ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΠΑΤΡΑΣ

ΠΑΡΑΡΤΗΜΑ ΑΜΑΛΙΑΔΑΣ

ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ ΕΦΑΡΜΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΣΤΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΤΗΝ ΟΙΚΟΝΟΜΙΑ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ:

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΑΝΑΠΤΥΞΗ ΕΡΓΑΛΕΙΟΥ ΑΥΤΟΜΑΤΗΣ ΕΞΑΓΩΓΗΣ ΤΟΥ ΜΟΝΤΕΛΟΥ ΔΕΔΟΜΕΝΩΝ (ODJECT DATA MODEL) ΑΠΕΙΚΟΝΙΖΟΝΤΑΣ ΤΟ ΣΧΗΜΑ ΜΙΑΣ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ ΣΕ ΚΩΔΙΚΑ VB.NET ΚΑΙ C#.NET

DESINGING AND DEVELOPMENT A TOOL FOR AUTOMETED EXPORTATION OF THE ODJECT DATA MODEL REPRESENTING THE SCHEMA OF A DATA BASE IN CODE OF VB.NET AND C#.NET

ΠΑΠΑΜΙΚΡΟΥΛΕΑΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

ΧΕΙΜΩΝΑΣ ΝΙΚΟΛΑΟΣ

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: ΧΟΧΟΛΗΣ ΔΙΟΝΥΣΙΟΣ

ΑΜΑΛΙΑΔΑ 2011

Ευχαριστήσουμε θερμά τον καθηγητή μας Χόχολη Διονύσιο για την πολύτιμη βοήθεια του σε όλη τη διάρκεια αυτής της εργασίας

Πίνακας περιεχομένων

Περίληψη

Τα τελευταία χρόνια οι εφαρμογές λογισμικού παρουσιάζουν αυξανόμενες απαιτήσεις όσον αφορά τον χρόνο υλοποίησης καθώς και την ποιότητα του παραγόμενου κώδικα. Για τον λόγο αυτό έχουν αναπτυχθεί αρκετές σύγχρονες τεχνολογίες και μεθοδολογίες σχεδίασης και υλοποίησης εφαρμογών με σκοπό την ταχύτερη και αποδοτικότερη υλοποίηση. Για παράδειγμα σχεδόν όλες οι σύγχρονες εφαρμογές σχεδιάζονται και υλοποιούνται με την χρήση της πολυεπίπεδης αρχιτεκτονικής(n-tier architecture) καθώς και ειδικών τεχνικών σχεδίασης(design patterns) όπως είναι το table module (ενότητα πίνακα) και το μοντέλο τομέα(domain model). Ωστόσο όλες οι τεχνικές παρουσιάζουν μειονεκτήματα και πλεονεκτήματα και για αυτόν τον λόγο θα πρέπει να επιλέγεται η κατάλληλη τεχνική ανάλογα με τις ειδικές απαιτήσεις της κάθε υλοποίησης.

Η συγκεκριμένη πτυχιακή αφορά την μελέτη και αξιολόγηση, μέσω συγκεκριμένων πρακτικών πειραμάτων, των προαναφερθέντων τεχνικών και μυθολογιών με στόχο την εξαγωγή σημαντικών συμπερασμάτων για το ποια θα πρέπει να επιλεχθεί και σε ποιά περίπτωση. Τα πειράματα υλοποιούνται χρησιμοποιώντας την πλατφόρμα .NET η οποία παρέχει την τεχνολογία ADO .NET η οποία βασίζεται στην τεχνική σχεδίασης table module (ενότητα πίνακα) Μια επίσης σύγχρονη τεχνολογία η οποία χρησιμοποιείται στην πλατφόρμα .NET ,αλλά δεν περιέχεται σε αυτήν, είναι το NHibernate η οποία βασίζεται στην τεχνική σχεδίασης μοντέλο τομέα(domain model).

Σκοπός της συγκεκριμένης πτυχιακής είναι η μελέτη και αξιολόγηση των δυο παραπάνω τεχνολογιών και η παρουσίαση των πλεονεκτημάτων και μειονεκτημάτων που παρουσιάζουν στην υλοποίηση πολυεπίπεδων εφαρμογών.

Abstract

In recent years, software applications are increasing demands on the timing of implementation and the quality of the code. For this reason are developed a number of modern technologies and methodologies for designing and implementing applications for faster and more efficient implementation. For example, almost all modern applications are designed and implemented using a layered architecture (n-tier architecture) as well as specific technical design such as table module and the domain model. However, all techniques have advantages and disadvantages and for this reason should be chosen the appropriate technique depending on the specific requirements of each implementation.

This thesis concerns the study and evaluation, through specific practical experiments, the above techniques and mythology with the aim of drawing significant conclusions about what should be selected and in what circumstances. The experiments were carried out using the platform .NET that provides technology ADO. NET based on the technique table module. Another technology used in the platform. NET, but not contained in it is the NHibernate based on the technique domain model.

The purpose of this project is the study and evaluation of these two technologies and to present the advantages and disadvantages in implementing multi-tier applications.

Κεφάλαιο 1

Εισαγωγή

Εισαγωγή

Η πτυχιακή μας θέλει να μας βοηθήσει να εξοικειωθούμε με τις διάφορες σύγχρονες τεχνολογίες και μεθοδολογίες σχεδίασης και υλοποίησης εφαρμογών και συγκεκριμένα με την χρήση πολυεπίπεδης αρχιτεκτονικής σε συνδυασμό με την χρήση ADO.NET ή NHibernate. Ο στόχος είναι μετά την χρήση τους να μπορέσουμε να κατανοήσουμε τα πλεονεκτήματα και μειονεκτήματα τους ώστε να βρούμε τον πιο αποδοτικό τρόπο υλοποίησης μιας εφαρμογής με βάση τον χρόνο που απαιτείται για την υλοποίηση της, την σταθερότητα της και την ευελιξία που αποκτά.

Για να μπορέσουμε να κατανοήσουμε τις τεχνικές υλοποίησης των εφαρμογών θα υλοποιήσουμε τρία πειράματα.

- Στο πρώτο πείραμα θα κάνουμε χρήση της πολυεπίπεδης αρχιτεκτονικής σε συνδυασμό με την ADO.NET,
- Στο δεύτερο πείραμα θα κάνουμε χρήση της πολυεπίπεδης αρχιτεκτονικής σε συνδυασμό με ADO.NET και στοιχεία Nhibernate(θα χρησιμοποιήσουμε το domain model που είναι βασικό στοιχείο του NHibernate).
- Στο τρίτο πείραμα θα χρησιμοποιήσουμε πολυεπίπεδη αρχιτεκτονική σε συνδυασμό με Nhibernate.

Στην εποχή μας η διαχείριση των πληροφοριών συνήθως γίνεται με την χρήση μια αντικειμενοστραφούς γλώσσας προγραμματισμού(vb.net, c#, java) σε συνδυασμό με την χρήση σχεσιακών βάσεων δεδομένων. Αυτό έχει σαν αποτέλεσμα την δημιουργία και γρήγορη εξάπλωση των πληροφοριακών συστημάτων διαχείρισης δεδομένων. Τα πληροφοριακά συστήματα χρησιμοποιούνται από εταιρίες, οργανισμούς, σχολές και γενικά σε οποία περίπτωση υπάρχει μεγάλος όγκος δεδομένων. Στην πτυχιακή μας επιλέγουμε να πειραματιστούμε με την υλοποίηση μιας εφαρμογής η οποία θα μπορούσε να είναι ένα τμήμα κάποιου πιθανού πληροφοριακού συστήματος διαχείρισης μιας σχόλης.

Λόγο του ότι θέλουμε να συγκρίνουμε τις διάφορες σύγχρονες τεχνολογίες και μεθοδολογίες σχεδίασης και υλοποίησης εφαρμογών θα υλοποιήσουμε τρία πειράματα. Τα πειράματα αυτά με την υλοποίηση τους δημιουργούν την ίδια εφαρμογή τρεις φορές. Η εφαρμογή μας θα αποτελείται από μια φόρμα η οποία θα είναι ένα εργαλείο με το οποίο θα μπορεί κάποιος να συνδέεται στην βάση δεδομένων της σχολής και να βλέπει τα μαθήματα με τις εργαστηριακές τους ομάδες, τους φοιτητές που είναι εγγεγραμμένοι σε αυτές, αυτούς που δεν έχουν εγγραφτεί και να διαγραφεί ή να εγγράφει φοιτητές σε εργαστηριακές ομάδες.

Για να γίνει η υλοποίηση των πειραμάτων με τους πρώτους δυο τρόπους θα κάνουμε χρήση της γλώσσας προγραμματισμού VB.NET και των προγραμμάτων Visual Studio 2005 το οποίο είναι ένα περιβάλλον προγραμματισμού που υποστηρίζει την πλατφόρμα .NET η οποία παρέχει την

τεχνολογία ADO.NET η οποία βασίζεται στην τεχνική σχεδίασης table module (ενότητα πίνακα) και Microsoft Sql server 2005 οπού αυτό είναι το πρόγραμμα το οποίο περιέχει την βάση δεδομένων που θα χρησιμοποιήσουμε και επιτρέπει να γίνει η διαχείριση της.

Για να γίνει η υλοποίηση του τρίτου πειράματος έκτος από την χρήση των προγραμμάτων Visual Studio 2005 και Microsoft Sql server 2005 θα χρησιμοποιήσουμε Nhibernate το οποίο χρησιμοποιείται στην πλατφόρμα .NET ,αλλά δεν περιέχεται σε αυτήν και βασίζεται στην τεχνική σχεδίασης domain model (μοντέλο τομέα), το πρόγραμμα MyGeneration το οποίο είναι ένα πρόγραμμα που συνδέεται σε μια βάση δεδομένων και επιλεγούμε ποιους από τους πίνακες της βάσης θα χαρτογραφήσει για μας. Το τελευταίο εργαλείο που θα χρησιμοποιήσουμε είναι η ιστοσελίδα http://converter.telerik.com η οποία μετατρέπει κώδικα από την γλώσσα προγραμματισμού C# σε VB.NET και μας είναι απαραίτητη λόγο του ότι η χαρτογράφηση από το MyGeneration γένεται σε C# ενώ εμείς θέλουμε VB.NET.

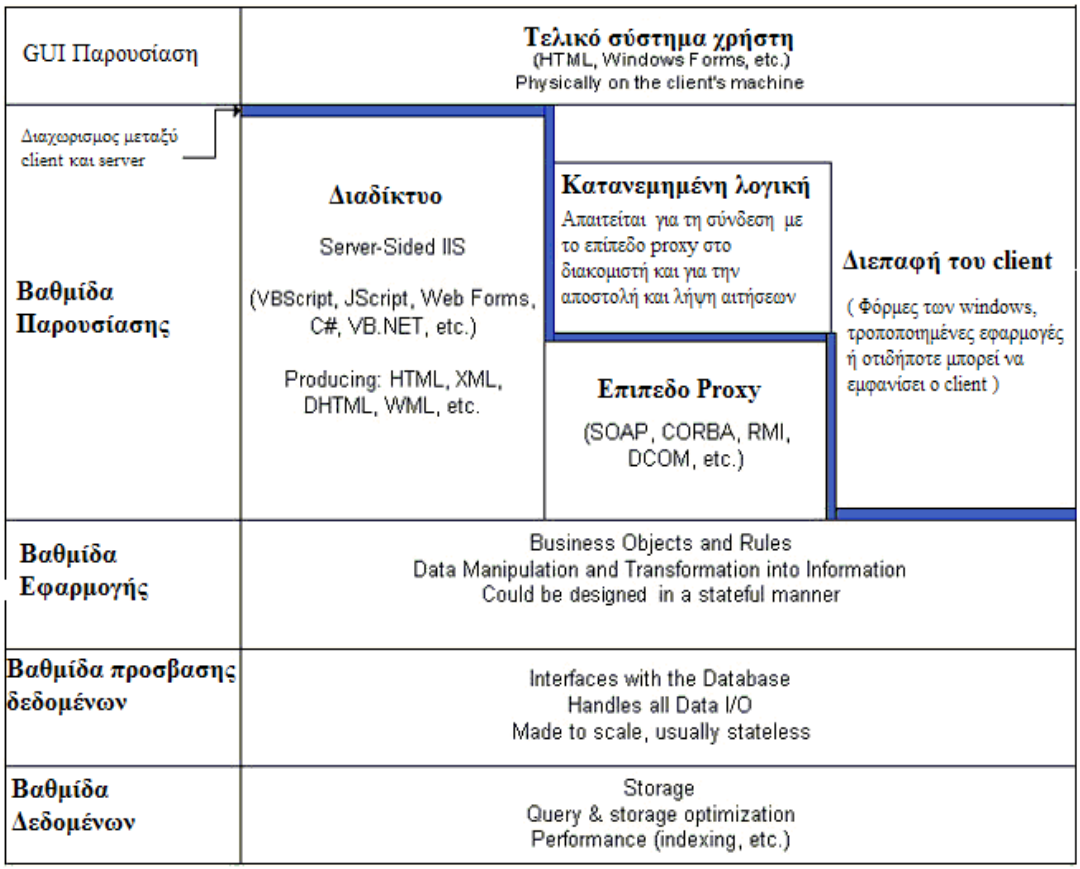
Η εργασία μας χωρίζεται σε δυο τμήματα. Το πρώτο τμήμα περιέχει το θεωρητικό υπόβαθρο και το δεύτερο τμήμα την υλοποίηση των πρακτικών πειραμάτων. Στο θεωρητικό κομμάτι περνούμε μια γεύση από το πώς δουλεύει η κάθε μια από τις τεχνικές που θα χρησιμοποιήσουμε μαθαίνουμε τα πλεονεκτήματα μειονεκτήματα τους τις συγκρίνουμε μεταξύ τους. Στο δεύτερο κομμάτι, το πρακτικό, έχουμε υλοποιήσει τα πειράματα μας με τους τρεις διαφορετικούς τρόπους βλέπουμε στην πράξη την θεωρία μας και κάνουμε την απαραίτητη σύγκριση ώστε να κατανοήσουμε πότε και γιατί χρησιμοποιούμε την κάθε τεχνολογία και μεθοδολογία σχεδίασης και υλοποίησης εφαρμογών.

Κεφάλαιο 2

Θεωρικό υπόβαθρο

N-tier Αρχιτεκτονική

Η N-tier αρχιτεκτονική παρέχει ένα μοντέλο για την δημιουργία μιας ευέλικτης και επαναχρησιμοποιήσιμης εφαρμογής. Με την ανάλυση μιας εφαρμογής σε επίπεδα, οι προγραμματιστές πρέπει μόνο να τροποποιήσουν ή να προσθέσουν ένα ειδικό επίπεδο, αντί να πρέπει να ξαναγράψουν ολόκληρη την εφαρμογή ξανά, εάν αποφασίσουν να αλλάξουν οι τεχνολογίες ή να κλιμακώσουν/επεκτήνουν την εφαρμογή. Στον όρο "N-tier," το "N" υποδηλώνει οποιονδήποτε αριθμό (όπως 2-tier, ή 4-tier) από διαφορετικές βαθμίδες που χρησιμοποιούνται στην αρχιτεκτονική .
Αρχιτεκτονικές εφαρμογών είναι μέρος του στρώματος 7 του OSI μοντέλου.



Εικόνα 1 N-tier **αρχιτεκτονική**

ΟΡΙΣΜΟΣ: Ένα πρόγραμμα που είναι βασισμένο σε n-tier αρχιτεκτονική είναι εκείνο που κατανέμεται σε τρεις ή περισσότερους ξεχωριστούς υπολογιστές σε ένα κατανεμημένο δίκτυο

Η δομή μιας N-tier εφαρμογής συνεπάγεται την χρήση client /server προγραμματιστικού μοντέλου. Όταν υπάρχουν περισσότερα από τρία επίπεδα ή βαθμίδες (tiers) που εμπλέκονται, οι πρόσθετες βαθμίδες στην εφαρμογή συνδέονται συνήθως με τη βαθμίδα της επιχειρηματικής λογικής.

Επιπρόσθετα στα πλεονεκτήματα του κατανεμημένου προγραμματισμού και των δεδομένων στο σύνολο ενός δικτύου, οι N-tier εφαρμογές, παρουσιάζουν το πλεονέκτημα ότι κάθε μία βαθμίδα(tier) μπορεί να τρέχει σε ένα κατάλληλο επεξεργαστή ή λειτουργικό σύστημα και μπορούν να ενημερώνονται ανεξάρτητα από τις άλλες βαθμίδες. Για την επικοινωνία μεταξύ των βαθμίδων προγράμματος θα πρέπει να χρησιμοποιούνται σαφώς ορισμένες διεπαφές προγράμματος(programming interfaces) (1) (2).

3-tier Αρχιτεκτονική

Στον σχεδιασμό λογισμικού, **multi-tier αρχιτεκτονική** (που συχνά αναφέρεται ως *n-tier αρχιτεκτονική)* είναι μια αρχιτεκτονική client-server κατά την οποία η παρουσίαση, η επεξεργασία της εφαρμογής, καθώς και η διαχείριση των δεδομένων είναι λογικά ξεχωριστές διαδικασίες. Η πιο διαδεδομένη χρήση της "multi-tier αρχιτεκτονικής" αναφέρεται στην ***αρχιτεκτονική τριών-επιπέδων.*** Η τριών επιπέδων αρχιτεκτονική είναι στην ουσία αρχιτεκτονική client-server στην οποία το περιβάλλον χρήστη, η λειτουργική λογική διαδικασία, η αποθήκευση δεδομένων και η πρόσβαση στα δεδομένα αναπτύσσονται και συντηρούνται ως ανεξάρτητες ενότητες, τις περισσότερες φορές σε ξεχωριστές πλατφόρμες. Η αρχιτεκτονική τριών επιπέδων έχει σκοπό να επιτρέπει σε οποιαδήποτε από τις τρεις βαθμίδες να αναβαθμιστεί ή να αντικατασταθεί ανεξάρτητα, από τις άλλες ανάλογα με τις απαιτήσεις ή την αλλαγή της τεχνολογίας (3).

Η 3-tier αρχιτεκτονική έχει τις ακόλουθες τρεις βαθμίδες:

Βαθμίδα Παρουσίασης (Presentation tier)

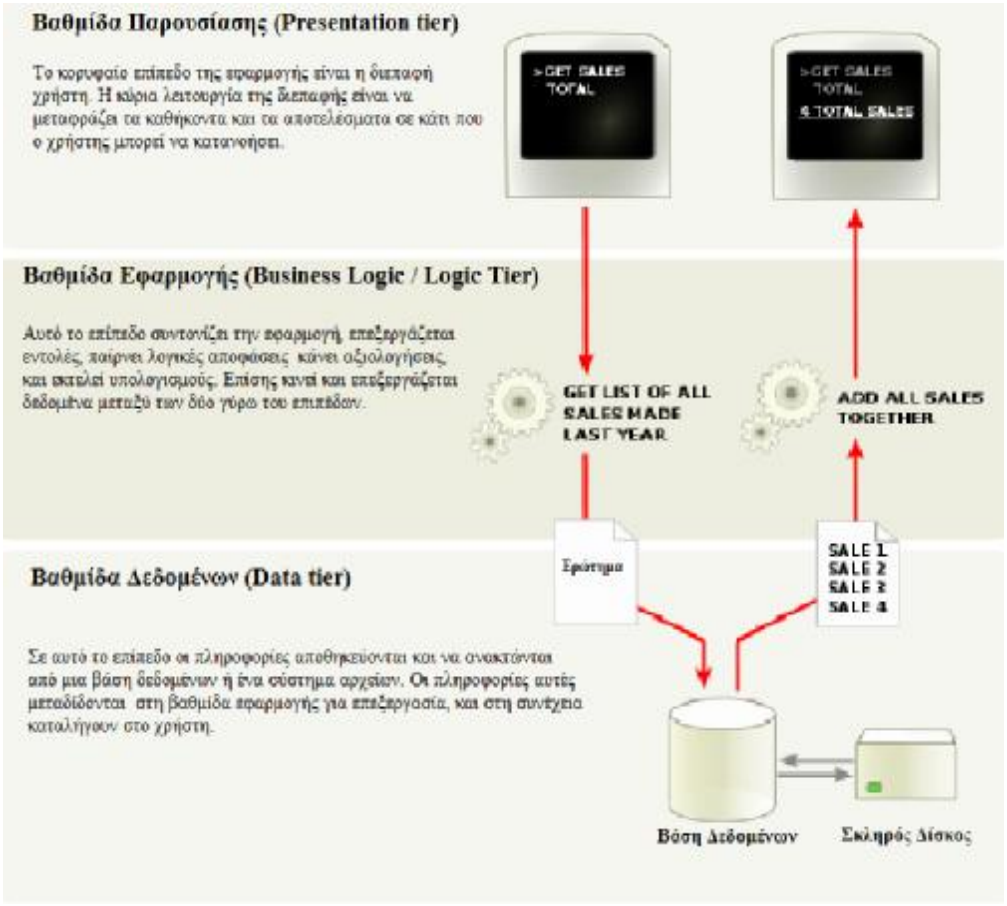
Το κορυφαίο επίπεδο της εφαρμογής είναι η διεπαφή χρήστη. Η κύρια λειτουργία της διεπαφής είναι να μεταφράζει τα καθήκοντα και τα αποτελέσματα σε κάτι που ο χρήστης μπορεί να κατανοήσει.

Βαθμίδα Εφαρμογής (Business Logic / Logic Tier)

Αυτό το επίπεδο συντονίζει την εφαρμογή, επεξεργάζεται εντολές, παίρνει λογικές αποφάσεις κάνει αξιολογήσεις, και εκτελεί υπολογισμούς. Επίσης κινεί και επεξεργάζεται δεδομένα μεταξύ των δύο γύρω του επιπέδων.

Βαθμίδα Δεδομένων (Data tier)

Σε αυτό το επίπεδο οι πληροφορίες αποθηκεύονται και ανακτώνται από μια βάση δεδομένων ή ένα σύστημα αρχείων. Οι πληροφορίες αυτές μεταδίδονται στη βαθμίδα εφαρμογής για επεξεργασία, και στη συνέχεια καταλήγουν στο χρήστη.



Εικόνα 2 3-tier αρχιτεκτονική

Στόχοι μιας καλής N-Tier Εφαρμογής

Ο N-tier σχεδιασμός προέκυψε ως αποτέλεσμα των αδυναμιών του client/server μοντέλου. Υπάρχουν πολλοί στόχοι που μια εφαρμογή με n-tier σχεδιασμό θα πρέπει να επιτύχει. Εδώ είναι μερικοί από αυτούς.

- Εάν αλλάξουν βασικές μέθοδοι πρόσβασης στα δεδομένα, ο κώδικας του client δεν θα πρέπει αναγκαστικά να αλλάξει.
- Όλες οι ρουτίνες πρόσβασης δεδομένων θα πρέπει να εμφανίζονται ως αντικείμενα αντί για κλήσεις λειτουργίας.
- Η SQL πρέπει να εξαλειφθεί από το κώδικα του client. Ο κώδικας του client θα πρέπει απλώς να ασχολείται με τις μεθόδους και τις ιδιότητες.
- Οι Πίνακες και τα ονόματα στηλών θα πρέπει να εξαλειφθούν από το κώδικα του client.
- Ο κώδικας του client δεν πρέπει ενδιαφέρεται από πού προέρχονται τα δεδομένα. Θα πρέπει απλώς να ενδιαφέρεται να μπορεί να ανακτήσει και να τροποποιήσει τα δεδομένα σε κάποιο αντικείμενο και το αντικείμενο θα φροντίσει για τις λεπτομέρειες.
- Η κωδικοποίηση που πρέπει να γίνετε στον client θα πρέπει να απλουστευθεί. Αντί να χρησιμοποιεί πολλές λειτουργίες, η εφαρμογή πρέπει να είναι σε θέση να χρησιμοποιεί αντικείμενα με ιδιότητες και μεθόδους.
- Γίνεται πιο εύκολη η δημιουργία και η χρήση των κλάσεων από το να καλούνται λειτουργίες.
- Γίνεται πιο εύκολο να προστεθεί λειτουργικότητα στις εφαρμογές, και να αλλάξει η λειτουργία του , χωρίς να σπάει ο κώδικας του client.

Πλεονεκτήματα N-Tier

- Ο διαχωρισμός των αρμοδιοτήτων της εφαρμογής σε πολλαπλά επίπεδα καθιστά ευκολότερη την κλιμάκωση/επεκταση της εφαρμογής.
- Η N-tier αρχιτεκτονική επιτρέπει να διαχωριστεί καλύτερα ο φόρτος εργασίας για τους προγραμματιστές. Με τη διάσπαση της εφαρμογής σε ξεχωριστά κομμάτια επιτρέπεται η παράλληλη ανάπτυξη των επιπέδων της εφαρμογής. Έτσι οι προγραμματιστές με διαφορετικές ειδικότητες μπορούν να εστιάσουν σε μια βαθμίδα που ταιριάζει καλύτερα στις δεξιότητες τους.
- Μια εφαρμογή με N-tier αρχιτεκτονική είναι πιο ευανάγνωστη και τα συστατικά της πιο επαναχρησιμοποιήσιμα. Με τον διαχωρισμό μιας εφαρμογή σε επίπεδα αποφεύγεται η γραφή πολύπλοκου κώδικα ο οποίος οδηγεί σε ατελείωτες γραμμές κώδικα με μπερδεμένη δομή ελέγχου και πολλές εμφωλευμένες δηλώσεις IF .
- Ο διαχωρισμός μιας εφαρμογής σε επίπεδα παρέχει ενθυλάκωση των διαφόρων αυτών επιπέδων και των συστατικών τους, η οποία οδηγεί στη δημιουργία μιας πιο σταθερής εφαρμογής.
- Η χρήση επιπέδων, επιτρέπει την ευκολότερη συντήρηση και υποστήριξη, αφού είναι πιο εύκολο να αλλαχτεί και να ενημερωθεί ένα συγκεκριμένο στοιχείο από το να γίνουν αλλαγές σε ολόκληρη την εφαρμογή (4) (5).

Μειονεκτήματα N-Tier

Αν και υπάρχουν πολλά πλεονεκτήματα σε μια καλή n-tier εφαρμογή, υπάρχουν και κάποια μειονεκτήματα.

