διαδικασίες. Η μηχανή ολοκλήρωσης του PowerCenter οδηγείται από τα μετα-δεδομένα, δημιουργώντας μια σχέση μεταξύ repository και μηχανής η οποία εξασφαλίζει την καλύτερη δυνατή ολοκλήρωση δεδομένων.

Τα εμπορικά εργαλεία που αναφέραμε παραπάνω απευθύνονται σε φυσικό επίπεδο και δεν ασχολούνται με λογική σχεδίαση και μοντελοποίηση. Αντίθετα, η δική μας προσέγγιση διαφοροποιείται στο ότι είναι επικεντρωμένη στη λογική σχεδίαση και μοντελοποίηση.

Μέρος ΙΙ

Σχεδίαση και Υλοποίηση Εργαλείου Σχεδίασης Διεργασιών ΕΜΦ

6

Ανάλυση και Σχεδίαση

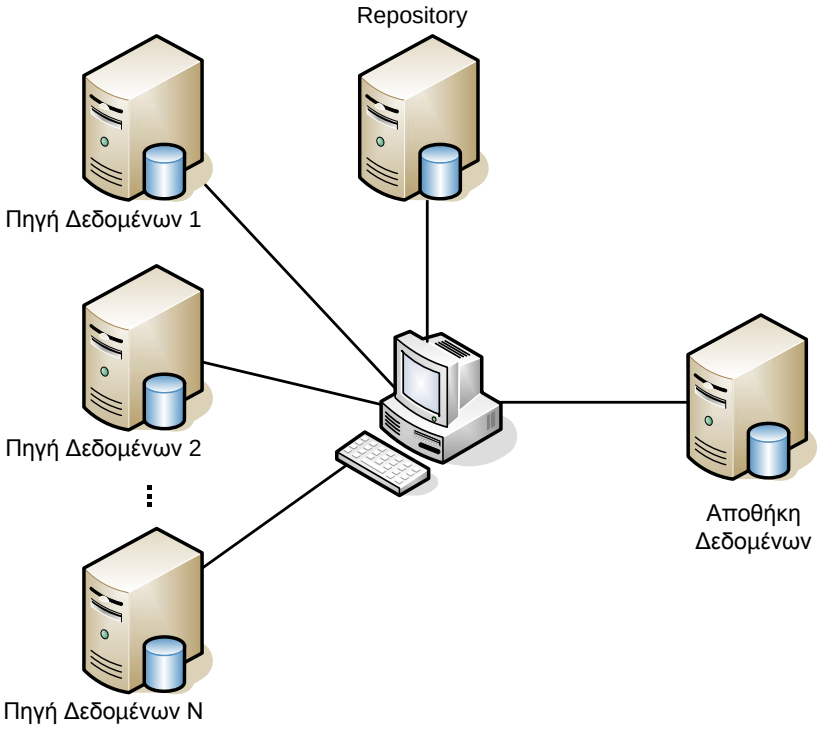
Το παρόν κεφάλαιο παρουσιάζει την ανάλυση και τη σχεδίαση του εργαλείου που κατασκευάστηκε. Στην ενότητα 6.1 περιγράφεται η αρχιτεκτονική του συστήματος, ενώ προσδιορίζονται και περιγράφονται τα λογικά τμήματα που το αποτελούν. Στη συνέχεια, στην ενότητα 6.2, αναλύονται και περιγράφονται αναλυτικά οι δυνατότητες που παρέχει στο χρήστη.

6.1 Περιγραφή Αρχιτεκτονικής

Για τη σωστή λειτουργία του εργαλείου, ο χρήστης πρέπει να έχει πρόσβαση στα εξής:

1. Στις Σχεσιακές Βάσεις Δεδομένων (πηγές δεδομένων) από τις οποίες επιθυμεί να εξάγει δεδομένα.
2. Στην Αποθήκη Δεδομένων στην οποία επιθυμεί να φορτωθούν τα αποτελέσματα των διεργασιών ΕΜΦ.
3. Σε κάποιο repository ώστε να μπορεί να χρησημοποιήσει τα απαραίτητα templates, να αποθηκεύσει το νέο σενάριο ή τις αλλαγές που έχει κάνει και να φορτώσει κάποιο υπάρχον σενάριο.

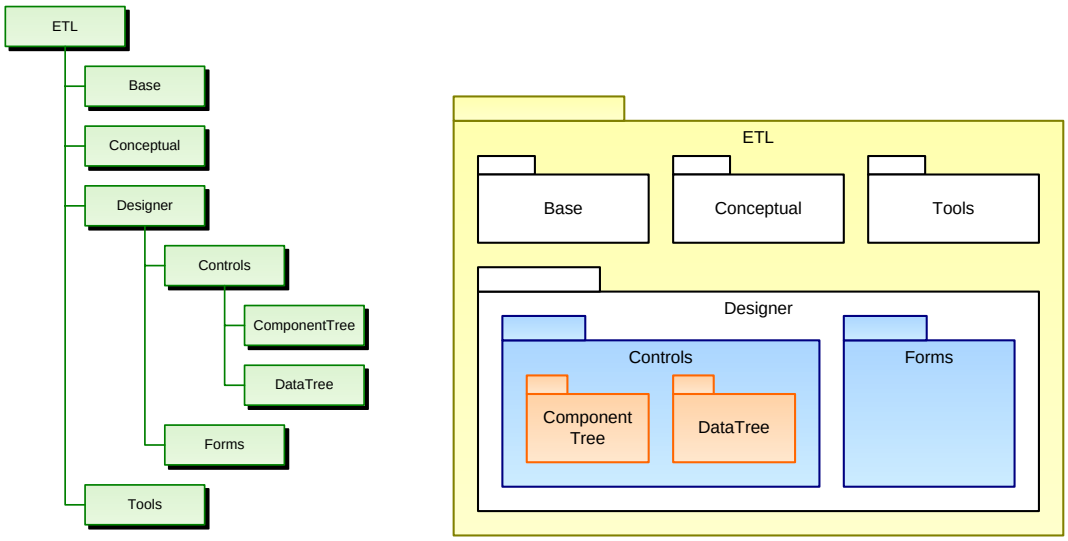
Οι παραπάνω απαιτήσεις περιγράφονται στο Σχήμα 6.1. Να σημειωθεί ότι οι πηγές δεδομένων, η Αποθήκη Δεδομένων και το repository δεν είναι απαραίτητο να βρίσκονται σε απομακρυσμένους υπολογιστές. Κάποια ή όλα από τα παραπάνω μπορεί να βρίσκονται εγκατεστημένα στον ίδιο υπολογιστή. Για λόγους γενικότητας, στο Σχήμα 6.1 δίνεται η πλήρης μορφή.



Σχήμα 6.1: Πλήρες διάγραμμα λειτουργίας εργαλείου

6.1.1 Διάρθρωση Εφαρμογής

Για σκοπούς ευκολίας και καλύτερης διαχείρισης, η εφαρμογή χωρίζεται σε διάφορα τμήματα (packages). Ο τρόπος που οργανώνονται τα τμήματα αυτά, φαίνεται στο Σχήμα 6.2.



(α) Δενδρική Δομή (β) UML

Σχήμα 6.2: Τα τμήματα (packages) στα οποία χωρίζεται η εφαρμογή.

ETL: Πρόκειται για το τμήμα το βασικό τμήμα στο οποίο περιέχονται όλα τα άλλα τμήματα της εφαρμογής (root package). Περιέχει επίσης και μία λειτουργική μονάδα (module) με όλες τις

συναρτήσεις και τις μεταβλητές που πρέπει να είναι ορατές από όλη την εφαρμογή.

ETL.Base: Είναι το θεμελιώδες κομμάτι της εφαρμογής. Εδώ ορίζονται αφηρημένα οι κλάσης που θα χρησιμοποιηθούν στη συνέχεια για τη δημιουργία των κόμβων και των ακμών που απαιτούνται για την ορθή αναπαράσταση ενός εννοιολογικού σεναρίου. Παράλληλα, περιέχει και δύο διεπαφές: μία για την ανανέωση της γραφικής απεικόνισης όταν ο χρήστης αλλάξει κάποια ιδιότητα στο σενάριό του, και μία για την εκτύπωση των σεναρίων του χρήστη.

ETL.Conceptual: Στο τμήμα αυτό εξειδικεύουμε τις αφηρημένες κλάσεις του τμήματος **ETL.Base** και δημιουργούμε τους κόμβους και τις ακμές που μας είναι απαραίτητες για την ορθή γραφική απεικόνιση ενός εννοιολογικού σεναρίου ΕΜΦ, όπως αυτό περιγράφεται στο Κεφάλαιο 2.

ETL.Designer: Πρόκειται για το τμήμα της εφαρμογής που περιέχει το γραφικό περιβάλλον το οποίο χρησιμοποιεί ο χρήστης για τη δημιουργία και την επεξεργασία ενός σεναρίου. Επιπλέον, το τμήμα αυτό περιέχει δύο άλλα τμήματα:

ETL.Designer.Controls: Περιέχει όλα τα γραφικά εργαλεία που δεν υπάρχουν στο περιβάλλον ανάπτυξης και τα οποία ήταν απαραίτητα για τη δημιουργία του γραφικού περιβάλλοντος. Δύο από αυτά τα εργαλεία, τα οποία λόγω της πολυπλοκότητας τους, και του ότι περιλαμβάνουν αρκετές κλάσεις, ανήκουν σε δικά τους τμήματα. Αυτά είναι τα **ETL.Designer.Controls.ComponentTree** και **ETL.Designer.Controls.DataTree**.

ETL.Designer.Forms: Περιέχει όλες τις φόρμες του εργαλείου, μέσω των οποίων γίνεται η εισαγωγή των δεδομένων καθώς και η επικοινωνία με το χρήστη.

ETL.Tools: Περιέχει εργαλεία που βοηθούν στην πληρότητα της εφαρμογής, όπως (α) δυνατότητα κρυπτογράφησης μίας συμβολοακολουθίας (για την προστασία των κωδικών πρόσβασης που αποθηκεύονται στο repository) και (β) αναγνώριση του τύπου ενός πεδίου στη βάση δεδομένων, έτσι ώστε ο χρήστης να γνωρίζει τον τύπο του πεδίου ανά πάσα στιγμή.

6.2 Περιγραφή Λειτουργιών

Η ενότητα αυτή παρουσιάζει αναλυτικά τις βασικές λειτουργίες της εφαρμογής.

6.2.1 Επιλογή Repository

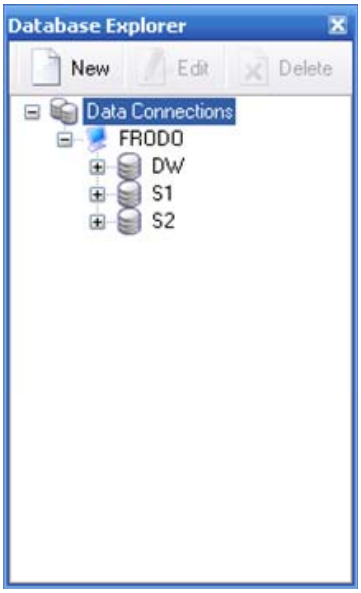
Για τη λειτουργία του εργαλείου απαιτείται η ύπαρξη ενός repository, το οποίο χρησιμοποιείται για την αποθήκευση και την ανάκτηση σεναρίων. Παράλληλα, στο repository υπάρχουν οι τύποι δεδομένων συναρτήσεων, μετασχηματισμών και περιορισμών ΕΜΦ (όπως ανάθεση υποκατάστατου κλειδιού, έλεγχος αναφορικής ακεραιότητας κ.α.) έτσι ώστε να μην απαιτείται ο εκ νέου ορισμός τους κάθε φορά που θα χρησιμοποιηθούν.

Σχήμα 6.3: Σύνδεση στο Repository

6.2.2 Αναγνώριση των Κατάλληλων Πηγών Δεδομένων

Όπως είδαμε και στο προηγούμενο κεφάλαιο, το πρώτο πρόβλημα που αντιμετωπίζει ο σχεδιαστής κατά την περίοδο της ανάλυσης και συλλογής των απαιτήσεων της Αποθήκης Δεδομένων, είναι η αναγνώριση των σχετικών πηγών δεδομένων. Μέσω κατάλληλων φορμών, ο σχεδιαστής μπορεί να ορίσει τις πηγές δεδομένων που επιθυμεί να προστεθούν στο σενάριό του. Στο Σχήμα 6.4(α), παρουσιάζεται η οθόνη προσθήκης μίας πηγής δεδομένων στο σενάριο, ενώ στο Σχήμα 6.4(β) υπάρχει η λίστα με τις πηγές δεδομένων που έχουν προστεθεί στο σενάριο μέχρι τώρα. Μέσω του ADO.NET API (που αποτελεί μέρος του Visual Studio .NET) υποστηρίζονται πολλά διαφορετικά ΣΔΒΔ, οι οποίοι είναι διαθέσιμοι στο χρήστη/σχεδιαστή, δίνοντας του την ευελιξία να μπορεί να χρησιμοποιήσει το εργαλείο σχεδόν σε κάθε περίπτωση.

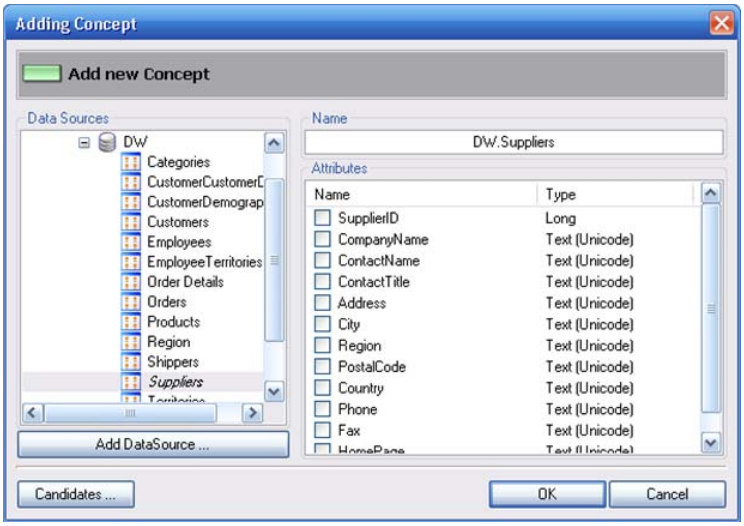
(α) Προσθέτοντας μία νέα πηγή δεδομένων



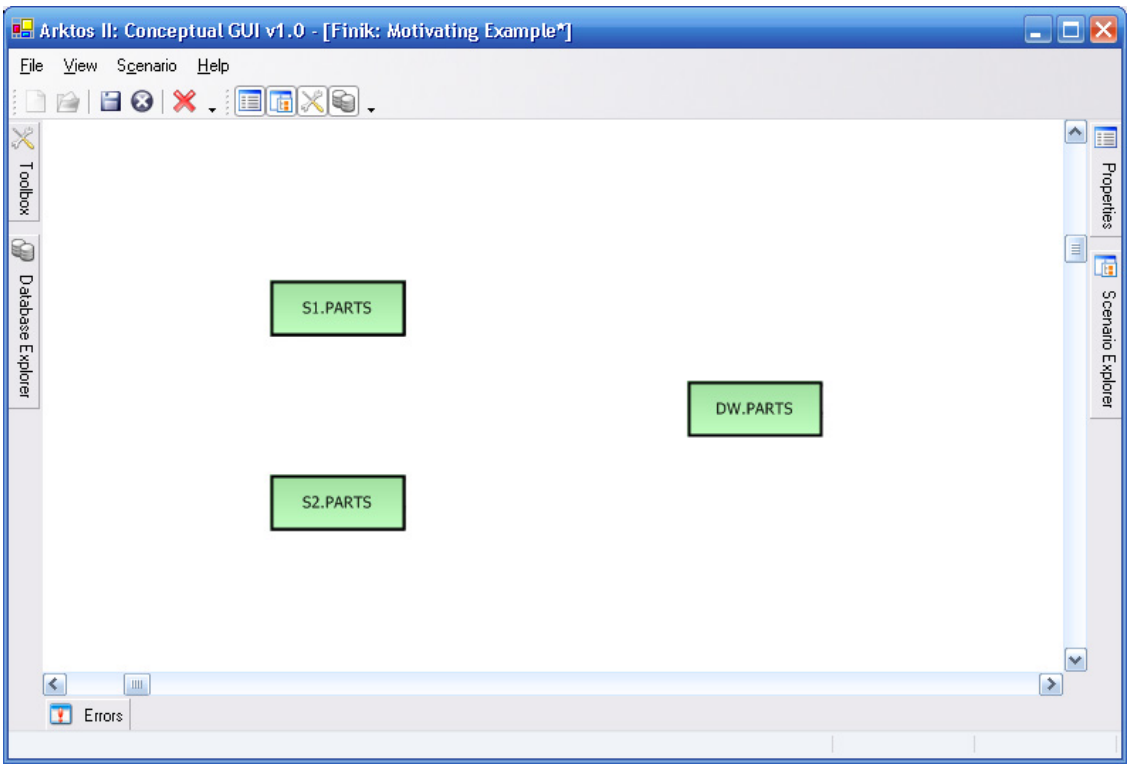
(β) Οι πηγές δεδομένων που ανήκουν στο σενάριο

Σχήμα 6.4: Αναγνώριση Πηγών Δεδομένων

Αφού αναγνωριστούν οι πηγές δεδομένων, ο σχεδιαστής τις τοποθετεί στην επιφάνεια σχεδιασμού του εργαλείου. Παράλληλα, από τη φόρμα που εμφανίζεται κατά την τοποθέτηση μίας πηγής δεδομένων (με τη μορφή έννοιας) στην επιφάνεια εργασίας, ο σχεδιαστής μπορεί να επιλέξει ποια από τα γνωρίσματά της θέλει να εμφανίζονται στο σενάριο. Η επιλογή αυτή φαίνεται στο Σχήμα 6.2.2, ενώ στο Σχήμα 6.2.2 παρουσιάζεται η επιφάνεια εργασίας αφότου έχουν προστεθεί οι έννοιες.



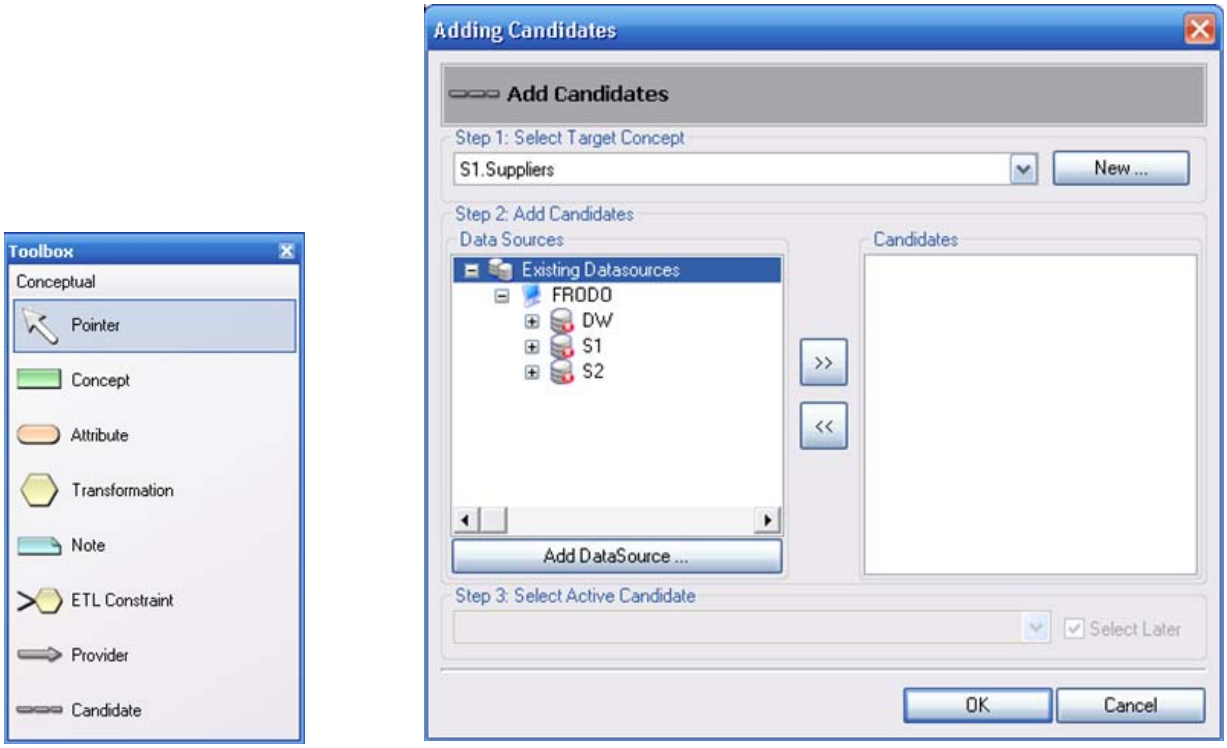
Σχήμα 6.5: Επιλογή γνωρισμάτων για την έννοια **DW.Suppliers**



Σχήμα 6.6: Στιγμιότυπο της επιφάνειας εργασίας αμέσως μετά την προσθήκη τριών εννοιών

6.2.3 Υποψήφιοι και Ενεργοί Υποψήφιοι

Όπως έχουμε ήδη αναφέρει, ο σχεδιαστής ενδέχεται να ανακαλύψει ότι περισσότερες από μία πηγές δεδομένων μπορούν να είναι υποψήφιες για να συμπληρώσουν μία έννοια. Σε αυτή την περίπτωση, ο σχεδιαστής τοποθετεί στο σενάριο όλες τις πιθανές περιπτώσεις και αναβάλλει τη λήψη της απόφασης αυτής για το μέλλον. Στο εργαλείο που κατασκευάσαμε, καλύψαμε την περίπτωση αυτή με την προσθήκη της επιλογής Provider (Σχήμα 6.7(α)) στην εργαλειοθήκη του εργαλείου, αλλά και με την επιλογή Add Candidates... που εμφανίζεται κατά την προσθήκη μίας νέας έννοιας στο σενάριο (κάτω αριστερά στο Σχήμα 6.2.2). Και οι δύο αυτές επιλογές οδηγούν στην οθόνη του Σχήματος 6.7(β), όπου ο σχεδιαστής ορίζει επιλέγει της υποψήφιες πηγές δεδομένων για την πλήρωση της έννοιας αυτής, και αποφασίζει (είτε τώρα είτε αργότερα) ποια έννοια θα αποτελεί τον ενεργό υποψήφιο.