- Δημιουργούνται πολλές κλάσεις. Αυτό μπορεί να οδηγήσει σε θέματα συντήρησης και μπορεί ακόμη να είναι ένα ζήτημα απόδοσης αφού χρειάζεται χρόνος για να δημιουργηθεί μια νέα κλάση κατά το χρόνο εκτέλεσης.
- Η N-tier αρχιτεκτονική δεν λειτουργεί καλά, όταν είναι άγνωστη η δομή των πινάκων από τους οποίους γίνετε η ανάκτηση δεδομένων.
- Δεν έχουμε την δυνατότητα να δημιουργήσουμε εκθέσεις γιατί οι συγγραφείς εκθέσεων δεν χρησιμοποιούν κλάσεις για να έχουν πρόσβαση σε δεδομένα

Στο τέλος, τα πλεονεκτήματα ενός κάλου N-tier σχεδιασμού θα υπερβαίνουν κατά πολύ τα μειονεκτήματα. Στις περιπτώσεις όπου απλά δεν μπορεί να χρησιμοποιηθεί N-tier, χρησιμοποιείτε η τυπική client / server μέθοδος ανάπτυξης (2).

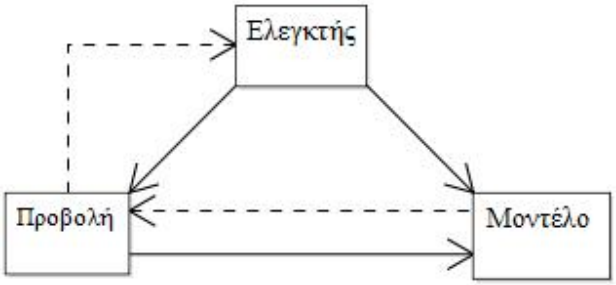
MVC

Η αρχιτεκτονική 3-tier δεν είναι η μοναδική επιλογή για την δημιουργία πολυεπίπεδων εφαρμογών. Μια άλλη παλιότερη στο χώρο και επιφανειακά παρόμια αρχιτεκτονική είναι η MVC.

Αρχιτεκτονική MVC

Η αρχιτεκτονική Model-View-Controller ασχολείται με τη διαίρεση των στοιχείων μιας εφαρμογής σε τρεις διαφορετικές κατηγορίες: Μοντέλο (Model), Προβολή (View) και Ελεγκτή (Controller). Οι συνιστώσες της αρχιτεκτονικής MVC έχουν μοναδική ευθύνη και κάθε συστατικό είναι ανεξάρτητο από το άλλο συστατικό. Οι αλλαγές σε ένα από τα στοιχεία έχουν μικρή ή καμία επίπτωση στα άλλα στοιχεία.

Εικόνα 3 Αρχιτεκτονική MVC



Ευθύνες των στοιχείων είναι:

Μοντέλο: Το **μοντέλο** είναι υπεύθυνο για την παροχή των στοιχείων από τη βάση δεδομένων και την αποθήκευση των δεδομένων στο χώρο αποθήκευσης δεδομένων. Όλη η επιχειρηματική λογική (business logic) εφαρμόζεται στο μοντέλο. Τα δεδομένα που έχουν εισαχθεί από τον χρήστη μέσω της θέας(view) ελέγχονται στο μοντέλο πριν από την αποθήκευση τους στη βάση δεδομένων. Η λογική της Πρόσβασης δεδομένων(Data access), επικύρωσης δεδομένων(Data validation) και αποθήκευσης δεδομένων(data saving) είναι μέρος του μοντέλου.

Προβολή: Η προβολή αντιπροσωπεύει την διεπαφή χρήστη της εφαρμογής και είναι υπεύθυνη για τη λήψη των εισροών (input) από το χρήστη, την αποστολή τους και, στη συνέχεια να λάβει τα αποτελέσματα και να εμφανίζει το αποτέλεσμα στο χρήστη.

Ελεγκτής: Ο Ελεγκτής είναι ενδιάμεσος μεταξύ Μοντέλου και Προβολής . Είναι υπεύθυνος για την παραλαβή της αίτησης από τον πελάτη(client). Μόλις το αίτημα από τον πελάτη ληφθεί ο ελεγκτής εκτελεί την κατάλληλη επιχειρηματική

λογική από το μοντέλο και στη συνέχεια παράγει το αποτέλεσμα στο χρήστη χρησιμοποιώντας το στοιχείο Προβολή (6).

Σύγκριση με την αρχιτεκτονική MVC

Εκ πρώτης όψεως, οι τρεις βαθμίδες μπορεί να φαίνονται παρόμοιο σχέδιο με το MVC (Model View Controller). Ωστόσο, τοπολογικά είναι διαφορετικά. Ένας βασικός κανόνας σε μια τριών επιπέδων αρχιτεκτονική είναι ότι το client tier δεν επικοινωνεί απευθείας με το data tier. Σε τριών επιπέδων μοντέλο κάθε επικοινωνία πρέπει να διέρχεται από το middleware tier. Εννοιολογικά η αρχιτεκτονική των τριών βαθμίδων είναι γραμμική. Ωστόσο, η αρχιτεκτονική MVC είναι τριγωνική: η Προβολή (View) στέλνει ενημερώσεις στο ελεγκτή(Controller), ο ελεγκτής ενημερώνει το μοντέλο, και η Προβολή (View) παίρνει ενημέρωση απευθείας από το μοντέλο ,σε αντίθεση με την N tier αρχιτεκτονική το MVC δεν αναφέρει ότι έχει κάποιο σταθερό μηχανισμό αποθήκευσης π.χ. μια βάση δεδομένων. Στην 3-tier αρχιτεκτονική μια εφαρμογή αναπτύσσεται σε 3 διαφορετικούς επεξεργαστές / διεργασίες: μια τέτοια διάσπαση είναι, πάνω απ 'όλα, φυσική. Η αρχιτεκτονική MVC καθιερώνει το διαχωρισμό των αρμοδιοτήτων μεταξύ των στοιχείων(components) σε επίπεδο περίπτωσης χρήσης, και δεν παραπέμπει σε φυσικά σύνορα.

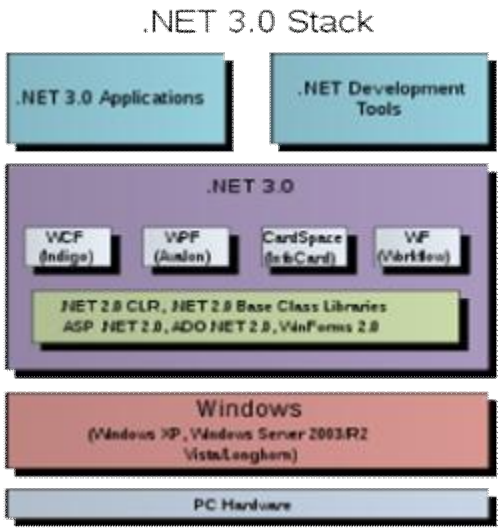
Από ιστορική άποψη, η ιδέα αρχιτεκτονική τριών βαθμίδων εμφανίστηκε στη δεκαετία του 1990 από τις παρατηρήσεις των κατανεμημένων συστημάτων *(π.χ.,* εφαρμογές web), όπου ο client, middleware και data tiers έτρεχαν σε φυσικά διαχωρισμένες πλατφόρμες. Η MVC προέρχεται από την προηγούμενη δεκαετία (από το έργο στο Xerox PARC στα τέλη της δεκαετίας του 1970 και στις αρχές του 1980) και βασίζεται στις παρατηρήσεις των εφαρμογών που έτρεξαν σε ένα ενιαίο γραφικό σταθμό εργασίας (3) (7) (8).

Τεχνολογίες ανάπτυξης N-tier εφαρμογών

Για την ανάπτυξη N-tier εφαρμογών έχουμε στην διάθεση μας αρκετές τεχνολογίες. Οι πιο διαδεδομένες είναι η ADO.NET, το Hibernate και το Nhibernate.

ADO.NET

Η **ADO.NET** είναι ένα σύνολο από στοιχεία του λογισμικού των ηλεκτρονικών υπολογιστών που μπορούν να χρησιμοποιηθούν από τους προγραμματιστές για την πρόσβαση σε δεδομένα και υπηρεσίες δεδομένων. Είναι ένα μέρος της base class library που περιλαμβάνεται με το Microsoft. NET Framework. Χρησιμοποιείται συνήθως από τους προγραμματιστές για να έχουν πρόσβαση και να τροποποιούν τα δεδομένα που αποθηκεύονται σε συστήματα σχεσιακών βάσεων δεδομένων (relational database systems), αν και μπορεί επίσης να χρησιμοποιηθεί για την πρόσβαση σε δεδομένα σε μη σχεσιακές πηγές. Η ADO.NET θεωρείται μερικές φορές μια εξέλιξη της ActiveX Data Objects (ADO) τεχνολογίας, αλλά άλλαξε τόσο εκτενώς, ώστε να μπορεί να θεωρηθεί ένα εντελώς νέο προϊόν.



Εικόνα 4 .NET **αρχιτεκτονική**

Αυτή η τεχνολογία είναι ένα μέρος του. NET Framework 3.0 (που αποτελεί μέρος του πλαισίου από την έκδοση 1.0)

Αρχιτεκτονική ADO.NET

Η ADO.NET αποτελείται από: τον πάροχο δεδομένων(Data provider), τα σύνολα δεδομένων(DataSets) και την ενότητα πίνακα(table module)

Πάροχος δεδομένων

Αυτές οι κλάσεις παρέχουν πρόσβαση σε μία πηγή δεδομένων, όπως ο Microsoft SQL Server ή Oracle_βάση δεδομένων και OLEDB πάροχος δεδομένων. Κάθε πηγή δεδομένων έχει το δικό της σύνολο των αντικειμένων παρόχου(provider objects), αλλά το καθένα έχει ένα κοινό σύνολο των κλάσεων χρησιμότητας(utility classes):

- **Σύνδεση** (Connection): Παρέχει ένα πλαίσιο που χρησιμοποιείται για την επικοινωνία με την πηγή δεδομένων. Επίσης, λειτουργεί ως αφηρημένο εργοστάσιο (abstract factory) για αντικείμενα εντολή (command objects).
- **Διοίκηση**(Command): Χρησιμοποιείται για να εκτελεστεί κάποια ενέργεια στην πηγή των δεδομένων, όπως ανάγνωση, ενημέρωση, διαγραφή σχεσιακών δεδομένων (relational data).
- **Παράμετρος** (Parameter): Περιγράφει μια μόνο παράμετρο σε μια εντολή. Ένα κοινό παράδειγμα είναι μια παράμετρος σε μια αποθηκευμένη διαδικασία.
- **DataAdapter**: Μια γέφυρα που χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ μιας πηγής δεδομένων και ενός αντικείμενου Dataset.
- **DataReader**: Χρησιμοποιείται για να επεξεργαστεί αποτελεσματικά μια μεγάλη λίστα αποτελεσμάτων μια προς μια εγγραφή. Επιτρέπει την πρόσβαση στις εγγραφές με read-only, forward-only τρόπο, δηλαδή, η πρόσβαση στα αρχεία πρέπει να γίνει με διαδοχική σειρά(sequential order). Δεν μπορούν να προσεγγιστούν τυχαία ούτε μπορεί μια εγγραφή η οποία έχει μεταποιηθεί προηγουμένως να προσπελαστεί πάλι.

Σύνολα δεδομένων

Τα Dataset αντικείμενα, είναι μια ομάδα από κλάσεις που περιγράφουν μια απλή in-memory σχεσιακή βάση δεδομένων (relational database). Οι κλάσεις αποτελούν μια ιεραρχία(containment hierarchy) :

- Ένα αντικείμενο **Dataset** αντιπροσωπεύει ένα σχήμα (είτε μια ολόκληρη βάση δεδομένων ή ένα υποσύνολο του ενός). Μπορεί να περιέχει πίνακες και τις σχέσεις μεταξύ των εν λόγω πινάκων.
    - Ένα **DataTable** αντικείμενο αποτελεί ένα ενιαίο πίνακα στη βάση δεδομένων. Έχει όνομα , σειρές και στήλες.
        - Ένα αντικείμενο **DataView** επικαλύπτει ένα DataTable ταξινομεί τα δεδομένα (όπως ακριβώς μια SQL "order by" ρήτρα) και

φιλτράρει τα αρχεία (όπως ακριβώς μια SQL ", where "ρήτρα) εάν έχει ρυθμιστεί ένα φίλτρο. Ένας in-memory δείκτης χρησιμοποιείται για να διευκολύνει αυτές τις διαδικασίες . Όλα τα DataTables έχουν προεπιλεγμένο ένα φίλτρο, ενώ κάθε αριθμός πρόσθετων DataViews μπορεί να οριστεί, μειώνοντας την αλληλεπίδραση με την υποκείμενη βάση δεδομένων και, συνεπώς, βελτιώνεται η επίδοση .

- Μια **DataColumn** αντιπροσωπεύει μια στήλη του πίνακα, με το όνομα και τον τύπο.
- Ένα αντικείμενο **DataRow** αντιπροσωπεύει μία μόνο γραμμή του πίνακα. Επιτρέπει την ανάγνωση και την αναπροσαρμογή των τιμών στην εν λόγω γραμμή, και την ανάκτηση τυχόν γραμμών που σχετίζονται με αυτό μέσω μιας, πρωτογενούς-κλειδιού ξένο-κλειδί, σχέσης (primary-key foreign-key relationship).
- Η **DataRowView** αντιπροσωπεύει μία μόνο γραμμή του DataView. Η διάκριση μεταξύ DataRow και DataRowView είναι σημαντική όταν γίνεται επανάληψη σε ένα σύνολο αποτελεσμάτων(result set).

§ Η **DataRelation** είναι μια σχέση μεταξύ πινάκων, π.χ. ένα πρωτεύον κλειδί-ξένο-κλειδί σχέση. Αυτό είναι χρήσιμο για να διευκολύνει τη λειτουργιά της DataRow στην ανάκτηση σχετικών σειρών.

o Ο **Περιορισμός** (Constraint) περιγράφει μια αναγκαστική ιδιοκτησία της βάσης δεδομένων, π.χ. η μοναδικότητα των αξιών σε μια στήλη “πρωτεύον κλειδί”. Όταν τα δεδομένα τροποποιούνται οποιεσδήποτε παραβάσεις προκύπτουν πρέπει να προκαλούν εξαιρέσεις.

Ένα σύνολο δεδομένων κατοικείται/γεμίζει από μια βάση δεδομένων μέσω ενός DataAdapter του οποίου οι ιδιότητες σύνδεση (Connection) και Διοίκηση (Command) έχουν τεθεί. Ωστόσο, ένα Dataset μπορεί να αποθηκεύσει το περιεχόμενό του σε XML (προαιρετικά με ένα σχήμα XSD), ή να γεμίσει(populate) από XML. Αυτό το καθιστά εξαιρετικά χρήσιμο για δικτυακές υπηρεσίες (web services), κατανεμημένα συστήματα πληροφορικής (distributed computing), και περιστασιακά-συνδεδεμένες εφαρμογές(occasionally-connected applications).

ΕΝΟΤΗΤΑ ΠΙΝΑΚΑ (TABLE MODULE)

Η ενότητα πίνακα είναι ένα υπόδειγμα που χειρίζεται την επιχειρηματική λογική όλων των γραμμών ενός πίνακα της βάσης δεδομένων ή μιας προβολής (view).

Η ενότητα πίνακα οργανώνει την επιχειρηματική λογική σε μία κλάση ανά πίνακα στη βάση δεδομένων, και ένα μόνο υπόδειγμα της κλάσης περιλαμβάνει τις διάφορες διαδικασίες που απαιτούνται.

Πώς λειτουργεί

Το πλεονέκτημα της ενότητας πίνακα είναι ότι μας επιτρέπει να “πακετάρουμε” τα δεδομένα μαζί με τη συμπεριφορά τους και έτσι εκμεταλλεύεται τα πλεονεκτήματα μιας σχεσιακής βάσης δεδομένων. Επιφανειακά η ενότητα πίνακα μοιάζει πολύ με ένα αντικείμενο. Η βασική διαφορά είναι ότι δεν έχει καμία ιδέα για την ταυτότητα των αντικείμενων που χρησιμοποιεί.

Συνήθως η ενότητα πίνακα χρησιμοποιείται με δομή δεδομένων που είναι προσανατολισμένη σε πίνακα. Τα tabular δεδομένα συνήθως προκύπτουν όταν καλούμε μια SQL κλάση που διατηρείται σε ένα σύνολο εγγραφών το οποίο μιμείται έναν SQL πίνακα. Συχνά χρειάζεται η συμπεριφορά από πολλές ενότητες πίνακα, ώστε να γίνει κάποια χρήσιμη εργασία. Πολλές φορές πολλαπλές ενότητες πίνακα λειτουργούν στο ίδιο Record Set.

Η ενότητα πίνακα μπορεί να είναι ένα υπόδειγμα, ή μπορεί να είναι μια συλλογή από στατικές μεθόδους. Τα πλεονεκτήματα ενός υποδείγματος είναι ότι επιτρέπει τη χρήση της κληρονομιάς αλλά και την αρχικοποίηση μιας ενότητας πίνακα με τη χρήση ενός υπάρχοντος συνόλου εγγραφών, ή με το αποτέλεσμα ενός ερωτήματος. Έπειτα με τη χρήση του υποδείγματος μπορούμε να χειραγωγήσουμε τις σειρές στο σύνολο εγγραφών.

Η λέξη “πίνακας” στο όνομα του προτύπου υποδεικνύει ότι υπάρχει μια ενότητα πίνακα ανά πίνακα στη βάση δεδομένων. Αυτό όμως δεν ισχύει πάντα. Είναι χρήσιμο να υπάρχει μια ενότητα πίνακα που θα περιέχει τις απόψεις ή τα ερωτήματα που χρησιμοποιούνται πιο συχνά. Η δομή της ενότητας πίνακα στην πραγματικότητα δεν εξαρτάται από τη δομή των πινάκων στη βάση δεδομένων, αλλά περισσότερο από τους εικονικούς πίνακες που αντιλαμβάνεται από την εφαρμογή, καθώς και τις απόψεις και τα ερωτήματα.

Πότε χρησιμοποιείται

Η ενότητα πίνακα βασίζεται σε μεγάλο βαθμό από table-oriented δεδομένα, έτσι προφανώς η χρήση της έχει νόημα όταν είναι απαραίτητη η πρόσβαση σε tabular δεδομένα χρησιμοποιώντας Record Set.

Ωστόσο, η ενότητα πίνακα δεν δίνει την πλήρη δύναμη των αντικειμένων στην οργάνωση πολύπλοκων λογικών. Δεν έχει άμεσες ,υπόδειγμα σε υπόδειγμα σχέσεις, και ο πολυμορφισμός δεν λειτουργεί καλά.

Η πιο γνωστή περίπτωση κατά την οποία χρησιμοποιείται είναι στο σχεδιασμό COM της Microsoft. Στο COM και .NET, το Record Set είναι ο κύριος χώρος αποθήκευσης των δεδομένων σε μια εφαρμογή. Οι βιβλιοθήκες ADO της Microsoft δίνουν ένα καλό μηχανισμό για την πρόσβαση σε σχεσιακά δεδομένα. Σε αυτή την περίπτωση η ενότητα πίνακα ταιριάζει την επιχειρηματική λογική στην εφαρμογή με ένα καλά οργανωμένο τρόπο, χωρίς να χάνει τον τρόπο με τον οποίο τα διάφορα στοιχεία δουλεύουν με tabular data.

ADO.NET και Visual Studio

Το Visual Studio IDE δημιουργεί εξειδικευμένες υποκατηγορίες των κλάσεων του DataSet για ένα συγκεκριμένο σχήμα βάσης δεδομένων, και επιτρέπει εύκολη πρόσβαση σε κάθε τομέα μέσω έντονα-δακτυλογραφημένων(strongly-typed) εντολών. Αυτό βοηθά στον εντοπισμό περισσότερων σφαλμάτων κατά τον έλεγχο (compile-time), καθιστώντας το Intellisense χαρακτηριστικό του IDE πιο αποδοτικό.

Entity Framework

Το ADO.NET Entity Framework είναι ένα σύνολο APIs δεδομένων πρόσβασης για το Microsoft .NET Framework. Μία οντότητα Framework Entity είναι ένα αντικείμενο το οποίο έχει ένα κλειδί που αντιπροσωπεύει το πρωτεύον κλειδί μιας datastore οντότητας. Μια εννοιολογική οντότητα, μοντέλο δεδομένων(Entity Data Model) (μοντέλο οντοτήτων-σχέσεων), αντιστοιχίζεται σε ένα datastore schema model. Χρησιμοποιώντας το Μοντέλο Δεδομένων οντότητας, η οντότητα-πλαισίου (Entity Framework) επιτρέπει στα στοιχεία που να αντιμετωπίζονται ως φορείς, ανεξάρτητα από τις υποκείμενες datastore παραστάσεις τους.

Η Οντότητα SQL είναι μια SQL-like γλώσσα για την αναζήτηση της οντότητας, μοντέλο δεδομένων(Entity Data Model) αντί των υποκειμένων datastore. Ομοίως, η LINQ επέκταση LINQ-to-Entities παρέχει δακτυλογραφημένα ερωτήματα για την οντότητα Μοντέλο Δεδομένων. Η οντότητα SQL και LINQ-to-Entities είναι εσωτερικά ερωτήματα που μετατρέπονται σε ένα Canonical Query Tree το οποίο στη συνέχεια μετατρέπεται σε ένα ερώτημα κατανοητό από τις υποκείμενες datastores (π.χ. σε SQL, στην περίπτωση μιας Σχεσιακής βάσης δεδομένων). Οι entities μπορούν να χρησιμοποιούν τις σχέσεις τους και οι αλλαγές τους δεσμεύονται πίσω στη datastore (9).

Πλεονεκτήματα της ADO.NET

Διαλειτουργικότητα (Interoperability)

Οι ADO.NET εφαρμογές παίρνουν πλεονέκτημα από την ελαστικότητα και ευρεία αποδοχή της XML. Επειδή XML είναι η διάταξη (Format) με την οποία στέλνουμε datasets μέσω δικτύου, όποιο στοιχείο (component) μπορεί να την διαβάσει μπορεί και να επεξεργαστεί τα δεδομένα. Το στοιχείο που λαμβάνει την XML δεν χρειάζεται να είναι στοιχείο της ADO.NET. Το στοιχείο που στέλνει το dataset το μόνο που κάνει είναι να το στείλει εκεί που πρέπει χωρίς να ενδιαφέρεται πως είναι φτιαγμένο το στοιχείο που το λαμβάνει.

Συντηρησιμότητα (Maintainability)

Κατά την λειτουργιά ενός συστήματος μικρές τροποποιήσεις είναι δυνατόν να γίνουν, αλλά σημαντικές αρχιτεκτονικές αλλαγές πολύ σπάνια επιχειρούνται γιατί είναι πολύ δύσκολο να γίνουν. Αυτό είναι λυπηρό γιατί στην πορεία των γεγονότων σημαντικές αλλαγές μπορεί να είναι απαραίτητες. Για παράδειγμα, καθώς μια εφαρμογή γίνεται γνωστή στους χρήστες, η αυξημένη εκτέλεση φορτώματος (increased performance load) το πιο πιθανό είναι να χρειάζεται αρχιτεκτονικές αλλαγές. Καθώς η εκτέλεση φορτώματος αυξάνεται σε μια server εφαρμογή, οι πόροι του συστήματος μπορεί να γίνουν σπάνιοι (scarce) και ο χρόνος απόκρισης θα μεγαλώνει. Αντιμέτωποι με αυτό το πρόβλημα οι σχεδιαστές λογισμικού, πλέον χωρίζουν σε επίπεδα (tiers) αλλά και σε διαφορετικά μηχανήματα, την επεξεργασία της επιχειρηματικής λογικής (business-logic) του Server από την επεξεργασία του περιβάλλοντος χρήστη (user interface). Αυτό έχει σαν αποτέλεσμα το ένα επίπεδο της server εφαρμογής να χωρίζεται σε δυο, μειώνοντας έτσι το πρόβλημα της έλλειψης πόρων. Το πρόβλημα δεν είναι ο σχεδιασμός μιας 3-tier εφαρμογής, αλλά ο αυξανόμενος αριθμός επιπέδων(tiers), όταν αρχίσει να λειτουργεί. Αν η αρχική εφαρμογή είναι φτιαγμένη σε ADO.NET χρησιμοποιώντας datasets, η μετατροπή είναι ευκολότερη. Όταν αντικαθιστούμε ένα επίπεδο με δυο, κανονίζουμε να επικοινωνούν/ανταλάσουν πληροφορίες. Επειδή τα επίπεδα μπορούν να μεταδίδουν δεδομένα μέσω datasets που έχουν XML μορφοποίηση, η επικοινωνία είναι ευκολότερη.

Προγραμματιστικότητα (Programmability)

Οι συνιστώσες της ADO.NET (ADO.NET data components) περιλαμβάνουν λειτουργίες πρόσβασης δεδομένων με διάφορους τρόπους που βοηθούν ώστε να γίνεται ο προγραμματισμός ευκολότερα και με λιγότερα λάθη. Οι ADO.NET κλάσεις δεδομένων που δημιουργούνται από τα εργαλεία των typed datasets έχουν σαν αποτέλεσμα να έχουμε πρόσβαση δεδομένων μέσω typed programming. Ο κώδικας για το typed dataset είναι ευκολότερο να διαβαστεί. Είναι επίσης ευκολότερο να γραφτεί επειδή η ολοκλήρωση του ερωτήματος παρέχεται. Συνοψίζοντας, ο κώδικας για τα typed dataset είναι ασφαλέστερος επειδή παρέχει κατά τον χρόνο ελέγχου (compile time) έλεγχο των τύπων.

Επεκτασιμότητα (Scalability)

Επειδή στο διαδίκτυο μπορεί να αυξηθεί κατά πολύ η ζήτηση των δεδομένων μας, η επεκτασιμότητα είναι πολύ σημαντική. Οι διαδικτυακές εφαρμογές έχουν ατελείωτη προμήθεια πιθανών χρηστών. Παρότι μια εφαρμογή μπορεί να εξυπηρετεί δώδεκα χρήστες καλά, μπορεί να μην λειτουργεί καλά αν εξυπηρετεί εκατό ή χίλιους. Μια εφαρμογή που καταναλώνει πηγές όπως κλειδιά βάσεων δεδομένων (database locks) και συνδέσεις βάσεων δεδομένων (database connections) δεν θα εξυπηρετεί καλά έναν μεγάλο αριθμό χρηστών επειδή η ζήτηση των χρηστών για τους περιορισμένους πόρους θα υπερβεί τελικά την προσφορά τους.