(α) Η εργαλειοθήκη της εφαρμογής

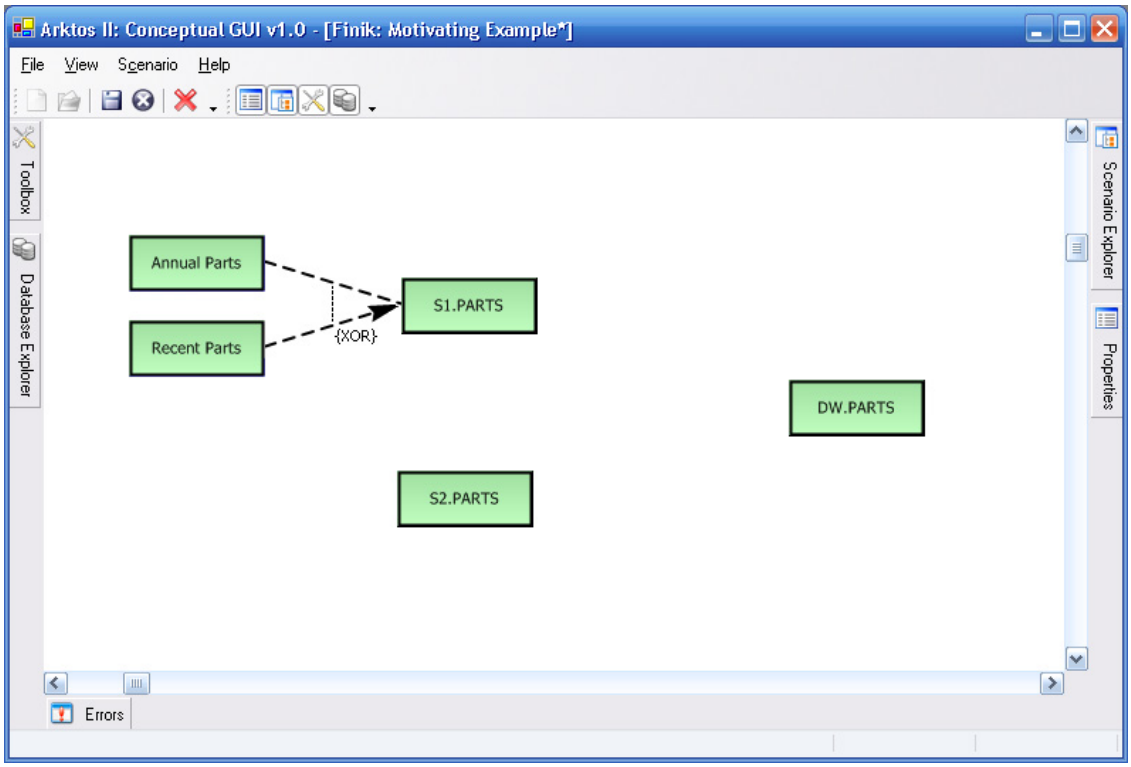
(β) Ορισμός Υποψηφίων για την έννοια **S1.Suppliers**

Σχήμα 6.7: Προσθήκη Υποψηφίων στο Σενάριο

Αφού γίνουν οι επιλογές για τους υποψηφίους και τον ενεργό υποψήφιο, το αποτέλεσμα που προκύπτει, εμφανίζεται στο Σχήμα 6.8.

6.2.4 Αντιστοίχηση γνωρισμάτων μεταξύ πηγών και στόχων

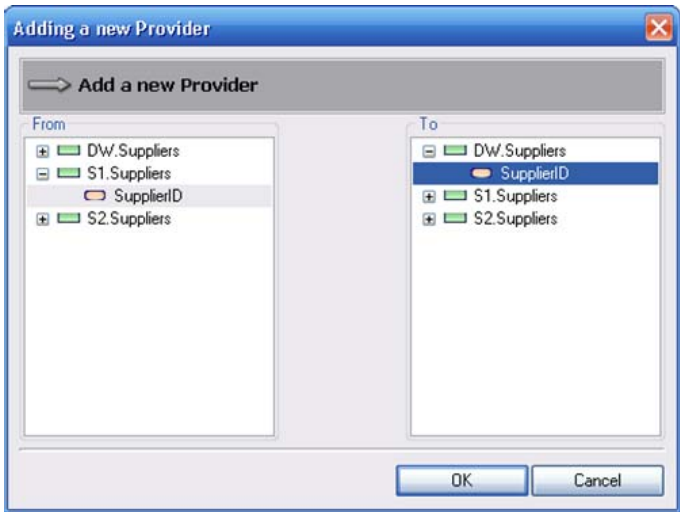
Όπως είδαμε και στο Κεφάλαιο 2, η πιο δύσκολη διαδικασία για το σχεδιαστή της Αποθήκης Δεδομένων είναι ο καθορισμός της αντιστοίχησης των γνωρισμάτων των πηγών με αυτών της Αποθήκης Δεδομένων. Στο εργαλείο αυτό, έχει δοθεί βάρος στην εύκολη αντιστοίχηση των εμπλεκομένων γνωρι-



Σχήμα 6.8: Στιγμιότυπο του σεναρίου, μετά την προσθήκη υποψηφίων.

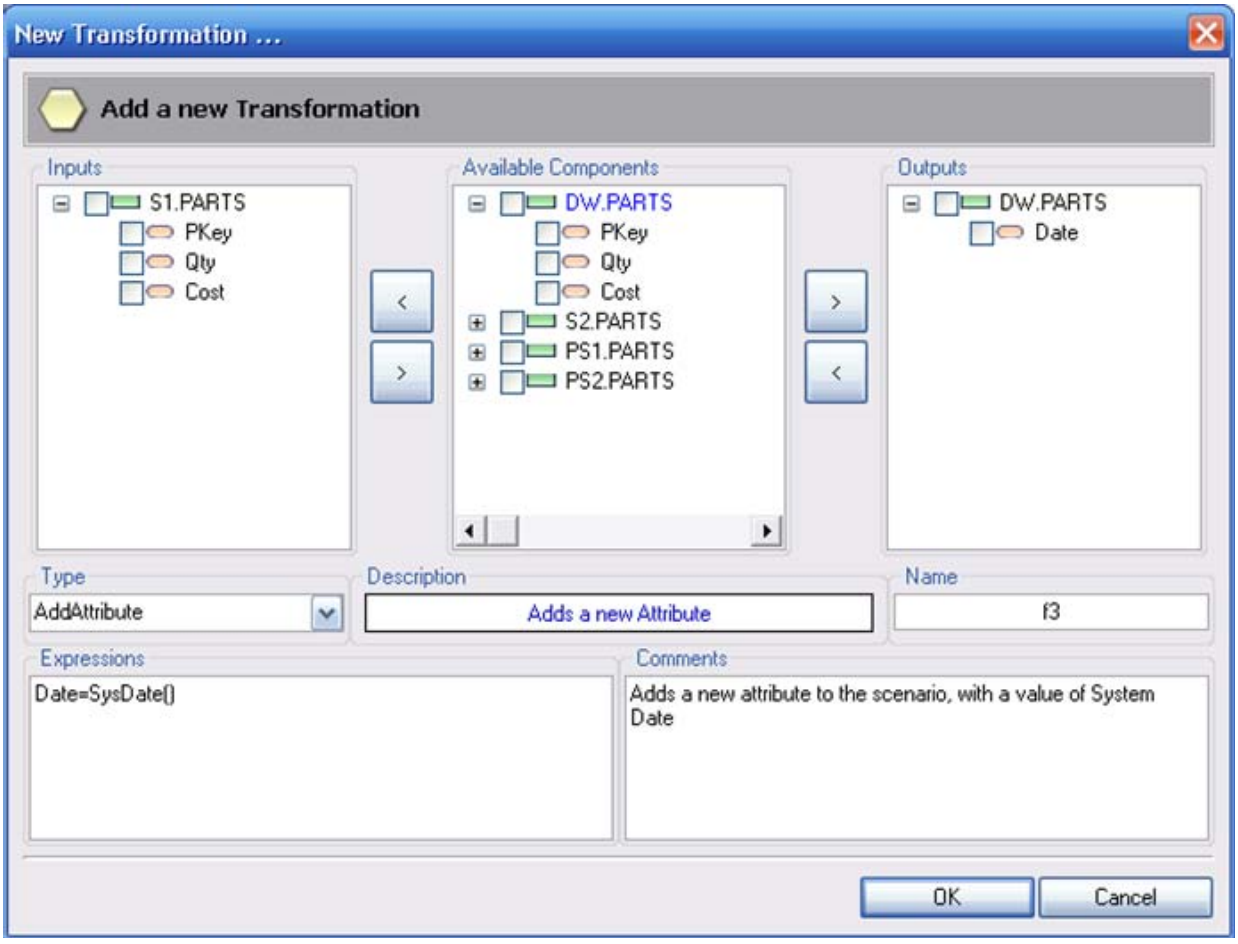
σμάτων, με το ενδιαφέρον να επικεντρώνεται στην επισήμανση των μετασχηματισμών και των εργασιών καθαρισμού που ενδέχεται να περικλείει η αντιστοίχηση αυτή.

Για την αποφυγή λαθών από μέρους του χρήστη/σχεδιαστή, η αντιστοίχηση γνωρισμάτων γίνεται μέσω απλών φορμών. Όπως έχουμε ήδη δει, η αντιστοίχηση των γνωρισμάτων μπορεί να είναι άμεση (χωρίς την παρεμβολή κάποιου μετασχηματισμού), είτε έμμεση (μέσω ενός ή περισσοτέρων μετασχηματισμών). Στο Σχήμα 6.2.4 παρουσιάζεται η φόρμα που χρησιμοποιείται για την άμεση αντιστοίχηση.



Σχήμα 6.9: Άμεση αντιστοίχηση γνωρισμάτων

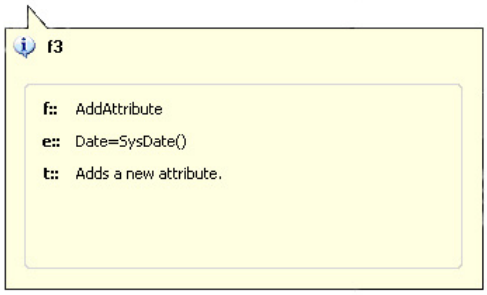
Στο Σχήμα 6.10 παρουσιάζεται ίσως η σημαντικότερη φόρμα του εργαλείου που κατασκευάσαμε. Μέσω της φόρμας αυτής, ο χρήστης/σχεδιαστής, εισάγει στο σενάριο κάποιο μετασχηματισμό, και ορίζει τα γνωρίσματα εισόδου και εξόδου του μετασχηματισμού αυτού. Στο συγκεκριμένο σχήμα, βλέπουμε την εισαγωγή ενός μετασχηματισμού τύπου **AddAttribute** στο σενάριο. Παρατηρείστε πως εκτός από τα γνωρίσματα εισόδου και εξόδου (πάνω αριστερά και δεξία στη φόρμα), ο χρήστης μπορεί να ορίσει επακριβώς το μετασχηματισμό εισάγοντας και τις απαιτούμενες εκφράσεις για την πλήρωση του μετασχηματισμού, αλλά και σύντομα σχόλια. Όπως θα θυμάται ο αναγνώστης από προηγούμενο κεφάλαιο, η πληροφορία αυτή δεν τοποθετείται στο μετασχηματισμό, αλλά σε ένα σχόλιο προσαρτημένο σε αυτό. Ακριβώς αυτό συμβαίνει και στην περίπτωση αυτή. Ο μετασχηματισμός χαρακτηρίζεται $f3$ (όπως φαίνεται και στο σχήμα) και ταυτόχρονα τοποθετείται στο σενάριο και ένα σχόλιο το οποίο περιέχει τη συμπληρωματική πληροφορία, δηλαδή (α) τον τύπο του μετασχηματισμού (**f::AddAttribute**), (β) τις απαιτούμενες εκφράσεις για τον τύπο αυτό (**e::Date=SysDate()**), αλλά και (γ) σύντομα σχόλια (**t::Adds a new attribute to the scenario, with a value of System Date**).



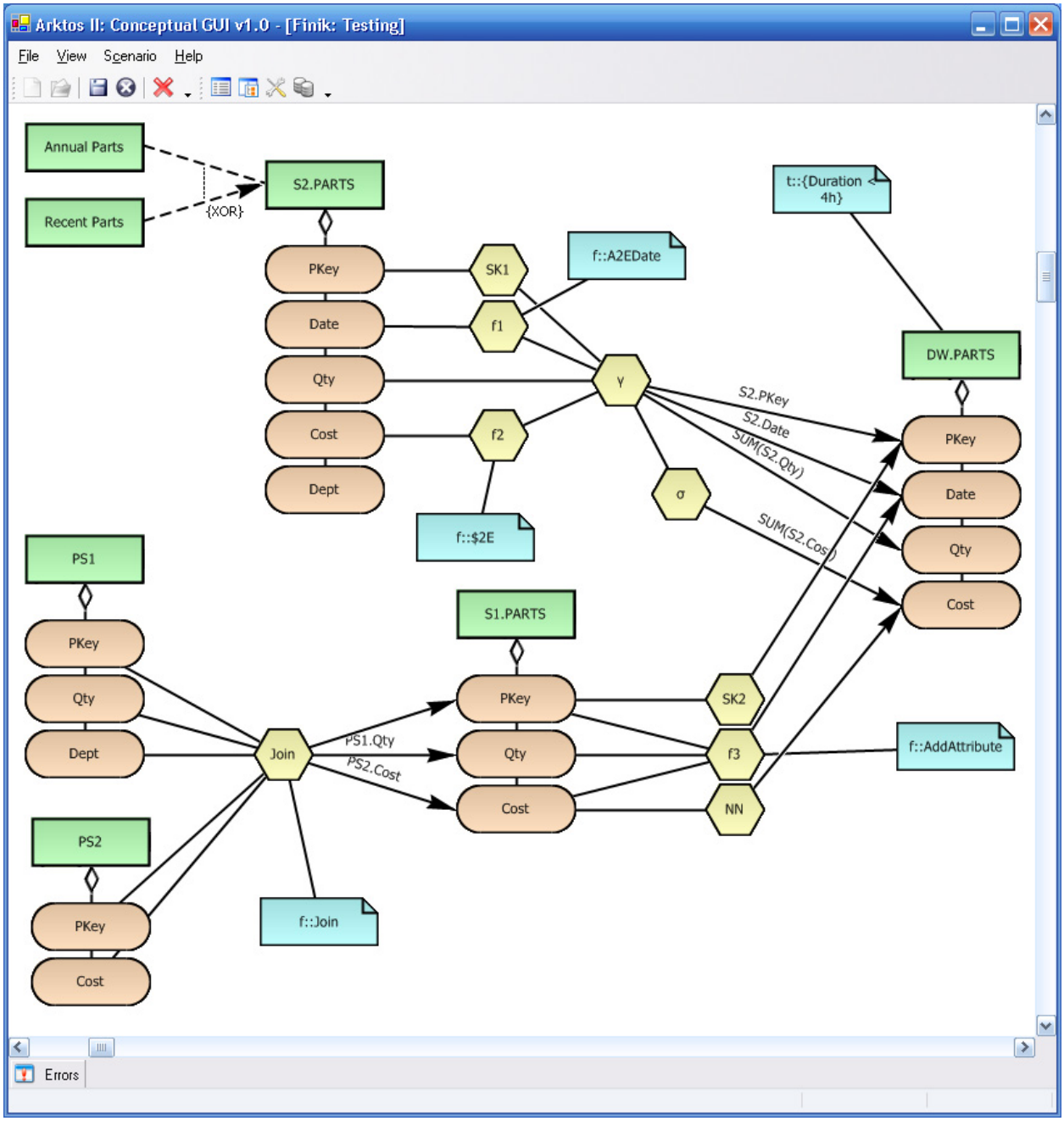
Σχήμα 6.10: Εισαγωγή νέου μετασχηματισμού στο σενάριο

Στο Σχήμα 6.12 φαίνεται τμήμα του Παραδείγματος 1. Φαίνεται η πλήρωση της έννοιας **DW.PARTS** από την έννοια **S2.PARTS**, μέσω των κατάλληλων μετασχηματισμών. Κατά την κατασκευή του εργαλείου, παρατηρήσαμε ότι η ύπαρξη στα σχόλια της πλήρους πληροφορίας, όπως αυτή παρουσιάστηκε

στο Κεφάλαιο 2, επιβάρυνε σημαντικά το σενάριο, έτσι αποφασίσαμε όπως η πληροφορία αυτή να παραμείνει κρυμμένη, και να εμφανίζεται όταν το επιθυμεί ο σχεδιαστής (Σχήμα 6.11).



Σχήμα 6.11: Εμφάνιση κρυμμένης πληροφορίας για το μετασχηματισμό **f3** που ορίζεται στο Σχήμα 6.10



Σχήμα 6.12: Τμήμα του Παραδείγματος 1, σχεδιασμένο στο εργαλείο που κατασκευάσαμε

7

Υλοποίηση

7.1 Πλατφόρμες και Προγραμματιστικά Εργαλεία

7.1.1 Επιλογή Γλώσσας Προγραμματισμού — Περιβάλλοντος Ανάπτυξης

Η ανάπτυξη του εργαλείου γραφικής αναπαράστασης εννοιολογικών σεναρίων ΕΜΦ έγινε σε Visual Basic .NET. Πιο συγκεκριμένα, χρησιμοποιήσαμε το περιβάλλον ανάπτυξης Visual Studio .NET 2003 της Microsoft. Η επιλογή της γλώσσας και του περιβάλλοντος ανάπτυξης στηρίχθηκε στα εξής:

- Το API (Application Interface) που προσφέρεται για τη διασύνδεση καθώς και την ανάπτυξη του γραφικού περιβάλλοντος (GUI) της εφαρμογής αποτέλεσε το βασικό κριτήριο για την απόφαση αυτή. Με δεδομένο ότι η εφαρμογή αποτελεί ένα εργαλείο γραφικής σχεδίασης, η ύπαρξη ενός ισχυρού περιβάλλοντος δημιουργίας GUI, αλλά και η ύπαρξη της βιβλιοθήκης GDI+ για την ανάπτυξη των σχημάτων των κόμβων, ήταν καταλυτική για την επιλογή της συγκεκριμένης πλατφόρμας.
- Το περιβάλλον ανάπτυξης του Visual Studio .NET είναι ένα πολύ δυνατό εργαλείο που διευκολύνει σημαντικά την διαδικασία ανάπτυξης και ελέγχου μίας εφαρμογής.
- Η ύπαρξη του πακέτου ADO.NET στο Visual Studio .NET, μας επέτρεπε την εύκολη διασύνδεση με βάσεις δεδομένων σχεδόν κάθε τύπου, επιτρέποντας στην τελική εφαρμογή να λειτουργεί κάτω από διαφορετικές συνθήκες χωρίς τη συγγραφή επιπλέον κώδικα.

7.1.2 Επιλογή Συστήματος Διαχείρισης Βάσεων Δεδομένων

Για Σύστημα Διαχείρισης Βάσεων Δεδομένων επιλέχθηκε ο Microsoft SQL Server 2000, ο οποίος αποτελεί ένα αρκετά πλήρες σύστημα, το οποίο μεταξύ άλλων συνεργάζεται πολύ καλά με το Visual Studio .NET. Επιπλέον, έχει τα εξής επιθυμητά χαρακτηριστικά:

- Συνοδεύεται από πλήρη και κατατοπιστική τεκμηρίωση, η οποία παρέχεται ως μέρος του συστήματος, αλλά και ως μέρος του Visual Studio .NET.
- Είναι ένα ευρέως διαδεδομένο Σύστημα Διαχείρισης Βάσεων Δεδομένων.

7.2 Λεπτομέρειες Υλοποίησης

7.2.1 Ανάλυση Κώδικα

Όπως αναφέρθηκε και στο Κεφάλαιο 6, η εφαρμογή είναι χωρισμένη σε τμήματα (στην ορολογία που επικρατεί στο .NET Framework, δεν ονομάζονται packages, αλλά *namespaces*). Τα τμήματα στα οποία χωρίζεται ο κώδικάς μας είναι:

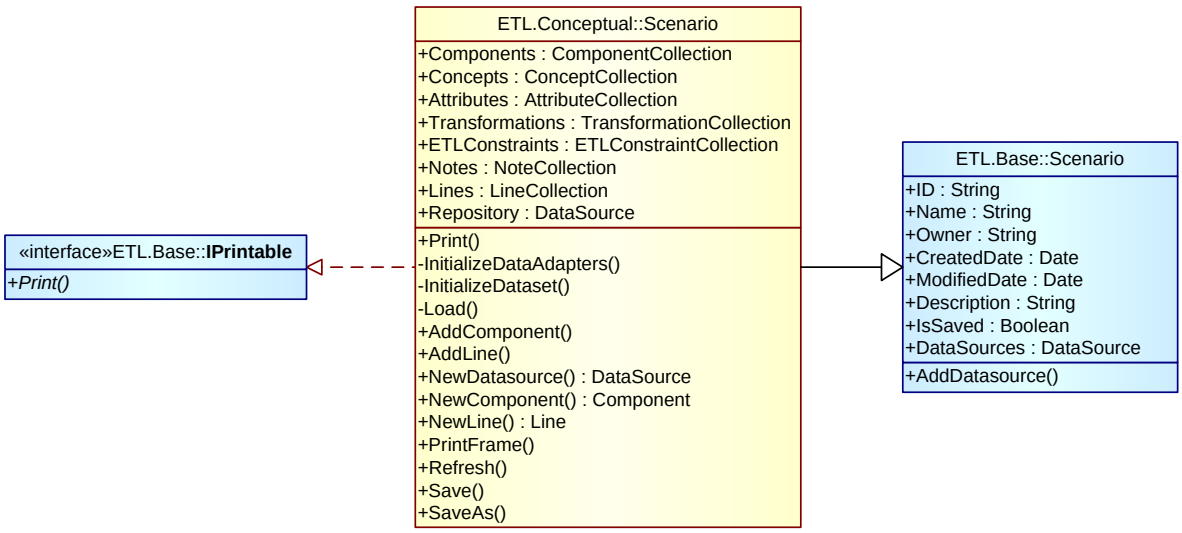
1. ETL.Base
2. ETL.Conceptual
3. ETL.Designer
   (a) ETL.Designer.Controls
      i. ETL.Designer.Controls.ComponentTree
      ii. ETL.Designer.Controls.DataTree
   (b) ETL.Designer.Forms
4. ETL.Tools

Η περιγραφή των παραπάνω τμημάτων έγινε στην ενότητα 6.2. Σε αυτή την ενότητα θα δώσουμε περισσότερες λεπτομέρειες σχετικά με την υλοποίησή τους. Κάθε τμήμα μπορεί να διαχωριστεί με τη σειρά του στις διάφορες κλάσεις και λειτουργικές μονάδες (Modules) που περιλαμβάνει.