Η ADO.NET παρέχει επεκτασιμότητα, ενθαρρύνοντας τους προγραμματιστές για τη διατήρηση των περιορισμένων πόρων. Επίσης, επειδή κάθε ADO.NET εφαρμογή έχει αποσυνδεδεμένη πρόσβαση στα δεδομένα, δεν διατηρεί κλειδιά βάσεων δεδομένων ή ενεργές συνδέσεις βάσης δεδομένων για μεγάλη χρονική διάρκεια (10).

Μειονεκτήματα της ADO.NET

Διαχειριζόμενη μόνο πρόσβαση (Managed-Only Access)

Η χρήση της ADO.NET αρχιτεκτονικής γίνεται μόνο με διαχειριζόμενο κώδικα. Αυτό σημαίνει ότι δεν υπάρχει διαλειτουργικότητα COM στην ADO.NET. Έτσι για να χρησιμοποιηθεί ο SQL Server Data Provider και άλλα χαρακτηριστικά, όπως Datasets , XML, εσωτερική αποθήκευση δεδομένων, και ούτω καθεξής, ο κώδικας πρέπει να εκτελείται σύμφωνα με το CLR(Common Language Runtime).

Μόνο δύο πάροχοι δεδομένων

Δυστυχώς, εάν πρέπει να έχουμε πρόσβαση σε οποιαδήποτε δεδομένα που απαιτούν ένα πρόγραμμα οδήγησης που δεν μπορεί να χρησιμοποιηθεί είτε μέσω ενός παρόχου OLEDB ή τον SQL Server Data Provider, αν χρησιμοποιηθεί κάποιος άλλος παροχος τότε έχουμε πρόβλημα με τις επιδόσεις καθότι επικαλούνται πολλαπλά επίπεδα αφαίρεσης, καθώς και διαλογή COM.

Απότομη καμπύλη μάθησης

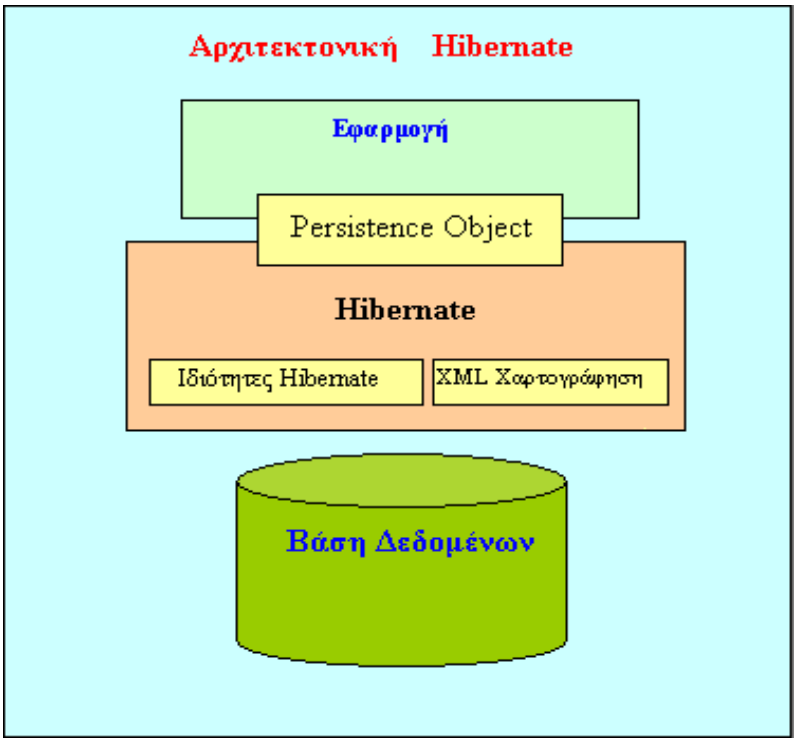
Παρά το παραπλανητικό όνομα, η ADO.NET δεν είναι απλώς μια νέα έκδοση του ADO και δεν πρέπει καν να θεωρηθεί ως άμεσος διάδοχος. Η ADO.NET θα πρέπει να θεωρηθεί περισσότερο ως μια βιβλιοθήκη κλάσεων για πρόσβαση σε δεδομένα με τη χρήση του. NET Framework. Η δυσκολία στην εκμάθηση της χρήσης ADO.NET είναι ότι φαίνεται οικεία και αυτό είναι που προκαλεί ορισμένες παγίδες. Οι προγραμματιστές πρέπει να μάθουν ότι ακόμη και αν ορισμένο συντακτικό μπορεί να εμφανιστεί το ίδιο, υπάρχει πράγματι ένα σημαντικό ποσό διαφοράς στην εσωτερική λειτουργία πολλών κλάσεων (11) (12).

HIBERNATE

Το Hibernate είναι βιβλιοθήκη σχεσιακής χαρτογράφησης αντικειμένου για τη γλώσσα προγραμματισμού Java, που παρέχει ένα πλαίσιο(Framework) για τη χαρτογράφηση ενός αντικειμενοστραφούς μοντέλου τομέα σε μια παραδοσιακή σχεσιακή βάση δεδομένων. Λύνει αντικείμενο-σχεσιακά προβλήματα αναντιστοιχίας αντικαθιστώντας τις persistence προσβάσεις στη βάση δεδομένων με λειτουργίες χειρισμού αντικειμένων υψηλού επιπέδου. Το Hibernate είναι δωρεάν λογισμικό ανοικτού κώδικα που διανέμεται με την αδεία της GNU. Κύριο χαρακτηριστικό του είναι η χαρτογράφηση από κλάσεις Java σε πίνακες της βάσης δεδομένων (και από τύπους δεδομένων Java σε SQL τύπους δεδομένων). Παρέχει επίσης ερωτήματα δεδομένων και εγκαταστάσεις ανάκτησης. Δημιουργεί SQL κλάσεις με σκοπό να διαχειρίζεται αυτόματα το σύνολο των αποτελεσμάτων και τη μετατροπή των αντικειμένων, διατηρώντας την εφαρμογή φορητή για όλες τις υποστηριζόμενες SQL βάσεις δεδομένων, με δυνατότητα μεταφοράς δεδομένων που γίνεται με πολύ μικρή επιβάρυνση των επιδόσεων.

Αρχιτεκτονική Hibernate

Το ακόλουθο διάγραμμα περιγράφει την αρχιτεκτονική υψηλού επιπέδου Hibernate αλλά και πως το Hibernate (αδρανοποίηση) χρησιμοποιεί τη βάση δεδομένων και τις παραμέτρους (configuration data), για την παροχή υπηρεσιών και αντικειμένων στην εφαρμογή.



Εικόνα 5 **Αρχιτεκτονική** Hibernate

Για να χρησιμοποιηθεί το hibernate (αδρανοποίηση), απαιτείται η δημιουργία κλάσεων Java οι οποίες αναπαριστάνουν τον πίνακα μιας βάσης δεδομένων και στη συνέχεια απαιτείται και η αντιστοίχιση (map) των μεταβλητών των κλάσεων με τις στήλες της βάσης δεδομένων. Έπειτα το Hibernate (Αδρανοποίηση) μπορεί να χρησιμοποιηθεί για να προβεί σε ενέργειες στη βάση δεδομένων όπως η επιλογή (select), τοποθέτηση (insert), ενημέρωση (update) και διαγραφή (delete) των εγγραφών του πίνακα. Το Hibernate δημιουργεί αυτόματα το ερώτημα (query) για την εκτέλεση αυτών των εργασιών.

Η αρχιτεκτονική Hibernate έχει τέσσερις βασικές συνιστώσες: τη Διαχείριση Σύνδεσης (Connection Management), τη Διαχείριση Συναλλαγής (Transaction management), τη Σχεσιακή χαρτογράφηση Αντικείμενου (Object relational mapping) και το Μοντέλο Τομέα (DOMAIN MODEL)

Διαχείριση Σύνδεσης (Connection Management)

Η υπηρεσία διαχείρισης σύνδεσης του Hibernate (Hibernate Connection management service) παρέχει αποτελεσματική διαχείριση των συνδέσεων βάσης δεδομένων. Η Σύνδεση στη βάση δεδομένων είναι το πιο δαπανηρό μέρος της αλληλεπίδρασης με τη βάση. Απαιτούνται πολλοί πόροι για να ανοίγει και να κλείνει η σύνδεση με τη βάση δεδομένων.

Διαχείριση Συναλλαγής (Transaction management):

Η υπηρεσία διαχείρισης συναλλαγών παρέχει στο χρήστη τη δυνατότητα να εκτελέσει στη βάση δεδομένων περισσότερες από μία εντολές (statements) τη φορά.

Σχεσιακή χαρτογράφηση Αντικειμένου (Object relational mapping):

Η Σχεσιακή χαρτογράφηση Αντικείμενου είναι η τεχνική της αναπαράστασης των δεδομένων ενός σχεσιακού μοντέλου από ένα μοντέλο αντικειμένων. Αυτό το μέρος του hibernate χρησιμοποιείται για την επιλογή, εισαγωγή, ενημέρωση και διαγραφή των αρχείων που αποτελούν τον υποκείμενο πίνακα.

Το Hibernate είναι πολύ καλό εργαλείο στη σχεσιακή χαρτογράφηση αντικείμενου, αλλά στην διαχείριση σύνδεσης και διαχείριση συναλλαγών, μειονεκτεί σε επιδόσεις και δυνατότητες. Έτσι συνήθως χρησιμοποιείται με άλλα εργαλεία διαχείρισης σύνδεσης και διαχείρισης συναλλαγών.

Το Hibernate παρέχει μεγάλη ευελιξία στη χρήση του. Λέγεται "Lite" αρχιτεκτονική όταν χρησιμοποιείται μόνο η σχεσιακή χαρτογράφηση Αντικείμενου. Ενώ όταν χρησιμοποιούνται και οι τρεις βασικές συνιστώσες του (διαχείριση σύνδεσης, διαχείριση συναλλαγής, και σχεσιακή χαρτογράφηση αντικείμενου) λέγεται "Full Cream" αρχιτεκτονική (13).

Μοντέλο Τομέα (DOMAIN MODEL)

Στη χειρότερη περίπτωσή της, η επιχειρηματική λογική μπορεί να είναι πολύ περίπλοκη. Οι κανόνες και η λογική περιγράφουν πολλές διαφορετικές περιπτώσεις και αλλαγές συμπεριφοράς, και αυτή είναι η πολυπλοκότητα για την οποία τα αντικείμενα έχουν σχεδιαστεί να λειτουργούν. Ένα μοντέλο τομέα δημιουργεί ένα δίκτυο διασυνδεδεμένων αντικειμένων, όπου κάθε αντικείμενο αντιπροσωπεύει κάποιο σημαντικό στοιχείο, είτε αυτό είναι μια επιχείρηση ή μια φόρμα παραγγελίας.

Υπάρχουν δύο μορφές του μοντέλου τομέα.

- Το απλό μοντέλο τομέα, που μοιάζει με το σχέδιο μιας βάσης δεδομένων, έχει ένα αντικείμενο τομέα για κάθε πίνακα της βάσης δεδομένων.
- Το πλούσιο μοντέλο τομέα, που έχει διαφορετική εμφάνιση από το σχέδιο μιας βάσης δεδομένων, περιέχει την κληρονομιά, τις στρατηγικές, καθώς και άλλα πρότυπα, και ένα πολύπλοκο δίκτυο διασυνδεδεμένων αντικειμένων. Το πλούσιο μοντέλο τομέα είναι καλύτερο για πιο πολύπλοκη λογική, αλλά είναι πιο δύσκολο να χαρτογραφηθεί στη βάση δεδομένων

Ένα ΟΟ μοντέλο τομέα συχνά μοιάζει με ένα μοντέλο βάσης δεδομένων, αλλά εξακολουθούν να έχουν πολλές διαφορές. Ένα μοντέλο τομέα ανακατεύει δεδομένα και διαδικασίες, έχει χαρακτηριστικά πολλών τιμών, ένα περίπλοκο δίκτυο συσχετίσεων, και χρησιμοποιεί κληρονομιά (inheritance).

Πώς λειτουργεί

Βάζοντας ένα μοντέλο τομέα σε μια εφαρμογή πρέπει να εισάγουμε και ένα επιπλέων στρώμα αντικειμένων τα οποία μοντελοποιούν την βαθμίδα εφαρμογής στην οποία δουλεύουμε. Σ` αυτό το επιπλέων στρώμα υπάρχουν αντικείμενα που μιμούνται τα δεδομένα της εφαρμογής και αντικείμενα που συλλαμβάνουν τους κανόνες που χρησιμοποιεί. Συνήθως τα δεδομένα και η επεξεργασία συνδυάζονται σε ένα σύμπλεγμα διαδικασιών κοντά στα δεδομένα τα οποία συνεργάζονται. Δεδομένου ότι μια εφαρμογή υπόκειται σε πολλές αλλαγές, είναι σημαντικό να μπορούμε να τροποποιήσουμε, να κατασκευάσουμε, και να δοκιμάσουμε αυτό το στρώμα εύκολα. Γι` αυτό το λόγο το μοντέλο τομέα πρέπει να είναι όσο το δυνατόν πιο ανεξάρτητο απο άλλα μέρη του συστήματος.

Με τη χρήση του μοντέλου τομέα υπάρχει μια σειρά από διαφορετικά πεδία που μπορούν να χρησιμοποιηθούν. Η απλούστερη περίπτωση είναι μια ενιαία εφαρμογή χρήστη όπου όλο το γράφημα αντικείμενου διαβάζεται από ένα αρχείο και τοποθετείται στη μνήμη. Μια desktop εφαρμογή μπορεί να λειτουργήσει με αυτό τον τρόπο, αλλά μια multitiered εφαρμογή που έχει πάρα πολλά αντικείμενα

δεν μπορεί. Βάζοντας κάθε αντικείμενο στη μνήμη καταναλώνει πάρα πολύ μνήμη και παίρνει πάρα πολύ καιρό.

Χωρίς τη χρήση μιας ΟΟ βάσης δεδομένων η μεταφορά των αντικειμένων μεταξύ μνήμης και δίσκου θα πρέπει να γίνει χειροκίνητα. Συνήθως ο σκοπός μιας συνεδρίας είναι να παίρνει ένα γράφημα όλων των αντικειμένων που εμπλέκονται στη βάση. Όμως τις περισσότερες φορές δεν παίρνει όλα τα αντικείμενα και συνήθως ούτε όλες τις κλάσεις και αυτό οφείλεται στον τρόπο με τον οποίο έχουν χαρτογραφηθεί τα αντικείμενα στη βάση δεδομένων.

Ένα συνηθισμένο πρόβλημα στη βαθμίδα εφαρμογής είναι τα " πρησμένα " αντικείμενα τομέα. Αυτό οφείλεται στο ότι δεν είναι πάντα ξεκάθαρο πώς πρέπει να χωριστούν οι ευθύνες/συμπεριφορά των αντικειμένων. Το πρόβλημα με το διαχωρισμό των ευθηνών/συμπεριφοράς αντικειμένων είναι ότι μπορεί να οδηγήσει σε επανάληψη. Η Επανάληψη μπορεί γρήγορα να οδηγήσει σε μεγαλύτερη πολυπλοκότητα και ασυνέπεια, αλλά έχει διαπιστωθεί ότι "πρησμένα" αντικείμενα εμφανίζονται πολύ λιγότερο από ό, τι είχε προβλεφθεί. Αν αυτό συμβεί, είναι σχετικά εύκολο να εντοπιστεί και να διορθωθεί.

Πότε χρησιμοποιείται

Εάν το πώς χρησιμοποιείται το μοντέλο τομέα είναι δύσκολο, το πότε είναι ακόμα πιο δύσκολο, λόγω της ασάφειας και απλότητάς του. Έτσι το πότε χρησιμοποιείται εξαρτάται από την πολυπλοκότητα της εφαρμογής. Αν η εφαρμογή έχει περίπλοκη και συνεχώς εναλλασσόμενη επιχειρηματική βαθμίδα η χρήση του είναι απαραίτητη. Από την άλλη πλευρά αν η εφαρμογή είναι απλή τότε δεν απαιτείται η χρήση του. Ένας ακόμα παράγοντας που περιορίζει την χρήση του μοντέλου τομέα είναι ότι είναι δύσκολο κάποιος να εξοικειωθεί με την χρήση του καθώς είναι δύσκολο και χρειάζεται πάρα πολύ εξάσκηση.

Χαρακτηριστικά

Το Hibernate προσφέρει τα παρακάτω στον προγραμματιστή:

- **Παραγωγικότητα**: Στην ανάπτυξη λογισμικού ένα μεγάλο μέρος της προγραμματιστικής προσπάθειας αφιερώνεται στην διεπαφή της εφαρμογής με τη βάση δεδομένων. Το Hibernate αυτοματοποιώντας τις βασικές λειτουργίες (CRUD – Create Read Update Delete) επιτρέπει αρχικά στον προγραμματιστή να επικεντρώνει την προσπάθειά του στη λογική της εφαρμογής (business logic). Επίσης, υπάρχει η δυνατότητα να ακολουθηθούν δύο στρατηγικές ανάπτυξης λογισμικού: είτε αρχίζοντας από το μοντέλο δεδομένων είτε από τη βάση δεδομένων. Αυτό μειώνει σε μεγάλο βαθμό το χρόνο ανάπτυξης.

- **Συντηρησιμότητα**: Με τη χρήση του Hibernate γράφονται σημαντικά λιγότερες γραμμές κώδικα και ο κώδικας είναι πιο κατανοητός και καλογραμμένος. Αυτό κάνει την συντήρηση της εφαρμογής ευκολότερη.

- **Ανεξαρτησία από τη βάση δεδομένων**: Με τη συμβατότητα του Hibernate με διαφορετικές βάσεις δεδομένων και τη δυνατότητα σύνδεσής του με τη βάση μέσω δηλώσεων οριζομένων σε ειδικό αρχείο η αναπτυσσόμενη εφαρμογή μπορεί με ελάχιστες τροποποιήσεις να χρησιμοποιηθεί με βάσεις δεδομένων διαφορετικών κατασκευαστών. Το γεγονός αυτό στερεί μεν από το Hibernate την εκμετάλλευση των ιδιαίτερων χαρακτηριστικών της χρησιμοποιούμενης βάσης, όμως, και σε αυτή την περίπτωση, δίνεται η δυνατότητα χρήσης πηγαίας SQL μέσα στο Hibernate που εκμεταλλεύεται τα ιδιαίτερα αυτά χαρακτηριστικά (14).

Πλεονεκτήματα HIBERNATE

Αναπαράσταση Συσχετίσεων (Relational Persistence)

Το Hibernate (αδρανοποίηση) είναι ευέλικτη και ισχυρή ORM (object relational mapping) λύση για την χαρτογράφηση Java κλάσεων σε πίνακες της βάσης δεδομένων. Το Hibernate, από μόνο του φροντίζει να αποτυπώνει την αναπαράσταση δεδομένων (data representation) ενός σχεσιακού μοντέλου δεδομένων (relational data model) σε ένα μοντέλο αντικειμένου (object model) αλλά και το αντίστοιχο σχήμα της βάσης δεδομένων (corresponding database schema), χρησιμοποιώντας αρχεία XML, έτσι ο προγραμματιστής δεν χρειάζεται να γράψει κώδικα για αυτό.

Διαφάνεια (Transparent Persistence)

Η αυτόματη χαρτογράφηση των Java αντικειμένων με πίνακες της βάσης δεδομένων και αντίστροφα, ονομάζεται Διαφανής Εμμονή. Το Hibernate παρέχει Εμμονή Διαφάνειας και ο προγραμματιστής δεν χρειάζεται να γράψει κώδικα για να χαρτογραφήσει τους πίνακες της βάσης δεδομένων με αντικείμενα της εφαρμογής κατά τη διάρκεια της αλληλεπίδρασης με RDBMS (Relational database management system).

Υποστήριξη για Query Language

Το Hibernate παρέχει μια ισχυρή γλώσσα ερωτημάτων, την Hibernate Query Language (η οποία είναι ανεξάρτητη από τον τύπο της βάσης δεδομένων) η σύνταξη της οποίας μοιάζει με την σύνταξη της SQL και περιλαμβάνει πλήρη υποστήριξη για πολυμορφικά ερωτήματα (polymorphic queries). Το Hibernate υποστηρίζει επίσης και την χρήση “καθαρών” SQL δηλώσεων (statements) .

Εξαρτώμενος κώδικας βάσης δεδομένων (Database Dependent Code)

Στο Hibernate η αντιστοίχιση μεταξύ των πινάκων και των αντικειμένων μιας εφαρμογής γίνεται με XML αρχεία. Αν υπάρχει αλλαγή στη βάση δεδομένων ή σε κάποιο πίνακα τότε το μόνο που χρειάζεται είναι να αλλάξει το XML αρχείο με τις ιδιότητες και όχι ο κώδικας με τον οποίο έγινε η χαρτογράφηση.

Κόστος συντήρησης(Maintenance Cost)

Το Hibernate μειώνει τις γραμμές κώδικα με το να κάνει την χαρτογράφηση object-table αυτόματα και επιστρέφει τα αποτέλεσμα στην εφαρμογή σε μορφή Java αντικειμένων. Ανακουφίζει τον προγραμματιστή από το να διαχειρίζεται ο ίδιος τα ανθεκτικά δεδομένα(persistent data), συνεπώς, μειώνεται ο χρόνος ανάπτυξης και το κόστος συντήρησης.

Βελτιστοποίηση της απόδοσης (Optimize Performance)

Η αποθήκευση των δεδομένων στην Κρυφή Μνήμη (caching) είναι η κατακράτηση των δεδομένων, συνήθως σε εφαρμογές για τον περιορισμό της

πρόσβασης στο δίσκο. Η κρυφή μνήμη του Hibernate τίθεται σε εφαρμογή στο χώρο εργασίας. Σχεσιακές δυάδες τοποθετούνται σ` αύτη τη κρυφή μνήμη μέσω ενός ερωτήματος. Αυτό βελτιώνει τις επιδόσεις, εάν η client εφαρμογή διαβάζει ίδια δεδομένα πολλές φορές .Η Αυτόματη Διαφανής Εμμονή επιτρέπει στον προγραμματιστή να επικεντρωθεί περισσότερο την επιχειρηματική λογική και όχι αυτόν στον κώδικα της εφαρμογή.

Αυτοματοποίηση εκδόσεων και ώρα Σφράγισης (Automatic Versioning and Time Stamping)

Με βάση την έκδοση από τη βάση δεδομένων μπορεί κανείς να είναι βέβαιος ότι οι αλλαγές που έγιναν από ένα άτομο δεν αλλάζουν κατά λάθος από κάποιον άλλο. Το Hibernate δίνει τη δυνατότητα στον προγραμματιστή να ορίσει ένα πεδίο τύπου έκδοση (version) για την εφαρμογή, έτσι το πεδίο αυτό ενημερώνεται από το Hibernate κάθε φορά που αλλάζει κάτι στην βάση δεδομένων. Αυτό σημαίνει ότι αν δυο προγραμματιστές ξεκινήσουν να τροποποιούν τη βάση δεδομένων όταν ο πρώτος αποθηκεύσει τις αλλαγές ο δεύτερος δεν θα μπορεί γιατί δεν θα έχει ενημερωμένα (updated) δεδομένα.

Open-Source, Η άδεια χρήσης προϊόντος έχει μηδενικό κόστος

Το Hibernate είναι open source και δωρεάν για χρήση τόσο για την ανάπτυξη όσο και για την παραγωγή εφαρμογών.

Επιχείρηση-κλάση Αξιοπιστία και επεκτασιμότητα (Enterprise-Class Reliability and Scalability)

Το Hibernate επεκτείνεται καλά σε οποιοδήποτε περιβάλλον, δεν έχει σημασία αν χρησιμοποιείται σε εσωτερικό Intranet που εξυπηρετεί εκατοντάδες χρήστες ή σε κρίσιμες εφαρμογές που εξυπηρετούν εκατοντάδες χιλιάδες χρηστών

Μειονεκτήματα HIBERNATE

- **Απότομη καμπύλη μάθησης**
  Για να εξοικειωθεί κάποιος με το hibernate χρειάζεται επιμονή, βοήθεια και εξάσκηση

- **Η χρήση του Hibernate είναι γενικά για τις εφαρμογές οι οποίες:**
  - Είναι απλές και χρησιμοποιούν μία βάση δεδομένων που δεν αλλάζει ποτέ
  - Τοποθετούν δεδομένα σε πίνακες μιας βάσης δεδομένων και δεν έχουν αντικείμενα που αντιστοιχίζονται σε δύο διαφορετικούς πίνακες
- **Το Hibernate αυξάνει τα επιπλέον τα επίπεδα και την πολυπλοκότητα των εφαρμογών**

- **Ο καθένας που θέλει να διατηρεί μια εφαρμογή που χρησιμοποιεί Hibernate θα πρέπει να είναι εξοικειωμένος με αυτό.**

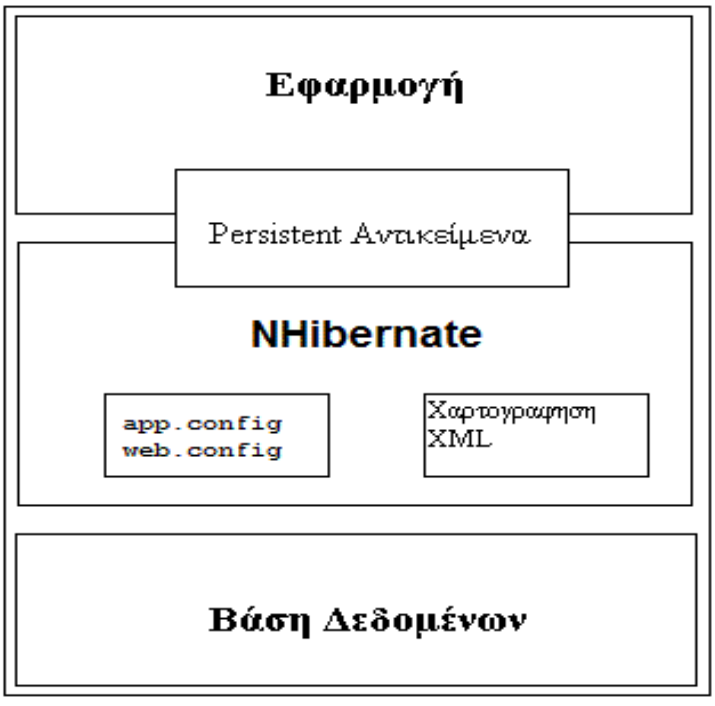
- **Για πολύπλοκα δεδομένα, η χαρτογράφηση από αντικείμενα σε πίνακες (Object-to-tables) και αντίστροφα μειώνει τις επιδόσεις και αυξάνει το χρόνο της μετατροπής** (15) (16).