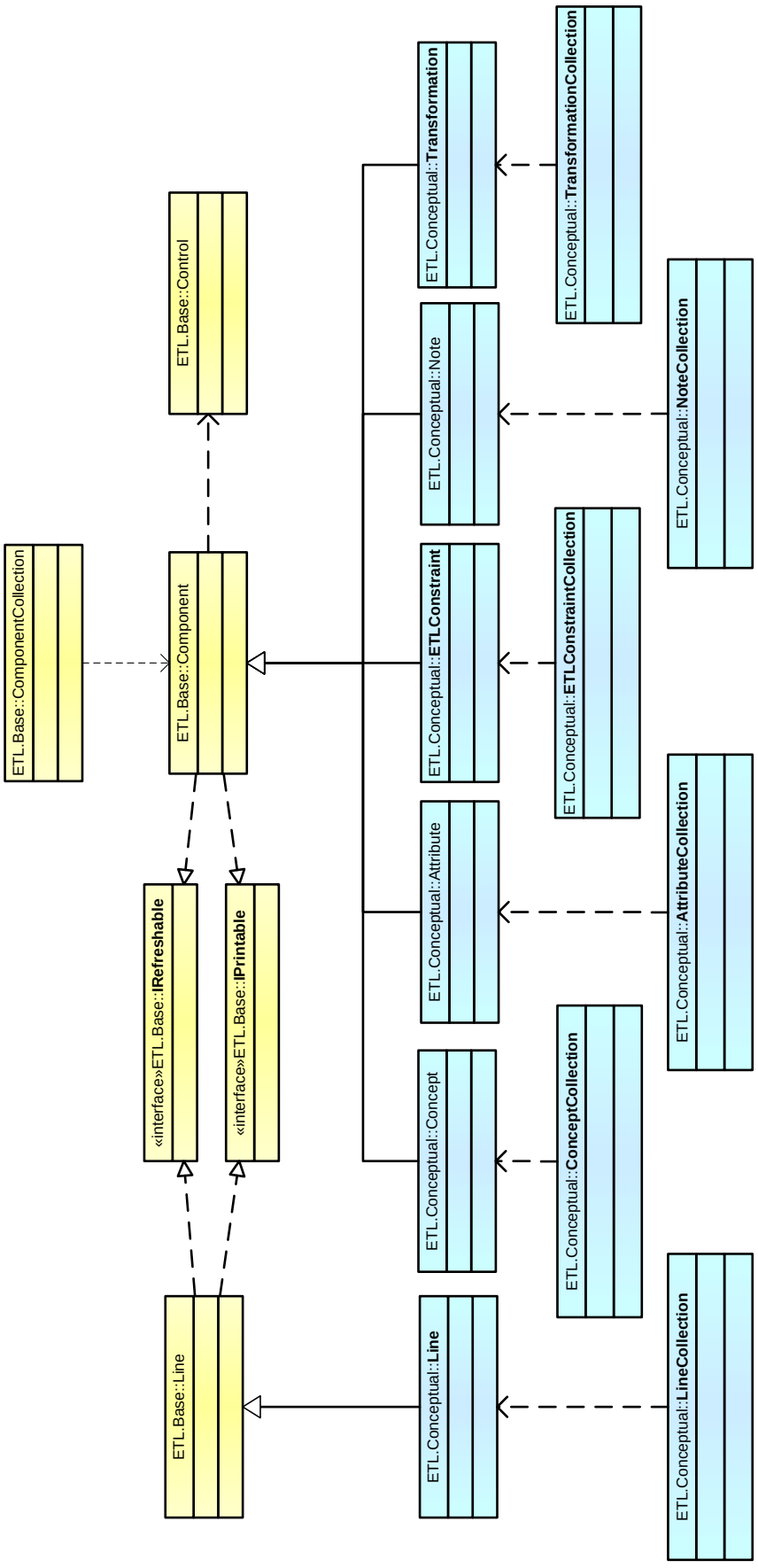
7.2.2 Διαγράμματα UML

Στη συνέχεια υπάρχουν σχήματα που απεικονίζουν τον τρόπο διασύνδεσης των κλάσεων της εφαρμογής. Στο Σχήμα 7.2 απεικονίζεται το στατικό διάγραμμα των κλάσεων που περιλαμβάνονται στα τμήματα **ETL.Base** και **ETL.Conceptual** και που σχηματίζουν το μοντέλο σχεδίασης. Από τις κλάσεις αυτές απουσιάζουν οι συναρτήσεις και οι μεταβλητές τους, οι οποίες θα παρουσιαστούν στη συνέχεια αναλυτικά.

Επίσης, από το Σχήμα 7.2 απουσιάζουν οι κλάσεις **ETL.Base::Scenario** και **ETL.Conceptual::Scenario**. Η παράληψη αυτή έγινε για σκοπούς καθαρότητας του διαγράμματος, διότι οι παραπάνω δύο κλάσεις εξαρτώνται από όλες, σχεδόν, τις άλλες κλάσεις. Επιπλέον, οι κλάσεις αυτές είναι πολύ σημαντικές στην υλοποίηση της εφαρμογής, έτσι εμφανίζονται μόνες τους, αναλυτικά με όλες τις συναρτήσεις τους, στο Σχήμα 7.1.



Σχήμα 7.1: Οι κλάσεις **ETL.Base.Scenario** και **ETL.Conceptual.Scenario**



Σχήμα 7.2: Οι κλάσεις **ETL.Base** και **ETLConceptual**

7.3 Περιγραφή Κώδικα

7.3.1 Διεπαφές

Η γλώσσα προγραμματισμού Visual Basic .NET, όπως οι περισσότερες σύγχρονες γλώσσες, δεν επιτρέπει την πολλαπλή κληρονομικότητα. Με δεδομένο αυτό, ο μόνος τρόπος να προστεθούν κατά ενιαίο τρόπο, κάποιες συναρτήσεις τόσο στους κόμβους, όσο και στις ακμές, ήταν μέσω διεπαφών.

ETL.Base.IPrintable

```
Public Interface IPrintable
  Sub Print(ByVal g As System.Drawing.Graphics)
End Interface
```

Η διεπαφή (interface) αυτή περιέχει τη συνάρτηση **Print** η οποία χρησιμοποιείται για την εκτύπωση κάποιου κόμβου ή κάποιας ακμής του σεναρίου. Κατά συνέπεια, όλοι οι κόμβοι και όλες οι ακμές πρέπει να υλοποιούν τη συνάρτηση αυτή, έτσι ώστε να είναι δυνατή η εκτύπωση.

ETL.Base.IRefreshable

```
Public Interface IRefreshable
  Sub Refresh()
End Interface
```

Η διεπαφή αυτή περιέχει τη συνάρτηση **Refresh** η οποία χρησιμοποιείται για την ανανέωση κάποιου κόμβου ή κάποιας ακμής. Με την κλήση της συνάρτησης αυτής σε επίπεδο σεναρίου, όλοι οι κόμβοι, και όλες οι ακμές που αποτελούν το σενάριο ξαναδιαβάζουν τις ιδιότητές τους (όπου μπορεί να έχει γίνει κάποια αλλαγή) και επανασχεδιάζονται στην επιφάνεια σχεδιασμού.

7.3.2 Λειτουργικές Μονάδες (modules)— Απαριθμήσεις (enumerations)

ETL.MetaData

```
Public Module MetaData
  ...
End Module
```

Στη λειτουργική αυτή μονάδα περιέχονται μεταβλητές που έχουν εμβέλεια όλη την εφαρμογή, δομές δεδομένων, απαριθμήσεις (enumerations) καθώς και βοηθητικές συναρτήσεις και κλάσης που απαιτούνται σε διάφορα σημεία της εφαρμογής.

Δομές Δεδομένων

```
Public Structure TransformationType
  Public Description As String
  Public Name As String
End Structure
```

Χρησιμοποιείται για τη δημιουργία λιστών με μετασχηματισμούς.

Απαριθμήσεις

```
Public Enum ComponentTypes
  Concept = 0
  Attribute
  Transformation
  Note
  ETLConstraint
End Enum
```

Οι διάφοροι τύποι των κόμβων που υποστηρίζονται στην εφαρμογή μας.

```
Public Enum EdgeSizes
  Small = 0
  Normal = 1
  Large = 2
End Enum
```

Τα διαθέσιμα μεγέθη για τα βέλη στις άκρες των ακμών.

```
Public Enum Themes
  Office2003 = 0
  OfficeXP
  WindowsDefault
End Enum
```

Οι διαθέσιμες επιλογές για την εμφάνιση της εφαρμογής.

```
Public Enum EdgeTypes
  None
  PartOf
  Provider
End Enum
```

Οι διαθέσιμοι τύποι για τα άκρα των ακμών.

```
Public Enum Direction
  North = 0
  NorthEast
  East
  SouthEast
  South
  SouthWest
  West
  NorthWest
  Center
End Enum
```

Τα διαθέσιμα σημεία για τη σύνδεση μίας ακμής σε κάποιο κόμβο.

```
Public Enum LineTypes
  Empty = 0
  ActiveCandidate
  Candidate
  PartOf
  Provider
  ProviderIn
  ProviderOut
  SerialComposition
  CollapsedIn
  CollapsedOut
End Enum
```

Οι διαθέσιμες επιλογές για τον τύπο μίας ακμής. Ανάλογα με τον τύπο της ακμής γίνεται και η επιλογή των άκρων της, το αν θα είναι συνεχής ή διακεκομμένη, τα σημεία σύνδεσής της με τους κόμβους και γενικά η όλη σχεδίασή της.

```
Public Enum ErrorCodes
  None = 0
  InputMissing = 1
  OutputMissing = 2
  NoActiveCandidate = 4
  SingleCandidate = 8
  SameInputOutputAttribute = 16
  NamingPrinciple = 32
  MultipleOutputSchema = 64
  InputsChanged = 128
  OutputsChanged = 256
End Enum
```

Οι διάφοροι κωδικοί σφάλματος που μπορούν να προκύψουν κατά τη διάρκεια της σχεδίασης ενός σεναρίου. Η επιλογή των κωδικών έγινε με τέτοιο τρόπο ώστε με τη χρήση των λογικών τελεστών AND, NOT και OR να μπορούμε να ελέγξουμε για την ύπαρξη, να ορίσουμε ή να αφαιρέσουμε ένα σφάλμα

σε κάποιο κόμβο με τη χρήση μίας και μόνο μεταβλητής.

```
Public Enum ErrorTypes
  FatalError = 0
  Warning
End Enum
```

Οι δύο κατηγορίες σφαλμάτων του εργαλείου. Σε περίπτωση σφάλματος της πρώτης κατηγορίας, το σενάριο *σίγουρα* δεν είναι σωστό, ενώ σε περίπτωση σφάλματος της δεύτερης κατηγορίας, το σενάριο *ενδέχεται* να μην είναι σωστό.

Συναρτήσεις

```
Public Function LoadUserPreferences(ByVal folder As String) As Boolean
```

Η συνάρτηση αυτή διαβάζει από το δίσκο το αρχείο με τις ρυθμίσεις της εφαρμογής. Σε περίπτωση που το αρχείο αυτό δεν βρεθεί, τότε καλεί τη συνάρτηση **CreateUserPreferences**.

```
Private Function CreateUserPreferences() As DataSet
```

Η συνάρτηση αυτή δημιουργεί εκ νέου το αρχείο ρυθμίσεων χρησιμοποιώντας την προκαθορισμένη επιλογή για κάθε ρύθμιση.

```
Public Sub SaveUserPreferences(ByVal folder As String)
```

Η συνάρτηση αυτή αναλαμβάνει την αποθήκευση του αρχείου με τις ρυθμίσεις στο δίσκο.

```
Public Sub SaveErrorReport(ByVal ex As Exception)
```

Η συνάρτηση αυτή είναι βοηθητική, και χρησιμοποιείται σε περιπτώσεις όπου έχουμε σφάλματα στην εκτέλεση του προγράμματος, όπως αδυναμία στη σύνδεση στη βάση κ.λ.π., και αποθηκεύει στο δίσκο μία λεπτομερή αναφορά του προβλήματος, έτσι ώστε τυχόν λάθη στον προγραμματισμό της εφαρμογής να μπορούν εύκολα να αναγνωριστούν και να διορθωθούν.

Κλάσεις

```
Public NotInheritable Class ComponentOptions
  ...
  Public Property Color() As System.Drawing.Color
  Public Property Size() As System.Drawing.Size
  ...
End Class
```

Η κλάση αυτή χρησιμοποιείται για την αποθήκευση των ιδιοτήτων *χρώματος* (**Color**) και *μεγέθους* (**Size**) για ένα κόμβο. Στη λειτουργική μονάδα **MetaData** υπάρχει ένα στιγμιότυπο αυτής της κλάσης για κάθε τύπο κόμβου (Έννοιας, Γνωρίσματος, Μετασχηματισμού, κ.λ.π.).

```
Public NotInheritable Class GlobalOptions
  ...
  Public Property PenWidth() As Integer
  Public Property LineColor() As System.Drawing.Color
  Public Property ForeColor() As System.Drawing.Color
  Public Property SelectionColor() As System.Drawing.Color
  Public Property ErrorColor() As System.Drawing.Color
  Public Property AntiAliasLines() As Boolean
  Public Property AntiAliasText() As Boolean
  Public Property LineWidth() As Integer
  Public Property EdgeSize() As EdgeSizes
  Public Property Theme() As Themes
  ...
End Class
```

Η κλάση αυτή χρησιμοποιείται για την αποθήκευση των ιδιοτήτων που αφορούν ολόκληρη την εφαρμογή. Περιέχει ιδιότητες για την αλλαγή του πάχους του περιγράμματος των κόμβων (**PenWidth**), το χρώμα των ακμών (**LineColor**), το χρώμα του κειμένου και του περιγράμματος των κόμβων (**ForeColor**), όπως και τα χρώματα του περιγράμματος όταν ο κόμβος είναι επιλελεγμένος (**SelectionColor**) ή παρουσιάζει κάποιο σφάλμα (**ErrorColor**).

Ακόμα, υπάρχουν ιδιότητες που ελέγχουν αν θα υπάρξει εξομάλυνση (antialias) των ακμών (**AntiAliasLines**) και του κειμένου (**AntiAliasText**). Τέλος, στην κλάση αυτή, μπορούν να αποθηκευτούν επιλογές του χρήστη που αφορούν το πάχος των ακμών (**LineWidth**), του μεγέθους των βελών στις άκρες των ακμών, αλλά και της εμφάνισης του παραθύρου της εφαρμογής (**Theme**).

Η λειτουργική μονάδα **MetaData** περιέχει ένα στιγμιότυπο της κλάσης αυτής, το οποίο δημιουργείται πριν από την εκτέλεση της εφαρμογής. Στο στιγμιότυπο αυτό φορτώνεται το αρχείο ρυθμίσεων, έτσι ώστε οι ρυθμίσεις που αφορούν τόσο τη σχεδίαση, όσο και τη συμπεριφορά της εφαρμογής να είναι άμεσα προσβάσιμες. Αμέσως μετά την έξοδο του χρήστη από την εφαρμογή, οι ρυθμίσεις που υπάρχουν στο στιγμιότυπο αυτό της κλάσης αποθηκεύονται ξανά στο δίσκο.

```
Public NotInheritable Class Datasource
  Implements IDisposable
    ...
End Class
```

Η κλάση αυτή χρησιμοποιείται για λειτουργίες που έχουν σχέση με τις πηγές δεδομένων και τη συμβολοακολουθία σύνδεσης σε αυτές. Για το σκοπό αυτό περιέχει τις ακόλουθες απαριθμήσεις, ιδιότητες και συναρτήσεις:

```
Public Enum Providers
  AS400 = 0                 ' AS/400
  ActiveDirectoryService    ' Active Directory Service
  DB2                       ' IBM DB2
  FoxPro                    ' Foxpro
  IndexServer               ' Index Server
  InternetPublishing        ' Internet Publishing
  MicrosoftAccess           ' Microsoft Access
  MicrosoftSQLServer        ' Microsoft SQL Server
  Oracle                    ' Oracle
  SimpleProvider            ' Simple Provider
End Enum
```

Η απαρίθμηση αυτή περιέχει τις πηγές δεδομένων στις οποίες μπορεί να συνδεθεί η εφαρμογή μας. Ανάλογα με την επιλογή του σχεδιαστή, η συμβολοακολουθία σύνδεσης (**ConnectionString**) δημιουργείται κατάλληλα, και χρησιμοποιείται για τη σύνδεση στην πηγή αυτή.

```
Public ReadOnly Property ID() As String
```

```
Public Property Name() As String
```

```
Public Property Location() As String
```

```
Public Property Provider() As Providers
```

```
Public Property Datasource() As String
```

```
Public Property UserID() As String
```

```
Public Property Password() As String
```

```
Public Property InitialCatalog() As String
```

```
Public Property PackageCollection() As String
```

```
Public Property UseIntegratedSecurity() As Boolean
```

Για την επιτυχή σύνδεση στην Αποθήκη Δεδομένων ή σε κάποια από τις πηγές δεδομένων του σεναρίου, απαιτείται μία συμβολοακολουθία σύνδεσης, η οποία σχηματίζεται τη στιγμή της σύνδεσης και είναι διαφορετική για κάθε πηγή δεδομένων. Εξαρτάται από τον τύπο του ΣΔΒΔ που χρησιμοποιείται σε κάθε περίπτωση, από το χαρακτηριστικό και τον κωδικό του χρήστη με τα οποία συνδεόμαστε, το όνομα της βάσης, κ.ο.κ. Για να γίνει εφικτή η εύκολη αλλαγή των στοιχείων αυτών (κυρίως για διόρθωση λαθών), επιλέχθηκε η αποθήκευση στο repository όχι της πλήρους συμβολοακολουθίας σύνδεσης, αλλά των επιμέρους τμημάτων που τη σχηματίζουν.

Το παραπάνω σύνολο από ιδιότητες μας επιτρέπουν την πρόσβαση στα δομικά στοιχεία της συμβολοακολουθίας σύνδεσης που περιγράψαμε παραπάνω. Δεν είναι απαραίτητο να υπάρχουν όλα, αντίθετα, κάθε τύπος πηγής δεδομένων απαιτεί την ύπαρξη μόνο τμήματος αυτών.

```
Public ReadOnly Property ConnectionString() As String
```

Η ιδιότητα αυτή επιστρέφει την πλήρη συμβολοακολουθία σύνδεσης, όπως σχηματίζεται από τις τιμές των παραπάνω ιδιοτήτων.

Εκτός από τις παραπάνω ιδιότητες, υπάρχουν και οι ακόλουθες στατικές (**static**) ιδιότητες:

```
Public Shared ReadOnly Property ProviderString(ByVal type As Providers) As String
```

```
Public Shared ReadOnly Property ProviderName(ByVal type As Providers) As String
```

```
Public Shared ReadOnly Property ConnectionString(ByVal provider As Providers, _
  ByVal userid As String, ByVal password As String, _
  ByVal datasource As String, ByVal initialCatalog As String, _
  ByVal packageCollection As String, ByVal location As String, _
  Optional ByVal integratedSecurity As Boolean = False) As String
```

Οι ιδιότητες αυτές χρησιμοποιούνται για σκοπούς ευκολίας, χωρίς να απαιτείται στιγμιότυπο της κλάσης αυτής. Η ιδιότητα **ProviderString** επιστρέφει τη συμβολοακολουθία που περιγράφει τον τύπο της πηγής δεδομένων. Η ιδιότητα **ProviderName** επιστρέφει μία σύντομη περιγραφή για τον τύπο της πηγής δεδομένων, έτσι ώστε όταν ο χρήστης καλείται να επιλέξει τον τύπο της πηγής δεδομένων, να μπορεί να το κάνει εύκολα, χωρίς να χρειάζεται να μαντεύει. Τέλος, η ιδιότητα **ConnectionString** δημιουργεί και επιστρέφει τη συμβολοακολουθία σύνδεσης σε κάποια πηγή δεδομένων με βάση τα στοιχεία που του δίνει ο χρήστης.

7.3.3 Κλάσεις

ETL.Base.Control

```
Public Class Control
  Inherits System.Windows.Forms.Control
  ...
End Class
```

Η κλάση αυτή περιέχει συναρτήσεις για:

- το σχεδιασμό κάποιου κόμβου (έννοιας, γνωρίσματος, μετασχηματισμού, κ.ο.κ.) την επιφάνεια σχεδίασης,
- τη μετακίνηση του αντικειμένου πάνω στην επιφάνεια σχεδίασης,
- τη μεταφορά των πράξεων του χρήστη στην υπόλοιπη εφαρμογή, μέσω γεγονότων.

ETL.Base.Component

```
Public MustInherit Class Component
  Implements IDisposable, IPrintable, IRefreshable
  ...
End Class
```

Αποτελεί αφηρημένη (abstract) κλάση, και λειτουργεί ως "περιτύλιγμα" (wrapper) για την κλάση **Control**. Σκοπός της είναι να ομαδοποιήσει όλα τα χαρακτηριστικά που υπάρχουν σε κάθε κόμβο και να κρύψει από το χρήστη όλα τα υπόλοιπα. Με την κλάση αυτή, δίνεται η δυνατότητα στο χρήστη να βλέπει και να αλλάζει τις ιδιότητες κάθε κόμβου, στο κατάλληλο παράθυρο στα δεξιά της περιοχής σχεδίασης (παράθυρο Properties). Παράλληλα, από την κλάση αυτή και με την προσθήκη των απαιτούμενων ιδιοτήτων και την προσαρμογή των συναρτήσεων σχεδιασμού, δημιουργούνται όλοι οι άλλοι κόμβοι (έννοια, γνώρισμα, κ.λ.π.) έτσι ώστε να επιτευχθεί το κατάλληλο αποτέλεσμα.

```
Protected Friend MustOverride Sub PaintBackground( _
  ByVal g As System.Drawing.Graphics, ByVal location As System.Drawing.Point)
```

```
Protected Friend MustOverride Sub Paint( _
  ByVal g As System.Drawing.Graphics, ByVal location As System.Drawing.Point)
```

Οι συναρτήσεις **PrintBackground** και **Paint** αποτελούν τις συναρτήσεις σχεδιασμού, τόσο στην οθόνη όσο και στον εκτυπωτή. Η πρώτη είναι απαραίτητη για τη σχεδίαση του φόντου κάθε κόμβου, ενώ η δεύτερη απαιτείται για τη σχεδίαση του περιγράμματος και του κειμένου. Οι συναρτήσεις αυτές δηλώνονται στην κλάση **Component** και υλοποιούνται σε κάθε κλάση–κόμβο στη συνέχεια. Για παράδειγμα, η κλάση **Attribute** που θα δούμε στη συνέχεια, και που αποτελεί επέκταση της κλάσης **Component**, υλοποιεί τις συναρτήσεις αυτές με τέτοιο τρόπο, ώστε να σχεδιάζουν ένα οβάλ σχήμα (το σύμβολο του γνωρίσματος), να το γεμίζουν με κατάλληλο χρώμα και να γράφουν στο κέντρο αυτού το όνομα του γνωρίσματος.

```
Public MustOverride Sub Refresh() Implements IRefreshable.Refresh
```

Αποτελεί τη συνάρτηση που ορίζεται στη διεπαφή **IRefreshable**. Όμως, λόγω της διαφορετικότητας των κόμβων, η συνάρτηση αυτή δεν υλοποιείται, αλλά ορίζεται ως **MustOverride**, επιβάλλοντας έτσι την υλοποίηση της σε όλες τις κλάσεις που προέρχονται από την κλάση **Component**.

```
Public Overridable Sub Print(ByVal g As System.Drawing.Graphics) _
  Implements IPrintable.Print
```

Αντίθετα με τη συνάρτηση της διεπαφής **IRefreshable**, η συνάρτηση της διεπαφής **IPrintable** υλοποιείται στην κλάση **Component**. Αυτό συμβαίνει διότι δεν απαιτείται κάτι συγκεκριμένο από αυτή την συνάρτηση εκτός από την κλήση των συναρτήσεων εκτύπωσης (**Paint** και **PaintBackground**) με κατάλληλες παραμέτρους (τη θέση του κόμβου στην επιφάνεια σχεδίασης).

```
Public MustOverride ReadOnly Property Type() As ComponentTypes
```

Η ιδιότητα **Type** επιστρέφει τον τύπο κάθε κόμβου, όπως αυτός ορίζεται κατά τη δημιουργία της αντίστοιχης κλάσης.

```
Public Overridable Sub AddInput(ByVal comp As ETL.Base.Component)
```

Προσθέτει ένα κόμβο στη συλλογή των κόμβων εισόδου.

```
Public Overridable Sub AddOutput(ByVal comp As ETL.Base.Component)
```

Προσθέτει ένα κόμβο στη συλλογή των κόμβων εξόδου.

ETL.Base.ComponentCollection

```
Public Class ComponentCollection
  Implements ICollection, IEnumerable
    ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από αναφορές προς κόμβους. Χρησιμοποιείται για να κρατήσουμε *αναφορές* προς κόμβους διαφορετικών τύπων (Έννοιες, Γνωρίσματα, Μετασχηματισμούς, κ.λ.π). Για τη δημιουργία της κλάσης αυτής, χρησιμοποιήσαμε δύο διεπαφές του .NET Framework, τις ICollection και IEnumerable.

Η διεπαφή **ICollection** παρέχει στην κλάση αυτή όλες τις απαραίτητες συναρτήσεις για να συμπεριφέρεται ως συλλογή. Οι κυριότερες από τις συναρτήσεις αυτές είναι:

```
Public Sub CopyTo(ByVal array As System.Array, ByVal index As Integer) _
  Implements System.Collections.ICollection.CopyTo
```

Εισάγει ένα σύνολο από κόμβους στη συλλογή.

```
Public ReadOnly Property Count() As Integer _
  Implements System.Collections.ICollection.Count
```

Επιστρέφει τον αριθμό των κόμβων που περιέχει η συλλογή αυτή.

Η διεπαφή **IEnumerable** παρέχει στην κλάση αυτή τη δυνατότητα να συμπεριφέρεται σαν διπλά συνδεδεμένη λίστα μέσω ενός iterator. Απαιτεί την υλοποίηση της συνάρτησης:

```
Public Function GetEnumerator() As System.Collections.IEnumerator _
  Implements System.Collections.IEnumerable.GetEnumerator
```

Επιστρέφει ένα iterator για τα στοιχεία της συλλογής.

Εκτός από τις συναρτήσεις που απαιτούνται από τις δύο αυτές διεπαφές, η κλάση αυτή περιέχει και μερικές συναρτήσεις για τη συμπεριφορά σαν λίστα και για την άμεση πρόσβαση στα στοιχεία της.

```
Public Function Add(ByVal value As Component) As Integer
```

Προσθέτει έναν κόμβο στη συλλογή αυτή και επιστρέφει τη θέση του.

```
Public Sub Clear()
```

Αφαιρεί όλους τους κόμβους από τη συλλογή.

```
Public Function Contains(ByVal value As Component) As Boolean
```

Ελέγχει αν υπάρχει κάποιος κόμβος στη συλλογή.

```
Public Function Contains(ByVal UID As String) As Boolean
```

Ελέγχει αν υπάρχει κόμβος με το συγκεκριμένο UID στη συλλογή.

```
Public Function IndexOf(ByVal value As Component) As Integer
```

Επιστρέφει τη θέση ενός συγκεκριμένου κόμβου στη στη συλλογή ή -1 σε περίπτωση που ο κόμβος αυτός δεν υπάρχει.

```
Public Function IndexOf(ByVal UID As String) As Integer
```

Επιστρέφει τη θέση ενός συγκεκριμένου κόμβου στη συλλογή, χρησιμοποιώντας το UID του. Σε περίπτωση που ο κόμβος αυτός δεν υπάρχει, επιστρέφει -1.

```
Default Public Property Item(ByVal index As Integer) As Component
```

Επιστρέφει τον κόμβο που βρίσκεται σε συγκεκριμένη θέση στη συλλογή.

```
Default Public ReadOnly Property Item(ByVal UID As String) As Component
```

Επιστρέφει τον κόμβο της συλλογής που έχει το συγκεκριμένο UID.

```
Public Sub Remove(ByVal value As Component)
```

Αφαιρεί τον κόμβο από τη συλλογή.

```
Public Sub RemoveAt(ByVal index As Integer)
```

Αφαιρεί τον κόμβο που βρίσκεται σε συγκεκριμένη θέση στη συλλογή.

DataSourceCollection

```
Public Class DataSourceCollection
  Implements ICollection, IEnumerable
    ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από πηγές δεδομένων. Σε κάθε **Scenario** υπάρχει ένα στιγμιότυπο της κλάσης αυτής, στο οποίο κρατούνται όλες οι πηγές δεδομένων που έχουν προστεθεί στο σενάριο αυτό. Οι συναρτήσεις που παρέχει η κλάση αυτή είναι όμοιες με αυτές που υπάρχουν στην κλάση **ComponentCollection** και δεν θα περιγραφούν ξανά.

ETL.Base.Line

```
Public MustInherit Class Line
  Inherits System.Windows.Forms.Control
  Implements IRefreshable, IPrintable
  ...
End Class
```

Η κλάση αυτή αποτελεί τη βάση για κάθε ακμή που υπάρχει στην εφαρμογή μας και έχει δηλωθεί ως αφηρημένη (abstract). Περιέχει ένα σύνολο από συναρτήσεις που καλούνται αυτόματα όταν δημιουργηθεί κάποιο συμβάν (όπως τη μετακίνηση του κόμβου στον οποίο συνδέονται κ.λ.π.), συναρτήσεις για τη σχεδίαση τόσο της ακμής, όσο και των άκρων της. Επιπλέον, περιέχει και ένα σύνολο από βοηθητικές στατικές συναρτήσεις.

```
Private Sub myBeginComponent_Moved(ByVal sender As Base.Component, _
  ByVal e As Base.ComponentEventArgs) Handles myBeginComponent.Moved
```

```
Private Sub myEndComponent_Moved(ByVal sender As Base.Component, _
  ByVal e As Base.ComponentEventArgs) Handles myEndComponent.Moved
```

```
Private Sub myBeginComponent_Disposed(ByVal sender As Component, _
  ByVal e As ComponentEventArgs) Handles myBeginComponent.Disposed
```

```
Private Sub myEndComponent_Disposed(ByVal sender As Component, _
  ByVal e As ComponentEventArgs) Handles myEndComponent.Disposed
```

```
Private Sub myBeginComponent_VisibilityChanged(ByVal sender As Component, _
  ByVal e As ComponentEventArgs) Handles myBeginComponent.VisibilityChanged
```

```
Private Sub myEndComponent_VisibilityChanged(ByVal sender As Component, _
  ByVal e As ComponentEventArgs) Handles myEndComponent.VisibilityChanged
```

Οι παραπάνω συναρτήσεις καλούνται αυτόματα όταν προκύψει κάποιο συμβάν σε κάποιο από τους κόμβους αρχής ή/και τέλους της ακμής αυτής. Οι πρώτες δύο (**myBeginComponent_Moved** και **myBeginComponent_Moved**) καλούνται όταν μετακινηθούν οι κόμβοι αρχής και τέλους της ακμής αντίστοιχα. Οι επόμενες δύο (**myBeginComponent_Disposed** και **myEndComponent_Disposed**) καλούνται όταν οι κόμβοι αυτοί διαγραφούν. Τέλος, οι τελευταίες δύο συναρτήσεις (**myBeginComponent_VisibilityChanged** και **myBeginComponent_VisibilityChanged**) καλούνται όταν κάποιος από τους κόμβους αρχής και τέλους ήταν ορατός και τώρα δεν είναι ή το αντίθετο. Αυτό γίνεται έτσι ώστε να μην έχουμε ακμές που να προέρχονται ή να καταλήγουν σε "αόρατους" κόμβους.

ETL.Base.Scenario

```
Public MustInherit Class Scenario
  Implements IDisposable
  ...
End Class
```

Η αφηρημένη κλάση αυτή αποτελεί τη βάση για το εννοιολογικό σενάριο που θα δημιουργήσουμε στη συνέχεια.

Η κλάση αυτή ορίζει μερικά συμβάντα, μέσω των οποίων ειδοποιούνται άλλες κλάσεις για το τι συμβαίνει κάθε στιγμή στο σενάριο αυτό.

```
Public Event Changed As ScenarioEventHandler
```

```
Public Event Saved As ScenarioEventHandler
```

```
Public Event ComponentAdded As ScenarioEventHandler
```

```
Public Event ComponentRemoved As ScenarioEventHandler
```

```
Public Event DataSourceAdded As ScenarioEventHandler
```

Η κλάση αυτή περιέχει τα παραπάνω συμβάντα που εκτελούνται όταν γίνει κάποια αλλαγή στο σενάριο (**Changed**), όταν το σενάριο αποθηκευτεί (**Saved**), όταν προστεθεί (**ComponentAdded**) ή όταν αφαιρεθεί (**ComponentRemoved**) ένας κόμβος από το σενάριο και, τέλος, όταν προστεθεί μία νέα πηγή δεδομένων (**DataSourceAdded**).

Επιπλέον, υπάρχουν και μερικές ιδιότητες που απαιτούνται για τον ορισμό ενός σεναρίου:

```
Public Property Name() As String
```

```
Public Property Owner() As String
```

```
Public Property CreatedDate() As Date
```

```
Public Property ModifiedDate() As Date
```

```
Public Property Description() As String
```

```
Public ReadOnly Property IsSaved() As Boolean
```

Μέσω των ιδιοτήτων αυτών, ο χρήστης μπορεί να δει και να αλλάξει, το όνομα του σεναρίου (**Name**), το όνομα του χρήστη στον οποίο ανήκει το σενάριο αυτό (**Owner**), την ημερομηνία δημιουργίας του σεναρίου (**CreatedDate**), την ημερομηνία που έγινε η τελευταία αλλαγή στο σενάριο αυτό (**ModifiedDate**), όπως και μία σύντομη περιγραφή του σεναρίου (**Description**). Επιπλέον, μπορούμε εύκολα να δούμε αν στο σενάριο υπάρχουν αλλαγές που δεν έχουν αποθηκευτεί (**IsSaved**).

```
Public ReadOnly Property DataSources() As ETL.Base.DataSourceCollection
```

```
Public Overridable Overloads Sub AddDataSource(ByVal ds As Datasource)
```

Τέλος, η κλάση αυτή περιέχει μία συλλογή από πηγές δεδομένων (**DataSources**) και μία συνάρτηση για την προσθήκη πηγών δεδομένων στη συλλογή αυτή (**AddDatasource**).

ETL.Conceptual.Attribute

```
Public Class Attribute
  Inherits ETL.Base.Component
    ...
End Class
```

Αποτελεί επέκταση της κλάσης **ETL.Base.Component**. Υλοποιώντας τις συναρτήσεις **PaintBackground** και **Paint** όπως απαιτείται από τη μητρική κλάση, η κλάση αυτή αποκτά δυνατότητες σχεδίασης στην οθόνη και ταυτόχρονα, σχεδίασης στον εκτυπωτή. Παράλληλα, προσθέτει τις παρακάτω συναρτήσεις και ιδιότητες:

```
Public Property ParentConcept() As ETL.Conceptual.Concept
```

Μέσω αυτής της ιδιότητας μπορούμε να έχουμε πρόσβαση στην *έννοια* στην οποία ανήκει το *γνώρισμα* αυτό.

```
Public Property DatasourceID() As String
```

Η ιδιότητα αυτή επιστρέφει τον κωδικό που αντιστοιχεί στην πηγή δεδομένων από την οποία προέρχεται το συγκεκριμένο γνώρισμα. Μέσω της ιδιότητας αυτής μπορούμε να αλλάξουμε τον κωδικό αυτό, αντιστοιχώντας το γνώρισμα αυτό με κάποια άλλη πηγή δεδομένων.

```
Public Property DataAddress() As String
```

Η ιδιότητα αυτή επιστρέφει μία *διεύθυνση* που αντιστοιχεί μονοσήμαντα με κάποιο γνώρισμα κάποιας πηγής δεδομένων. Η διεύθυνση αυτή ορίζεται ως ακολούθως:

```
<COLUMN_NAME>.<TABLE_NAME>@<DATASOURCE_ID>
```

Όπου DATASOURCE_ID το πρωτεύον κλειδί της πηγής δεδομένων στην οποία ανήκει το γνώρισμα αυτό (*βλέπε* Ενότητα 7.4.2), TABLE_NAME το όνομα του πίνακα στον οποίο ανήκει η στήλη αυτή και COLUMN_NAME το όνομα της στήλης, σε περίπτωση σχεσιακής βάσης δεδομένων. Σε περίπτωση αρχείου κειμένου ή αρχείου COBOL, TABLE_NAME και COLUMN_NAME αντιστοιχούν στο όνομα του αρχείου και στο όνομα του γνωρίσματος αντίστοιχα.

```
Public Property DataType() As Tools.Data.DBType
```

Η ιδιότητα αυτή παρέχει πρόσβαση στον τύπο δεδομένων του γνωρίσματος αυτού. Οι διαθέσιμοι τύπου παρέχονται από το ADO.NET με τη μορφή ακεραίων αριθμών, τους οποίους εμείς αντιστοιχήσαμε μέσω του API Documentation του Visual Studio .NET σε μία απαρίθμηση, έτσι ώστε να μπορούμε εύκολα να ξέρουμε σε τι αντιστοιχεί ο κάθε αριθμός.

```
Public ReadOnly Property DBType() As String
```

Επιστρέφει σε μορφή κειμένου τον τύπο του γνωρίσματος αυτού. Η ιδιότητα αυτή δεν χρειαζόταν για τη λειτουργία του προγράμματος, αλλά θεωρήσαμε χρήσιμο να μπορεί ο χρήστης να μπορεί να βλέπει τον τύπο των δεδομένων κάθε γνωρίσματος εύκολα κατά τη διάρκεια του σχεδιασμού.

ETL.Conceptual.AttributeCollection

```
Public Class AttributeCollection
  Implements ICollection, IEnumerable
    ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από γνωρίσματα (attributes). Περιέχει τις αντίστοιχες ιδιότητες και συναρτήσεις που περιέχει και η κλάση **ETL.Base.ComponentCollection**, έτσι δεν θα περιγραφεί εκ νέου.

ETL.Conceptual.Concept

```
Public Class Concept
  Inherits ETL.Base.Component
    ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Component** υλοποιώντας ένα κόμβο τύπου *έννοιας*. Όπως και η κλάση **ETL.Conceptual.Attribute** (και όλες οι κλάσεις που υλοποιούν κόμβους και θα αναφερθούν στη συνέχεια), υλοποιεί της συναρτήσεις **Paint** και **PaintBackground** για να σχεδιάσει στην οθόνη και στον εκτυπωτή. Επιπλέον προσθέτει τις ακόλουθες ιδιότητες και συναρτήσεις:

```
Public Property Attributes() As AttributeCollection
```

Η ιδιότητα αυτή μας δίνει πρόσβαση σε μία συλλογή που περιλαμβάνει τα γνωρίσματα της έννοιας αυτής.

```
Public ReadOnly Property Candidates() As ETL.Conceptual.ConceptCollection
```

Η ιδιότητα αυτή επιστρέφει τη συλλογή που περιλαμβάνει τις έννοιες που είναι υποψήφιες για την πλήρωση της έννοιας αυτής.

```
Public Property DataSourceID() As String
```

Η ιδιότητα αυτή επιστρέφει τον κωδικό που αντιστοιχεί στην πηγή δεδομένων από την οποία προέρχεται η συγκεκριμένη έννοια.

```
Public Property DataAddress() As String
```

Η ιδιότητα αυτή επιστρέφει τη διεύθυνση που αντιστοιχεί σε αυτή την έννοια. Η διεύθυνση αυτή έχει τη μορφή:

```
<TABLE_NAME>@<DATASOURCE_ID>
```

όπου DATASOURCE_ID το πρωτεύον κλειδί της πηγής δεδομένων στην οποία ανήκει η έννοια αυτή και TABLE_NAME το όνομα του πίνακα (σε περίπτωση σχεσιακής βάσης δεδομένων) ή του αρχείου (σε περίπτωση αρχείων κειμένου ή αρχείων COBOL) στο οποίο αντιστοιχεί η έννοια αυτή.

```
Public ReadOnly Property IsCollapsed() As Boolean
```

Η ιδιότητα αυτή επιστρέφει **True** σε περίπτωση που τα γνωρίσματα της έννοιας αυτής είναι κρυμμένα και **False** σε κάθε άλλη περίπτωση.

```
Public ReadOnly Property HiddenLines() As ETL.Conceptual.LineCollection
```

Μέσω της ιδιότητας αυτής έχουμε πρόσβαση σε μία συλλογή *κρυμμένων* ακμών. Οι ακμές αυτές αντιστοιχούν στις ακμές που συνδέουν έννοιες και μετασχηματισμούς (ή έννοιες με έννοιες), όταν τα γνωρίσματα τους είναι κρυμμένα (*βλέπε* Ενότητα 2.1.1).

```
Public Property IsCandidate() As Boolean
```

Η ιδιότητα αυτή επιτρέπει τον διαχωρισμό των κλάσεων τύπου **Concept** σε *έννοιες* (**False**) και *υποψήφιες* έννοιες (**True**).

```
Public Property IsActiveCandidate() As Boolean
```

Μέσω της ιδιότητας αυτής μπορούμε να ξεχωρίσουμε ένα *υποψήφιο* από ένα *ενεργό υποψήφιο*. Στην πρώτη περίπτωση έχει την τιμή **False** ενώ στη δεύτερη, την τιμή **True**.

```
Public Property ActiveCandidate() As ETL.Conceptual.Concept
```

Η ιδιότητα αυτή μας παρέχει πρόσβαση στην έννοια που αποτελεί τον ενεργό υποψήφιο για την έννοια αυτή. Σε περίπτωση που δεν έχουν οριστεί υποψήφιοι για την έννοια αυτή, ή κανένας από τους υποψήφιους αυτούς δεν έχει χαρακτηριστεί ως ενεργός, τότε επιστρέφεται **Nothing**.

```
Public Sub AddAttribute(ByVal attr As ETL.Conceptual.Attribute, _
  Optional ByVal isNew As Boolean = True)
```

Η συνάρτηση αυτή προσθέτει το δοθέν γνώρισμα στη συλλογή γνωρισμάτων της έννοιας αυτής. Σε περίπτωση που το γνώρισμα μόλις έχει δημιουργηθεί (**isNew=True**), τότε δημιουργεί και τη γραμμή που συνδέει την έννοια αυτή με το γνώρισμα.

```
Public Sub RemoveAttribute(ByVal attr As ETL.Conceptual.Attribute)
```

Η συνάρτηση αυτή αφαιρεί ένα γνώρισμα από τη συλλογή των γνωρισμάτων της έννοιας αυτής.

```
Public Sub RefreshAttributes()
```

Η συνάρτηση αυτή καλείται για να ανανεώσει τη σχεδίαση των γνωρισμάτων της έννοιας. Η τοποθέτηση των γνωρισμάτων μίας έννοιας γίνεται αυτόματα από το πρόγραμμα, έτσι, απαιτείται μία συνάρτηση που να αναλαμβάνει αυτή ακριβώς την εργασία. Καλείται όταν μετακινηθεί η έννοια ή όταν προσθέσουμε/αφαιρέσουμε κάποιο γνώρισμα.

```
Public Sub RefreshCandidates()
```

Η συνάρτηση αυτή λειτουργεί όπως ακριβώς και η προηγούμενη (**RefreshAttribures**), με μόνη διαφορά ότι ενεργεί στους *υποψηφίους* της έννοιας αυτής και όχι στα γνωρίσματά της.

```
Public Sub AddCandidate(ByVal con As ETL.Conceptual.Concept, _
  Optional ByVal isNew As Boolean = True)
```

Μέσω της συνάρτησης αυτής γίνεται η προσθήκη νέων υποψηφίων στη συλλογή των υποψηφίων (**Candidates**) της έννοιας αυτής. Σε περίπτωση που έχουμε νέο υποψήφιο (**isNew=True**), τότε δημιουργείται και η ακμή που το συνδέει με την έννοια αυτή.

```
Public Sub RemoveCandidate(ByVal con As ETL.Conceptual.Concept)
```

Η συνάρτηση αυτή αφαιρεί μία υποψήφια έννοια από τη συλλογή των υποψηφίων για την έννοια.

```
Public Sub Expand()
```

```
Public Sub Collapse()
```

Οι δύο αυτές συναρτήσεις αναλαμβάνουν το έργο της εμφάνισης ή μη των γνωρισμάτων της έννοιας. Με κλήση της πρώτης (**Expand**) τα γνωρίσματα της έννοιας (και τυχόν σχέσεις τους με άλλα γνωρίσματα ή/και μετασχηματισμούς) εμφανίζονται στο σχεδιαστή. Αντίθετα, με κλήση της δεύτερης (**Collapse**), έχουμε απόκρυψη από το χρήστη/σχεδιαστή αυτής της λεπτομέρειας.

Παράλληλα με τις παραπάνω συναρτήσεις, η κλάση **Concept** περιέχει και μερικά *συμβάντα* (**Events**):

```
Public Event Collapsed As ConceptEventHandler
```

```
Public Event Expanded As ConceptEventHandler
```

```
Public Event BeforeActiveCandidateChanged As ConceptEventHandler
```

```
Public Event AfterActiveCandidateChanged As ConceptEventHandler
```

Τα συμβάντα αυτά εκτελούνται όταν κρύψουμε τα γνωρίσματα της έννοιας (**Collapsed**), όταν τα εμφανίσουμε (**Expanded**), αμέσως πριν την αλλαγή του ενεργού υποψήφιου της έννοιας (**BeforeActiveCandidateChanged**) και αμέσως μετά την αλλαγή (**AfterActiveCandidateChanged**). Τα συμβάντα αυτά υλοποιήθηκαν για την ειδοποίηση των συναρτήσεων σχεδίασης και ελέγχου που υπάρχουν σε άλλες κλάσεις και στο σχεδιαστικό περιβάλλον.

ETL.Conceptual.ConceptCollection

```
Public Class ConceptCollection
  Implements ICollection, IEnumerable
  ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από *έννοιες* (concepts). Περιέχει αντίστοιχες ιδιότητες και συναρτήσεις με αυτές που περιέχει η κλάση **ETL.Base.ComponentCollection**.

ETL.Conceptual.ETLConstraint

```
Public Class ETLConstraint
  Inherits ETL.Base.Component
  ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Component** υλοποιώντας ένα κόμβο τύπου *περιορισμού ΕΜΦ*. Υλοποιεί τις συναρτήσεις **Paint** και **PaintBackground** για να σχεδιάσει στην οθόνη και στον εκτυπωτή. Επιπλέον, προσθέτει τις ακόλουθες ιδιότητες και συναρτήσεις:

```
Public Property Attributes() As ETL.Conceptual.AttributeCollection
```

```
Public Property ConstraintType() As String
```

```
Public Sub AddAttribute(ByVal attr As ETL.Conceptual.Attribute, _
  Optional ByVal isNew As Boolean = False)
```

```
Public Sub RemoveAttribute(ByVal attr As ETL.Conceptual.Attribute)
```

Μέσω της ιδιότητας **Attributes** έχουμε πρόσβαση στη συλλογή με τα γνωρίσματα στα οποία επιβάλλεται ο περιορισμός ΕΜΦ και μέσω της ιδιότητας **ConstraintType** περιγράφεται ο τύπος του περιορισμού αυτού. Μέσω των συναρτήσεων **AddAttribute** και **RemoveAttribute** μπορούμε να προσθέσουμε και να αφαιρέσουμε ένα γνώρισμα στη συλλογή των γνωρισμάτων, αντίστοιχα.

```
Public Event AttributeAdded As ETL.Base.Component.ComponentEventHandler
```

```
Public Event AttributeRemoved As ETL.Base.Component.ComponentEventHandler
```

Εκτός από τις παραπάνω συναρτήσεις και ιδιότητες, η κλάση αυτή ορίζει και μερικά συμβάντα για την εύκολη ενημέρωση άλλων κλάσεων για το τι συμβαίνει στην κλάση αυτή. Συγκεκριμένα, ορίζουμε τα συμβάντα **AttributeAdded** και **AttributeRemoved** τα οποία εκτελούνται όταν προστεθεί ή αφαιρεθεί ένα γνώρισμα στη συλλογή των γνωρισμάτων αντίστοιχα.

ETL.Conceptual.ETLConstraintCollection

```
Public Class ETLConstraintCollection
  Implements ICollection, IEnumerable
  ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από *περιορισμούς ΕΜΦ* (ETL Constraints). Περιέχει αντίστοιχες συναρτήσεις και ιδιότητες με αυτές που περιέχει η κλάση **ETL.Base.ComponentCollection**, έτσι δεν θα περιγραφεί ξανά.

ETL.Conceptual.Note

```
Public Class Note
  Inherits ETL.Base.Component
  ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Component** υλοποιώντας ένα κόμβο τύπο *σχολίου*. Υλοποιεί τις συναρτήσεις **Paint** και **PaintBackground** για να σχεδιάσει στην οθόνη και τον εκτυπωτή. Επιπλέον, προσθέτει και την ακόλουθη ιδιότητα:

```
Public Property ParentTransformation() As ETL.Conceptual.Transformation
```

Μέσω της ιδιότητας αυτής, ο σχεδιαστής μπορεί να δει ή και να αλλάξει τον μετασχηματισμό στον οποίο αναφέρεται το σχόλιο αυτό.