NHIBERNATE

Το NHibernate είναι ένα κομμάτι του πυρήνα του Hibernate το οποίο χρησιμοποιείται από τη Java για το .NET Framework. Διαχειρίζεται απλά persisting .NET αντικείμενα από και προς μια υποκείμενη σχεσιακή βάση δεδομένων. Δεδομένης της XML περιγραφής των οντοτήτων και των σχέσεων, το NHibernate δημιουργεί αυτόματα SQL για τη φόρτωση και την αποθήκευση των αντικειμένων. Προαιρετικά, τα χαρτογραφημένα μεταδεδομένα μπορούν να περιγραφτούν από κάποια από τα χαρακτηριστικά του πηγαίου κώδικα (17).

Αρχιτεκτονική NHibernate

Το NHibernate υποστηρίζει εμμονή διαφάνειας, οι κλάσεις των αντικειμένων δεν χρειάζεται να ακολουθήσουν ένα αυστηρά περιορισμένο μοντέλο προγραμματισμού. Οι persistent κλάσεις δεν χρειάζεται να υλοποιήσουν κάποια διασύνδεση ή να κληρονομήσουν από κάποια ειδική κλάση. Αυτό δίνει τη δυνατότητα να σχεδιαστεί η επιχειρηματική λογική χρησιμοποιώντας απλά .NET (CLR) αντικείμενα και object-oriented ιδίωμα. Λόγω του ότι το NHibernate είναι κομμάτι του Hibernate όλες οι γνώσεις και η υπάρχουσα τεκμηρίωση για το Hibernate ισχύουν άμεσα και στο Nhibernate



Εικόνα 6 **Αρχιτεκτονική N**Hibernate

Το παραπάνω διάγραμμα δείχνει πως το NHibernate χρησιμοποιεί τη βάση δεδομένων και τις παραμέτρους, για να παρέχει υπηρεσίες (και αντικείμενα) για την εφαρμογή (17) (18).

Χαρακτηριστικά του NHibernate:

Το πρώτο μέρος είναι η POCO[1], η οποία αλληλεπιδρά με άλλους .NET κώδικες (19).

Το δεύτερο μέρος είναι ένα αρχείο χαρτογράφησης, το οποίο περιγράφει πώς η POCO αντιστοιχίζεται στο σχήμα μιας σχεσιακής βάσης δεδομένων.

Το τρίτο μέρος είναι η ίδια η διαμόρφωση του NHibernate, η οποία περιέχει το διακομιστή της βάσης δεδομένων και τη συμβολοσειρά σύνδεσης.

Το τέταρτο μέρος είναι η χρήση του μοντέλου τομέα στην αναπαράσταση της βαθμίδας εφαρμογής (20).

[11] Η **POCO** είναι ένα αρκτικόλεξο για **Plain Old CLR Object** και χρησιμοποιείται από προγραμματιστές με στόχο την Common Language Runtime (**CLR**) του .NET Framework. Εκτός από την Java ο όρος αυτός χρησιμοποιείται για να δείξει την αντίθεση ενός αντικειμένου σε σύγκριση με ένα άλλο το οποίο είναι σχεδιασμένο για να χρησιμοποιείται με πολύπλοκα, ειδικά frameworks αντικειμένου όπως για παράδειγμα το ORM. Σε .NET όρους, η λέξη αυτή χρησιμοποιείται πιο συχνά με την προγραμματική της έννοια. Αυτό σημαίνει ότι χρησιμοποιείται για να δείξει την διαφορά μεταξύ μιας non serviced συνιστώσας και ενός standard αντικειμένου.

Πλεονεκτήματα Nhibernate

- **Το NHibernate είναι ένα ευέλικτο και προσαρμόσιμο πλαίσιο**.

- **Το NHibernate παρέχει όλα τα χαρακτηριστικά που απαιτούνται για την γρήγορη κατασκευή ενός προηγμένου persistence layer.**

- **Διαχείριση διαγραμμάτων**

Το NHibernate είναι ικανό να φορτώσει και να αποθηκεύσει ολόκληρα τα διαγράμματα των διασυνδεδεμένων αντικειμένων, διατηρώντας παράλληλα τις σχέσεις μεταξύ τους.

- **Αυτοματοποίηση**

Αν του δοθούν οι πληροφορίες χαρτογράφησης αναφέροντας του με ποιον τρόπο θα πρέπει να φορτώνονται και να σώζονται οι οντότητες, το NHibernate μπορεί να τις μετακινεί από και προς μια σχεσιακή βάση δεδομένων αυτόματα. Επίσης μπορεί και να εκτελεί αποτελεσματικά λειτουργίες CRUD (Create, Read, Update, Delete) με την κλήση μόνο μίας μεθόδου.

- **Παρακολούθηση και αυτοματοποίηση του μοτίβου εργασίας**

Σε μια περίπλοκη συναλλαγή που περιλαμβάνει πολλές ενημερώσεις και διαγραφές, εάν πρέπει να παρακολουθηθούν χειροκίνητα οι οντότητες που θα αποθηκευτούν ή θα διαγράφουν και φροντίζοντας ταυτόχρονα να φορτωθεί κάθε οντότητα μόνο μία φορά, σύντομα η κατάσταση θα γίνει πολύ περίπλοκη. Το NHibernatee ακολουθεί το μοτίβο της μονάδας εργασίας για την επίλυση αυτού του προβλήματος και έτσι διευκολύνει την εφαρμογή των συναλλαγών. Αν οι οντότητες συσχετιστούν με το NHibernatee τότε, αυτό παρακολουθεί τη φόρτωση και την αποθήκευση των αλλαγών μόνο όταν αυτό απαιτείται. Στο τέλος της συναλλαγής, το NHibernatee αυτόματα εφαρμόζει όλες τις αλλαγές με τη σωστή σειρά και επειδή παρακολουθεί όλες τις οντότητες, μπορεί να απλοποιήσει σημαντικά την εφαρμογή και να αυξήσει την απόδοση της.

- **Lazy loading**

Η τροποποίηση ενός χρήστη μπορεί να γίνει με δυο τρόπους, είτε φορτώνοντας όλη τη συλλογή των αντικειμένων του, το οποίο είναι αναποτελεσματικό είτε αφήνοντας τη συλλογή μη-αρχικοποιημένη με αποτέλεσμα να περιορίζεται η δυνατότητα τροποποίησης του χρήστη. Το NHibernate υποστηρίζει μια δυνατότητα που ονομάζεται *lazy loading* για την επίλυση αυτού του προβλήματος. Κατά τη φόρτωση του χρήστη, μπορούμε να αποφασίσουμε αν θα γίνει η φόρτωση των αντικειμένων ή όχι. Εάν επιλέξουμε να μην φορτωθεί, η συλλογή αρχικοποιείται διαφανώς όταν χρειαστεί.

- **Χρήση Cache**

Ο χάρτης ταυτοτήτων του NHibernate χρησιμοποιεί ένα χώρο προσωρινής αποθήκευσης (cache)για να αποφευχθεί η φόρτωση μιας οντότητας πολλές φορές. Αυτός ο χώρος μπορεί να χρησιμοποιηθεί από συναλλαγές και εφαρμογές κάνοντας τες πιο αποτελεσματικές.

- **Χαρτογράφηση οντοτήτων**

Με βάση το γεγονός ότι το NHibernate μπορεί να φορτώσει και να αποθηκεύσει οντότητες, μπορούμε να συμπεράνουμε ότι γνωρίζει πώς, κάθε οντότητα χαρτογραφείται στη βάση δεδομένων. Όταν ζητήσουμε μια οντότητα με το αναγνωριστικό της, το NHibernate ξέρει πώς να την βρεί. Άρα Θα πρέπει να μπορούμε να εκφράσουμε ένα ερώτημα χρησιμοποιώντας το όνομα της οντότητας και τις ιδιότητες της, και στη συνέχεια το NHibernate μπορεί να το μετατρέψει σε αντίστοιχη ερώτηση SQL κατανοητή από τη σχεσιακή βάση δεδομένων.

- **Παραγωγικότητα**

Ένα εργαλείο σαν το NHibernate κάνει τον προγραμματιστή πιο παραγωγικό. Εξαλείφει μεγάλο μέρος της χρονοβόρας εργασίας και επιτρέπει στον προγραμματιστή να επικεντρώνεται στα προβλήματα της βαθμίδας εφαρμογής. Το NHibernate αν χρησιμοποιείται σε συνδυασμό με τα κατάλληλα εργαλεία μειώνει σημαντικά το χρόνο ανάπτυξης της εφαρμογής.

- **Συντηρησιμότητα**

Το NHibernate βελτιώνει ουσιαστικά τη δυνατότητα συντήρησης, όχι μόνο επειδή μειώνει τον αριθμό των γραμμών του κώδικα, αλλά και επειδή παρέχει μια ζώνη(buffer) μεταξύ του μοντέλου αντικειμένων και της σχεσιακής αναπαράστασης

- **Φορητότητα**

Το NHibernate δημιουργεί ένα επίπεδο αφαίρεσης μεταξύ της εφαρμογής με τη βάση δεδομένων και την SQL διάλεκτο. Έτσι το γεγονός ότι υποστηρίζει μια σειρά από διαφορετικές βάσεις δεδομένων παρέχει ένα επίπεδο φορητότητας στην εφαρμογή.

Μειονεκτήματα Nhibernate

- Αυξημένος χρόνος εκκίνησης λόγω της προετοιμασίας των μεταδεδομένων.
- Τεράστια καμπύλη μάθησης αν κάποιος δεν έχει ξανασχοληθεί με ORM.
- Συγκριτικά δυσκολότερος ο συγχρονισμός της παραγόμενης SQL.
- Δύσκολο να βρεθεί δικαίωμα χρήσης συνεδρίας εάν χρησιμοποιείται σε μη-τυπικά περιβάλλοντα.
- Δεν είναι κατάλληλο για εφαρμογές που δεν έχουν καθαρό τομέα του μοντέλου αντικειμένου (domain object model) (δεν έχουν όλες οι εφαρμογές την ανάγκη για έναν καθαρό τομέα του μοντέλου αντικειμένου).
- Αντιμετωπίζει πολλές δυσκολίες, αν το σχήμα της βάσης δεδομένων δεν είναι σχεδιασμένο σωστά.
- Το να γίνει XML χαρτογράφηση μπορεί να είναι χρονοβόρο σε μεγάλες βάσεις δεδομένων με εκατοντάδες και χιλιάδες πίνακες, όψεις (views) και αποθηκευμένες διαδικασίες. Υπάρχουν εργαλεία που βοηθούν στο να γίνει η χαρτογράφηση αλλά ακόμα και με αυτά απαιτείται πολλή χειρωνακτική εργασία (21).

Σύγκριση αρχιτεκτονικών N-tier εφαρμογών

Η ADO.NET η Hibernate και η NHibernate χρησιμοποιούνται στην ανάπτυξη N-tier εφαρμογών. Γι` αυτό το λόγο αναγκαστικά, έχουν αρκετά κοινά σημεία. Για παράδειγμα και οι τρεις τεχνολογίες είναι ευέλικτες και εύκολα προσαρμόσιμες στις απαιτήσεις της εφαρμογής, και οι τρεις μπορούν να χρησιμοποιήσουν SQL[2] , χρησιμοποιούν XML είτε για χαρτογράφηση είτε για επικοινωνία και μεταφορά δεδομένων και η εξοικείωση με οποιαδήποτε από τις τρείς είναι αρκετά χρονοβόρα. Παρά τους κοινούς στόχους και τις ομοιότητες τους έχουν και κάποιες διαφορές. Οι περισσότερες από αυτές δεν είναι ικανές ώστε να δημιουργήσουν ανταγωνιστικό πλεονέκτημα σε κάποια από αυτές[3].

Το στοιχείο που επιδρά περισσότερο στην απόδοση, την ταχύτητα και αξιοπιστία τους είναι ο τρόπος με τον οποίο οι τεχνολογίες αυτές αναπαριστούν την βαθμίδα εφαρμογής μιας N-tier εφαρμογής. Η χρήση του Table module από την ADO.NET και η χρήση του Domain model από τις Hibernate και NHibernate αποτελούν την κύρια διαφοροποίηση αυτών των τεχνολογιών. Συνεπώς η σύγκρισή μας θα εστιαστεί στις αντιθέσεις των Domain model και Table module.

Σύγκριση ενότητας πίνακα και μοντέλου τομέα

Για την οργάνωση της λογικής στην βαθμίδα εφαρμογής έχουμε δύο μοτίβα: το μοντέλο τομέα και την ενότητα πίνακα. Πολύ επιφανειακά μπορεί να πει κάποιος ότι η ενότητα πίνακα και ο τομέας μοντέλου μοιάζουν μεταξύ τους επειδή χρησιμοποιούν και οι δυο κλάσεις και αναφορές προόδου. Η σημαντική διαφορά τους είναι ότι το μοντέλο τομέα έχει για κάθε ένα πίνακα της βάσης δεδομένων και ένα υπόδειγμά του, ενώ η ενότητα πίνακα έχει ένα υπόδειγμα για όλους τους πίνακες. Αυτό οφείλεται στο ότι η ενότητα πίνακα έχει φτιαχτεί να δουλεύει με Record Set. Έτσι η επικοινωνία μεταξύ βάσης και της ενότητας πίνακα είναι πιο εύκολη και πιο γρήγορη.

Η οργάνωση της βαθμίδας εφαρμογής βάση πινάκων και όχι βάση διαδικασιών παρέχει καλύτερη διάρθρωση και είναι πιο εύκολο να βρεθούν και να απομακρυνθούν προβλήματα που δημιουργούνται, όπως για παράδειγμα η επικάλυψη. Όμως με αυτό το τρόπο δεν μπορούμε να χρησιμοποιήσουμε αρκετές τεχνικές που παρέχει το μοντέλο τομέα, όπως κληρονομία, στρατηγικές.

[2] Το Hibernate μπορεί να χρησιμοποιήσει SQL αλλά αποδίδει καλύτερα με την χρήση της HQL

[3] Μια σημαντική διαφορά μεταξύ της ADO.NET και των Hibernate/NHibernate είναι ότι η ADO.NET είναι εμπορικό προϊόν ενώ οι άλλες δυο είναι προγράμματα ανοιχτού κώδικα.

Πώς διαλέγουμε μεταξύ των δύο εξαρτάται από το πόσο πολύπλοκη είναι η λογική στη βαθμίδα εφαρμογής.

Όσο η λογική της βαθμίδας εφαρμογής είναι απλή, το μοντέλο τομέα δεν είναι καθόλου καλό, γιατί απαιτεί παρά πολύ χρόνο ώστε να γίνει κατανοητό και πολύ χρόνο για την ανάπτυξή του.

Όσο πιο πολύπλοκη είναι η λογική που χρησιμοποιούμε, το μοντέλο τομέα φαίνεται να είναι πιο σωστή επιλογή. Όμως έχει δύο πολύ σημαντικά προβλήματα. Πρώτον, έχει απότομη καμπύλη μάθησης και δεύτερον αντιμετωπίζει προβλήματα στη σύνδεσή του με σχεσιακές βάσεις δεδομένων (αντικειμενοστραφή μοντέλα και σχεσιακά μοντέλα δεν μπορούν να συνεργαστούν σωστά). Η ενότητα πίνακα αν και δεν δουλεύει εξίσου καλά με το μοντέλο τομέα σε πολύπλοκη λογική συνεργάζεται άψογα με σχεσιακές βάσεις δεδομένων.

Ανεξάρτητα από την πολυπλοκότητα της λογικής η χρήση της ενότητας τομέα εξαρτάται από το αν υποστηρίζεται η χρήση RecordSet από το περιβάλλον προγραμματισμού. Η ενότητα πίνακα είναι πιο ελκυστική αν δουλεύουμε σε .NET ή Visual studio τα οποία περιέχουν αρκετά εργαλεία τα οποία δουλεύουν με RecordSet.

Μεγάλο πλεονέκτημα της ενότητας πίνακα είναι και το πώς ταιριάζει με την υπόλοιπη αρχιτεκτονική. Πολλές βαθμίδες παρουσίασης φτιάχνονται για να δουλεύουν τα αποτελέσματα από SQL ερωτήματα τα οποία είναι οργανωμένα βάση ενός recordset. Έτσι αφού η ενότητα πίνακα δουλεύει με Record Set είναι πολύ εύκολο να κάνουμε ερωτήματα στη βάση και να επεξεργαστούμε δεδομένα. Αρκετές πλατφόρμες και συγκεκριμένα η COM και η .NET της Microsoft χρησιμοποιούν αυτόν το τρόπο ανάπτυξης εφαρμογών.

Η ενότητα πίνακα δεν μπορεί να χρησιμοποιήσει πλήρως τη δύναμη των αντικειμένων στην οργάνωση πολύπλοκων λογικών. Δεν έχει άμεσες υπόδειγμα-σε-υπόδειγμα σχέσεις, και ο πολυμορφισμός δεν λειτουργεί καλά. Γι` αυτό το λόγο αν έχουμε πολύπλοκη λογική το μοντέλο τομέα είναι καλύτερη επιλογή. Παρόλο που τα μειονεκτήματα της ενότητας πίνακα μας δείχνουν ότι το μοντέλο τομέα υπερέχει, καταλήγουμε στο συμπέρασμα ότι τα πλεονεκτήματά του δεν είναι αρκετά σημαντικά για να επικαλύψουν τα πλεονεκτήματα της ενότητας πίνακα. Αυτό κυρίως οφείλεται στο γεγονός ότι αν και μερικές φορές είμαστε σε θέση να επιλέξουμε τα εργαλεία με βάση την αρχιτεκτονική μας, και θεωρητικά αυτός είναι ο σωστός τρόπος. Στην πράξη, ωστόσο, ταιριάζουμε την αρχιτεκτονική μας με τα εργαλεία που έχουμε. Από τα δυο πρότυπα η ενότητα πίνακα έχει ισχυρότερα πλεονεκτήματα όταν έχει τα εργαλεία που χρειάζεται. Είναι μια πολύ ισχυρή επιλογή στα .NET περιβάλλοντα προγραμματισμού μιας και βασίζεται στα RecordSet (23).

Κεφάλαιο 3

Υλοποίηση πρακτικών πειραμάτων

Στις προηγούμενες ενότητες πήραμε μια γεύση για την λειτουργία καθώς και τα πλεονεκτήματα, μειονεκτήματα της ADO.NET και του Nhibernate.Σ` αυτή την ενότητα θα προσπαθήσουμε να δούμε στην πράξη τι από αυτά ισχύει και πόσο δύσκολο ή εύκολο είναι για έναν προγραμματιστή με μεταπηδήσει από την μια τεχνολογία στην άλλη.

Αυτό θα προσπαθήσουμε να το πετύχουμε με την υλοποίηση τριών πειραμάτων (εφαρμογών) που στην ουσία θα κάνουν το ίδιο πράγμα αλλά κάθε ένα από αυτά θα χρησιμοποιήσει διαφορετικά στοιχεία από την θεωρεία.

Εργαλεία πειραμάτων

Για την υλοποίηση των πειραμάτων μας θα χρησιμοποιήσουμε Microsoft SQL Server 2005 ο οποίος θα μας βοηθήσει στο να έχουμε κάπου την βάση δεδομένων, το Microsoft Visual Studio 2005 το οποίο θα είναι το εργαλείο στο οποίο θα γίνει η υλοποίηση των πειραμάτων, το πρόγραμμα MyGeneration το οποίο θα το χρησιμοποιήσουμε στο τρίτο πείραμα για την χαρτογράφηση της βάσης δεδομένων την ιστοσελίδα http://converter.telerik.com/ η οποία είναι ένα πρόγραμμα που μετατρέπει την γλώσσα C# σε VB.Net και Visual Basic.net που είναι η γλώσσα προγραμματισμού με την οποία θα υλοποιήσουμε όλα τα πειράματα.

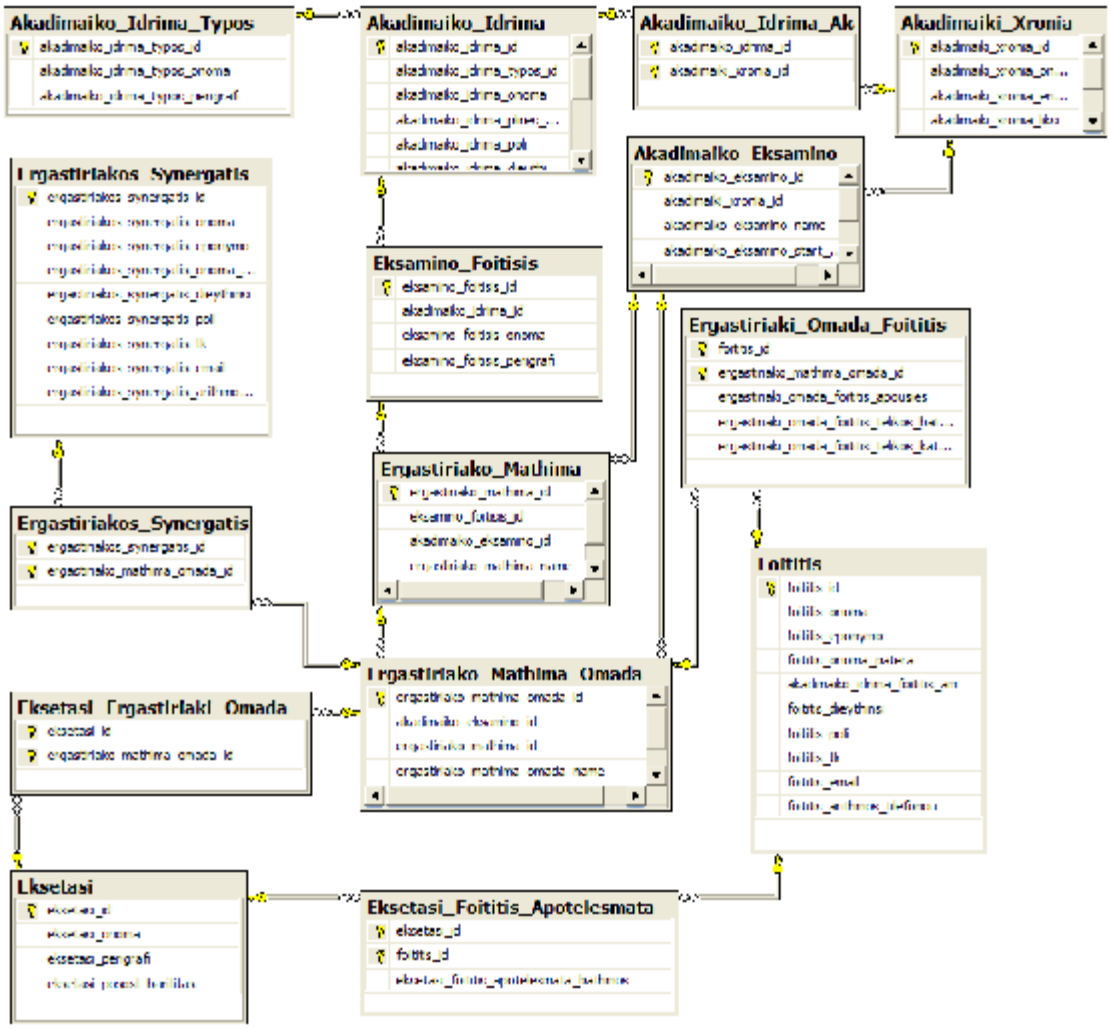
Για να μπορέσουν τα πειράματά μας να δουλέψουν πρέπει να χρησιμοποιηθεί μια βάση δεδομένων. Λόγω του ότι ο σκοπός μας είναι η σύγκριση της ADO.NET και του Nhibernate δεν ασχοληθήκαμε με την δημιουργία της βάσης δεδομένων την οποία θα χρησιμοποιήσουμε, αλλά επιλέξαμε να χρησιμοποιήσουμε μια ήδη υπάρχουσα βάση.

Το όνομα της είναι ateidb περιέχει τους εξής πίνακες:

- Akadimaiki_Xronia
- Akadimaiko_Eksamino
- Akadimaiko_Idrima
- Akadimaiko_Idrima_Akadimaiki_Xronia
- Akadimaiko_Idrima_Typos
- Eksamino_Foitisis
- Eksetasi
- Eksetasi_Ergastiriaki_Omada
- Eksetasi_Foititis_Apotelesmata
- Ergastiriaki_Omada_Foititis
- Ergastiriako_Mathima

- Ergastiriako_Mathima_Omada
- Ergastiriakos _Synergatis
- Ergastiriakos_Synergatis_Ergastiriaki_Omada
- Foititis

καθώς και τις μεταξύ τους σχέσεις.



Εικόνα 7 **Διάγραμμα βάσης δεδομένων** ateidb

Από την συγκεκριμένη βάση χρησιμοποιούμε τους παρακάτω πίνακες:

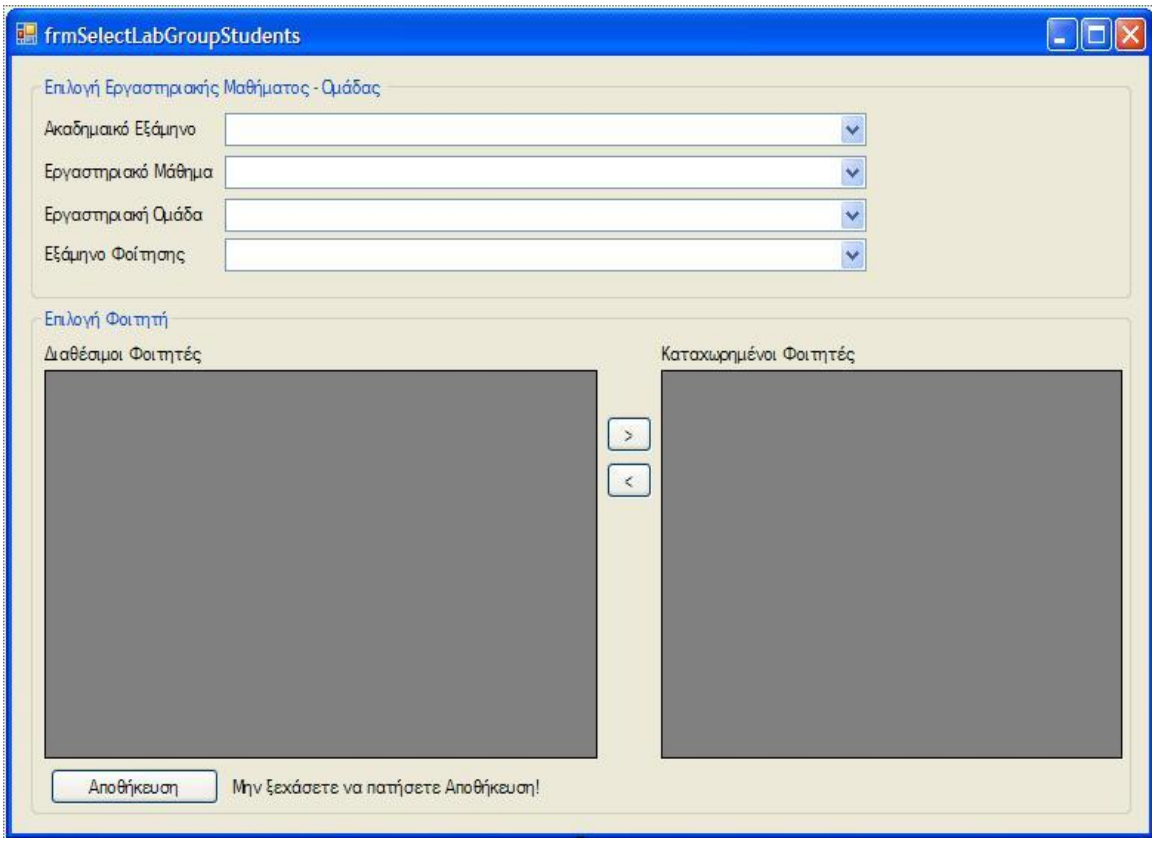
- Akadimaiko_Eksamino
- Eksamino_Foitisis
- Ergastiriaki_Omada_Foititis
- Ergastiriako_Mathima
- Ergastiriako_Mathima_Omada
- Foititis

με τους υπόλοιπους πίνακες δεν ασχολούμαστε καθώς δεν μας επηρεάζουν στην υλοποίηση των πειραμάτων.

Τρόπος λειτουργίας των πειραμάτων

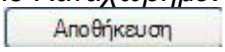
Όπως αναφέραμε και προηγουμένως θα υλοποιήσουμε τρία πειράματα, με το όνομα LabManagementSystem, τα οποία θέλουμε και να συγκρίνουμε, γι` αυτόν λοιπόν το λόγο και τα τρία κάνουν ακριβώς τα ίδια πράγματα. Λόγω του ότι η βάση δεδομένων μας απεικονίζει στην ουσία τις οντότητες μιας σχολής αλλά και τις μεταξύ τους σχέσεις, τα πειράματα αποτελούν ένα εργαλείο με το οποίο θα μπορεί κάποιος να συνδεθεί στη βάση δεδομένων και θα έχει την δυνατότητα να γράφει ή να διαγράφει φοιτητές στις εργαστηριακές ομάδες του μαθήματος που επιλέγει.

Για να γίνει η χρήση του κάθε πειράματος πιο απλή και να είναι προσιτή σε κάθε χρήστη, δημιουργήσαμε μια φόρμα πάνω στην οποία θα γίνονται οι τροποποιήσεις όλων των εργαστηριακών ομάδων.



Εικόνα 8 Η κοινή φόρμα των πειραμάτων

Η φόρμα μας αποτελείται από τα παρακάτω πεδία και κουμπιά:

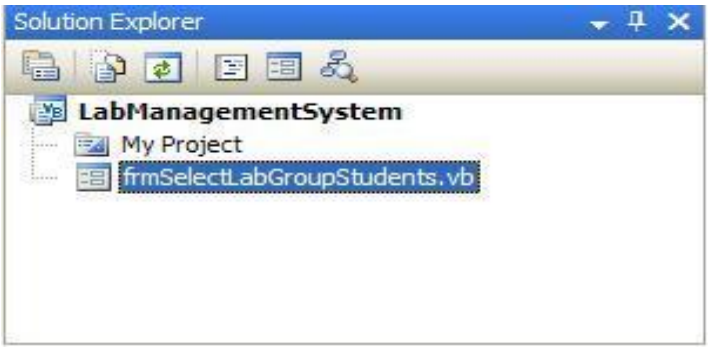
- *Ακαδημαϊκό εξάμηνο:* το οποίο είναι το τρέχων εξάμηνο π.χ. χειμερινό 2010-2011
- *Εργαστηριακό Μάθημα:* το οποίο περιέχει τα μαθήματα που έχουν εργαστήρια π.χ. Προγραμματισμός Η/Υ
- *Εργαστηριακή ομάδα* εδώ εμφανίζονται οι ομάδες του κάθε εργαστηρίου π.χ. Ομάδα Α
- *Εξάμηνο φοίτησης:* όπου εμφανίζεται το εξάμηνο του οδηγού σπουδών στο οποίο ανήκει το Εργαστηριακό μάθημα π.χ. Β
- *Διαθέσιμοι φοιτητές:* εδώ εμφανίζονται οι φοιτητές του εργαστηριακού μαθήματος που έχουμε επιλέξει στο προηγούμενο πεδίο οι οποίοι δεν έχουν ενταχθεί σε κάποια ομάδα του
- *Καταχωρημένοι φοιτητές:* εδώ εμφανίζονται οι φοιτητές που έχουν ενταχθεί στην ομάδα που έχουμε επιλέξει
- Το κουμπί  : το οποίο μεταφέρει κάποιον φοιτητή που έχουμε επιλέξει, από το πεδίο *Διαθέσιμοι φοιτητές στο* πεδίο *Καταχωρημένοι φοιτητές*
- Το κουμπί  : το οποίο μεταφέρει κάποιον φοιτητή που έχουμε επιλέξει, από το πεδίο *Καταχωρημένοι φοιτητές στο* πεδίο *Διαθέσιμοι φοιτητές*
- Το κουμπί  : το οποίο αποθηκεύει στην βάση δεδομένων τις αλλαγές που έχουμε κάνει στη φόρμα

Αν για παράδειγμα θελήσουμε να γράψουμε κάποιον φοιτητή σε κάποια ομάδα το μόνο που πρέπει να κάνουμε είναι να τον επιλέξουμε από το πεδίο *Διαθέσιμοι φοιτητές* και να πατήσουμε το κουμπί με το βέλος που έχει φορά προς τα δεξιά, για να διαγράψουμε κάποιον τον επιλέγουμε από το πεδίο *Καταχωρημένοι φοιτητές* και πατάμε το κουμπί με το βέλος που έχει φορά προς τα αριστερά. Τέλος όταν ήμαστε σίγουροι για τις αλλαγές που έχουμε κάνει στην ομάδα που έχουμε επιλέξει πατάμε το κουμπί *Αποθήκευση* ώστε η εφαρμογή μας να ενημερώσει την βάση δεδομένων και να αποθηκευτούν οι αλλαγές που έχουμε κάνει.

Υλοποίηση πρώτου πειράματος

Με βάση την θεωρεία η εφαρμογή μας θα πρέπει να έχει στοιχεία πολυεπίπεδης αρχιτεκτονικής και να χρησιμοποιεί ADO.NET.

Στον πολυεπίπεδο προγραμματισμό μια εφαρμογή πρέπει να χωρίζεται σε βαθμίδες όπως: Βαθμίδα παρουσίασης, Βαθμίδα εφαρμογής και Βαθμίδα δεδομένων. Στην εφαρμογή μας η Βαθμίδα παρουσίασης και Βαθμίδα εφαρμογής συνενώνονται σε μια βαθμίδα η οποία ονομάζεται *frmSelectLabGroupStudents* στην οποία περιέχονται το λειτουργικό περιβάλλον που διαχειρίζεται ο χρήστης αλλά και οι διεργασίες της εφαρμογής. Τη βαθμίδα δεδομένων αποτελεί στην ουσία η βάση δεδομένων ateidb.



Εικόνα 9 Οι συνενωμένες Βαθμίδες παρουσίασης και εφαρμογής

Η αρχιτεκτονική ADO.NET αποτελείται από:

- **Πάροχο δεδομένων(Data provider)** ο οποίος έχει κλάσεις που παρέχουν πρόσβαση σε μία πηγή δεδομένων. Στην συγκεκριμένη εφαρμογή χρησιμοποιούμε τις connection, command και DataAdapter. Λόγω του ότι χρησιμοποιούμε MSSQL Server μετατρέπονται σε sqlConnection sqlCommand, sqlDataAdapter.

```
Public Shared Function GetDataValue(ByVal sqlCommand As String) As String
    Dim connectionString As String = GetConnectionString()
    Dim ateidbConnection As SqlClient.SqlConnection = New SqlClient.SqlConnection(connectionString)
    Dim command As New SqlClient.SqlCommand(sqlCommand, ateidbConnection)

    Dim adapter As SqlClient.SqlDataAdapter = New SqlClient.SqlDataAdapter()
    adapter.SelectCommand = command
    Dim table As New DataTable
    Dim ret As String

    table.Locale = System.Globalization.CultureInfo.InvariantCulture
    'MessageBox.Show(sqlCommand)
    adapter.Fill(table)

    If Not table.Rows(0).IsNull(0) Then
        ret = table.Rows(0).Item(0).ToString
    Else
        ret = ""
    End If

    Return ret
End Function
```

Εικόνα 10 **Λειτουργία που χρησιμοποιεί τις** sqlConnection , sqlCommand **και** sqlDataAdapter

- **Σύνολα δεδομένων(Datasets)** είναι μια ομάδα από κλάσεις που περιγράφουν μια απλή σχεσιακή βάση δεδομένων. Η Εφαρμογή μας χρησιμοποιεί DatagridView, DataTable, Datarow

```
Private Sub btnInsertStudents_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnInsertStudents.Click
    Dim foititis, onoma, epwnimo As String
    Dim dt As DataTable
    dt = DataGridView2.DataSource
    If (DataGridView1.SelectedRows.Count > 0) Then
        For Each dgv1Row As DataGridViewRow In DataGridView1.SelectedRows

            foititis = dgv1Row.Cells("AM").Value
            onoma = dgv1Row.Cells("ONOMA").Value
            epwnimo = dgv1Row.Cells("EPWNYMO").Value

            Dim row As DataRow = dt.NewRow
            row("AM") = foititis
            row("ONOMA") = onoma
            row("EPWNYMO") = epwnimo
            dt.Rows.Add(row)

            add_foititi(foititis, "add")

            DataGridView1.Rows.Remove(dgv1Row)
        Next
```

Εικόνα 11 **Τμήμα μεθόδου που χρησιμοποιεί** DatagridView, DataTable, DataRow

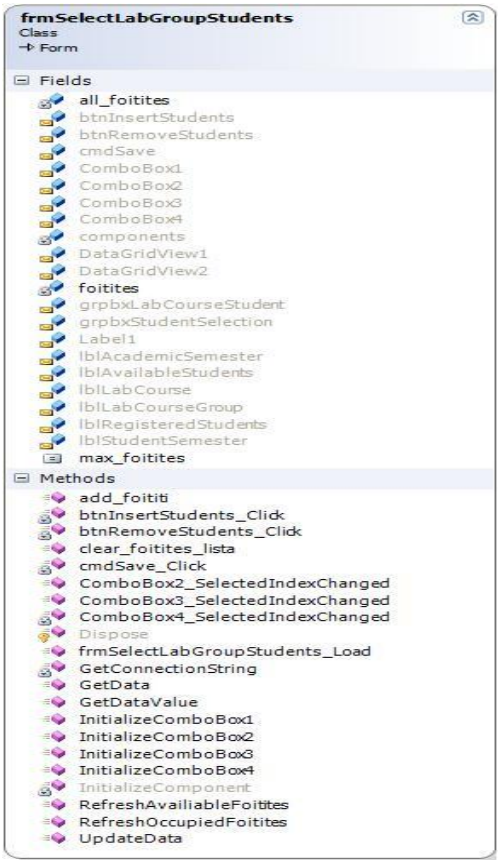
- **Ενότητα πίνακα(table module)** είναι ο τρόπος με τον οποίο διαχειριζόμαστε δομές δεδομένων προσανατολισμένες σε πίνακα (tabular data). Σ` αυτή την εφαρμογή για την διαχείριση της βάσης δεδομένων χρησιμοποιούμε sql ερωτήματα.

```
Public Sub RefreshAvailiableFoitites()
    Try
        With Me.DataGridView1
            .AutoGenerateColumns = True
            Dim c As String = String.Empty
            c = "SELECT foititis_id AS 'AM', foititis_onoma AS 'ONOMA', foititis_eponymo AS 'EPΩNYMO' " & _
                    "FROM Foititis " & _
                    "WHERE foititis_id NOT IN (" & _
                        "SELECT foititis_id FROM Ergastiriaki_Omada_Foititis WHERE ergastiriako_mathima_omada_id NOT IN (" & _
                            "SELECT ergastiriako_mathima_omada_id FROM Ergastiriako_Mathima_Omada WHERE ergastiriako_mathima_id NOT IN (" & _
                                "SELECT ergastiriako_mathima_id FROM Ergastiriako_Mathima WHERE ergastiriako_mathima_name= '" & ComboBox2.Text & "')" & _
                            ")" & _
                        ")" & _
                    "ORDER BY AM;"

            .DataSource = GetData(c)
        End With
    Catch ex As SqlClient.SqlException

    End Try
End Sub
```

Εικόνα 12 **Μέθοδος που χρησιμοποιεί** sql **ερώτημα**

Περιεχόμενα πειράματος

Σ` αυτό το σημείο θα δούμε πιο αναλυτικά τις λειτουργίες, μεθόδους και την κεντρική κλάση της εφαρμογής.



Εικόνα 13 **Το** Class diagram **με τις μεθόδους και τα στοιχεία της πρώτης εφαρμογής**

Κεντρική κλάση

Η *frmSelectLabGroupStudents* είναι η κεντρική κλάση της εφαρμογής και περιέχει όλες τις λειτουργίες και τις μεθόδους.

Λειτουργίες (functions)

- **GetConnectionString**: είναι μια λειτουργία που περιέχει τις απαραίτητες πληροφορίες για να γίνει η σύνδεση της εφαρμογής με την βάση δεδομένων.
- **GetData**: χρησιμοποιεί την GetConnectionString για να κάνει την σύνδεση στη βάση και χρησιμοποιεί τα sql ερωτήματα κάποιων μεθόδων για να εξάγει δεδομένα από την βάση. Έπειτα δημιουργεί ένα datatable και αποθηκεύει σε αυτό τα δεδομένα που εξάγει. Χρησιμοποιείται από τις μεθόδους InitializeComboBox1-4, RefreshOccupiedFoitites, RefreshAvailiableFoitites
- **UpdateData**: χρησιμοποιεί την GetConnectionString για να κάνει την σύνδεση στη βάση και χρησιμοποιεί sql ερωτήματα τα οποία δεν επιστρέφουν κάποια τιμή, τέτοια ερωτήματα είναι κυρίως αυτά που κάνουν αποθήκευση ή διαγραφή. Επίσης και αυτή δημιουργεί datatable για να αποθηκεύει τα δεδομένα των sql ερωτημάτων. Χρησιμοποιείται από την μέθοδο cmdSave_Click.
- **GetDataValue**: χρησιμοποιεί την GetConnectionString για να κάνει την σύνδεση στη βάση και αποθηκεύει τα δεδομένα ενός sql ερωτήματος σε datatable αλλά αναλαμβάνει να επιστρέψει τη πρώτη στήλη από τη πρώτη γραμμή από τα αποτελέσματα του sql ερωτήματος. Χρησιμοποιείται από την μέθοδο cmdSave_Click.

Μέθοδοι (sub)

- ***frmSelectLabGroupStudents_Load***: χρησιμοποιείται για να ενεργοποιήσει τις μεθόδους InitializeComboBox1 και InitializeComboBox2
- ***InitializeComboBox1****:* χρησιμοποιείται για να γεμίσει το πρώτο πεδίο (Ακαδημαϊκό Εξάμηνο) της φόρμας το οποίο είναι ένα ComboBox. Περιέχει την μεταβλητή C την οποία χρησιμοποιούμε σαν string[4] που περιέχει ένα sql ερώτημα με το οποίο περνούμε από την βάση δεδομένων το όνομα των Ακαδημαϊκών εξαμήνων από τον πίνακα Akadimaiko_Eksamino. Για να εμφανιστούν τα δεδομένα στο πεδίο πρέπει

[4] String είναι τύπος δεδομένων που χαρακτηρίζει τις μεταβλητές μιας εφαρμογής

να κάνουμε bind[5], έτσι δίνουμε σαν πηγή δεδομένων (Datasource) του ComboBox την λειτουργία GetData η οποία θα χρησιμοποιεί την C και ορίζουμε ποια από τα δεδομένα του ερωτήματος θα εμφανίζονται(Display Member).

- ***InitializeComboBox2****:* χρησιμοποιείται για να γεμίσει το δεύτερο πεδίο (Εργαστηριακό Μάθημα) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει ακριβώς όπως η InitializeComboBox1 όμως αλλάζει το sql ερώτημα. Εδώ το ερώτημα επιστρέφει το όνομα των εργαστηριακών μαθημάτων από τον πίνακα Ergastiriako_Mathima .
- ***InitializeComboBox3****:* χρησιμοποιείται για να γεμίσει το τρίτο πεδίο (Εργαστηριακή Ομάδα) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει ακριβώς όπως η InitializeComboBox1 όμως αλλάζει το sql ερώτημα. Εδώ το ερώτημα επιστρέφει το όνομα των εργαστηριακών ομάδων, που ανήκουν στο εργαστηριακό μάθημα το οποίο έχει επιλεχτεί στο δεύτερο πεδίο, από τον πίνακα Ergastiriako_Mathima_Omada.
- ***InitializeComboBox4****:* χρησιμοποιείται για να γεμίσει το τέταρτο πεδίο (Εξάμηνο Φοίτησης) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει ακριβώς όπως η InitializeComboBox1 όμως αλλάζει το sql ερώτημα. Εδώ το ερώτημα επιστρέφει το όνομα του Εξαμήνου φοίτησης, στο οποίο βρίσκετε το επιλεγμένο μάθημα από το δεύτερο πεδίο, από τον πίνακα Eksamino_Foitisis.
- ***ComboBox2_SelectedIndexChanged***: χρησιμοποιείται για να ελέγχει το περιεχόμενο του δεύτερου πεδίου και σε οποιαδήποτε αλλαγή του να ενεργοποιεί τις μεθόδους InitializeComboBox3 και InitializeComboBox4.
- ***ComboBox3_SelectedIndexChanged***: η χρήση της βοηθάει στον έλεγχο του περιεχομένου του τρίτου πεδίου και αν συμβεί οποιαδήποτε αλλαγή ενεργοποιεί τις μεθόδους RefreshAvailiableFoitites και RefreshOccupiedFoitites.
- **RefreshAvailiableFoitites**: γεμίζει το πέμπτο πεδίο (Διαθέσιμοι Φοιτητές) της φόρμας το οποίο είναι ένα DataGridView. Περιέχει την μεταβλητή C την οποία χρησιμοποιούμε σαν string (δεν έχουμε πρόβλημα που χρησιμοποιούμε την ίδια μεταβλητή γιατί τώρα βρίσκεται σε άλλη μέθοδο) που περιέχει ένα πολύπλοκο sql ερώτημα με το οποίο παίρνουμε από την βάση δεδομένων τα ονόματα των φοιτητών που δεν είναι γραμμένοι σε καμία ομάδα, του επιλεγμένου στο δεύτερο πεδίο, μαθήματος. Για να εμφανιστούν τα δεδομένα στο πεδίο πρέπει να κάνουμε bind έτσι δίνουμε

[5] Bind ή Data bind είναι η διαδικασία με την οποία "δένουμε" τα δεδομένα σε κάποια λειτουργία. Πχ ένα ComboBox ή ένα DataGridView

σαν πηγή δεδομένων (Datasource) του DataGridView την λειτουργία *GetData* η οποία θα χρησιμοποιεί την C

- **RefreshOccupiedFoitites:** γεμίζει το έκτο πεδίο (Καταχωρημένοι Φοιτητές) της φόρμας το οποίο είναι ένα DataGridView. Ο τρόπος λειτουργίας είναι ο ίδιος(χρησιμοποιεί την C που περιέχει ένα sql ερώτημα και κάνουμε bind για να εμφανιστούν τα δεδομένα) όμως αλλάζει το ερώτημα. Εδώ το ερώτημα επιστρέφει τα ονόματα των φοιτητών που είναι γραμμένοι στην επιλεγμένη εργαστηριακή ομάδα στο τρίτο πεδίο η οποία ανήκει στο επιλεγμένο μάθημα που βρίσκεται στο δεύτερο πεδίο.
- **clear_foitites_lista**: Αδειάζει τη λίστα με το όνομα all_foitites από οποιαδήποτε εγγραφή περιέχει. Χρησιμοποιείται από την μέθοδο cmdSave_Click.
- **add_foititi**: χρησιμοποιείται από τις μεθόδους btnInsertStudents_Click και btnRemoveStudents_Click για να ενημερώνει την λίστα all_foitites όσο γίνονται αλλαγές στα DataGridView(πεδία 5 και 6).
- **btnInsertStudents_Click**: είναι η μέθοδος που ενεργοποιείται όταν πατήσουμε το κουμπί  . Αυτή η μέθοδος είναι υπεύθυνη για την εγγραφή των φοιτητών(μεταφέρει τους φοιτητές από το πεδίο 5 στο πεδίο 6). Χρησιμοποιεί ένα DataTable το οποίο έχει σαν πηγή δεδομένων (Datasource) την πηγή του έκτου πεδίου. Όταν επιλέξουμε την σειρά του πεδίου που περιέχει τον φοιτητή που θέλουμε να εγγράψουμε (να τον μεταφέρουμε στο πεδίο 6) ενεργοποιεί την μέθοδο add_foititi η οποία ενημερώνει την λίστα all_foitites προσθέτοντάς της τον συγκεκριμένο φοιτητή. Επειδή όμως ενεργοποιείται από την μέθοδο btnInsertStudents_Click όταν προσθέσει τον φοιτητή στην λίστα προσθέτει και τον χαρακτηρισμό (λέξη) add . Αν δεν έχουμε επιλέξει όλη τη σειρά την επιλέγει αυτόματα.
- **btnRemoveStudents_Click**: είναι η μέθοδος που ενεργοποιείται όταν πατήσουμε το κουμπί  . Αυτή η μέθοδος είναι υπεύθυνη για την διαγραφή των φοιτητών(μεταφέρει τους φοιτητές από το πεδίο 6 στο πεδίο 5). Χρησιμοποιεί ένα DataTable το οποίο έχει σαν πηγή δεδομένων (Datasource) την πηγή του πέμπτου πεδίου. Όταν επιλέξουμε την σειρά του πεδίου που περιέχει τον φοιτητή που θέλουμε να διαγράψουμε (να τον μεταφέρουμε στο πεδίο 5) ενεργοποιεί την υποκλάση add_foititi η οποία ενημερώνει την λίστα all_foitites προσθέτοντάς της τον συγκεκριμένο φοιτητή Επειδή όμως ενεργοποιείται από την μέθοδο btnRemoveStudents_Click όταν προσθέσει τον φοιτητή στην λίστα προσθέτει και τον χαρακτηρισμό (λέξη) remove. Αν δεν έχουμε επιλέξει όλη τη σειρά την επιλέγει αυτόματα.

- **cmdSave_Click**: Αυτή η μέθοδος ενεργοποιείται όταν πατήσουμε το κουμπί Αποθήκευση και είναι υπεύθυνη για να μεταφερθούν οι αλλαγές που έχουν γίνει στα DatagridView(πεδία 5 και 6) στην βάση δεδομένων. Όταν πατήσουμε το κουμπί Αποθήκευση τότε η cmdSave_Click ενεργοποιεί τη λειτουργία GetDataValue η οποία χρησιμοποιεί ένα ερώτημα με το οποίο μπορεί και βρίσκει τον κωδικό (id) της εργαστηριακής ομάδας στην οποία θα γίνουν οι αλλαγές. Έπειτα ψάχνει την λίστα all_foitites για να βρει αυτούς με τον χαρακτηρισμό add και ενεργοποιεί τη λειτουργία UpdateData η οποία χρησιμοποιεί ένα sql ερώτημα με το οποίο προσθέτει στην ομάδα που έχει βρει πιο πριν, τους φοιτητές. Για τους φοιτητές με τον χαρακτηρισμό remove ενεργοποιεί πάλι τη λειτουργία UpdateData η οποία χρησιμοποιεί ένα sql ερώτημα με το οποίο διαγράφει από την ομάδα που έχει βρει πιο πριν, τους φοιτητές.

Περιγραφή του πειράματος

Όταν γίνει εκκίνηση της εφαρμογής η πρώτη μέθοδος που ενεργοποιείται είναι η *frmSelectLabGroupStudents_Load* η οποία με τη σειρά της ενεργοποιεί τις μεθόδους *InitializeComboBox1* και *InitializeComboBox2*. Οι δυο αυτές μέθοδοι γεμίζουν με δεδομένα τα δυο πρώτα πεδία της φόρμας(*Ακαδημαϊκό Εξάμηνο, Εργαστηριακό Μάθημα*).

Όταν επιστραφούν τα δεδομένα και γεμίσουν τα δύο πρώτα πεδία ενεργοποιείται η μέθοδος ComboBox2_SelectedIndexChanged η οποία ελέγχει αν γίνονται αλλαγές στο περιεχόμενο του δεύτερου πεδίου. Επειδή η μέθοδος InitializeComboBox2 γεμίζει με δεδομένα το δεύτερο πεδίο, του αλλάζει το αρχικό του περιεχόμενο (Δεν περιείχε τίποτα) η ComboBox2_SelectedIndexChanged ενεργοποιεί τις InitializeComboBox3 και InitializeComboBox4

Η *InitializeComboBox3* γεμίζει το τρίτο πεδίο (Εργαστηριακή Ομάδα) με τις ομάδες που περιέχει το επιλεγμένο εργαστηριακό μάθημα του δεύτερου πεδίου και η *InitializeComboBox4 (Εξάμηνο Φοίτησης)* γεμίζει το τέταρτο πεδίο το οποίο περιέχει το εξάμηνο στο οποίο βρίσκεται το επιλεγμένο εργαστηριακό μάθημα.