ETL.Conceptual.NoteCollection

```
Public Class NoteCollection
  Implements ICollection, IEnumerable
  ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από *σχόλια* (Notes). Περιέχει αντίστοιχες συναρτήσεις και ιδιότητες με αυτές που περιέχει η κλάση **ETL.Base.ComponentCollection**.

ETL.Conceptual.Transformation

```
Public Class Transformation
  Inherits ETL.Base.Component
  ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Component** υλοποιώντας ένα κόμβο τύπου *μετασχηματισμού*. Υλοποιεί τις συναρτήσεις **Paint** και **PaintBackground** για να σχεδιάσει στην οθόνη και στον εκτυπωτή. Επιπλέον, προσθέτει τις ακόλουθες ιδιότητες και συναρτήσεις:

```
Public Property Notes() As ETL.Conceptual.NoteCollection
Public ReadOnly Property TransformationType() As String
Public Sub SetTransformationType(ByVal type As String)
Public Overrides Sub AddInput(ByVal comp As ETL.Base.Component)
Public Overrides Sub AddOutput(ByVal comp As ETL.Base.Component)
Public Sub AddNote(ByVal note As ETL.Conceptual.Note, _
  Optional ByVal isNew As Boolean = True)
Public Sub RemoveInput(ByVal comp As ETL.Base.Component)
Public Sub RemoveOutput(ByVal comp As ETL.Base.Component)
Public Property Expressions() As String
Public Property Description() As String
```

Με τις ιδιότητες **Notes** και **TransformationType** έχουμε πρόσβαση στη συλλογή με τα σχόλια που αφορούν το μετασχηματισμό και στον τύπο του μετασχηματισμού, αντίστοιχα. Ο τύπος του μετασχηματισμού μπορεί να αλλάξει εύκολα με κλήση της συνάρτησης **SetTransformationType**, ενώ μέσω των συναρτήσεων **AddInput** και **AddOutput** μπορούμε να ορίσουμε ένα κόμβο ως κόμβο εισόδου ή ως κόμβο εξόδου. Η συνάρτηση **AddNote** προσθέτει ένα σχόλιο στη συλλογή με τα σχόλια του μετασχηματισμού αυτού. Μέσω των συναρτήσεων **RemoveInput** και **RemoveOutput** μπορούμε να αφαιρέσουμε κόμβους από τις συλλογές των κόμβων εισόδου και εξόδου αντίστοιχα. Τέλος, οι ιδιότητες **Expressions** και **Description** μας δίνουν πρόσβαση στα expressions του μετασχηματισμού, καθώς και μία σύντομη περιγραφή αυτού, αντίστοιχα.

ETL.Conceptual.TransformationCollection

```
Public Class TransformationCollection
  Implements ICollection, IEnumerable
  ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από *μετασχηματισμούς* (Transformations). Περιέχει αντίστοιχες ιδιότητες και συναρτήσεις με αυτές που περιέχει η κλάση **ETL.Base.Component** και έτσι δεν θα περιγραφεί ξανά.

ETL.Conceptual.Line

```
Public Class Line
  Inherits ETL.Base.Line
  ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Line** υλοποιώντας μία *ακμή*. Υλοποιεί τις συναρτήσεις **PaintForeground** και **PaintBackground** για το σχεδιασμό στην οθόνη και στον εκτυπωτή. Επιπλέον, προσθέτει τις ακόλουθες ιδιότητες:

```
Public Property LineType() As LineTypes
```

Επιτρέπει την πρόσβαση στον τύπο της ακμής. Μέσω της ιδιότητας αυτής ο χρήστης μπορεί να δει και να αλλάξει τον τύπο της ακμής, με την αλλαγή του αυτή να είναι άμεσα ορατή στη σχεδίαση της ακμής στην οθόνη.

```
Public Property BeginComponent() As ETL.base.Component
```

Μέσω της ιδιότητας αυτής μπορούμε να δούμε ή να ορίσουμε από ποιο κόμβο ξεκινά η ακμή αυτή.

```
Public Property EndComponent() As ETL.Base.Component
```

Μέσω της ιδιότητας αυτής μπορούμε να δούμε ή να ορίσουμε σε ποιο κόμβο τερματίζεται η ακμή αυτή.

ETL.Conceptual.LineCollection

```
Public Class LineCollection
  Implements ICollection, IEnumerable
  ...
End Class
```

Η κλάση αυτή αποτελεί μία συλλογή από *ακμές* (Lines). Εκτός από τις ιδιότητες και συναρτήσεις που περιέχονται και στην κλάση **ETL.Base.ComponentCollection** περιέχει και τις ακόλουθες:

```
Public Function Contains(ByVal beginComp As ETL.Base.Component, _
  ByVal endComp As ETL.Base.Component) As Boolean
```

Ελέγχει αν υπάρχει στη συλλογή ακμή που να ξεκινά από τον κόμβο **beginComp** και να καταλήγει στον κόμβο **endComp**.

```
Public Function IndexOf(ByVal beginComp As ETL.Base.Component, _
  ByVal endComp As ETL.Base.Component) As Integer
```

Επιστρέφει τη θέση μίας ακμής που ξεκινά από τον κόμβο **beginComp** και καταλήγει στον κόμβο **endComp**, στη συλλογή. Σε περίπτωση που η ακμή αυτή δεν υπάρχει στη συλλογή, επιστρέφει την τιμή -1.

```
Default Public ReadOnly Property Item(ByVal beginComp As ETL.Base.Component, _
  ByVal endComp As ETL.Base.Component) As ETL.Conceptual.Line
```

Επιστρέφει την ακμή που ξεκινά από τον κόμβο **beginComp** και καταλήγει στον κόμβο **endComp**.

ETL.Conceptual.Scenario

```
Public Class Scenario
  Inherits ETL.Base.Scenario
  Implements ETL.Base.IPrintable
  ...
End Class
```

Επεκτείνει την κλάση **ETL.Base.Scenario**, δημιουργώντας ένα *εννοιολογικό σενάριο*. Υλοποιώντας τη συνάρτηση **Print** που προέρχεται από τη διεπαφή **IPrintable**, αποκτούμε τη δυνατότητα εκτύπωσης του σεναρίου. Η κλάση αυτή περιέχει τις ακόλουθες ιδιότητες:

```
Public ReadOnly Property Components() As ETL.Base.ComponentCollection
Public ReadOnly Property Concepts() As ConceptCollection
Public ReadOnly Property Attributes() As AttributeCollection
Public ReadOnly Property Transformations() As TransformationCollection
Public ReadOnly Property Notes() As NoteCollection
Public ReadOnly Property ETLConstraints() As ETLConstraintCollection
Public ReadOnly Property Lines() As ETL.Conceptual.LineCollection
Public ReadOnly Property Repository() As Datasource
```

Οι παραπάνω ιδιότητες επιστρέφουν μία συλλογή με όλους τους κόμβους που περιέχονται στο σενάριο (**Components**), μία συλλογή με όλες τις ακμές (**Lines**) όπως και την πηγή δεδομένων που αποτελεί το repository στο οποίο είναι αποθηκευμένο το σενάριο αυτό (**Repository**). Ακόμα, οι ιδιότητες **Concepts**, **Attributes**, **Transformations**, **Notes** και **ETLConstraints** επιστρέφουν συλλογές που περιέχουν όλες τις *έννοιες*, τα *γνωρίσματα*, τους *μετασχηματισμούς*, τα *σχόλια* και τους *περιορισμούς ΕΜΦ* που περιέχει το σενάριο.

Επιπλέον, υπάρχουν και αρκετές συναρτήσεις:

```
Public Overloads Sub AddComponent(ByVal component As ETL.Conceptual.Concept)
```

```
Public Overloads Sub AddComponent(ByVal component As ETL.Conceptual.Attribute)
```

```
Public Overloads Sub AddComponent(ByVal component As ETL.Conceptual.Transformation)
```

```
Public Overloads Sub AddComponent(ByVal component As ETL.Conceptual.Note)
```

```
Public Overloads Sub AddComponent(ByVal component As ETL.Conceptual.ETLConstraint)
```

Η παραπάνω υπερφορτωμένη συνάρτηση αναλαμβάνει την προσθήκη ενός κόμβου στο σενάριο (και παράλληλα στην κατάλληλη συλλογή).

```
Public Overloads Sub AddLine(ByVal ln As ETL.Conceptual.Line)
```

Η συνάρτηση αυτή προσθέτει μία ακμή στο σενάριο (όπως και στη συλλογή ακμών του σεναρίου).

```
Public Overloads Function NewDatasource(ByVal id As String) As Datasource
```

```
Public Overloads Function NewDatasource(ByVal name As String, _
  ByVal location As String, ByVal provider As Datasource.Providers, _
  ByVal datasource As String, ByVal userid As String, ByVal password As String, _
  ByVal initialCatalog As String, ByVal package As String, _
  ByVal integratedSecurity As Boolean) As Datasource
```

Η υπερφορτωμένη συνάρτηση αυτή επιστρέφει μία πηγή δεδομένων. Στην πρώτη περίπτωση δέχεται ως όρισμα τον κωδικό της πηγής δεδομένων αυτής και δημιουργεί χρησιμοποιώντας στοιχεία από το repository, ενώ στη δεύτερη περίπτωση τη δημιουργεί χρησιμοποιώντας τα στοιχεία που της περνά ως ορίσματα ο χρήστης.

```
Public Overloads Function NewComponent(ByVal id As String) As ETL.Base.Component
```

```
Public Overloads Function NewComponent(ByVal name As String, _
  ByVal type As ComponentTypes, ByVal left As Single, ByVal top As Single, _
  ByVal address As String, ByVal parentID As String, _
  ByVal dataType As String) As ETL.Base.Component
```

Η υπερφορτωμένη συνάρτηση αυτή επιστρέφει ένα κόμβο. Στην πρώτη περίπτωση δέχεται ως όρισμα τον κωδικό του κόμβου αυτού και τον επιστρέφει, αφού τον δημιουργήσει με τα στοιχεία που υπάρχουν στο repository για τον κόμβο αυτό. Στη δεύτερη περίπτωση, ο κόμβος δημιουργείται χρησιμοποιώντας στα στοιχεία που δίνονται ως ορίσματα από το χρήστη.

```
Public Overloads Function NewLine(ByVal id As String) As ETL.Conceptual.Line
```

```
Public Overloads Function NewLine(ByVal beginID As String, ByVal endID As String, _
  ByVal lineType As LineTypes, ByVal text As String) As ETL.Conceptual.Line
```

Η υπερφορτωμένη συνάρτηση αυτή επιστρέφει μία ακμή. Όπως και στις δύο προηγούμενες περιπτώσεις (για πηγές δεδομένων και κόμβους), η πρώτη εκδοχή της συνάρτησης δέχεται ως όρισμα τον κωδικό της ακμής και τη δημιουργεί αντλώντας στοιχεία από το repository, ενώ στη δεύτερη εκδοχή, τη δημιουργεί με τα στοιχεία που δίνονται από το χρήστη.

```
Public Sub Refresh()
```

```
Public Function Save() As Boolean
```

Οι συναρτήσεις αυτές είναι σημαντικές διότι, η πρώτη (**Refresh**) ανανεώνει το σχήμα του σεναρίου, προωθώντας σε κάθε κόμβο και σε κάθε ακμή τυχόν αλλαγές που έχουν γίνει στο σενάριο, ενώ η δεύτερη (**Save**), όπως είναι προφανές από την ονομασία της, αναλαμβάνει την αποθήκευση του σεναρίου στο repository.

ETL.Conceptual.XORLine

```
Public Class XORLine
  Inherits System.Windows.Forms.Control
  Implements ETL.Base.IPrintable
  ...
End Class
```

Η κλάση αυτή είναι εντελώς βοηθητική, και δημιουργήθηκε για να καλύψει την απαίτηση για μία κάθετη ακμή με την ένδειξη XOR που χρησιμοποιείται στην περίπτωση των *σχέσεων υποψηφίων* για μία έννοια. Με την υλοποίηση της συνάρτησης **Print** αποκτά τη δυνατότητα εκτύπωσης.

7.3.4 Φόρμες

ETL.Designer.frmMain

```
Public Class frmMain
  Inherits System.Windows.Forms.Form
  ...
End Class
```

Η φόρμα αυτή αποτελεί τη βασική οθόνη του εργαλείου, μέσω της οποίας γίνονται όλες οι λειτουργίες της εφαρμογής.

ETL.Designer.Forms.frmGeneric

```
Public Class frmGeneric
  Inherits System.Windows.Forms.Form
  ...
End Class
```

Η φόρμα αυτή δημιουργήθηκε σαν βάση για τις περισσότερες φόρμες που θα ακολουθήσουν. Σκοπός της ήταν η ομαδοποίηση των χαρακτηριστικών που υπάρχουν σε κάθε φόρμα (όπως τα πλήκτρα OK και Cancel), έτσι ώστε να δωθεί σε όλες τις φόρμες μία ενιαία εμφάνιση και λειτουργικότητα.

ETL.Designer.Forms.frmAttribute

```
Public Class frmAttribute
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Η φόρμα αυτή χρησιμοποιείται για την εισαγωγή/αλλαγή ενός γνωρίσματος στο σενάριο.

ETL.Designer.Forms.frmCandidate

```
Public Class frmCandidate
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης ορίζει κάποιες έννοιες ως υποψήφιες. Επιλέγει τη βασική έννοια καθώς και τουλάχιστον δύο υποψήφιες έννοιες για την τροφοδοσία της έννοιας αυτής. Επίσης, έχει τη δυνατότητα να ορίσει κάποια από τις υποψήφιες έννοιες ως ενεργή υποψήφια.

ETL.Designer.Forms.frmConcept

```
Public Class frmConcept
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Η φόρμα αυτή χρησιμοποιείται για τον ορισμό και την αλλαγή μίας έννοιας στο σενάριο. Ο χρήστης καλείται να επιλέξει (ή να ορίσει) την πηγή δεδομένων και τον πίνακα στα οποία αντιστοιχεί η έννοια αυτή, καθώς και τα γνωρίσματα που επιθυμεί ο σχεδιαστής να είναι ορατά κατά τη διάρκεια της σχεδίασης του σεναρίου. Σε περίπτωση που το επιθυμεί, ο χρήστης, μπορεί να ορίσει και υποψήφιες έννοιες για την έννοια αυτή.

ETL.Designer.Forms.frmDatabase

```
Public Class frmDatabase
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης μπορεί να εισάγει στο σενάριο μία νέα πηγή δεδομένων, ή να ορίσει το repository. Ανάλογα με τον τύπο της πηγής δεδομένων που θα επιλέξει ο χρήστης, είναι και

τα στοιχεία που πρέπει να εισάγει ο χρήστης για τον ορθό ορισμό της πηγής.

ETL.Designer.Forms.frmETLConstraint

```
Public Class frmETLConstraint
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Με χρήση της φόρμας αυτής, ο χρήστης μπορεί να τοποθετήσει ένα νέο περιορισμό ΕΜΦ στο σενάριό του. Για το σκοπό αυτό θα πρέπει να επιλέξει την έννοια και τα γνωρίσματα στα οποία θέλει να επιβάλλεται ο περιορισμός, τον τύπο του περιορισμού καθώς και τυχόν εκφράσεις που απαιτούνται. Επίσης, όπως και στην περίπτωση του μετασχηματισμού, δίνεται η δυνατότητα στο χρήστη να εισάγει μία σύντομη περιγραφή για τον περιορισμό που εισάγει.

ETL.Designer.Forms.frmLine

```
Public Class frmLine
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης μπορεί να εισάγει κείμενο σε κάποια ακμή του σεναρίου, έτσι ώστε να γίνει το σενάριο πιο κατανοητό.

ETL.Designer.Forms.frmLogin

```
Public Class frmLogin
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης συνδέεται σε κάποιο repository με χρήση του ονόματος χρήστη και ενός συνθηματικού. Παράλληλα μπορεί να ορίσει και να χρησιμοποιήσει και νέο repository.

ETL.Designer.Forms.frmNewScenario

```
Public Class frmNewScenario
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Η φόρμα αυτή χρησιμοποείται σε δύο περιπτώσεις: (α) για τη δημιουργία ενός νέου σεναρίου, όπου ο χρήστης καλείται να δώσει τα στοιχεία (όνομα και περιγραφή) του σεναρίου που επιθυμεί να δημιουργήσει και (β) για τυχόν αλλαγές των στοιχείων αυτών του σεναρίου.

ETL.Designer.Forms.frmNote

```
Public Class frmNote
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης εισάγει ένα σχόλιο στο σενάριο. Καλείται να επιλέξει τον κόμβο (έννοια, μετασχηματισμός, περιορισμός ΕΜΦ) στον οποίο θα προσαρτηθεί το σχόλιο καθώς και το κείμενο που επιθυμεί να αναγράφεται στο σχόλιο αυτό.

ETL.Designer.Forms.frmOpenScenario

```
Public Class frmOpenScenario
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Η φόρμα αυτή χρησιμοποιείται για την επιλογή από το χρήστη του σεναρίου που επιθυμεί να φορτώσει, μεταξύ των σεναρίων που βρίσκονται στο repository.

ETL.Designer.Forms.frmOptions

```
Public Class frmOptions
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης μπορεί να αλλάξει πολλές από τις παραμέτρους του προγράμματος, έτσι ώστε να το προσαρμόσει στις προτιμήσεις του.

ETL.Designer.Forms.frmPrintPreview

```
Public Class frmPrintPreview
  Inherits System.Windows.Forms.Form
  ...
End Class
```

Χρησιμοποιείται για να εμφανίσουμε στο χρήστη μία προεπισκόπιση της εκτύπωσης του σεναρίου.

ETL.Designer.Forms.frmProvider

```
Public Class frmProvider
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Χρησιμοποιείται για να ορίσουμε μία σχέση παροχής μεταξύ δύο γνωρισμάτων (χωρίς τη χρήση μετασχηματισμού).

ETL.Designer.Forms.frmTransformation

```
Public Class frmTransformation
  Inherits ETL.Designer.Forms.frmGeneric
  ...
End Class
```

Μέσω της φόρμας αυτής, ο χρήστης ορίζει/αλλάζει τα στοιχεία ενός μετασχηματισμού. Δίνονται επιλογές για τις εισόδους και τις εξόδους του μετασχηματισμού, τον τύπο και τις εκφράσεις (expressions) του, καθώς και μία σύντομη περιγραφή αυτού.

7.4 Περιγραφή Σχήματος Repository

Στο Σχήμα 7.3 περιγράφεται διαγραμματικά το σχήμα του repository που χρησιμοποιούμε και στη συνέχεια περιγράφονται με λεπτομέρεια όλοι οι πίνακες που το αποτελούν.

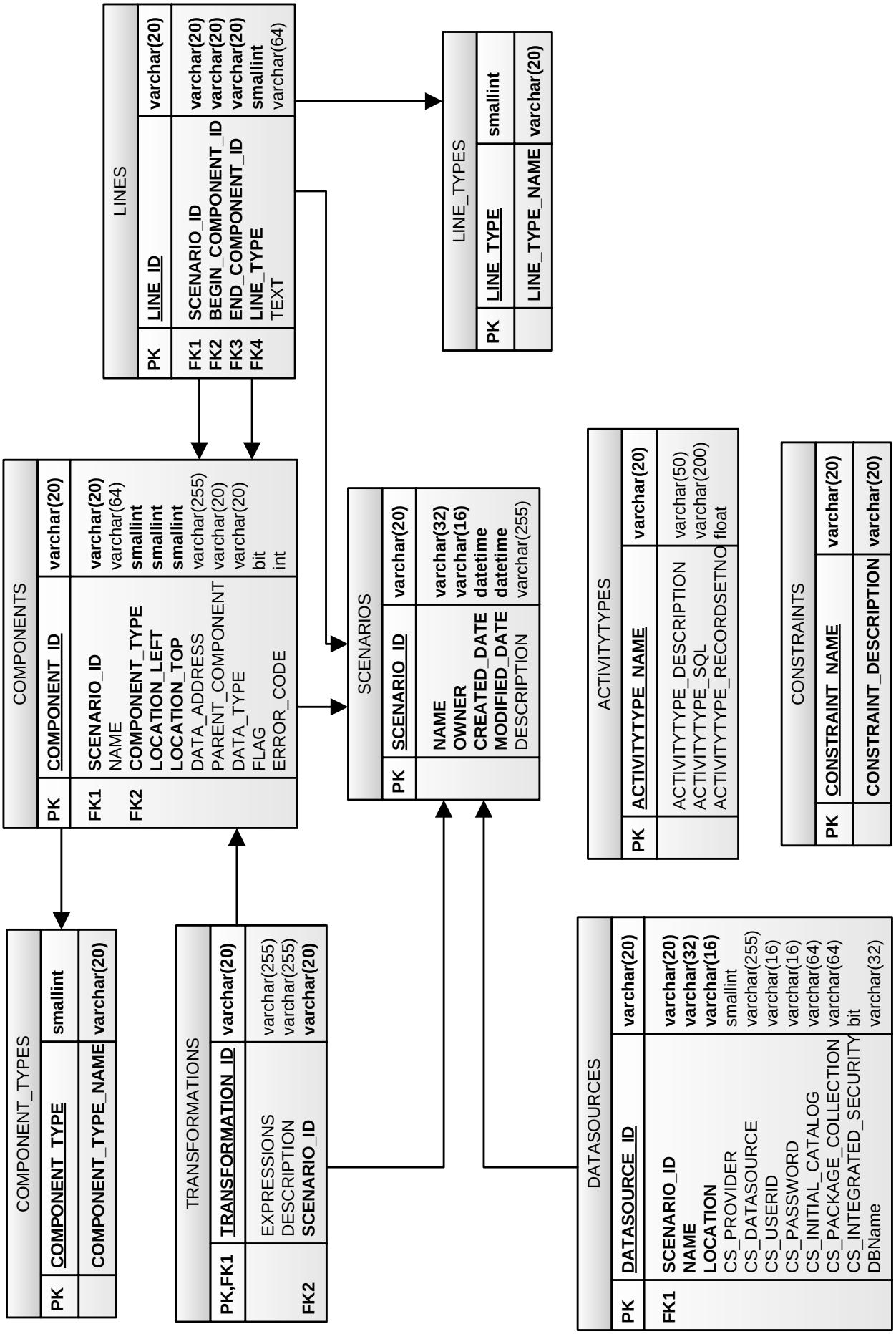
7.4.1 Scenarios

Ο πίνακας αυτός περιέχει τα βασικά στοιχεία ενός σεναρίου.

SCENARIO_ID (VARCHAR(20)): Το πρωτεύον κλειδί του πίνακα. Όπως και στους άλλους πίνακες παράγεται με τη μετατροπή του χρόνου δημιουργίας του (μετρημένο σε διαστήματα των 100-nanoseconds από την 1/1/0001) στο δεκαεξαδικό σύστημα για ελαχιστοποίηση του μεγέθους.

NAME (VARCHAR(32)): Το όνομα του σεναρίου. Αυτό παρουσιάζεται στο χρήστη, όταν αυτός ζητήσει τα διαθέσιμα σενάρια.

OWNER (VARCHAR(16)): Το όνομα του χρήστη στον οποίο ανήκει το σενάριο.



Σχήμα 7.3: Σχήμα Repository

CREATED_DATE (DATETIME): Η ημερομηνία που δημιουργήθηκε το σενάριο.

MODIFIED_DATE (DATETIME): Η ημερομηνία κατά την οποία το σενάριο αποθηκεύτηκε τελευταία φορά.

DESCRIPTION (VARCHAR(255)): Μία σύντομη περιγραφή του σεναρίου, έτσι ώστε ο χρήστης να μπορεί να αναγνωρίζει εύκολα τα σενάριά του.