Με την επιστροφή των δεδομένων ενεργοποιείται η μέθοδος *ComboBox3_SelectedIndexChanged* η οποία ελέγχει αν γίνονται αλλαγές στο περιεχόμενο του τρίτου πεδίου. Επειδή η μέθοδος *InitializeComboBox3* γεμίζει με δεδομένα το τρίτο πεδίο, του αλλάζει το αρχικό του περιεχόμενο, η *ComboBox3_SelectedIndexChanged* ενεργοποιεί τις *RefreshAvailiableFoitites* και *RefreshOccupiedFoitites* οι οποίες γεμίζουν το πεμπτο πεδίο (Διαθέσιμοι Φοιτητές) και το εκτο πεδίο (Καταχωρημένοι Φοιτητές)

Αν θέλουμε να γράψουμε κάποιον φοιτητή στην ομάδα που έχουμε επιλέξει στο τρίτο πεδίο τον επιλέγουμε από το πεδίο Διαθέσιμοι Φοιτητές και πατάμε το κουμπί  το οποίο ενεργοποιεί την μέθοδο btnInsertStudents_Click η οποία γράφει τον φοιτητή και παράλληλα με την βοήθεια της μεθόδου add_foititi θυμάται ότι έγινε η εγγραφή του.

Αν θέλουμε να διαγράψουμε κάποιον φοιτητη από την ομαδα που έχουμε επιλέξει στο τρίτο πεδίο τον επιλεγουμε από το πεδίο Καταχωρημένοι Φοιτητές και παταμε το κουμπι  το οποίο ενεργοποιει την μέθοδο *btnRemoveStudents_Click* η οποία διαγράφει τον φοιτητη και παράληλα με την βοήθεια της μεθόδου *add_foititi* θυμάται ότι εγινε η διαγράφή του.

Όταν ολοκληρώσουμε τις αλλαγές που θέλουμε να κάνουμε πατάμε το κουμπί Αποθήκευση το οποίο ενεργοποιεί την μέθοδο cmdsave_Click η οποία με την βοήθεια τις λειτουργίας GetDataValue βρίσκει σε ποια εργαστηριακή ομάδα κάναμε αλλαγές. Αφού βρει την ομάδα χρησιμοποιεί τη λειτουργία UpdateData σε συνδυασμό με ένα sql ερώτημα το οποίο προσθέτει φοιτητές στην ομάδα, αν βρει στην λίστα add_foititi φοιτητές που έχουν εγγραφθεί. Για τους φοιτητές που θα βρει στην λίστα ότι έχουν διαγραφθεί η λειτουργία UpdateData χρησιμοποιείται σε συνδυασμό με ένα sql ερώτημα το οποίο θα τους διαγράψει από την ομάδα.

Συμπεράσματα

Με τον συγκεκριμένο τρόπο υλοποίησης, την χρήση των αρχιτεκτονικών του πολυεπίπεδου προγραμματισμού και ADO.NET δημιουργούμε σχετικά γρήγορα μια εφαρμογή. Αυτό βέβαια οφείλεται στο ότι τα εργαλεία που χρησιμοποιούμε έχουν μεγάλη συμβατότητα μεταξύ τους. Αυτός ο τρόπος δημιουργεί και κάποια προβλήματα. Δημιουργούμε πάρα πολλά sql ερωτήματα που στην πλειοψηφία τους είναι αρκετά πολύπλοκα. Τόσο η υλοποίησή τους αλλά και η κατανόησή τους σε κάποια συντήρηση ή επέκταση της εφαρμογής θα είναι πολλή δύσκολη.

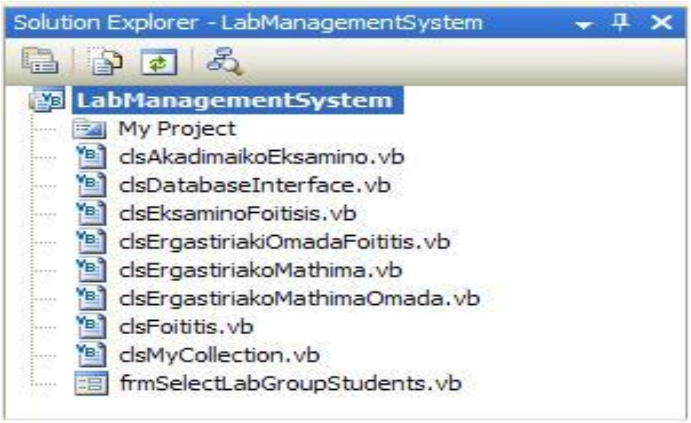
Η εσωτερική λίστα που έχουμε φτιάξει ώστε να κρατάμε τις αλλαγές των δεδομένων μέχρι να αποφασίσουμε να τις αποθηκεύσουμε στην βάση, αν τοποθετηθεί σε μια μεγαλύτερης κλίμακας εφαρμογή θα δεσμεύει πάρα πολύ μνήμη με αποτέλεσμα να είναι η εφαρμογή πιο αργή και λιγότερο αξιόπιστη λόγω του ότι υπάρχει και ο κίνδυνος να “μπουχτίσει” η λίστα με αποτέλεσμα να σταματήσει η λειτουργία της εφαρμογής.

Όλες οι μέθοδοι της εφαρμογής που χρησιμοποιούν κάποιο sql ερώτημα κάνουν, κάθε φόρα που ενεργοποιούνται, σύνδεση με την βάση δεδομένων. Αυτό έχει σαν αποτέλεσμα να υπάρχει και εδώ κίνδυνος να σταματήσει η λειτουργία της εφαρμογής, λόγω των πολλαπλών προσπαθειών της εφαρμογής να συνδεθεί στη βάση δεδομένων (μπορεί κάποια φορά να αποτύχει).

Υλοποίηση δεύτερου πειράματος

Ο δεύτερος τρόπος υλοποίησης διαφέρει κατά πολύ από τον πρώτο διότι, ενώ ο πρώτος βασίζεται στις αρχιτεκτονικές του πολυεπίπεδου προγραμματισμού και ADO.Net, όπως ακριβώς επιτάσσει η θεωρία, εδώ θα προσπαθήσουμε να εισάγουμε και κάποια στοιχεία της NHibernate αρχιτεκτονικής (θα χρησιμοποιήσουμε το domain model αντί για το table module).

Στον πολυεπίπεδο προγραμματισμό μια εφαρμογή πρέπει να χωρίζεται σε βαθμίδες: όπως Βαθμίδα παρουσίασης, Βαθμίδα εφαρμογής και Βαθμίδα δεδομένων. Στην εφαρμογή μας η Βαθμίδα παρουσίασης είναι η frmSelectLabGroupStudents, Βαθμίδα εφαρμογής είναι η clsDatabaseInterface και Βαθμίδα δεδομένων είναι η βάση δεδομένων ateidb.



Εικόνα 14 **Βαθμίδες παρουσίασης και εφαρμογής μαζί με άλλα στοιχεία της εφαρμογής**

Η χρήση της ADO.Net αρχιτεκτονικής τροποποιείται με αποτέλεσμα κάποια από τα στοιχεία της να μην τα χρησιμοποιούμε και να εισάγουμε άλλα στην θέση τους για να μας βοηθήσουν στην υλοποίηση της εφαρμογής.

Η τροποποιημένη από εμάς ADO.Net αρχιτεκτονική αποτελείται από:

- **Πάροχο δεδομένων(Data provider)**: ο οποίος έχει κλάσεις που παρέχουν πρόσβαση σε μία πηγή δεδομένων. Στην συγκεκριμένη εφαρμογή χρησιμοποιούμε τις Connection, Command, Parameter και DataReader.

```
Public Function GetAkadimaikaEksamina() As ArrayList
    Dim akadimaikaEksamina As New ArrayList
    Dim sqlString As String = "Select * From Akadimaiko_Eksamino"
    Dim command As SqlCommand
    Dim reader As SqlDataReader
    Dim con As SqlConnection
    Try
        con = GetConnection()
        con.Open()
        command = New SqlCommand(sqlString, con)
        reader = command.ExecuteReader()

        While (reader.Read())
            Dim akadimaikoEksamino As New clsAkadimaikoEksamino(reader.GetInt32(0), reader.GetInt32(1))
            akadimaikoEksamino.akadimaiko_eksamino_name = reader.GetString(2)
            akadimaikoEksamino.akadimaiko_eksamino_start_date = reader.GetDateTime(3)
            akadimaikoEksamino.akadimaiko_eksamino_end_date = reader.GetDateTime(4)
            If Not reader.IsDBNull(5) Then
                akadimaikoEksamino.akadimaiko_eksamino_perigrafi = reader.GetString(5)
            End If
            akadimaikaEksamina.Add(akadimaikoEksamino)
        End While
    Catch e As Exception
        Throw e
    Finally
        Try
            If Not reader Is Nothing Then
                If Not reader.IsClosed Then
                    reader.Close()
                End If
            End If
            con.Close()
        Catch e As SqlException
            Throw e
        End Try
    End Try
    Return akadimaikaEksamina
End Function
```

Εικόνα 15 **Λειτουργία που χρησιμοποιεί τις** Connection, Command, Parameter **και** DataReader

- **Σύνολα δεδομένων(Datasets)**: είναι μια ομάδα από κλάσεις που περιγράφουν μια απλή σχεσιακή βάση δεδομένων. Η Εφαρμογή μας χρησιμοποιεί DataGrid.

```
Public Sub RefreshOccupiedFoitites()

    Dim foititesErgastiriakisOmadas As ArrayList = dbInterface.GetOccupiedStudentsByErgastiriakiOmada(ComboBox3.SelectedValue)

    Dim ts As New DataGridTableStyle
    ts.MappingName = "ArrayList"
    With Me.DataGrid2
        .DataSource = foititesErgastiriakisOmadas

        Dim cs As New DataGridTextBoxColumn
        cs.MappingName = "akadimaiko_aidrima_foititia_am"
        cs.HeaderText = "A.M."
        ts.GridColumnStyles.Add(cs)

        cs = New DataGridTextBoxColumn
        cs.MappingName = "foititis_onoma"
        cs.HeaderText = "ΟΝΟΜΑ"
        ts.GridColumnStyles.Add(cs)

        cs = New DataGridTextBoxColumn
        cs.MappingName = "foititis_eponymo"
        cs.HeaderText = "ΕΠΩΝΥΜΟ"
        ts.GridColumnStyles.Add(cs)

        .TableStyles.Clear()
        .TableStyles.Add(ts)

    End With
End Sub
```

Εικόνα 16 **Μέθοδος που χρησιμοποιεί** DataGrid

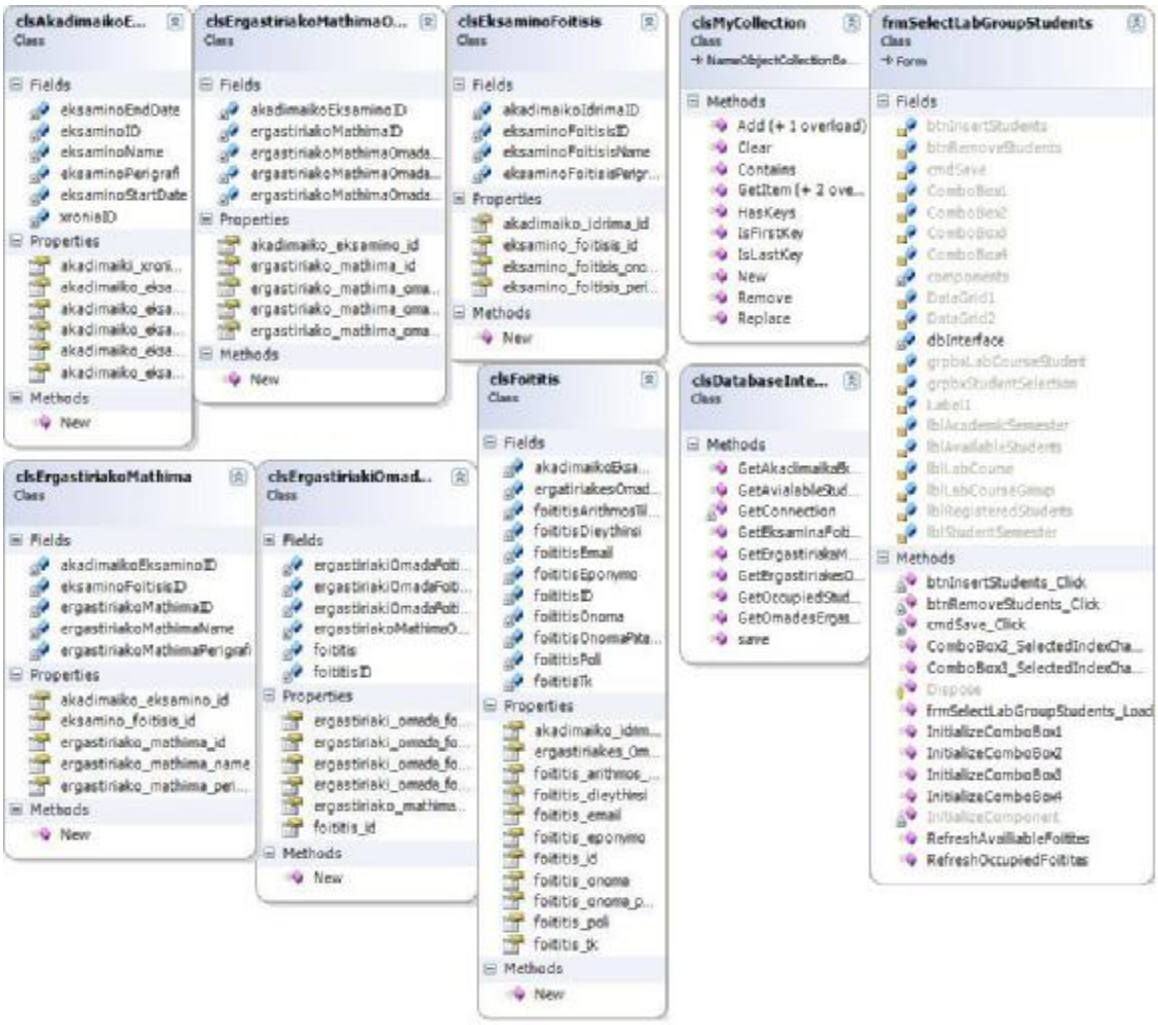
- **Μοντέλο τομέα (Domain model)**: μας βοηθάει να δημιουργήσουμε ένα σχέδιο της βάσης δεδομένων στην εφαρμογή μας και να χρησιμοποιήσουμε αντικείμενα για να διαχειριστούμε τους πίνακες της βάσης δεδομένων.

```
Public Function GetErgastiriakesOmadesFoititi(ByVal foititisID As Integer) As clsMyCollection
    Dim ergastiriakesOmadesFoititi As New clsMyCollection
    Dim sqlString As String = "SELECT * FROM Ergastiriaki_Omada_Foititis WHERE foititis_id = @FOITITISID"
    Dim command As SqlCommand
    Dim reader As SqlDataReader
    Dim con As SqlConnection
    Try
        con = GetConnection()
        con.Open()
        command = New SqlCommand(sqlString, con)
        command.Parameters.AddWithValue("@FOITITISID", foititisID)
        reader = command.ExecuteReader()
        While (reader.Read())
            Dim ergastiriakiOmadaFoititis As New clsErgastiriakiOmadaFoititis(reader.GetInt32(0), reader.GetInt32(1))
            If Not reader.IsDBNull(2) Then
                ergastiriakiOmadaFoititis.ergastiriaki_omada_foititis_apousies = reader.GetInt16(2)
            End If
            If Not reader.IsDBNull(3) Then
                ergastiriakiOmadaFoititis.ergastiriaki_omada_foititis_telikos_bathmos = reader.GetDecimal(3)
            End If
            If Not reader.IsDBNull(4) Then
                ergastiriakiOmadaFoititis.ergastiriaki_omada_foititis_telikos_katastasi = reader.GetString(4)
            End If
            ergastiriakesOmadesFoititi.Add(ergastiriakiOmadaFoititis.ergastiriako_mathima_omada_id, ergastiriakiOmadaFoititis)
        End While
```

Εικόνα 17 **Τμήμα μεθόδου που γεμίζει με δεδομένα ένα πίνακα που έχουμε δημιουργήσει στην εφαρμογή μας και αντιπροσωπεύει πίνακα της βάσης δεδομένων**

Περιεχόμενα πειράματος

Σ` αυτό το σημείο θα δούμε πιο αναλυτικά τις λειτουργίες και μεθόδους που βρίσκονται στις βαθμίδες εφαρμογής (*frmSelectLabGroupStudents*) και παρουσίασης(*clsDatabaseInterface*) καθώς και τις κλάσεις που αναπαριστούν την βάση δεδομένων της εφαρμογής μας.



Εικόνα 18 **Το** Class Diagram **της δεύτερης εφαρμογής με τους χαρτογραφημένους πίνακες της βάσης και τις υποκλάσεις με τις μεθόδους και τα στοιχεία τους**

Βαθμίδα παρουσίασης

Η Βαθμίδα παρουσίασης είναι στην ουσία η κλάση *frmSelectLabGroupStudents* και αποτελείται από τις παρακάτω μεθόδους που για μπορέσουμε να τις χρησιμοποιήσουμε δημιουργούμε το αντικείμενο *dbInterface* το οποίο ανήκει στην υποκλάση *clsDatabaseInterface*(Βαθμίδα εφαρμογής) .

- **frmSelectLabGroupStudents_Load**: χρησιμοποιείται για να ενεργοποιήσει τις μεθόδους *InitializeComboBox1 και InitializeComboBox2*
- **InitializeComboBox1**: χρησιμοποιείται για να γεμίσει το πρώτο πεδίο (Ακαδημαϊκό Εξάμηνο) της φόρμας το οποίο είναι ένα ComboBox. Περιέχει την μεταβλητή akadimaikaEksamina την οποία χρησιμοποιούμε

σαν ArrayList[6] και την γεμίζουμε με δεδομένα μέσω του αντικειμένου dbInterface το οποίο χρησιμοποιεί την λειτουργία GetAkadimaikaEksamina για να βρει τα ακαδημαϊκά εξάμηνα που περιέχονται στην κλάση clsAkadimaikoEksamino. Για να εμφανιστούν τα δεδομένα στο πεδίο πρέπει να κάνουμε bind. Δίνουμε σαν πηγή δεδομένων (Datasource) του ComboBox τη μεταβλητή akadimaikaEksamina, επειδή όμως η μεταβλητή akadimaikaEksamina περιέχει όλο τον πίνακα Akadimaiko_Eksamino ορίζουμε στο ComboBox πια δεδομένα θα δείχνει με την εντολή DisplayMember στη συγκεκριμένη περίπτωση το όνομα του Ακαδημαϊκού Εξαμήνου και κρατάμε στην μνήμη τον αριθμό του(ID) με την εντολή ValueMember.

- **InitializeComboBox2**: χρησιμοποιείται για να γεμίσει το δεύτερο πεδίο (Εργαστηριακό Μάθημα) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει ακριβώς όπως η InitializeComboBox1 όμως αλλάζει η λειτουργία που χρησιμοποιεί το αντικείμενο dbInterface στην GetErgastiriakaMathimata και μας επιστρέφει τα εργαστηριακά μαθήματα που περιέχονται στην υποκλάση clsErgastiriakoMathima. Πάλι κάνουμε bind και του δίνουμε εντολή να δείχνει το όνομα του εργαστηριακού μαθήματος και να κρατάει στην μνήμη τον αριθμό του.
- **InitializeComboBox3**: χρησιμοποιείται για να γεμίσει το τρίτο πεδίο (Εργαστηριακή Ομάδα) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει σχεδόν όπως η InitializeComboBox1. Το αντικείμενο dbInterface σε συνδυασμό με την λειτουργία GetOmadesErgastiriakwnMathimatwn χρησιμοποιεί και τον αριθμό του εργαστηριακού μαθήματος που είχαμε κρατήσει στην μνήμη, για να μπορέσει να μας επιστρέψει από την υποκλάση clsErgastiriakoMathimaOmada τις εργαστηριακές ομάδες του μαθήματος που έχουμε επιλέξει στο δεύτερο πεδίο. Πάλι κάνουμε bind και του δίνουμε εντολή να δείχνει το όνομα της εργαστηριακής ομάδας και να κρατάει στην μνήμη τον αριθμό της (id).
- **InitializeComboBox4**: χρησιμοποιείται για να γεμίσει το τέταρτο πεδίο (Εξάμηνο Φοίτησης) της φόρμας το οποίο είναι ένα ComboBox. Δουλεύει ακριβώς όπως η InitializeComboBox3 όμως αλλάζει η λειτουργία που χρησιμοποιεί το αντικείμενο dbInterface στην GetEksaminaFoitisis και μας επιστρέφει από την υποκλάση clsEksaminoFoitisis το εξάμηνο στο οποίο βρίσκεται το μάθημα που έχουμε επιλέξει στο δεύτερο πεδίο. Πάλι κάνουμε bind και του δίνουμε εντολή να δείχνει το όνομα του εξαμήνου φοίτησης και να κρατάει στην μνήμη τον αριθμό του (id).

[6] Δομή δεδομένων που περιέχει μια απλή λίστα με τιμές

- **ComboBox2_SelectedIndexChanged**: χρησιμοποιείται για να ελέγχει το περιεχόμενο του δεύτερου πεδίου και σε οποιαδήποτε αλλαγή του να ενεργοποιεί τις μεθόδους InitializeComboBox3 και InitializeComboBox4.
- **ComboBox3_SelectedIndexChanged**: η χρήση της βοηθάει στον έλεγχο του περιεχομένου του τρίτου πεδίου και αν συμβεί οποιαδήποτε αλλαγή ενεργοποιεί τις μεθόδους RefreshAvailiableFoitites και RefreshOccupiedFoitites.
- **RefreshAvailiableFoitites**: χρησιμοποιείται για να γεμίσει το πέμπτο πεδίο (Διαθέσιμοι Φοιτητές) της φόρμας το οποίο είναι ένα DataGrid. Περιέχει την μεταβλητή foititesErgastiriakisOmadas την οποία χρησιμοποιούμε σαν ArrayList και την γεμίζουμε με δεδομένα μέσω του αντικειμένου dbInterface, το οποίο χρησιμοποιεί την λειτουργία GetAvialableStudentsByErgastiriakiMathima σε συνδυασμό με τον αριθμό του εργαστηριακού μαθήματος (ComboBox2.SelectedValue) που έχουμε επιλέξει στο δεύτερο πεδίο, για να μας επιστρέψει από την υποκλάση clsErgastiriakiOmadaFoititis τους φοιτητές οι οποίοι δεν έχουν γραφτεί σε καμία εργαστηριακή ομάδα του εργαστηριακού μαθήματος που έχουμε επιλέξει στο δεύτερο πεδίο. Μετά δημιουργούμε τη μεταβλητή ts η οποία ουσιαστικά είναι ένα κενό DataGrid και της ορίζουμε ότι τα δεδομένα που θα χρησιμοποιήσει είναι τύπου ArrayList. Για να εμφανιστούν τα δεδομένα και εδώ πρέπει να κάνουμε bind. Ορίζουμε σαν πηγή δεδομένων του DataGrid την μεταβλητή foititesErgastiriakisOmadas και στην συνέχεια πρέπει να φτιάξουμε τις στήλες του DataGrid ώστε να εμφανίσουμε τα δεδομένα που θέλουμε. Φτιάχνουμε την μεταβλητή cs και ορίζουμε ότι θα είναι DataGridTextBoxColumn (στήλη). Δημιουργούμε τρείς στήλες. Στην πρώτη με την εντολή MappingName εμφανίζουμε τον αριθμό μητρώου του φοιτητή, με την εντολή HeaderText της δίνουμε το όνομα Α.Μ. και την προσθέτουμε στην μεταβλητή ts, στην δεύτερη στήλη εμφανίζουμε το όνομα του φοιτητή την ονομάζουμε ΟΝΟΜΑ και την προσθέτουμε στην μεταβλητή ts, στην τρίτη εμφανίζουμε το επώνυμο του φοιτητή την ονομάζουμε ΕΠΩΝΥΜΟ και την προσθέτουμε στην μεταβλητή ts.
- **RefreshOccupiedFoitites**: χρησιμοποιείται για να γεμίσει το έκτο πεδίο (Καταχωρημένοι Φοιτητές) της φόρμας το οποίο είναι ένα DataGrid. Περιέχει την μεταβλητή foititesErgastiriakisOmadas την οποία χρησιμοποιούμε σαν ArrayList και την γεμίζουμε με δεδομένα μέσω του αντικειμένου dbInterface, το οποίο χρησιμοποιεί την λειτουργία GetOccupiedStudentsByErgastiriakiOmada σε συνδυασμό με τον αριθμό (id) της εργαστηριακής ομάδας (ComboBox3.SelectedValue) που έχουμε επιλέξει στο τρίτο πεδίο, για να μας επιστρέψει από την υποκλάση

clsErgastiriakiOmadaFoititis τους φοιτητές οι οποίοι έχουν γραφτεί στην εργαστηριακή ομάδα που έχουμε επιλέξει στο τρίτο πεδίο. Έπειτα ακολουθούμε ακριβώς τα ίδια βήματα με την RefreshAvailiableFoitites για να εμφανίζουμε τα δεδομένα.