7.4.2 Datasources

Στο συγκεκριμένο πίνακα αποθηκεύονται όλες οι απαιτούμενες πληροφορίες για τη σύνδεση του εργαλείου με μία πηγή ή μία Αποθήκη Δεδομένων. Η φύλαξη της πληροφορίας αυτής κρίθηκε απαραίτητη, έτσι ώστε τόσο ο σχεδιαστής να μπορεί να συνδέεται με τις πηγές δεδομένων και να βλέπει τη μορφή τους, όσο και για να μπορούμε, όταν το σενάριο ολοκληρωθεί, να εξάγουμε το σενάριο σε κατάλληλη μορφή έτσι ώστε να μπορεί να επεξεργαστεί σε λογικό επίπεδο και τελικά να υλοποιηθεί σε φυσικό επίπεδο.

DATASOURCE_ID (VARCHAR(20)): Το πρωτεύον κλειδί του πίνακα.

SCENARIO_ID (VARCHAR(20): Το σενάριο στο οποίο ανήκει αυτή η πηγή δεδομένων.

NAME (VARCHAR(32): Το όνομα της πηγής δεδομένων, όπως αυτή εμφανίζεται στο σενάριο.

LOCATION (VARCHAR(16): Το όνομα του υπολογιστή στον οποίο βρίσκεται η πηγή αυτή. Χρησιμοποιείται μόνο για την ομαδοποίηση των πηγών, έτσι ώστε σε περιπτώσεις σεναρίων με δεκάδες πηγές σε πολλά διαφορετικά μηχανήματα, ο σχεδιαστής να μπορεί να βρει εύκολα την πηγή που αναζητεί.

CS_⋆: Στις στήλες της μορφής αυτής αποθηκεύονται τα διάφορα τμήματα της συμβολοακολουθίας που απαιτείται για τη σύνδεση της εφαρμογής σε μία πηγή δεδομένων.

DBName (VARCHAR(32)): Στη στήλη αυτή τοποθετείται τυχόν alias που επιθυμεί να δώσει ο χρήστης σε κάποια πηγή δεδομένων.

7.4.3 Components

Ο πίνακας αυτός περιέχει στοιχεία για τους κόμβους του σεναρίου.

COMPONENT_ID (VARCHAR(20)): Το πρωτεύον κλειδί του πίνακα.

SCENARIO_ID (VARCHAR(20)): Το σενάριο στο οποίο ανήκει αυτός ο κόμβος.

NAME (VARCHAR(64)): Το κείμενο το οποίο αναγράφεται πάνω στον κόμβο, κατά τη σχεδίασή του στην οθόνη.

COMPONENT_TYPE (SMALLINT): Ο τύπος του κόμβου (όπως αυτός ορίζεται στον πίνακα COMPONENT_TYPES.

LOCATION_LEFT (SMALLINT): Η απόσταση της πάνω αριστερά γωνίας του κόμβου από την αριστερή άκρη της επιφάνειας σχεδίασης.

LOCATION_TOP (SMALLINT): Η απόσταση της πάνω αριστερά γωνίας του κόμβου από την πάνω άκρη της επιφάνειας σχεδίασης.

DATA_ADDRESS (VARCHAR(255)): Αποτελεί τη "διεύθυνση" της πηγής με την οποία συνδέεται ο κόμβος αυτός για την εισαγωγή ή την εξαγωγή δεδομένων στο σενάριο. Ο τρόπος σχηματισμού της διεύθυνσης αυτής περιγράφεται αναλυτικά για κάθε περίπτωση στην περιγραφή των κλάσεων **ETL.Conceptual.Attribute** και **ETL.Conceptual.Concept**.

PARENT_COMPONENT (VARCHAR(20)): Ο κόμβος στον οποίο ανήκει ο κόμβος αυτός. Χρησιμοποιείται σε περιπτώσεις:

- *γνωρισμάτων* (δείχνει την *έννοια* στην οποία ανήκει το γνώρισμα), και
- *σχολίων* (δείχνει τους μετασχηματισμούς στους οποίους αναφέρεται).

DATA_TYPE (VARCHAR(20)): Ανάλογα με τον τύπο του κόμβου στον οποίο αναφέρεται, περιέχει και διαφορετικά δεδομένα. Υπάρχουν οι ακόλουθες περιπτώσεις:

- *γνωρίσματος*, όπου αναφέρει τον τύπο του γνωρίσματος στο οποίο αναφέρεται (δηλαδή, αν είναι VARCHAR, INTEGER κ.ο.κ.).
- *μετασχηματισμού*, όπου περιγράφει τον τύπο του μετασχηματισμού, όπως αυτός αναφέρεται στον πίνακα ACTIVITYTYPES,
- *περιορισμού ΕΜΦ*, όπου περιγράφει τον τύπο του περιορισμού, όπως αυτός αναφέρεται στον πίνακα CONSTRAINTS.

FLAG (BIT): Μέσω της στήλης αυτής, αποθηκεύεται στο repository η κατάσταση των διαφόρων κόμβων. Σε περίπτωση που ο κόμβος είναι

- *μετασχηματισμός*, δείχνει αν ο χρήστης επιθυμεί να φαίνεται το σχόλιο που συνοδεύει το μετασχηματισμό αυτό.
- *περιορισμός ΕΜΦ*, δείχνει αν ο χρήστης επιθυμεί να φαίνεται το σχόλιο που συνοδεύει τον Περιορισμό ΕΜΦ αυτό.
- *έννοια*, δείχνει αν ο χρήστης επιθυμεί να φαίνονται τα γνωρίσματα της έννοιας αυτής.

ERROR_CODE (INT): Ο (τυχόν) κωδικός σφάλματος κάθε κόμβου.

7.4.4 ActivityTypes

Περιέχει στοιχεία για τους μετασχηματισμούς που έχουν οριστεί μέχρι στιγμής στο repository. Ο σχεδιαστής, έχοντας ορίσει[1] κάποιο μετασχηματισμό σε κάποιο σενάριο, μπορεί να τον χρησιμοποιήσει ξανά σε κάθε μελλοντικό σενάριο, χωρίς να απαιτείται ο εκ νέου ορισμός του.

ACTIVITYTYPE_NAME (VARCHAR(20)): Το όνομα του μετασχηματισμού.

ACTIVITYTYPE_DESCR (VARCHAR(200): Μία σύντομη περιγραφή του μετασχηματισμού.

ACTIVITYTYPE_SQL (VARCHAR(400)): Η πρότυπη SQL πρόταση που ορίζει το μετασχηματισμό.

7.4.5 Constraints

Περιέχει τους διαθέσιμους *περιορισμούς ΕΜΦ* που υπάρχουν στο repository.

CONSTRAINT_NAME (VARCHAR(20)): Το όνομα του περιορισμού ΕΜΦ.

CONSTRAINT_DESCRIPTION (VARCHAR(200)): Μία σύντομη περιγραφή του περιορισμού αυτού.

7.4.6 Lines

Ο πίνακας αυτός περιέχει τις ακμές του σεναρίου.

LINE_ID (VARCHAR(20)): Το πρωτεύον κλειδί του πίνακα.

SCENARIO_ID (VARCHAR(20)): Το σενάριο στο οποίο ανήκει η ακμή αυτή.

BEGIN_COMPONENT_ID (VARCHAR(20)): Ο κόμβος από τον οποίο αρχίζει η ακμή.

END_COMPONENT_ID (VARCHAR(20)): Ο κόμβος στον οποίο καταλήγει η ακμή.

LINE_TYPE (SMALLINT): Ο τύπος της ακμής, όπως αυτός ορίζεται στον πίνακα LINE_TYPES.

TEXT (VARCHAR(64): Το κείμενο που αναγράφεται πάνω στην ακμή.

7.4.7 Line_Types

Περιέχει τους διαθέσιμους τύπους ακμών στο Repository.

LINE_TYPE (SMALLINT): Το πρωτεύον κλειδί του πίνακα αυτού.

LINE_TYPE_NAME (VARCHAR(20)): Το όνομα του τύπου αυτού.

[1] Ο ορισμός μετασχηματισμών και περιορισμών ΕΜΦ ανήκουν στο λογικό μέρος των διεργασιών ΕΜΦ, έτσι ξεφεύγουν από τον σκοπό αυτής της διπλωματικής εργασίας. Για περισσότερες πληροφορίες για το σκοπό αυτό, δείτε [41].

7.4.8 COMPONENT_TYPES

Περιέχει τους διαθέσιμούς τύπους κόμβων στο Repository.

COMPONENT_TYPE (SMALLINT): Το πρωτεύον κλειδί του πίνακα αυτού.

COMPONENT_TYPE_NAME (VARCHAR(20)): Το όνομα του τύπου αυτού.

7.4.9 TRANSFORMATIONS

Περιέχει λεπτομέρειες για τους μετασχηματισμούς που περιέχονται στα σενάρια.

TRANSFORMATION_ID (VARCHAR(20)): Το πρωτεύον κλειδί του πίνακα αυτού, το οποίο συγχρόνως είναι και ξένο κλειδί προς τον πίνακα **COMPONENTS**.

SCENARIO_ID (VARCHAR(20)): Το σενάριο στο οποίο ανήκει ο μετασχηματισμός για τον οποίο αναφέρεται η εγγραφή αυτή. Παρόλο που η πληροφορία αυτή φαίνεται και στον πίνακα **COMPONENTS** κρίθηκε αναγκαίο να τοποθετηθεί και στον πίνακα αυτό, έτσι ώστε να αποφευχθεί η χρήση Join κατά τη φόρτωση ενός σεναρίου από το repository.

EXPRESSIONS (VARCHAR(255)): Εδώ αποθηκεύονται οι διάφορες εκφράσεις (expressions) που απαιτούνται για τη σωστή εφαρμογή του μετασχηματισμού.

DESCRIPTION (VARCHAR(255)): Μία σύντομη περιγραφή του μετασχηματισμού.

8

Χρήση Εργαλείου

8.1 Εγκατάσταση της Εφαρμογής

Η εγκατάσταση της εφαρμογής είναι μία τυπική εγκατάσταση σε περιβάλλον Windows. Η εγκατάσταση της εφαρμογής γίνεται με εκτέλεση του αρχείου "SetupConceptualArktosII.exe". Απαιτείται από το χρήστη εισαγωγή της διαδρομής του φακέλου στον οποίο θα εγκατασταθεί η εφαρμογή καθώς και το όνομα του φακέλου που θα δημιουργηθεί στο μενού Start των Windows. Το πρόγραμμα εγκατάστασης αναλαμβάνει να δημιουργήσει τους απαιτούμενους φακέλους για την εγκατάσταση και να εγκαταστήσει τα απαραίτητα dll αρχεία στη σωστή τους θέση.

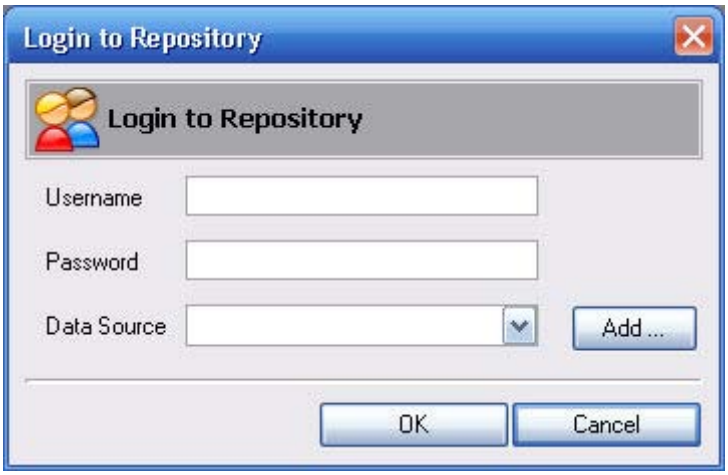
Στον ίδιο φάκελο με τα αρχεία της εφαρμογής, αντιγράφονται και το αρχείο βοήθειας (Arktos II - Conceptual GUI.chm) και το αρχείο με τις SQL προτάσεις που χρησιμοποιείται για τη δημιουργία του repository.

8.2 Εκκίνηση της Εφαρμογής

8.2.1 Login

Για τη χρήση του εργαλείου απαιτείται σύνδεση με κάποιο repository. Στην πρώτη οθόνη του εργαλείου, ο χρήστης καλείται να επιλέξει από μία λίστα το repository που επιθυμεί να χρησιμοποιήσει και δίνει το αναγνωριστικό όνομα (**Username**) και τον κωδικό (**Password**) που απαιτούνται για τη σύνδεση

στο repository αυτό.



Αν το repository στο οποίο επιθυμεί ο χρήστης να συνδεθεί δεν υπάρχει στη λίστα, τότε μπορεί να το προσθέσει χρησιμοποιώντας το πλήκτρο **Add...**.

8.2.2 Προσθήκη Repository

Η εισαγωγή ενός νέου repository γίνεται μέσω του παραθύρου του σχήματος.

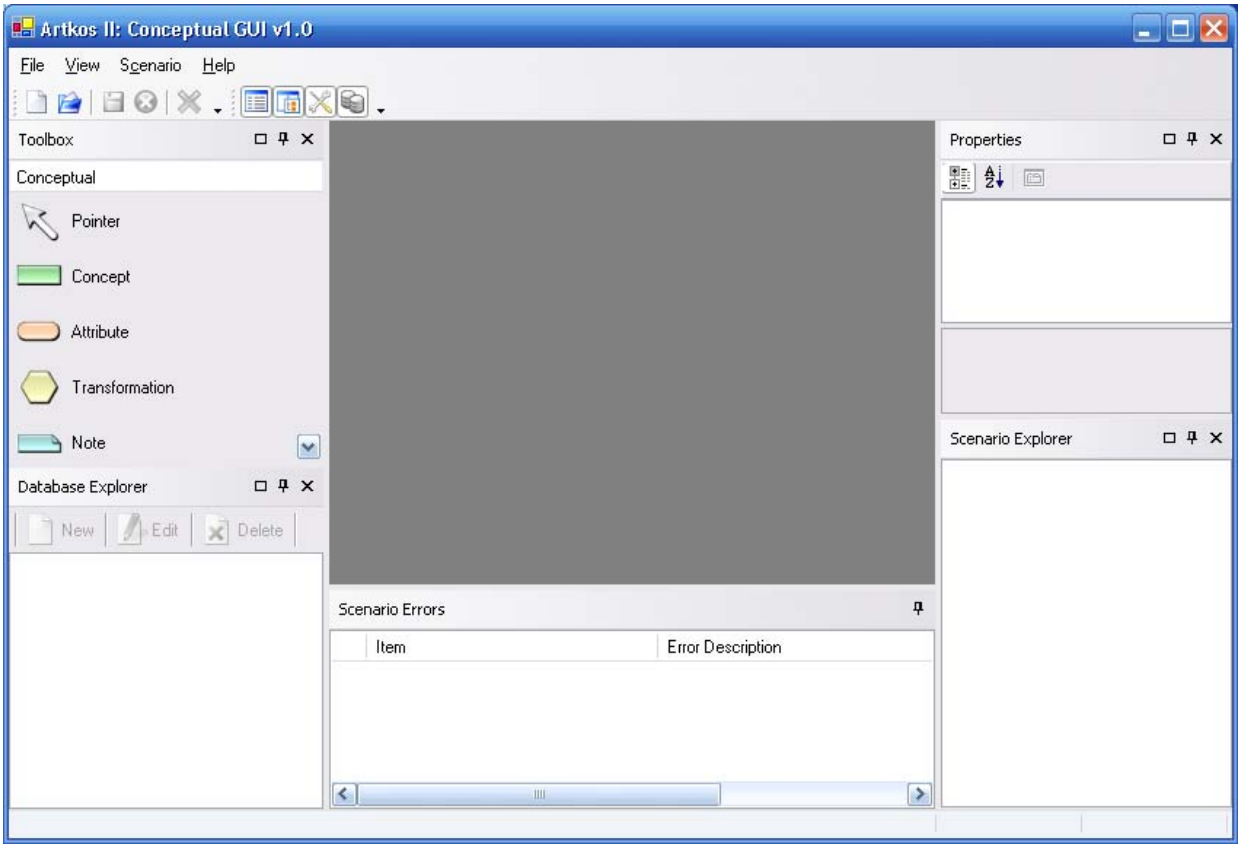
Στο παράθυρο αυτό, από τη λίστα **Provider** ο χρήστης επιλέγει τον τύπο του ΣΔΒΔ στο οποίο επιθυμεί να συνδεθεί. Ανάλογα με το ΣΔΒΔ που θα επιλεχθεί, θα είναι και τα στοιχεία που πρέπει να συμπληρώσει για την επιτυχή σύνδεση.

Στην περίπτωση του παραπάνω σχήματος, έχουμε σύνδεση σε Microsoft SQL Server, όπου τα απαι-

τούμενα στοιχεία είναι το **Datasource**, δηλαδή ο υπολογιστής στον οποίο βρίσκεται εγκατεστημένος ο SQL Server· το **Initial Catalog**, δηλαδή η βάση δεδομένων στον SQL Server· το αναγνωριστικό όνομα (**Username**) και ο κωδικός χρήσης **Password**). Το username και το password δεν είναι απαραίτητα αν η σύνδεση γίνει με χρήση **Windows Based Authentication**[1].

8.3 Κεντρική Οθόνη Προγράμματος

Μετά την επιλογή ενός repository και την επιτυχή σύνδεση σε αυτό με χρήση του αναγνωριστικού ονόματος και του κωδικού χρήσης, εμφανίζεται η κεντρική οθόνη του προγράμματος.



Η κεντρική οθόνη του προγράμματος περιέχει πέντε εργαλειοθήκες. Στα αριστερά της οθόνης υπάρχει το **Toolbox** που αποτελεί την κεντρική εργαλειοθήκη του προγράμματος μέσω της οποίας γίνονται όλες οι λειτουργίες σχεδιασμού κάποιου σεναρίου. Επίσης στα αριστερά βλέπουμε το **Database Explorer** που είναι μία λίστα με όλες τις πηγές δεδομένων που έχουμε προσθέσει στο σενάριο. Στα δεξιά του παραθύρου βλέπουμε τις εργαλειοθήκες **Properties** και **Scenario Explorer**. Η πρώτη μας παρουσιάζει χρήσιμες πληροφορίες για κάθε κόμβο του σεναρίου, ενώ μας επιτρέπει να αλλάξουμε κάποιες επιλογές αυτού, ενώ η δεύτερη αποτελεί μία λίστα με όλους τους κόμβους που έχουμε προσθέσει στο σενάριο, ομαδοποιημένους κατάλληλα. Τέλος, στο κάτω μέρος του παραθύρου εμφανίζεται η εργαλειοθήκη **Scenario Errors**, η οποία αποτελεί μία λίστα με όλες τις παραβιάσεις κάποιου περιορισμού

[1] Η επιλογή αυτή πρέπει να επιτρέπεται από τον server

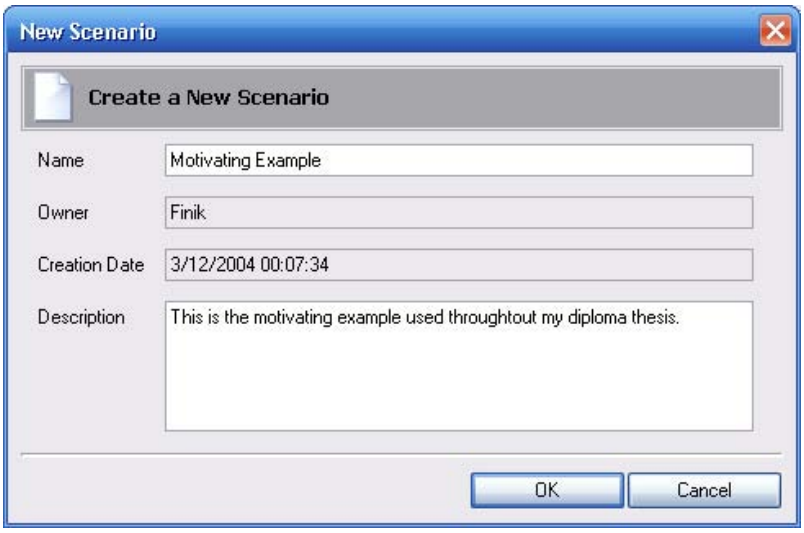
στη σχεδίαση του εννοιολογικού σεναρίου.

8.3.1 Μενού

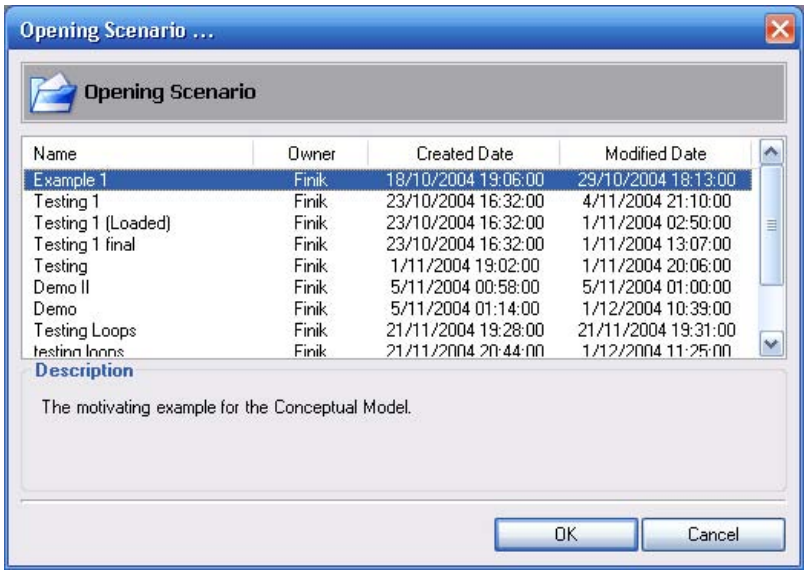
Εκτός από τις εργαλειοθήκες που βοηθούν στο σχεδιασμό ενός σεναρίου, βασικό ρόλο παίζουν και τα μενού επιλογών στην κορυφή του παραθύρου. Οι διαθέσιμες επιλογές είναι:

File

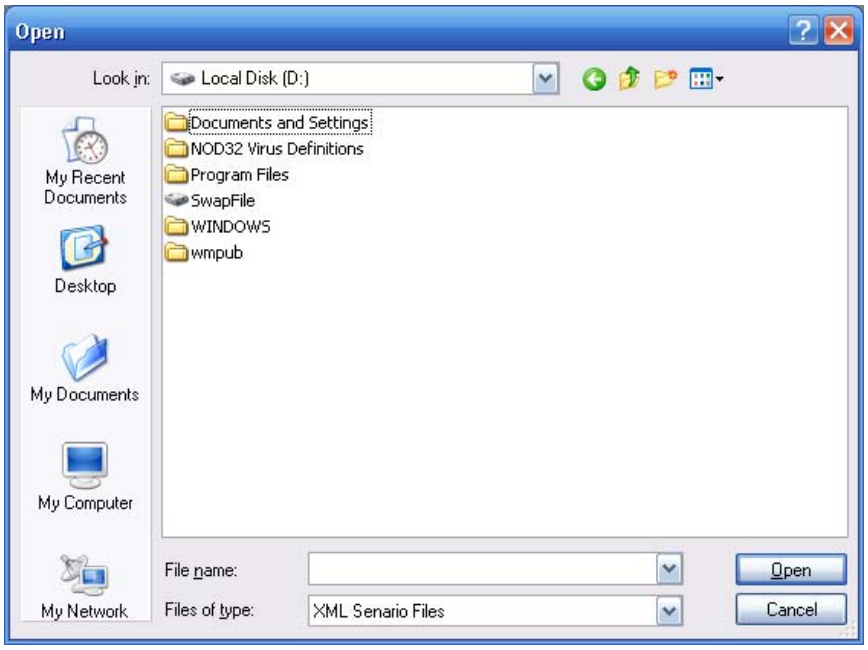
- **New Scenario**: Εμφανίζει το παράθυρο δημιουργίας ενός νέου σεναρίου.