- **btnRemoveStudents_Click**: ενεργοποιείται όταν πατήσουμε το κουμπί < και διαγράφει τον φοιτητή που έχουμε επιλέξει(τον μεταφέρει από το έκτο πεδίο στο πέμπτο). Εδώ ορίζουμε τέσσερεις μεταβλητές. Οι πρώτες δύο είναι: η *cmOccupiedStudents* την οποία χρησιμοποιούμε σαν *CurrencyManager*[7] και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid2 και η cmAvailableStudents την οποία χρησιμοποιούμε σαν CurrencyManager και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid1. Οι επόμενες δύο είναι: η occupiedStudents την οποία χρησιμοποιούμε σαν ArrayList και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid2 και η availableStudents την οποία χρησιμοποιούμε σαν ArrayList και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid1. Έπειτα κάνουμε δύο ελέγχους. Με τον πρώτο εξετάζουμε αν το πλήθος των στοιχείων της occupiedStudents είναι μεγαλύτερο από το μηδέν, αν είναι τότε ορίζουμε δύο μεταβλητές: την foititis όπου την εξισώνουμε με το επιλεγμένο στοιχείο της occupiedStudents και την θέση του που μας δίνεται από την cmOccupiedStudents (occupiedStudents.Item (cmOccupiedStudents.Position)) και την removeAt που την χρησιμοποιούμε σαν Integer[8](ακέραιο) και την εξισώνουμε με την cmOccupiedStudents.Position που περιέχει τη θέση ενός επιλεγμένου στοιχείου. Με τον δεύτερο έλεγχο εξετάζουμε αν η μεταβλητή removeAt περιέχει κάποιο στοιχείο, αν περιέχει, το αφαιρούμε από την cmOccupiedStudents.Position. Έπειτα αφαιρούμε την removeAt από την occupiedStudents με την εντολή RemoveAt, προσθέτουμε την μεταβλητή foititis στην availableStudents με την εντολή Add, και εκτελούμε την εντολή Refresh στις cmAvailableStudents και cmOccupiedStudents.
- **btnInsertStudents_Click**: ενεργοποιείται όταν πατήσουμε το κουμπί > και εγγράφει τον φοιτητή που έχουμε επιλέξει(τον μεταφέρει από το πέμπτο πεδίο στο έκτο). Εδώ ορίζουμε τέσσερεις μεταβλητές. Οι πρώτες δύο είναι: η *cmOccupiedStudents* την οποία χρησιμοποιούμε σαν *CurrencyManager* και της ορίζουμε σαν πηγή δεδομένων την πηγή του

[7] Ο CurrencyManager είναι μια κλάση της VB.Net η οποία μπορεί να συγχρωνίσει τα εργαλεία μιας φόρμας (στην περίπτωσή μας το πεμπτο και έκτο πεδίο) τοποθετόντας κάποιους δείκτες στα στοιχεία που περιέχονται σ` αυτά.

[8] Είναι τύπος δεδομένων που χαρακτηρίζει τις μεταβλητές μιας εφαρμογής.

DataGrid2 και η cmAvailableStudents την οποία χρησιμοποιούμε σαν CurrencyManager και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid1. Οι επόμενες δύο είναι: η occupiedStudents την οποία χρησιμοποιούμε σαν ArrayList και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid2 και η availableStudents την οποία χρησιμοποιούμε σαν ArrayList και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid1. Έπειτα κάνουμε τρείς ελέγχους. Με τον πρώτο εξετάζουμε αν το πλήθος των στοιχείων της availableStudents είναι μεγαλύτερο από το μηδέν, αν είναι τότε ορίζουμε δύο μεταβλητές: την foititis όπου την εξισώνουμε με το επιλεγμένο στοιχείο της availableStudents και την θέση του που μας δίνεται από την cmAvailableStudents (availableStudents.Item(cmAvailableStudents.Position)) και την removeAt που την χρησιμοποιούμε σαν Integer και την εξισώνουμε με την cmAvailableStudents.Position που περιέχει τη θέση ενός επιλεγμένου στοιχείου. Με τον δεύτερο έλεγχο εξετάζουμε αν η μεταβλητή removeAt περιέχει κάποιο στοιχείο, αν περιέχει, το αφαιρούμε από την cmAvailableStudents.Position. Δημιουργούμε την μεταβλητή ergastiriakiOmadaFoititis και την χρησιμοποιούμε σαν clsErgastiriakiOmadaFoititis[9] προσθέτοντάς της τα ορίσματα foititis.foititis_id(είναι ο αριθμός του φοιτητή που περιέχεται στον πίνακα foititis) και ComboBox3.SelectedValue(είναι ο αριθμός της εργαστηριακής ομάδας που έχουμε επιλέξει στο τρίτο πεδίο).Με τον τρίτο έλεγχο εξετάζουμε αν ο αριθμός του φοιτητή που έχουμε επιλέξει σχετίζεται με τον αριθμό της επιλεγμένης ομάδας, αν δεν σχετίζονται τότε προσθέτουμε την foititis στην occupiedStudents με την εντολή Add. Αφαιρούμε την μεταβλητή removeAt από την availableStudents με την εντολή RemoveAt, και εκτελούμε την εντολή Refresh στις cmAvailableStudents και cmOccupiedStudents.

- **cmdSave_Click**: χρησιμοποιεί το αντικείμενο dbInterface μέσω της μεθόδου save (η οποία χρησιμοποιεί την: ComboBox3.SelectedValue όπου είναι ο αριθμός της επιλεγμένης εργαστηριακής ομάδας και DataGrid2.DataSource που είναι η τροποποιημένη πηγή δεδομένων από την occupiedStudents) για να αποθηκεύσει τις αλλαγές στην βάση δεδομένων.

[9] clsErgastiriakiOmadaFoititis είναι μια κλάση που έχουμε φτιάξει στην εφαρμογή μας που αντιπροσωπεύει έναν πίνακα από την βάση δεδομένων ο οποίος περιέχει τους φοιτητές και τις εργαστηριακές ομάδες που σχετίζονται με αυτόν.

Βαθμίδα εφαρμογής

Η Βαθμίδα εφαρμογής είναι η κλάση clsDatabaseInterface και αποτελείται από τις παρακάτω λειτουργίες και υποκλάση:

Λειτουργίες

- **GetConnection**:είναι η λειτουργία που περιέχει τις απαραίτητες πληροφορίες που χρειάζονται όλες οι υπόλοιπες λειτουργίες και υποκλάση για να συνδεθούν με τη βάση δεδομένων.
- **GetAkadimaikaEksamina**:είναι η λειτουργία που μας επιστρέφει τα ακαδημαϊκά εξάμηνα. Χρησιμοποιεί αρχικά πέντε μεταβλητές, την *akadimaikaEksamina* σαν ArrayList, την *command* σαν sqlCommand ώστε να μπορεί να χρησιμοποιεί sql, την *reader* σαν SqlDataReader για να μπορεί να διαβάζει, την *con* που περιέχει τις πληροφορίες για την σύνδεση με την βάση δεδομένων και την *sqlString* που είναι το ερώτημα που μας επιστρέφει τα ακαδημαϊκά εξάμηνα από την βάση. Έπειτα κάνουμε την σύνδεση στη βάση και ενεργοποιούμε στις *command* και *reader*. Αφού έχει συνδεθεί στη βάση και συγκεκριμένα στον πίνακα που ζητάμε μέσω του ερωτήματος ενεργοποιείται η *reader* που δημιουργεί την μεταβλητή *akadimaikoEksamino* και όσο υπάρχουν δεδομένα στον πινακα που διαβάζει τα προσθέτει σε αυτή.Όταν τελειώσει, τα δεδομένα προστήθενται στην *akadimaikaEksamina* απενεργοποιείται η *reader* κλείνει η σύνδεση με την βάση δεδομένων και μέσω της εντολής *Return* η *akadimaikaEksamina* μας επιστρέφει τα δεδομένα που περιέχει και γεμίζει έτσι η *clsAkadimaikoEksamino*.
- **GetErgastiriakaMathimata**:δουλεύει ακριβώς όπως και η GetAkadimaikaEksamina αλλα αυτή μας επιστρέφει τα εργαστηριακα μαθήματα οποτε αλλάζει μονο το ερωτημα που περιεχει και γεμιζει την *clsErgastiriakoMathima.*
- **GetOmadesErgastiriakwnMathimatwn**: δουλεύει όπως και η GetAkadimaikaEksamina αλλα αυτή μας επιστρέφει τις εργαστηριακές ομάδες που εχει το κάθε εργαστηριακό μάθημα. Εδώ λοιπόν υπάρχει η διαφορά με τις προηγούμενες λειτουργίες. Το ερώτημα που έχουμε φτιάξει συνδυάζει δύο πινάκες για να μπορέσει να πάρει αποτελέσματα. Χρειαζόμαστε να πάρουμε για κάθε μάθημα όλες τις ομάδες του, έτσι χρησιμοποιούμε το σύμβολο @(*@ERGASTMATHIMAID*) με το οποίο δηλώνουμε ότι θέλουμε όλες τις συσχετίσεις που μπορεί να υπάρχουν ανάμεσα στους δύο πίνακες, και γεμίζουμε την *clsErgastiriakoMathimaOmada* .

- **GetEksaminaFoitisis**: δουλεύει ακριβώς όπως και η *GetOmadesErgastiriakwnMathimatwn* όμως αλλάζει το ερώτημα που χρησιμοποιει με αποτέλεσμα να μας επιστρέφει τα εξάμηνα στα οποία βρίσκονται τα εργστηριακα μαθήματα και γεμιζει την *clsEksaminoFoitisis*
- **GetErgastiriakesOmadesFoititi**: δουλεύει ακριβώς όπως και η *GetOmadesErgastiriakwnMathimatwn* όμως αλλάζει το ερώτημα που χρησιμοποιει με αποτέλεσμα να μας επιστρέφει τις εργαστηριακες ομαδες και τους φοιτητές που βρίσκονται σε αυτές και γεμίζει την clsErgastiriakiOmadaFoititis.
- **GetFoititisByID**: δουλεύει ακριβώς όπως και η *GetOmadesErgastiriakwnMathimatwn* όμως αλλάζει το ερώτημα που χρησιμοποιει με αποτέλεσμα να μας επιστρέφει τους φοιτητές που υπάρχουν και γεμιζει την clsFoititis.
- **GetOccupiedStudentsByErgastiriakiOmada**: δουλεύει ακριβώς όπως και η *GetOmadesErgastiriakwnMathimatwn* όμως αλλάζει το ερώτημα που χρησιμοποιει με αποτέλεσμα να μας επιστρέφει τους φοιτητες που είναι γραμένοι σε κάποια ομάδα και τους αποθηκεύει στην μεταβλητη *foititesErgastiriakisOmadas*. Η λειτουργία αυτή χρησιμοποιείται από την υποκλάση *RefreshOccupiedFoitites*, η οποία της ορίζει ποιας ομάδας τους φοιτητές να επιστρέψει.
- **GetAvialableStudentsByErgastiriakiMathima**: δουλεύει ακριβώς όπως και η *GetOmadesErgastiriakwnMathimatwn* όμως αλλάζει το ερώτημα που χρησιμοποιεί με αποτέλεσμα να μας επιστρέφει τους φοιτητές που δεν είναι γραμμένοι σε κάποιο μάθημα και τους αποθηκεύει στην μεταβλητή *foititesErgastiriakisOmadas*. Η λειτουργία αυτή χρησιμοποιείται από την υποκλάση *RefreshAvailiableFoitites*, η οποία της ορίζει ποιου μαθήματος τους φοιτητές να επιστρέψει.

Μέθοδος

- **Save**: ενεργοποιείται από την υποκλάση cmdSave_Click όταν πατήσουμε το κουμπί Αποθήκευση. Χρησιμοποιεί τέσσερεις μεταβλητές: την sqlString που περιέχει ένα ερώτημα το οποίο διαγράφει από τον πίνακα Ergastiriaki_Omada_Foititis τις εγγραφές που σχετίζονται με κάποια εργαστηριακή ομάδα που την παίρνει από την υποκλάση cmdSave_Click, την command, την con, και trans που είναι απαραίτητη για να εφαρμοστούν οι αλλαγές που θέλουμε. Έπειτα κάνουμε την σύνδεση στη βάση ενεργοποιούμε στις command και trans και για κάθε φοιτητή που περιέχεται στην foititesErgastiriakisOmadas εισάγουμε στον πίνακα

Ergastiriaki_Omada_Foititis με την βοήθεια ενός ερωτήματος όλες τις απαραίτητες πληροφορίες.

Χαρτογράφηση

Η χαρτογράφηση της βάσης δεδομένων επειδή έγινε χειροκίνητα φτιάξαμε μόνο τις έξι υποκλάσεις που μας ήταν απαραίτητες. Οι υποκλάσεις που αναπαριστούν τους πίνακες της βάσης μας είναι σχεδόν ίδιες. Οι διαφορές τους είναι οι διαφορετικές μεταβλητές που έχουν καθώς αναπαριστούν και διαφορετικούς πίνακες. Άλλη μια διαφορά τους είναι ότι δεν έχουμε δημιουργήσει συσχετισμούς(References) σε όλες παρά μόνο στις απολύτως απαραίτητες. Με αυτό τον τρόπο διαφοροποιούνται οι: clsFoititis που την έχουμε συνδέσει με την clsMyCollection, η οποία είναι μια έτοιμη υποκλάση που χρησιμοποιούμε και περιέχει κάποιους απαραίτητους κατασκευαστές (Constructors) και η clsErgastiriakiOmadaFoititis που την έχουμε συνδέσει με την clsFoititis.

Περιγραφή του πειράματος

Όταν γίνει εκκίνηση της εφαρμογής πρώτα ενεργοποιείται η μέθοδος clsDatabaseInterface (βαθμίδα εφαρμογής) και μετά η λειτουργία GetAkadimaikaEksamina η οποία είναι υπεύθυνη να γεμίσει την κλάση clsAkadimaikoEksamino. Για να μπορέσει να το κάνει αυτό καλεί την λειτουργία GetConnection και με την χρήση ενός sql ερωτήματος επιστρέφει από την βάση δεδομένων όλα τα ακαδημαϊκά εξάμηνα. Όταν επιστραφούν τα δεδομένα ενεργοποιείται από την κλάση frmSelectLabGroupStudents η μέθοδος InitializeComboBox1 και μέσω του αντικειμένου dbInterface (το οποίο έχουμε φτιάξει για να ενώσουμε την υποκλάση clsDatabaseInterface και την κλάση frmSelectLabGroupStudents) δέχεται τα δεδομένα που χρειάζονται για το πεδίο Ακαδημαϊκό *εξάμηνο.*

Μετά ενεργοποιείται η μέθοδος frmSelectLabGroupStudents_Load η οποία καλεί την InitializeComboBox1 ώστε να εμφανιστούν τα δεδομένα της στην φόρμα και μετά καλεί την InitializeComboBox2, επειδή όμως δεν περιέχει τίποτα ενεργοποιείται από την υποκλάση clsDatabaseInterface η λειτουργία GetConnection και μετά η GetErgastiriakaMathimata η οποία επιστρέφει με τη χρήση ενός ερωτήματος όλα τα εργαστηριακά μαθήματα στην υποκλάση clsErgastiriakoMathima. Έπειτα ενεργοποιείται η μέθοδος InitializeComboBox2 όπου δέχεται τα απαραίτητα δεδομένα για το πεδίο Εργαστηριακό Μάθημα. Επειδή η frmSelectLabGroupStudents_Load ενεργοποίησε την

InitializeComboBox2 το περιεχόμενο του δεύτερου πεδίου άλλαξε από άδειο και περιέχει δεδομένα. Η μέθοδος ComboBox2_SelectedIndexChanged ελέγχει μήπως γίνουν αλλαγές στο περιεχόμενο του δεύτερου πεδίου, τώρα που έγιναν, ενεργοποιεί την μέθοδο InitializeComboBox3.

Λόγω του ότι η InitializeComboBox3 είναι άδεια, ενεργοποιείται πάλι η GetConnection και χρησιμοποιείται από την λειτουργία GetOmadesErgastiriakwnMathimatwn η οποία επιστρέφει στην υποκλάση *clsErgastiriakoMathimaOmada* όλες τις ομάδες των εργαστηριακών μαθημάτων. Επειδή το τρίτο πεδίο της φόρμας περιέχει δεδομένα, ενεργοποιείται η μέθοδος ComboBox3_SelectedIndexChanged η οποία ελέγχει για τυχόν αλλαγές στο περιεχόμενό του και επειδή άλλαξε ενεργοποιεί την RefreshAvailiableFoitites η οποία γεμίζει με δεδομένα το πέμπτο πεδίο. Έτσι ενεργοποιείται πάλι η GetConnection που τώρα την χρησιμοποιεί η λειτουργία GetAvialableStudentsByErgastiriakiMathima η οποία ελέγχει ποιοι φοιτητές δεν είναι γραμμένοι σε καμία από τις εργαστηριακές ομάδες των μαθημάτων και τους αποθηκεύει στην ArrayList *foititesErgastiriakisOmadas.* Έπειτα ενεργοποιείται η μέθοδος *RefreshAvailiableFoitites* η οποία γεμίζει το πεδίο *Διαθέσιμοι Φοιτητές* με τους φοιτητές που δεν είναι γραμμένοι σε καμία από τις ομάδες του επιλεγμένου μαθήματος στο δεύτερο πεδίο.

Η μέθοδος ComboBox3_SelectedIndexChanged ενεργοποιεί και την RefreshOccupiedFoitites όπου και αυτή είναι άδεια, οπότε ενεργοποιείται πάλι η GetConnection που τώρα χρησιμοποιείται από την λειτουργία GetOccupiedStudentsByErgastiriakiOmada η οποία ελέγχει ποιοι φοιτητές είναι γραμμένοι στις εργαστηριακές ομάδες των μαθημάτων και τους αποθηκεύει στην ArrayList *foititesErgastiriakisOmadas.* Για να το κάνει αυτό πρέπει να ελέγξει για κάθε φοιτητή αν ο αριθμός του (ID) περιέχεται στην υποκλάση clsErgastiriakiOmadaFoititis η οποία αντιπροσωπεύει ένα πίνακα ο οποίος περιέχει τους φοιτητές και τις ομάδες στις οποίες έχουν γραφτεί. Οπότε για κάθε φοιτητή καλεί την GetConnection που χρησιμοποιείται από την *GetErgastiriakesOmadesFoititi* η οποία επιστρέφει δεδομένα στην υποκλάση clsErgastiriakiOmadaFoititis. Αφού ελέγξει για κάθε φοιτητή ενεργοποιείται η υποκλάση RefreshOccupiedFoitites και εμφανίζονται στο πεδίο *Καταχωρημένοι Φοιτητές,* οι φοιτητές που έχουν γραφτεί στην επιλεγμένη ομάδα(του τρίτου πεδίου) του επιλεγμένου μαθήματος(του δεύτερου πεδίου).

Η μέθοδος ComboBox2_SelectedIndexChanged ενεργοποιεί και την InitializeComboBox4, επειδή όμως είναι άδεια, ενεργοποιείται η GetConnection που χρησιμοποιείται από την GetEksaminaFoitisis η οποία γεμίζει την υποκλάση *clsEksaminoFoitisis* με τα εξάμηνα στα οποία περιέχονται τα εργαστηριακά μαθήματα. Τώρα η InitializeComboBox4 μπορεί και εμφανίζει στο πεδίο *Εξάμηνο*

Φοίτησης το εξάμηνο στο οποίο βρίσκετε το επιλεγμένο μάθημα του δεύτερου πεδίου.

Σε αυτό το σημείο στην φόρμα μας έχουμε επιλεγμένο ένα μάθημα, μια από τις εργαστηριακές του ομάδες, τους φοιτητές που δεν είναι γραμμένοι σε καμία από τις ομάδες και τους φοιτητές που έχουν γραφτεί στην επιλεγμένη ομάδα. Αν θελήσουμε να επιλέξουμε άλλο μάθημα ή άλλη ομάδα δεν χρειάζεται να ξανασυνδεθούμε στην βάση καθώς έχουμε τα δεδομένα που χρειαζόμαστε στις υποκλάσεις που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων.

Αν θέλουμε να γράψουμε στην ήδη επιλεγμένη ομάδα κάποιον από τους φοιτητές που δεν έχουν γραφτεί, επιλέγουμε τον φοιτητή και πατάμε το κουμπί [>] όπου ενεργοποιεί την μέθοδο *btnInsertStudents_Click* η οποία θα γράψει τον φοιτητή στην ομάδα (θα τον μεταφέρει από το πέμπτο στο έκτο πεδίο) και θα κρατήσει τις αλλαγές προσωρινά στην μνήμη μέχρι να γίνει αποθήκευση ή να τερματίσουμε την εφαρμογή. Αν θελήσουμε να διαγράψουμε κάποιον φοιτητή που είναι ήδη γραμμένος τον επιλέγουμε και πατάμε το κουμπί [<] όπου ενεργοποιεί την μέθοδο *btnRemoveStudents_Click* η οποία θα διαγράψει τον φοιτητή από την ομάδα (θα τον μεταφέρει από το έκτο στο πέμπτο πεδίο) και θα κρατήσει τις αλλαγές προσωρινά στην μνήμη μέχρι να γίνει αποθήκευση ή να τερματίσουμε την εφαρμογή.

Όταν ολοκληρώσουμε τις αλλαγές που θέλουμε να κάνουμε, πατάμε το κουμπί *Αποθήκευση* το οποίο ενεργοποιεί την μέθοδο *cmdSave_Click* η οποία λέει στην μέθοδο *save* μέσω του αντικειμένου *dbInterface* σε ποια ομάδα καναμε αλλαγές χρησιμοποιώντας το *ComboBox3.SelectedValue* που περιέχει τον αριθμο (ID) της ομάδας και *DataGrid2.DataSource* που μέχρι να πατήσουμε το κουμπί *Αποθήκευση* είναι οι προσωρινές αλλαγές που έχουν κρατηθεί στην μνήμη από την *btnInsertStudents_Click.*

Συμπεράσματα

Με αυτό τον τρόπο υλοποίησης της εφαρμογής αντιμετωπίσαμε σημαντικά προβλήματα. Για να γίνει η χαρτογράφηση των πινάκων της βάσης δεδομένων στην εφαρμογή χρειαστήκαμε αρκετό χρόνο καθώς φτιάξαμε ένα αρκετά μεγάλο κομμάτι κώδικα με το χέρι. Το πρόβλημα που έχουμε είναι ότι σε αυτό το κομμάτι χρειάστηκε να γράψουμε πολλές φορές τα ίδια πράγματα με πάρα πολλή προσοχή γιατί η διόρθωσή τους είναι πολλή δύσκολη καθώς περιέχουν ονόματα και μεταβλητές που προέρχονται από την βάση και αν γίνει κάτι λάθος βρίσκεται δύσκολα. Αν η εφαρμογή μας απαιτούσε την χρήση εκατό πινάκων ο χρόνος υλοποίησής της θα ήταν τεράστιος γιατί θα έπρεπε να κάνουμε την χαρτογράφηση με το χέρι. Αυτή η διαδικασία δεν θα έπρεπε να είναι το κύριο μέλημα ενός προγραμματιστή γιατί απαιτεί την σπατάλη χρόνου που κάποιες φορές δεν έχουμε. Θα έπρεπε να υπάρχει κάποιο εργαλείο για να κάνει αυτή τη δουλειά ώστε ο προγραμματιστής να συγκεντρώνεται σε πιο σημαντικά πράγματα.

Άλλο ένα σημαντικό πρόβλημα που συναντήσαμε οφειλόταν στην προσπάθεια μας να ξεφύγουμε από την λογική μιας εφαρμογής που χρησιμοποιεί Ενότητα πίνακα(table module) και να εισάγουμε κάποια στοιχεία της λογικής Μοντέλο τομέα (Domain model). Όλες οι ευκολίες που είχαμε χάθηκαν, δεν χρησιμοποιούμε DataTable οπότε αμέσως ακυρώνεται η χρήση του DataGridView στο οποίο γίνονται οι σημαντικές διεργασίες της εφαρμογής μας.

Αυτή η δυσκολία μας οδήγησε στην χρήση του DataGrid και του CurrencyManager. Ο CurrencyManager αντίθετα με τα πολύπλοκα sql ερωτήματα που η δημιουργία τους απαιτεί υπερβολικά πολύ χρόνο, κάνει την ίδια δουλειά με λιγότερο και πιο κατανοητό κώδικα.

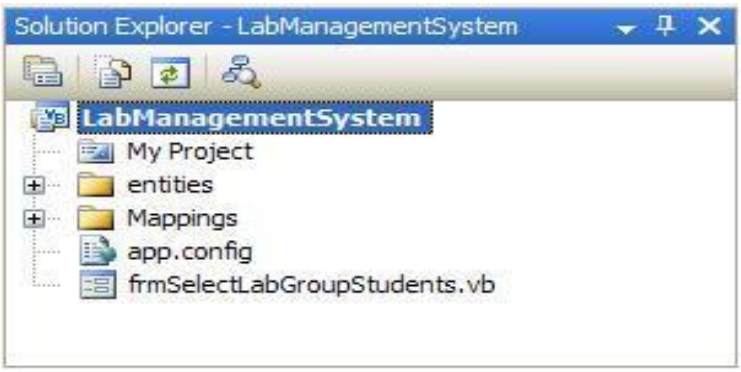
Η εφαρμογή μας χρησιμοποιεί πάλι sql ερωτήματα αλλά δεν είναι το ίδιο πολύπλοκα και δεν βρίσκονται πίσω από σχεδόν κάθε διεργασία, όπως προηγουμένως. Όλα τα πεδία της φόρμας στην προηγούμενη εφαρμογή έπαιρναν τα δεδομένα τους επικοινωνώντας απευθείας με την βάση δεδομένων. Η εφαρμογή τώρα επικοινωνεί με την βάση στην αρχή της λειτουργίας της όπου είναι απαραίτητο για να γεμίζει τους πίνακες της και να μπορεί να πάρει πληροφορίες αργότερα από αυτούς και στο τέλος αν υπάρχουν αλλαγές που πρέπει να δοθούν στη βάση δεδομένων. Η λιγότερη άμεση επικοινωνία με την βάση σημαίνει λιγότερη κατανάλωση μνήμης, λιγότερος χρόνος για να γίνουν κάποιες διεργασίες και λιγοστεύει η πιθανότητα εμφάνισης κάποιου προβλήματος, όπως να χαθεί η επικοινωνία με την βάση.

Με αυτό τον τρόπο υλοποίησης (την εισαγωγή κάποιων στοιχείων της λογικής Μοντέλο τομέα) μπορούμε να γράψουμε πιο κατανοητό κώδικα σε σημεία που πριν δεν μπορούσαμε, τροποποιούμε κάποιο κομμάτι της εφαρμογής μας πιο εύκολα καθώς έχουμε σε κάποια σημαντικά σημεία πολύ πιο απλό και κατανοητό κώδικα και η εφαρμογή μας αποκτά πιο αξιόπιστο κώδικα.

Υλοποίηση τρίτου πειράματος

Ο τρίτος τρόπος υλοποίησης ακολουθεί πιστά την θεωρία μας. Για την υλοποίηση της εφαρμογής μας θα κάνουμε χρήση των στοιχείων του πολυεπίπεδου προγραμματισμού και του NHibernate.

Στον πολυεπίπεδο προγραμματισμό μια εφαρμογή πρέπει να χωρίζεται σε βαθμίδες όπως Βαθμίδα παρουσίασης, Βαθμίδα εφαρμογής και Βαθμίδα δεδομένων. Στην εφαρμογή μας η Βαθμίδα παρουσίασης και Βαθμίδα εφαρμογής συνενώνονται σε μια βαθμίδα η οποία ονομάζεται *frmSelectLabGroupStudents* στην οποία περιέχονται το λειτουργικό περιβάλλον που διαχειρίζεται ο χρήστης αλλά και οι διεργασίες της εφαρμογής. Τη βαθμίδα δεδομένων αποτελεί στην ουσία η βάση δεδομένων ateidb.