- **Open Scenario**: Εμφανίζει μία λίστα με τα σενάρια που βρίσκονται στο repository στο οποίο είμαστε συνδεδεμένοι για να επιλέξει ο χρήστης αυτό που επιθυμεί να φορτώσει.



- **Import Scenario**: Φορτώνεται ένα σενάριο που έχει γίνει Export σε μορφή XML.



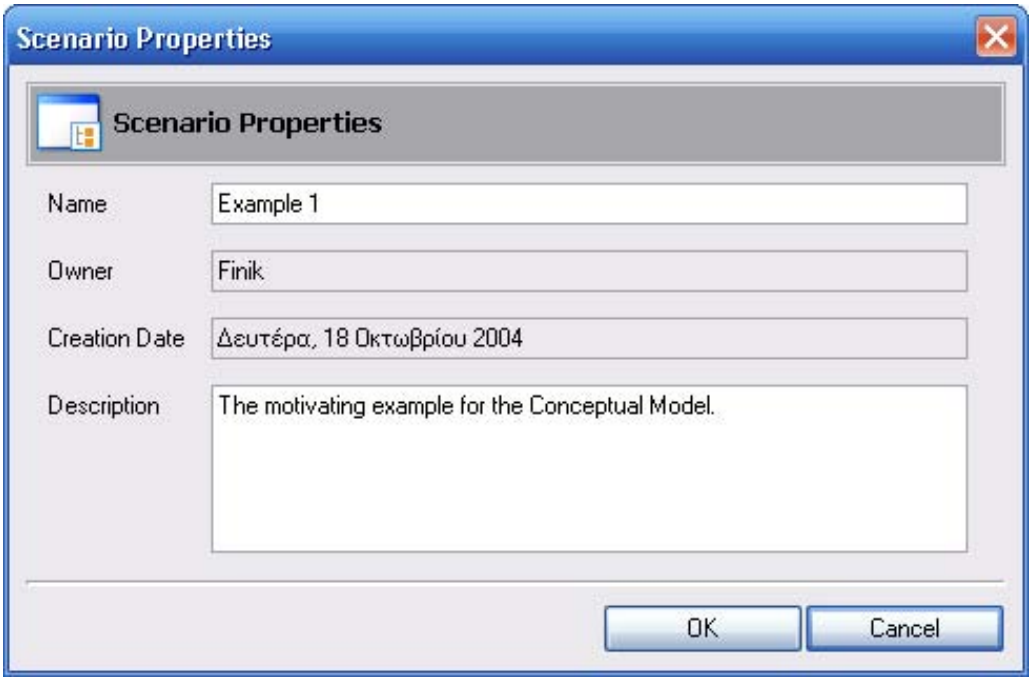
- **Close**: Κλείνει το ενεργό σενάριο, έτσι ώστε ο χρήστης να μπορεί να δημιουργήσει ένα νέο σενάριο ή να φορτώσει κάποιο από το repository για περαιτέρω επεξεργασία.
- **Change User**: Με την επιλογή αυτή μπορεί ο χρήστης να συνδεθεί είτε στο ίδιο repository με άλλο αναγνωριστικό όνομα χρήστη, είτε σε κάποιο άλλο repository.
- **Save Scenario**: Σώζει το σενάριο στο repository.
- **Export**: Δίνονται δύο διαφορετικές επιλογές:
  - **Export Scenario to XML**: Σώζει το σενάριο σε μορφή XML.
  - **Export for Logical GUI**: Σώζει το σενάριο σε μορφή κατάλληλη για να εισαχθεί στο πρόγραμμα επεξεργασίας λογικών σεναρίων για περαιτέρω επεξεργασία.
- **Print**: Εκτυπώνει το ενεργό σενάριο.
- **Print Preview**: Δίνει μία προεπισκόπηση της εκτύπωσης του σεναρίου.
- **Exit**: Τερματισμός του προγράμματος.

View

- **Properties**: Εμφανίζει/Κρύβει την εργαλειοθήκη **Properties**.
- **Scenario Explorer**: Εμφανίζει/Κρύβει την εργαλειοθήκη **Scenario Explorer**.
- **Toolbox**: Εμφανίζει/Κρύβει την εργαλειοθήκη **Toolbox**.
- **Database Explorer**: Εμφανίζει/Κρύβει την εργαλειοθήκη **Databas Explorer**.

Scenario

- **Delete Component(s)**: Διαγράφει όλους τους επιλεγμένους κόμβους του σεναρίου.
- **Align Components**: Υπομενού με επιλογές στοίχισης των κόμβων. Διαθέσιμες επιλογές είναι:
  - **Left**: Στοίχιση στα αριστερά.
  - **Center**: Στοίχιση στο κέντρο.
  - **Right**: Στοίχιση στα δεξιά.
  - **Top**: Στοίχιση στην κορυφή.
  - **Middle**: Στοίχιση στο μέση.
  - **Bottom**: Στοίχιση στο κάτω μέρος
- **Scenario Properties**: Εμφανίζει τα στοιχεία του σεναρίου, έτσι ώστε ο χρήστης να μπορεί να αλλάξει τον τίτλο ή την περιγραφή του σεναρίου αυτού.

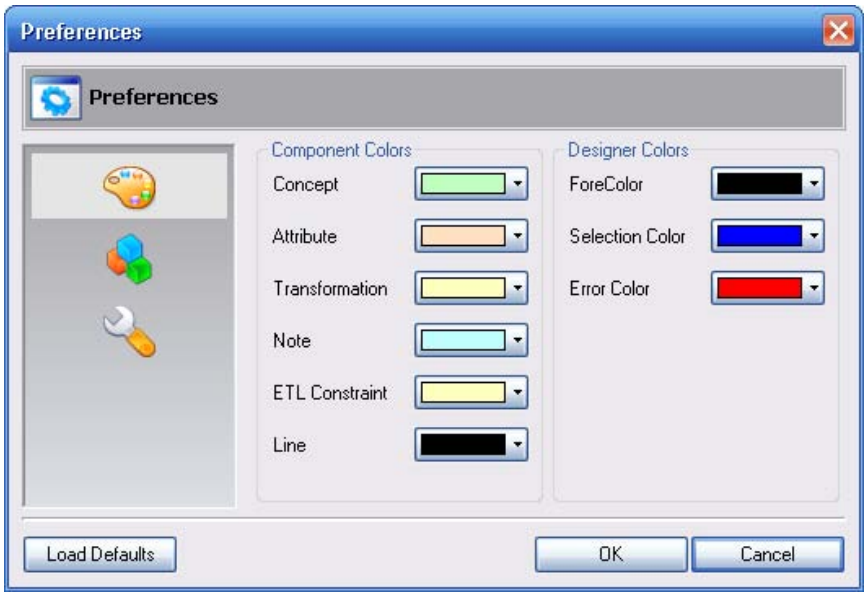


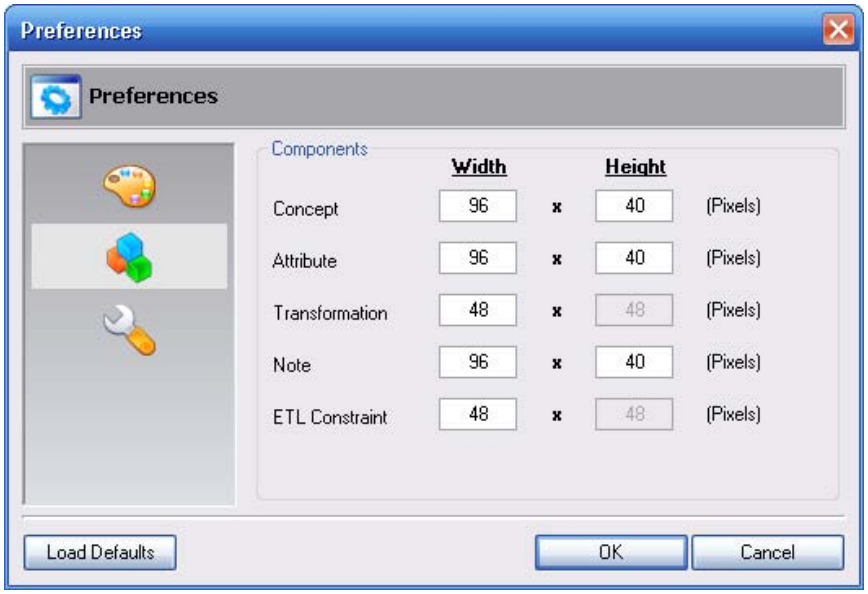
- **Preferences**: Εμφανίζει ένα παράθυρο, μέσω του οποίου ο χρήστης αλλάζει διάφορες επιλογές που αφορούν την εμφάνιση και την λειτουργία του προγράμματος.

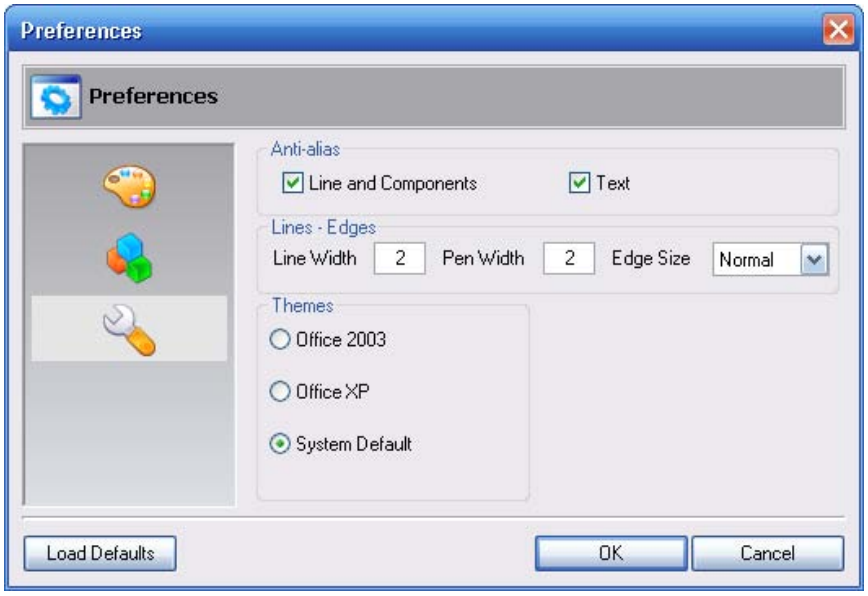
  Η πρώτη οθόνη επιλογών επιτρέπει στο χρήστη να επιλέξει τα χρώματα που επιθυμεί να χρησιμοποιούνται για το χρωματισμό των διαφόρων κόμβων, καθώς και τα διάφορα χρώματα για τις ακμές.

  Στη δεύτερη οθόνη υπάρχουν επιλογές για τα μεγέθη των κόμβων.

  Στην τρίτη οθόνη υπάρχουν επιλογές για τα πάχη των ακμών, για το αν θα εξομαλύνονται οι ακμές και το κείμενο καθώς και για την εμφάνιση της κεντρικής οθόνης του προγράμματος.


Preferences
Preferences
Component Colors
Concept
Attribute
Transformation
Note
ETL Constraint
Line
Designer Colors
ForeColor
Selection Color
Error Color
Load Defaults
OK
Cancel


Preferences
Preferences
Components
Width
Height
Concept 96 x 40 (Pixels)
Attribute 96 x 40 (Pixels)
Transformation 48 x 48 (Pixels)
Note 96 x 40 (Pixels)
ETL Constraint 48 x 48 (Pixels)
Load Defaults
OK
Cancel


Preferences
Preferences
Anti-alias
Line and Components
Text
Lines - Edges
Line Width 2
Pen Width 2
Edge Size Normal
Themes
Office 2003
Office XP
System Default
Load Defaults
OK
Cancel

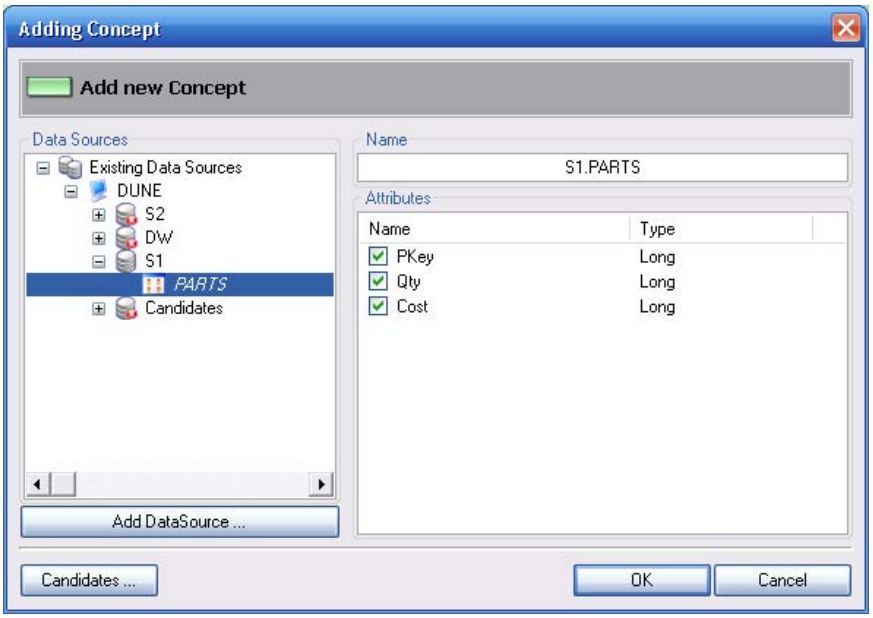
Help

- **Help**: Εμφάνιση του αρχείου βοήθειας του προγράμματος.
- **About**: Εμφανίζει ένα παράθυρο με πληροφορίες για την εφαρμογή και για τα πνευματικά δικαιώματα αυτής.

8.4 Εισαγωγή Κόμβων και Ακμών στο Σενάριο

8.4.1 Concept

Επιλέγοντας από την εργαλειοθήκη **Toolbox** το εικονίδιο της Έννοιας και στη συνέχεια κάνοντας κλικ στην επιφάνεια σχεδίασης, εμφανίζεται το παρακάτω παράθυρο, για τον ορισμό και την εισαγωγή μίας νέας έννοιας στο σενάριο.

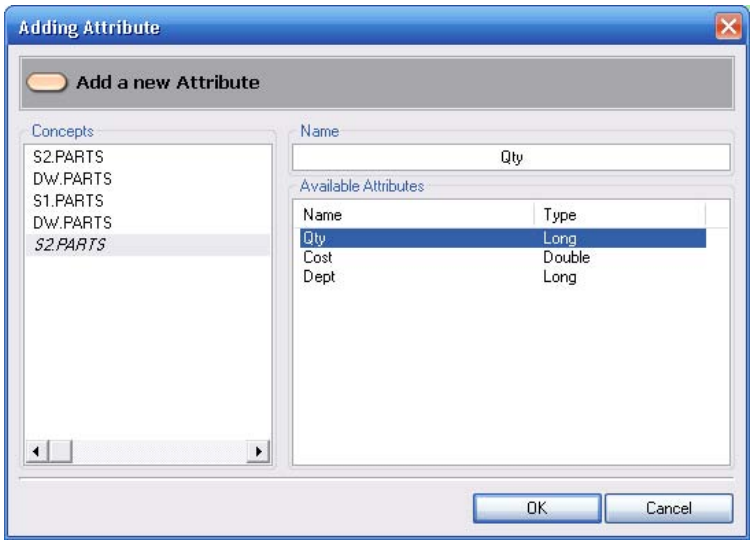


Στη λίστα στο αριστερό μέρος της οθόνης υπάρχουν όλες οι διαθέσιμες πηγές δεδομένων που έχουν προστεθεί στο σενάριο μέχρι τώρα. Επιλέγοντας μία έννοια από τη λίστα αυτή (δηλαδή ένα πίνακα σε περίπτωση σχεσιακών βάσεων δεδομένων), εμφανίζονται στην λίστα αριστερά, τα γνωρίσματα της έννοιας αυτής (οι στήλες του πίνακα). Από τα γνωρίσματα αυτά, ο χρήστης μπορεί να επιλέξει ποια επιθυμεί να προστεθούν αυτόματα στο σενάριο με την προσθήκη της έννοιας. Σε περίπτωση που δεν προσθέσει κάποιο από τα γνωρίσματα αυτά, μπορεί να το κάνει αργότερα.

Στην περίπτωση που η πηγή που επιθυμεί ο χρήστης να αντιστοιχεί με την έννοια που δημιουργεί δεν έχει προστεθεί ακόμη στο σενάριο, τότε μπορεί να την προσθέσει μέσω της επιλογής **Add Datasource**.

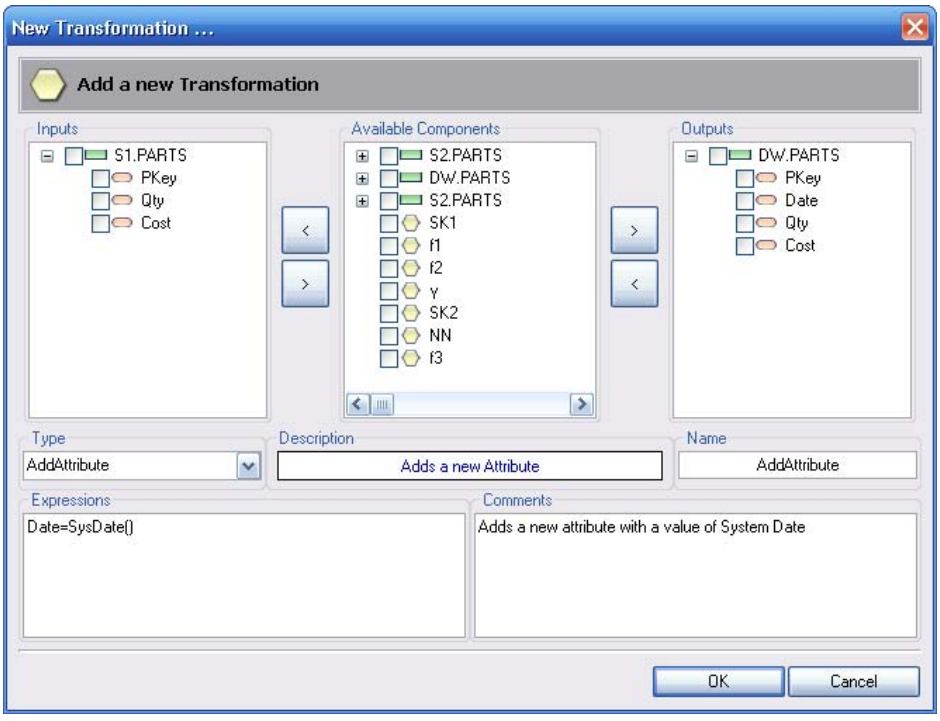
Ο χρήστης μπορεί να δώσει ένα όνομα στην Έννοια, το οποίο θα αναγράφεται στο σχήμα της και θα χρησιμοποιείται σε κάθε αναφορά στην έννοια αυτή. Επιπλέον, αν το επιθυμεί, μπορεί να ορίσει κάποιες έννοιες ως υποψήφιες για την πλήρωση της έννοιας αυτής (επιλογή **Candidates ...**).

8.4.2 Attribute



Κατά το σχεδιασμό ενός σεναρίου, ο σχεδιαστής μπορεί να να μην έχει προσθέσει κάποιο γνώρισμα σε κάποια έννοια, ή να το έχει διαγράψει από λάθος. Στην περίπτωση αυτή, επιλέγοντας το εργαλείο **Attribute** και κάνοντας κλικ στην επιφάνεια σχεδιασμού του σεναρίου, εμφανίζεται το παραπάνω παράθυρο. Από αυτό, ο χρήστης μπορεί να επιλέξει την έννοια στην οποία θέλει να προσθέσει κάποιο γνώρισμα και από τα γνωρίσματα που θα εμφανιστούν στη λίστα αριστερά, να επιλέξει αυτό που τον ενδιαφέρει. Σημειώστε εδώ ότι μόνο τα γνωρίσματα μίας έννοιας που δεν έχουν ήδη προστεθεί στο σενάριο, εμφανίζονται στη λίστα αυτή.

8.4.3 Transformation



Ίσως το πιο σημαντικό παράθυρο του εργαλείου. Μέσω αυτού, ο χρήστης μπορεί να προσθέσει μετασχηματισμούς στο σενάριο. Στη λίστα που βρίσκεται στη μέση (στο πάνω μέρος), απεικονίζονται όλοι οι διαθέσιμοι κόμβοι του σεναρίου. Ο χρήστης μπορεί να επιλέξει από αυτούς τα γνωρίσματα και τους μετασχηματισμούς που θα αποτελούν την είσοδο (**Inputs**) και την έξοδο **Outputs**) του μετασχηματισμού.

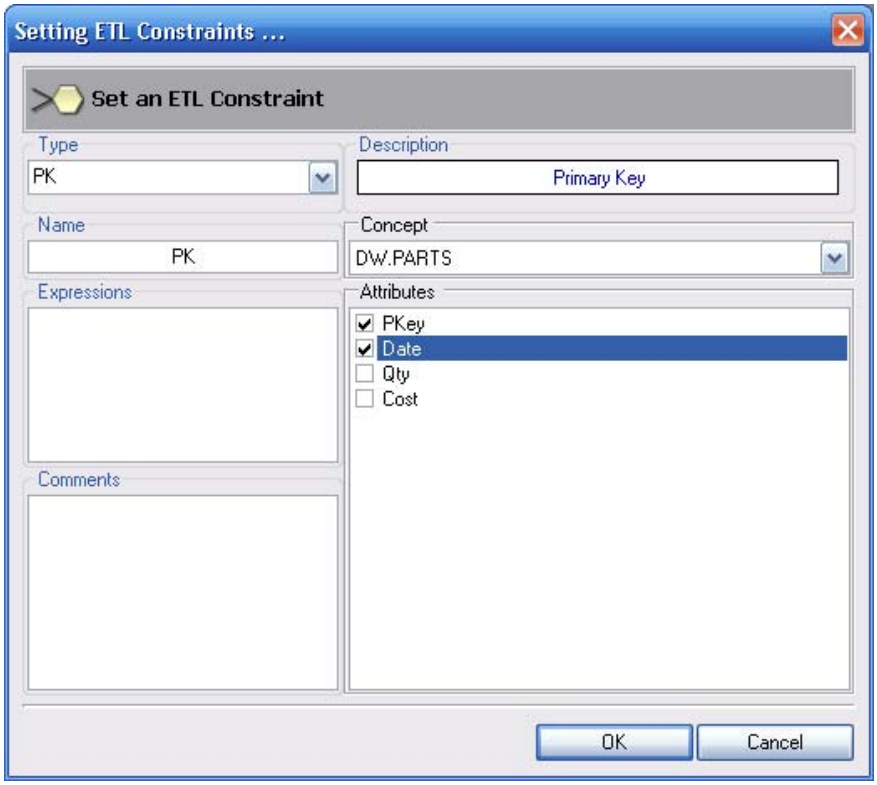
Στη συνέχεια, από τη λίστα **Type** πρέπει να επιλέξει τον τύπο του μετασχηματισμού. Σε περίπτωση που ο τύπος μετασχηματισμού που επιθυμεί ο χρήστης να εισάγει δεν είναι διαθέσιμος, μπορεί να τον προσθέσει, γράφοντας το όνομά του στο πεδίο αυτό. Σε αυτή την περίπτωση, όταν ο χρήστης φορτώσει το σενάριο αυτό στο εργαλείο λογικής σχεδίασης, θα ορίσει πλήρως τον μετασχηματισμό αυτό.

Στα πεδία **Expressions** και **Comments**, ο χρήστης εισάγει τις απαιτούμενες παραμέτρους για την ορθή εφαρμογή του μετασχηματισμού και σύντομα σχόλια για το μετασχηματισμό αυτό, αντίστοιχα.

Παράλληλα, ο χρήστης μπορεί στο πεδίο **Name** να δώσει ένα όνομα στο μετασχηματισμό, για την εύκολη αναγνώρισή του κατά τη διάρκεια του σχεδιασμού.

8.4.4 ETL Constraint

Εκτός από μετασχηματισμούς, στο σενάριο μπορούν να προστεθούν και Περιορισμοί ΕΜΦ. Η εισαγωγή τους γίνεται μέσω της παρακάτω οθόνης.



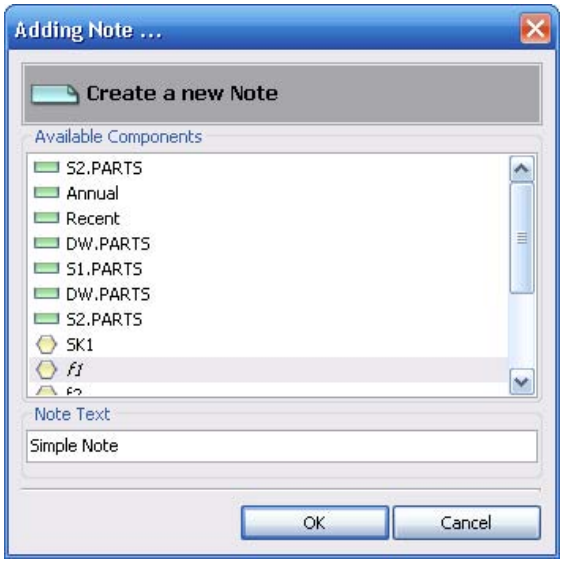
Για τον ορισμό (και την εισαγωγή) ενός Περιορισμού ΕΜΦ, ο χρήστης καλείται να επιλέξει τον τύπου του περιορισμού (**Type**)· την έννοια (**Concept**), σε γνωρίσματα της οποίας θα εφαρμοστεί ο περιορισμός· και τα γνωρίσματα αυτά (**Attributes**). Επιπλέον, ο χρήστης καλείται να δώσει ένα όνομα, το οποίο θα

χαρακτηρίζει τον περιορισμό αυτό στη συνέχεια στο σενάριο.

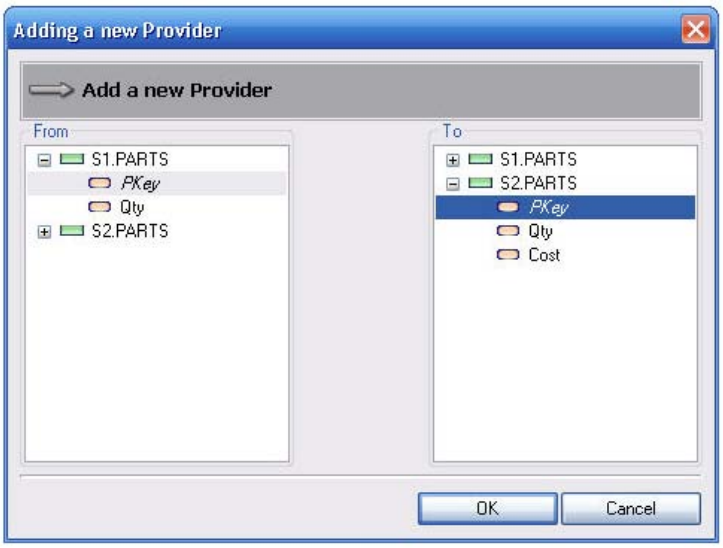
Όπως και στην περίπτωση των μετασχηματισμών, έτσι κι εδώ, ο χρήστης μπορεί να δώσει μέσω των πεδίων **Expressions** και **Comments** τις απαιτούμενες παραμέτρους για την εφαρμογή του περιορισμού και σύντομα σχόλια για αυτόν, αντίστοιχα.

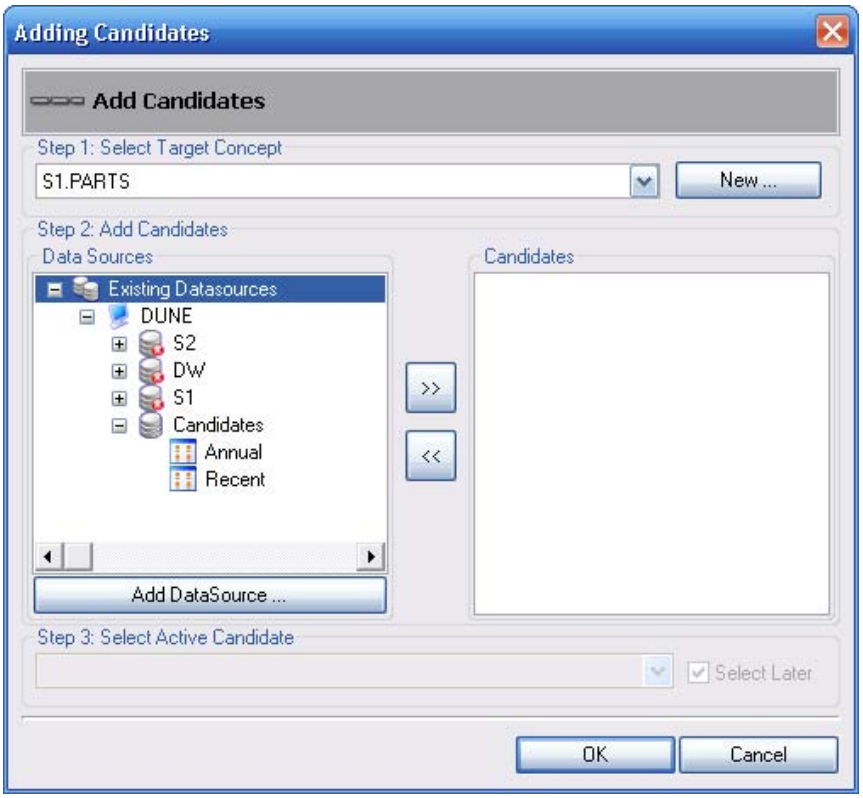
8.4.5 Note

Μέσω της παρακάτω οθόνης, ο χρήστης μπορεί να επισυνάψει σχόλιο σε κάποιο κόμβο (Έννοια, Μετασχηματισμό ή Περιορισμό ΕΜΦ). Αυτό γίνεται με επιλογή του κόμβου στον οποίο επιθυμεί να προσθέσει το σχόλιο από τη λίστα στο πάνω μέρος του παραθύρου και με συμπλήρωση του σχολίου, στο πεδίο **Note Text**.



8.4.6 Provider





Εκτός από τις Σχέσεις Παροχής μέσω κάποιου μετασχηματισμού, το μοντέλο υποστηρίζει και την εισαγωγή Σχέσεων Παροχής 1:1 μεταξύ δύο γνωρισμάτων. Ο ορισμός μίας τέτοιας σχέσης στο σενάριό μας γίνεται μέσω του παραπάνω παραθύρου. Το μόνο που απαιτείται από το χρήστη είναι η επιλογή από την λίστα στα αριστερά, του γνωρίσματος–πηγής και από τη λίστα στα δεξιά, του γνωρίσματος–στόχου.

8.4.7 Candidates

Η προσθήκη Υποψηφίων για κάποια έννοια του σεναρίου, γίνεται μέσω της παραπάνω φόρμας. Η διαδικασία εισαγωγής αποτελείται από τρία στάδια.

1. Επιλογή της Έννοιας–στόχου των Υποψηφίων. Από τη λίστα στο πάνω μέρος της οθόνης, ο χρήστης μπορεί εύκολα να επιλέξει την έννοια στην οποία θέλει να ορίσει υποψήφιους. Σε περίπτωση που η έννοια αυτή δεν βρίσκεται στο σενάριο, μπορεί να την εισάγει με χρήση του πλήκτρου **New**.

2. Προσθήκη Υποψηφίων. Στο στάδιο αυτό, ο σχεδιαστής καλείται να επιλέξει από τις διαθέσιμες πηγές δεδομένων αυτές που θα είναι υποψήφιες για την τροφοδοσία της έννοιας–στόχου. Σημειώνουμε εδώ ότι ο αριθμός των υποψηφίων για κάποια έννοια δεν μπορεί να είναι μικρότερος του 2.

3. Επιλογή Ενεργού Υποψηφίου. Η επιλογή του ενεργού υποψηφίου γίνεται μέσω της λίστας

στο κάτω μέρος της οθόνης. Εδώ σημειώνουμε ότι η επιλογή ενεργού υποψηφίου δεν είναι απαραίτητη, αλλά μπορεί να γίνει και αργότερα.

9

Επίλογος

Στο κεφάλαιο αυτό συνοψίζουμε τα αποτελέσματα της διπλωματικής εργασίας και παραθέτουμε τις μελλοντικές επεκτάσεις που έχουν νόημα και ενδιαφέρον να υλοποιηθούν.

9.1 Σύνοψη — Συμπεράσματα

Στην εργασία αυτή, αναπτύχθηκε ένα εργαλείο γραφικής σχεδίασης εννοιολογικών σεναρίων Εξαγωγής–Μετασχηματισμού–Φόρτωσης δεδομένων σε περιβάλλον Αποθηκών Δεδομένων, το οποίο επιτρέπει την εύκολη σχεδίαση αλλά και περαιτέρω ανάπτυξη τους. Το εργαλείο αυτό προσφέρει ένα εύχρηστο γραφικό περιβάλλον για τη δημιουργία των σεναρίων και τη δυνατότητα αποθήκευσής στους σε ένα repository. Με χρήση του μηχανισμού μετάβασης από το εννοιολογικό στο λογικό μοντέλο και του αντίστοιχου εργαλείου λογικής σχεδίασης που περιέχεται στο πρόγραμμα *Άρκτος II*, μπορούμε να χρησιμοποιήσουμε το repository σαν πηγή σεναρίων κατά την εξέλιξη της Αποθήκης Δεδομένων.

Το μεγαλύτερο όμως όφελος που αποκομίζουμε από τη χρήση του μοντέλου γραφικής απεικόνισης διεργασιών ΕΜΦ, και κατ' επέκταση του εργαλείου, είναι η δημιουργία μίας "εποπτικής εικόνας" του σεναρίου, στην οποία απεικονίζεται με σαφήνεια τόσο ο διαχωρισμός, όσο και οι συσχετίσεις των διαδικασιών που χρησιμοποιούμε, γεγονός που διευκολύνει σημαντικά την εξέλιξη του σεναρίου, ώστε να ακολουθεί τις συνεχώς μεταβαλλόμενες απαιτήσεις της Αποθήκης Δεδομένων. Προσφέραμε λοιπόν σχεδιαστική και μεθοδολογική θεμελίωση του σχεδιασμού, της ανάπτυξης και της υλοποίησης των διεργασιών ΕΜΦ.

9.2 Μελλοντικές Επεκτάσεις

Η βασικότερη επέκταση που μπορεί να γίνει στο σύστημα, ώστε αυτό να γίνει πιο ολοκληρωμένο είναι ίσως η συγχώνευση των δύο εργαλείων (εννοιολογικής και λογικής) σχεδίασης που περιέχονται στο πρόγραμμα *Άρκτος ΙΙ* σε ένα, και η δημιουργία έτσι ενός ολοκληρωμένου εργαλείου Εξαγωγής–Μετασχηματισμού–Φόρτωσης δεδομένων.

Αν αποφασίσουμε να κινηθούμε προς την αντίθετη κατεύθυνση, σε σχέση με την πρώτη επέκταση, και να ανεξαρτητοποιήσουμε πλήρως τα δύο εργαλεία, μία δεύτερη επέκταση που θα μπορούσε να γίνει, είναι η προσθήκη δυνατότητας (πλήρους) ορισμού νέων μετασχηματισμών στο repository.

Γλωσσάρι

active candidate relationship	σχέση ενεργού υποψηφίου
active candidate	ενεργός υποψήφιος
activity	διεργασία
attribute	γνώρισμα
candidate relationship	σχέση υποψηφίου
candidate	υποψήφιος
concept	έννοια
data staging area	μεταβατική περιοχή φόρτωσης δεδομένων
data store	χώρος αποθήκευσης δεδομένων
data warehouse	αποθήκη δεδομένων
etl constraint	περιορισμός ΕΜΦ
instance-of relationship	σχέση στιγμιοτύπου
note	σχόλιο
on-line analytical processing	συνεχής αναλυτική επεξεργασία
provider relationship	σχέση παροχής
serial composition	σειριακή σύνθεση (μετασχηματισμών)
transformation	μετασχηματισμός

Βιβλιογραφία

[1] Arktos II: Internal project of kdbms. Available at: http://www.dblab.ece.ntua.gr/ arktos.

[2] Ascential Software Inc. Available at: http://www.ascentialsoftware.com.

[3] Ascential Software Products. Data warehousing technology. Available at: http://www.ascentialsoftware.com/products/datastage.html.

[4] C. Ballard. *Data Modeling for Data Warehousing*. IBM Red Book, 1998. ISBN 0738402451.

[5] Pieter Bekaert, Bert Van Nuffelen, Bruynooghe Bruynooghe, David Gilis, and Marc Denecker. On the transformation of object-oriented conceptual models to logical theories. *Lecture Notes in Computer Science*, 2503:152–??, 2002.

[6] P.A. Bernstein and T. Bergstraesser. Meta-data support for data transformations using microsoft repository. *Bulletin of the Technical Committee on Data Engineering, Special Issue on Data Transformations*, 22(1):9–14, 1999.

[7] G. Booch, I. Jacobson, and J. Rumbaugh. *The Unified Modeling Language User Guide*. Addison–Wesly Pub Co, 1st edition, October 1998.

[8] V. Borkar, K. Deshmuk, and S. Sarawagi. Automaticaly Extracting Structure from Free Text Addresses. *Bulletin of the Technical Commitee on Data Engineering*, 23(4), 2000.

[9] M. Bouzeghoub, F. Fabret, and M. Matulovic. Modeling Data Warehouse Refreshment Process as a Workflow Application. In *Proceedings of International Workshop on Design and Managment of Data Warehouses, (DMDW)*, Heidelberg, Germany, 1999.

[10] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Information Integration: Conceptual Modeling and Reasonging Support. In *Proceedings of the International Conference on Cooperative Information Systems (COOPIS)*, pages 280–291, New York, USA, August 1998.

[11] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. A Principled Approach to Data Integration and Reconciliation in Data Warehousing. In *International Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, 1999.

[12] Workflow Management Coalition. Interface 1: Process definition interchange process model. Document Number WfMC TC–1016–P. Available at: www.wfmc.org, 1998.

[13] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An Extensible Data Cleaning Tool. In *ACM SIGMOD International Conference On the Management of Data*, page 590, Dallas, Texas, 2000.

[14] Gartner. Etl magic quadrant update: Market pressure increases, January 2003. Gartner's Strategic Data Management Research Note, M-19-1108.

[15] Giga Information Group. Market overview update: Etl, March 2002. Technical Report RPA-032002-00021.

[16] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: a Conceptual Model for Data Warehouses. *Invited Paper, International Journal of Cooperative Information Systems*, 7 (2&3), 1998.

[17] M. Golfarelli and S. Rizzi. Methodological Framework for Data Warehouse Design. In *ACM First International Workshop on Data Warehousing and OLAP (DOLAP'98)*, pages 3–9, Bethesda, Maryland, USA, November 1998.

[18] C. Graves, M. Scott, M. Benkovich, P. Turley, R. Skoglund, R. Dewson, S. Youness, D. Lee, S. Ferguson, T. Bain, and T. Joubert. *Professional SQL Server 2000 Data Warehousing with Analysis Services*. Wrox Press Ltd, 1st edition, October 2001.

[19] Karl Hahn, Carsten Sapia, and Markus Blaschka. Automatically generating OLAP schemata from conceptual graphical models. In Rokia Missaoui and Il-Yeol Song, editors, *Proceedings of the 3rd International Workshop on Data Warehousing and OLAP(DOLAP-00)*, pages 9–16, N. Y., November 10 2000. ACM Press.

[20] B. Husemann, J. Lechtenborger, and G. Vossen. Conceptual Data Warehouse Modeling. In *Proceedings of the 2nd International Workshop on Design and Managment of Data Warehouses, (DMDW)*, pages 6.1–6.11, Stockholm, Sweden, 2000.

[21] IBM. Ibm data warehouse manager. Available at: http://www-3.ibm.com/software/data/db2/datawarehouse.

[22] Informatica. Powercenter. Available at: http://www.informatica.com/products/data+integration/powercenter/default.htm.

[23] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis, editors. *Fundamentals of Data Warehouses*. Springer, Berlin, Heidelberg, New York, 2003.

[24] R. Kimbal. A Dimensional Modeling Manifesto. *DBMS Magazine*, August 1997.

[25] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley and Sons, February 1998.

[26] Microsoft. Data transformation services. Available at: http://www.microsoft.com.

[27] Microsoft Corp. OLEDB specification. Available at: http://www.microsoft.com/data/oledb.

[28] A. Monge. Within a Duplicate Detection System. *Bulletin of the Technical Commitee on Data Engineering*, 23(4), 2000.

[29] D.L. Moody and M.A.R. Kortink. From Enterprise Models to Dimensional Models: A methodology for Data Warehouse and Data Mart Design. In *Proceedings of the 2nd International Workshop on Design and Managment of Data Warehouses, (DMDW)*, Stockholm, Sweden, 5–6 June 2000. Available at: http://sunsite.informatik.rwthaachen.de/Publications/CEUR-WS/Vol-28/.

[30] T.B. Nguyrn, A. Min Tjoa, and R.R. Wagner. An Object Oriented Multidimensional Data Model for OLAP. In *Proceedings of the First International Conference on Web-Age Information Management (WAIM-99)*, Shangai, China, June 2000.

[31] Oracle. Oracle warehouse builder product page. Available at: http://otn.oracle.com/products/warehouse/content.html.

[32] Oracle. Oracle 9i warehouse builder, architectural white paper, April 2002.

[33] V. Peralta. Data Warehouse Logical Design from Multidimensional Conceptual Schemas. 2003.

[34] C. Phipps. Migrating an operational database schema to data warehouse schemas, 2002.

[35] C. Phipps and K. C. Davis. Automating data warehouse conceptual schema design and evaluation. pages 52–61, Toronto, Canada, 2002.

[36] E. Rahm and H. Do. Data Cleaning: Problems and Current Approaches. *Bulletin of the Technical Commitee on Data Engineering*, 23(4), 2000.

[37] V. Raman and J. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB)*, pages 381–390, Roma, Italy, 2001.

[38] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. Extending the E/R Model for the Multidimensional Paradigm. In *ER Workshops*, pages 105–116, 1998. Lecture Notes in Computer Science 1552 Springer 1999.

[39] A. Simitsis. *Modeling and Optimization of Extraction-Transformation-Loading (ETL) Processes in Data Warehouse Environments*. PhD thesis, School of Electrical and Computer Engineering, National Technical University of Athens, 2004.

[40] A. Simitsis, P. Vassiliadis, and T. Sellis. Optimizing ETL processes in DW enviroments. In *21st IEEE International Conference on Data Engineering*, Tokyo, Japan, 2005.

[41] Π. Γιωργαντάς. Γραφική Αναπαράσταση Διαδικασιών Εξαγωγής-Μετασχηματισμού-Φόρτωσης Δεδομένων, 2002.

[42] J.C. Trujillo, M. Palomar, and J. Gómez. Applying Object-Oriented Conceptual Modeling Techniques to the Design of Multidimensional Databases and OLAP Applications. In *Proceedings of the First International Conference on Web-Age Information Management (WAIM-99)*, pages 83–94, Shangai, China, June 2000.

[43] N. Tryfona, F. Busborg, and J.G.B. Christiansen. starER: A Conceptual Model for Data WarehouseDesign. In *ACM 2nd International Workshop on Data Warehousing and OLAP (DOLAP'99)*, pages 3–8, Kansas City, Missouri, USA, November 1999.

[44] A. Tsois. MAC: Conceptual Data Modeling for OLAP. In *3rd International Workshop on Design and Management of Data Warehouses (DMDW)*, pages 5.1–5.11, Interlaken, Switzerland, 2001.

[45] P. Vassiliadis, C. Quix, Y. Vassiliou, and M. Jarke. Data Warehouse Process Management. *Information Systems*, 26(3):205–236, June 2001.

[46] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. To appear in Information Systems, 2004.

[47] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *ACM Fifth International Workshop on Data Warehousing and OLAP (DOLAP'02)*, pages 14–21, McLean, Virginia, USA, November 2002.

[48] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Modeling ETL activities as graphs. In *Proceedings of Design and Management of Data Warehouses (DMDW'02) 4th International Workshop in conjuction with CAiSE'02*, pages 52–61, Toronto, Canada, 27 May 2002.

[49] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. ARKTOS: Towards the Modeling, Design, Control Execution of ETL Processes. *Information Systems*, 26(1):537–561, December 2001. Elsevier Science Ltd.

[50] Achim Ulbrich vom Ende and Michael Boehnlein. Deriving initial data warehouse structures from the conceptual data models of the underlying operational information systems, November 10 1999.

Ευρετήριο

Α

ανάστροφη σχεδίαση και συλλογή απαιτήσεων 5
αποθήκη δεδομένων 2
Αρκτος II 5, 6

Γ

γενικότητα 25
γενικοί κατασκευαστές 25
γνώρισμα 13, 30
γράφος αρχιτεκτονικής 35

Δ

διαχείριση 6
διεργασία 31
δομημένη οντότητα 31
δυαδικός μετασχηματισμός 27

Ε

ΕΜΦ 3
ενεργός υποψήφιος 17
εννοια 13
εννοιολογικό μοντέλο 5, 11
επίπεδο
  μετα-μοντέλου 25
  προτύπων 25
  σχήματος 25
επίπεδο μετασχηματισμού 54
επεκτασιμότητα 25
εξάρτηση 30

Κ

κύκλος ζωής 5
κατασκευή λογισμικού 5
κλάση
  εννοια 25
  γνώρισμα 25
  μετασχηματισμός 25
  περιορισμός ΕΜΦ 25
  σχέση 25

Λ

λειτουργία επεξεργασίας αρχείων 28
λειτουργίες μεταφοράς 28
λογικό μοντέλο 5
λογικός σχεδιασμός 5

Μ

μεταβατική περιοχή αποθήκευσης δεδομένων 4
μετασχηματισμός 15
μετασχηματισμοί ισοδύναμης σειράς 55
μετασχηματισμοί ισοδύναμου επιπέδου 54
μετασχηματιστής 43
μετρήσεις λογισμικού 5
μοναδιαίος μετασχηματισμός 27

Ο

ορος 30

Π

περιορισμός ΕΜΦ 15
πρότυπη μορφή 47

Ρ

ρύθμιση 5

Σ

σύνθετος μετασχηματισμός 27
σύνολο εγγραφών 31
σειριακή σύνθεση μετασχηματισμών 20
σχέση
  απορρέουσας παροχής (λογικό) 36
  ενεργού υποψηφίου 17
  μέρους (λογικό) 35
  παροχής 18
  παροχής (λογικό) 33, 35
  ρύθμισης (λογικό) 35
  στιγμιοτύπου 25
  στιγμιοτύπου (λογικό) 35
  υποψηφίου 17
σχήμα
  απόρριψης 32, 33
  εισόδου 32
  εξόδου 32
  λειτουργικότητας 33
  παραμέτρων 32
σχόλιο 16

σταθερά 30
συνάρτηση 31
συνεχής αναλυτική επεξεργασία 2

Τ

τύπος δεδομένων 30

Υ

υπευθυνότητα 30
υποψήφιος 17

Φ

φίλτρα 27
φίλτρο 43
φυσικό μοντέλο 5