Εικόνα 19 **Οι συνενωμένες βαθμίδες παρουσίασης και εφαρμογής μαζί με άλλα στοιχεία της εφαρμογής**

Το NHibernate αποτελεί κομμάτι του Hibernate, οπότε ότι ισχύει για το Hibernate ισχύει άμεσα και για το Nhibernate.

Η αρχιτεκτονική NHibernate αποτελείται από:

- **Διαχείριση Σύνδεσης (Connection Management)** Η υπηρεσία διαχείρισης σύνδεσης παρέχει αποτελεσματική διαχείριση των συνδέσεων βάσης δεδομένων.

```
Public Shared Function OpenSession() As ISession
    If factory Is Nothing Then
        Dim c As New Configuration()
        c.AddAssembly(Assembly.GetCallingAssembly())
        factory = c.BuildSessionFactory()
    End If
    Return factory.OpenSession()

End Function
```

Εικόνα 20 **Η λειτουργία την οποία χρησιμοποιούμε για να συνδεθούμε στην βάση δεδομένων**

- **Διαχείριση Συναλλαγής (Transaction management):**Η υπηρεσία διαχείρισης συναλλαγών παρέχει στο χρήστη τη δυνατότητα να εκτελέσει στη βάση δεδομένων περισσότερες από μία εντολές τη φορά.

```
Private Sub save()

    Dim ergastiriakiOmada As ErgastiriakoMathimaOmada = ComboBox3.SelectedItem
    Dim occupiedStudents As IList = DataGrid2.DataSource
    Dim availableStudents As IList = DataGrid1.DataSource


    Using sess As ISession = OpenSession()
        Using trans As ITransaction = sess.BeginTransaction()
            trans.Commit()
            For Each foitit As Foititis In occupiedStudents

                For Each ergasMathimaFoit In foitit.ErgastiriakiOmadaFoititis
```

Εικόνα 21 **Τμήμα υποκλάσης που κάνει χρήση του** Transaction Management

- **Σχεσιακή χαρτογράφηση Αντικειμένου (Object relational mapping):**Η Σχεσιακή χαρτογράφηση Αντικείμενου είναι η τεχνική της αναπαράστασης των δεδομένων ενός σχεσιακού μοντέλου από ένα μοντέλο αντικειμένων.
- **Μοντέλο τομέα (Domain model)**: μας βοηθάει να δημιουργήσουμε ένα σχέδιο της βάσης δεδομένων στην εφαρμογή μας και να χρησιμοποιήσουμε αντικείμενα για να διαχειριστούμε τους πίνακες της βάσης δεδομένων.

```
Public Sub RefreshAvailiableFoitites()
    Dim ergastiriakoMathima As ErgastiriakoMathima = ComboBox2.SelectedItem
    Dim foitites As IList(Of Foititis) = LoadFoitites()
    Dim availFoitites As IList(Of Foititis) = LoadFoitites()

    For Each ergastiriakiOmada In ergastiriakoMathima.ErgastiriakoMathimaOmada
        For Each ErgastiriakisOmadaFoititis In ergastiriakiOmada.ErgastiriakiOmadaFoititis
            For Each foit In foitites
                Dim contains As Boolean = False
                For Each FoititisErgastiriakisOmada In foit.ErgastiriakiOmadaFoititis
                    If FoititisErgastiriakisOmada.ErgastiriakoMathimaOmada.Equals(ErgastiriakisOmadaFoititis.ErgastiriakoMathimaOmada) Then
                        contains = True
                    End If
                Next

                If contains Then
                    availFoitites.Remove(foit)
                End If
            Next
        Next
    Next
```

Εικόνα 22 Τμήμα υποκλάσης που γεμίζει με δεδομένα ένα πίνακα που έχουμε δημιουργήσει στην εφαρμογή μας και αντιπροσωπεύει πίνακα της βάσης δεδομένων

Περιεχόμενα πειράματος

Σ` αυτό το σημείο θα δούμε αναλυτικότερα τις μεθόδους, τις λειτουργίες και τα υπόλοιπα στοιχεία της εφαρμογής.

Εικόνα 23 **Το** Class Diagram **της τρίτης εφαρμογής με τους χαρτογραφημένους πίνακες της βάσης και τις υποκλάσεις με τις μεθόδους και τα στοιχεία τους**

Λειτουργίες

- **OpenSession**: είναι η λειτουργία που κάνει την σύνδεση με την βάση δεδομένων και την χρησιμοποιούν οι υπόλοιπες υποκλάσεις και λειτουργίες
- **LoadAkadimaikoEksaminoFromDatabase**: στη λειτουργία αυτή ορίζουμε την μεταβλητή X την οποία χρησιμοποιούμε σαν *IList*[10]. Έπειτα καλείτε η *OpenSession* για να γίνει η σύνδεση με την βάση δεδομένων και κάνουμε ένα ερώτημα ώστε να επιστραφεί ο πίνακας *AkadimaikoEksamino* και στο τέλος εξισώνουμε την μεταβλητή *X* με το ερώτημα
- **LoadFoitites**:η λειτουργία αυτή δουλεύει ακριβώς όπως και η προηγούμενη με την διαφορά ότι η μεταβλητή της είναι *S* και το ερώτημα επιστρέφει τον πίνακα *Foititis*.

Μέθοδοι

- **InitializeComboBox1**:αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το πρώτο πεδίο της φόρμας (*Ακαδημαϊκό Εξάμηνο*) το οποίο είναι *ComboBox*. Ορίζουμε την μεταβλητή X την οποία θα χρησιμοποιήσουμε σαν *IList* και την εξισώνουμε με την λειτουργία *LoadAkadimaikoEksaminoFromDatabase.* Με αυτό τον τρόπο η μεταβλητή X περιέχει τα περιεχόμενα του ερωτήματος της λειτουργίας. Έπειτα πρέπει να κάνουμε bind στο *ComboBox* οπότε του ορίζουμε σαν πηγή δεδομένων την μεταβλητή X και του ορίζουμε με την εντολή DisplayMember τι θα εμφανίζει, εμείς του ζητάμε το όνομα του ακαδημαϊκού εξαμήνου.
- **InitializeComboBox2**: αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το δεύτερο πεδίο της φόρμας (*Εργαστηριακό Μάθημα*) το οποίο είναι *ComboBox.* Λόγο της χαρτογράφησης των πινάκων τις βάσης δεδομένων (την οποία θα δούμε πιο κάτω) δεν χρειάζεται να συνδεθούμε πάλι στην βάση. Ορίζουμε την μεταβλητή X την οποία εξισώνουμε με το επιλεγμένο αντικείμενο του πρώτου πεδίου. Έπειτα κάνουμε bind δίνοντας σαν πηγή δεδομένων τον πίνακα *ErgastiriakoMathima* που περιέχεται στην μεταβλητή X, ορίζουμε με την εντολή DisplayMember να εμφανίζεται

[10] Αντιπροσωπεύει συλλογή από αντικείμενα που μπορούν να προσπελαστούν ατομικά μέσω ενός δείκτη(index)

το όνομα του εργαστηριακού μαθήματος και με την εντολή ValueMember κρατάμε στην μνήμη τον αριθμό (ID) του μαθήματος.

- **InitializeComboBox3**: αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το τρίτο πεδίο της φόρμας (*Εργαστηριακή Ομάδα*) το οποίο είναι *ComboBox.* Ορίζουμε την μεταβλητή X την οποία εξισώνουμε με το επιλεγμένο αντικείμενο του δεύτερου πεδίου. Έπειτα κάνουμε bind δίνοντας σαν πηγή δεδομένων τον πίνακα ErgastiriakoMathimaOmada που περιέχεται στην μεταβλητή X, ορίζουμε με την εντολή DisplayMember να εμφανίζεται το όνομα της εργαστηριακής ομάδας και με την εντολή ValueMember κρατάμε στην μνήμη τον αριθμό (ID) της ομάδας.
- **InitializeComboBox4**: αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το τέταρτο πεδίο της φόρμας (Εξάμηνο Φοίτησης) το οποίο είναι ComboBox. Ορίζουμε την μεταβλητή C την οποία εξισώνουμε με το επιλεγμένο αντικείμενο του δεύτερου πεδίου. Ορίζουμε την μεταβλητή X την οποία χρησιμοποιούμε σαν ArrayList και της προσθέτουμε με την εντολή Add τον πίνακα EksaminoFoitisis που περιέχεται στην μεταβλητή C. Έπειτα κάνουμε bind, ορίζουμε σαν πηγή δεδομένων την μεταβλητή X και με την εντολή DisplayMember εμφανίζουμε το όνομα του εξαμήνου φοίτησης.
- **frmSelectLabGroupStudents_Load**: χρησιμοποιείται για να ενεργοποιήσει τις υποκλασεις *InitializeComboBox1 και InitializeComboBox2*
- **ComboBox2_SelectedIndexChanged**: χρησιμοποιείται για να ελέγχει το περιεχόμενο του δεύτερου πεδίου και σε οποιαδήποτε αλλαγη του να ενεργοποιεί τις υποκλάσεις *InitializeComboBox3* και *InitializeComboBox4*
- **ComboBox3_SelectedIndexChanged**: η χρήση της βοηθάει στον έλεγχο του περιεχομένου του τριτου πεδίου και αν συμβεί οποιαδήποτε αλλαγή ενεργοποιεί τις υποκλάσεις *RefreshAvailiableFoitites* και *RefreshOccupiedFoitites.*
- **RefreshAvailiableFoitites**: αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το πέμπτο πεδίο της φόρμας (Διαθέσιμοι Φοιτητές) το οποίο είναι ένα DataGrid. Ορίζουμε τρείς μεταβλητές. Η πρώτη είναι η ergastiriakoMathima που την χρησιμοποιούμε σαν την κλάση ErgastiriakoMathima και την εξισώνουμε με το επιλεγμένο αντικείμενο του δεύτερου πεδίου. Οι επόμενες δυο είναι η foitites και η availFoitites τις οποίες χρησιμοποιούμε σαν IList της κλάσης Foititis και τις εξισώνουμε με την λειτουργία LoadFoitites. Έπειτα ελέγχουμε όλες τις εργαστηριακές ομάδες, που περιέχονται στο επιλεγμένο αντικείμενο του δεύτερου πεδίου, για να βρούμε τους φοιτητές που δεν βρίσκονται σε καμία(ουσιαστικά

ελέγχουμε αν ο πίνακας *ErgastiriakiOmadaFoititis* περιέχει κάποιον φοιτητή γιατί σε αυτόν αποθηκεύονται οι φοιτητές και οι εργαστηριακές ομάδες στις οποίες είναι γραμμένοι). Κάθε φοιτητής που βρίσκετε σε κάποια ομάδα διαγράφεται από την μεταβλητή availFoitites. Μετά δημιουργούμε τη μεταβλητή ts η οποία ουσιαστικά είναι ένα κενό. Για να εμφανιστούν τα δεδομένα πρέπει να κάνουμε bind. Ορίζουμε σαν πηγή δεδομένων του DataGrid την μεταβλητή availFoitites και στην συνέχεια πρέπει να φτιάξουμε τις στήλες του DataGrid ώστε να εμφανίσουμε τα δεδομένα που θέλουμε. Φτιάχνουμε την μεταβλητή cs και ορίζουμε ότι θα είναι *DataGridTextBoxColumn* (στήλη). Δημιουργούμε τρείς στήλες. Στην πρώτη με την εντολή *MappingName* εμφανίζουμε τον αριθμό μητρόου του φοιτητή, με την εντολή *HeaderText* της δίνουμε το όνομα *Α.Μ.* και την προσθέτουμε στην μεταβλητή ts*,* στην δέυτερη στήλη εμφανίζουμε το όνομα του φοιτητη την ονομάζουμε *ΟΝΟΜΑ* και την προσθέτουμε στην μεταβλητή ts, στην τριτη εμφανίζουμε το επώνυμο του φοιτητή την ονομάζουμε *ΕΠΩΝΥΜΟ* και την προσθέτουμε στην μεταβλητή ts*.*

- **RefreshOccupiedFoitites**: αυτή η υποκλάση είναι υπεύθυνη να γεμίσει με δεδομένα το έκτο πεδίο της φόρμας (Καταχωρημένοι Φοιτητές) το οποίο είναι ένα *DataGrid*. Ορίζουμε τρείς μεταβλητές. Η πρώτη είναι η *ergastiriakiOmada* που την χρησιμοποιούμε σαν την κλάση *ErgastiriakoMathimaOmada* και την εξισώνουμε με το επιλεγμένο αντικείμενο του τρίτου πεδίου. Οι επόμενες δυο είναι η *foitites* και η *occupFoitites* τις οποίες χρησιμοποιούμε σαν *IList* της κλάσης *Foititis* και τις εξισώνουμε με την λειτουργία *LoadFoitites*. Έπειτα ελέγχουμε ποιοι από τους φοιτητές περιέχονται στην επιλεγμένη εργαστηριακή ομάδα του τρίτου πεδίου. Όποιος φοιτητής δεν περιέχεται τον αφαιρούμε από την μεταβλητή *occupFoitites* με την εντολή *Remove*. Μετά δημιουργούμε τη μεταβλητή *ts* η οποία ουσιαστικά είναι ένα κενό. Για να εμφανιστούν τα δεδομένα πρέπει να κάνουμε bind. Ορίζουμε σαν πηγή δεδομένων του *DataGrid* την μεταβλητή *availFoitites* και στην συνέχεια πρέπει να φτιάξουμε τις στήλες του *DataGrid* ώστε να εμφανίσουμε τα δεδομένα που θέλουμε. Φτιάχνουμε την μεταβλητή cs και ορίζουμε ότι θα είναι *DataGridTextBoxColumn* (στήλη). Δημιουργούμε τρείς στήλες. Στην πρώτη με την εντολή *MappingName* εμφανίζουμε τον αριθμό μητρόου του φοιτητή, με την εντολή *HeaderText* της δίνουμε το όνομα *Α.Μ.* και την προσθέτουμε στην μεταβλητή ts*,* στην δέυτερη στήλη εμφανίζουμε το όνομα του φοιτητη την ονομάζουμε *ΟΝΟΜΑ* και την προσθέτουμε στην μεταβλητή ts, στην τριτη εμφανίζουμε το επώνυμο του φοιτητή την ονομάζουμε *ΕΠΩΝΥΜΟ* και την προσθέτουμε στην μεταβλητή ts*.*

- **btnInsertStudents_Click**: ενεργοποιείται όταν πατήσουμε το κουμπί > και εγγράφει τον φοιτητή που έχουμε επιλέξει(τον μεταφέρει από το πέμπτο πεδίο στο έκτο). Εδώ ορίζουμε πέντε μεταβλητές. Οι πρώτες δύο είναι: η *cmOccupiedStudents* την οποία χρησιμοποιούμε σαν *CurrencyManager* και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid2 και η cmAvailableStudents την οποία χρησιμοποιούμε σαν *CurrencyManager* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid1*. Οι επόμενες δύο είναι: η *occupiedStudents* την οποία χρησιμοποιούμε σαν *IList* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid2* και η *availableStudents* την οποία χρησιμοποιούμε σαν *IList* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid1*. Η τελευταία είναι η ergastiriakiOmada την οποία χρησιμοποιούμε σαν τη κλάση ErgastiriakoMathimaOmada και την εξισώνουμε με το επιλεγμένο στοιχείο του τρίτου πεδίου. Έπειτα κάνουμε δύο ελέγχους. Με τον πρώτο εξετάζουμε αν το πλήθος των στοιχείων της *availableStudents* είναι μεγαλύτερο από το μηδέν, αν είναι τότε ορίζουμε δύο μεταβλητές: την *foititis* όπου την εξισώνουμε με το επιλεγμένο στοιχείο της *availableStudents* και την θέση του που μας δινεται από την *cmAvailableStudents (availableStudents.Item(cmAvailableStudents. Position))* και την *removeAt* που την χρησιμοποιούμε σαν *Integer* και την εξισώνουμε με την *cmAvailableStudents.Position* που περιέχει τη θέση ενός επιλεγμένου στοιχείου. Με τον δεύτερο έλεγχο εξετάζουμε αν η μεταβλητή *removeAt* περιέχει κάποιο στοιχείο, αν περιέχει το αφαιρούμε από την *cmAvailableStudents.Position*. Στη συνέχεια ορίζουμε τη μεταβλητή *ErgOmadaFoititis* την οποία χρησιμοποιούμε σαν τη κλάση *ErgastiriakiOmadaFoititis* και της προσθέτουμε μια νέα εγγραφή με τα ορίσματα foititis και ergastiriakiOmada. Έπειτα προσθέτουμε στην *occupiedStudents,*με την εντολη Add, την μεταβλητή *foititis* που περιέχει τον φοιτητή που εγγράψαμε, την αφαιρούμε από την *availableStudents* με την εντολή RemoveAt και αρχικοποιούμε τις *cmOccupiedStudents* και *cmAvailableStudents* με την εντολή *Refresh*.
- **btnRemoveStudents_Click**: ενεργοποιείται όταν πατήσουμε το κουμπί < και διαγράφει τον φοιτητή που έχουμε επιλέξει(τον μεταφέρει από το έκτο πεδίο στο πέμπτο). Εδώ ορίζουμε έξι μεταβλητές. Οι πρώτες δύο είναι: η *cmOccupiedStudents* την οποία χρησιμοποιούμε σαν *CurrencyManager* και της ορίζουμε σαν πηγή δεδομένων την πηγή του DataGrid2 και η cmAvailableStudents την οποία χρησιμοποιούμε σαν *CurrencyManager* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid1*. Οι επόμενες δύο είναι: η *occupiedStudents* την οποία

χρησιμοποιούμε σαν *IList* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid2* και η *availableStudents* την οποία χρησιμοποιούμε σαν *IList* και της ορίζουμε σαν πηγή δεδομένων την πηγή του *DataGrid1*. Η πέμπτη είναι η ergastiriakiOmada την οποία χρησιμοποιούμε σαν τη κλάση ErgastiriakoMathimaOmada και την εξισώνουμε με το επιλεγμένο στοιχείο του τρίτου πεδίου. Η έκτη είναι η ErgOmadaFoititis που την χρησιμοποιούμε σαν την κλάση ErgastiriakiOmadaFoititis και την θέτουμε σαν άδεια με την εντολή Nothing. Έπειτα κάνουμε τρείς ελέγχους. Με τον πρώτο εξετάζουμε αν το πλήθος των στοιχείων της *occupiedStudents* είναι μεγαλύτερο από το μηδέν, αν είναι τότε ορίζουμε δύο μεταβλητές: την *foititis* όπου την εξισώνουμε με το επιλεγμένο στοιχείο της *occupiedStudents* και την θέση του που μας δινεται από την *cmOccupiedStudents* (*occupiedStudents*.*Item* (*cmOccupiedStudents*.*Position*)) και την *removeAt* που την χρησιμοποιούμε σαν Integer (ακέραιο) και την εξισώνουμε με την *cmOccupiedStudents*.*Position* που περιέχει τη θέση ενός επιλεγμένου στοιχείου. Με τον δεύτερο έλεγχο εξετάζουμε αν η μεταβλητή *removeAt* περιέχει κάποιο στοιχείο, αν περιέχει το αφαιρούμε από την *cmOccupiedStudents*.*Position*. Με τον τρίτο έλεγχο εξετάζουμε αν ο φοιτητής που έχουμε επιλέξει βρίσκετε στην ergastiriakiOmada και αν βρίσκεται τον προσθέτουμε στην μεταβλητή ErgOmadaFoititis. Έπειτα με την εντολή Remove αφαιρούμε την μεταβλητή ErgOmadaFoititis από την ergastiriakiOmada και με την εντολή Add τον προσθέτουμε στην foititis. Έπειτα αφαιρούμε την *removeAt* από την *occupiedStudents* με την εντολή *RemoveAt*, προσθέτουμε την μεταβλητή *foititis* στην *availableStudents* με την εντολή *Add*, και εκτελούμε την εντολή *Refresh* στις *cmAvailableStudents* και *cmOccupiedStudents.*

- **Save:** ενεργοποιείται όταν πατήσουμε το κουμπί Αποθήκευση. Χρησιμοποιεί την λειτουργία *OpenSession* σε συνδυασμό με δυο δομές επανάληψης και την εντολή SaveOrUpdateCopy και αποθηκεύει στη βάση δεδομένων τις αλλαγές που έχουμε κάνει στην φόρμα.

App.Config

Το αρχείο App.config είναι ένα αρχείο xml το οποίο περιέχει πληροφορίες απαραίτητες για την εφαρμογή μας. Οι πληροφορίες αυτές είναι ότι θα χρησιμοποιήσουμε NHibernate για να διαχειριστούμε την σύνδεση με την βάση δεδομένων, το τύπο της sql διαλεκτού που χρησιμοποιεί η βάση καθώς και πληροφορίες για τον τρόπο σύνδεσης στην βάση και το όνομα της.

Χαρτογράφηση

Στην προηγούμενη εφαρμογή μας κάναμε την χαρτογράφηση των πινάκων με το χέρι εδώ λόγο της χρήσης NHibernate υπάρχει η δυνατότητα να χρησιμοποιήσουμε ένα άλλο πρόγραμμα για να γίνει η χαρτογράφηση. Χρησιμοποιήσαμε το MyGeneration 1,3 το οποίο συνδέσαμε με την βάση δεδομένων επιλέξαμε όλους τους πίνακες της βάσης δεδομένων μας και το πρόγραμμα έκανε την δουλειά μας. Το πρόβλημα με αυτόν τον τρόπο χαρτογράφησης είναι ότι όλα τα προγράμματα που βρήκαμε έβγαζαν αποτελέσματα στην γλώσσα προγραμματισμού C#, γι` αυτό το λόγο χρησιμοποιήσαμε μετά την ιστοσελίδα http://converter.telerik.com/ η οποία είναι ένα πρόγραμμα που μετατρέπει την γλώσσα C# σε VB.Net. Μετά και από αυτό έχουμε όλους τους πίνακες της βάσης σε κλάσεις οι οποίες έχουν και τις απαραίτητες σχέσεις μεταξύ τους καθώς και αρχεία .hbm.xml τα οποία περιέχουν απαραίτητες πληροφορίες για να μπορεί να χρησιμοποιεί το NHibernate τις κλάσεις που δημιουργήθηκαν. Τις κλάσεις καθώς και τα αρχεία .hbm.xml τα εισάγαμε στην εφαρμογή μας σε δυο φακέλους τις κλάσεις στον φάκελο entities και τα αρχεία .hbm.xml στον φάκελο Mappings.

Περιγραφή του πειράματος

Όταν γίνει εκκίνηση της εφαρμογής ενεργοποιείται η κλάση *frmSelectLabGroupStudents_Load* η οποία ενεργοποιεί δυο υποκλάσεις. Πρώτα ενεργοποιεί την υποκλάση *InitializeComboBox1* η οποία γεμίζει με δεδομένα το πρώτο πεδίο της φόρμας (*Ακαδημαϊκό Εξάμηνο*). Για να αποκτήσει τα δεδομένα που χρειάζεται ενεργοποιεί την λειτουργία *LoadAkadimaikoEksaminoFrom Database* η οποία για να συνδεθεί στην βάση δεδομένων ενεργοποιεί με την σειρά της την λειτουργία *OpenSession*. Όταν συνδεθεί στην βάση κάνει ένα ερώτημα για να πάρει όλα τα περιεχόμενα του πίνακα *AkadimaikoEksamino*, λόγο της χρήσης NHibernate αλλά και της χαρτογράφησης που έχουμε κάνει επιστρέφει δεδομένα για όλους τους πίνακες με τους οποίους σχετίζεται. Επιστρέφει στην υποκλάση τα δεδομένα του πίνακα *AkadimaikoEksamino* και αυτή εμφανίζει στην φόρμα το όνομα του ακαδημαϊκού εξαμήνου. Η δεύτερη υποκλάση που ενεργοποιεί είναι η *InitializeComboBox2*.Αύτη η υποκλάση γεμίζει με δεδομένα το δεύτερο πεδίο της φόρμας(*Εργαστηριακό Μάθημα*). Στηριζόμενοι στην βάση δεδομένων γνωρίζουμε ότι ο πίνακας *ErgastiriakoMathima* σχετίζεται με τον πίνακα *AkadimaikoEksamino*. Γι` αυτό το λόγο δεν συνδέεται η *InitializeComboBox2* πάλι με την βάση αλλά ζητάει από τον πίνακα

AkadimaikoEksamino να μας επιστρέψει τον πίνακα *ErgastiriakoMathima* και αφού γίνει αυτό η *InitializeComboBox2* εμφανίζει στην φόρμα τα ονόματα των μαθημάτων.

Λόγο του ότι γέμισε με δεδομένα το δεύτερο πεδίο ενεργοποιείται η υποκλάση *ComboBox2_SelectedIndexChanged* η οποία ελέγχει για τυχόν αλλαγές στο περιεχόμενο του. Αυτή με την σειρά της ενεργοποιεί δυο υποκλάσεις. Η πρώτη υποκλάση που ενεργοποιεί είναι η *InitializeComboBox3* η οποία γεμίζει με δεδομένα το τρίτο πεδίο της φόρμας (*Εργαστηριακή Ομάδα*).Στηριζόμενοι στην βάση δεδομένων γνωρίζουμε ότι ο πίνακας *ErgastiriakoMathima* σχετίζεται με τον πίνακα *ErgastiriakoMathimaOmada* οπότε παίρνουμε από αυτόν τα δεδομένα του πίνακα *ErgastiriakoMathimaOmada* και με βάση το επιλεγμένο μάθημα του δεύτερου πεδίου εμφανίζουμε στην φόρμα το όνομα των εργαστηριακών ομάδων που ανήκουν σε αυτό. Πρώτου ενεργοποιηθεί η δεύτερη υποκλάση ενεργοποιείται η υποκλάση *ComboBox3_SelectedIndexChanged* η οποία ελέγχει για τυχόν αλλαγές στο περιεχόμενο του τρίτου πεδίου. Αυτή η υποκλάση ενεργοποιεί άλλες δυο. Η πρώτη είναι η *RefreshAvailiableFoitites* η οποία γεμίζει δεδομένα το πέμπτο πεδίο της φόρμας(*Διαθέσιμοι Φοιτητές*). Λόγο του σχεδιασμού της βάσης δεδομένων ο πίνακας *Foititis* που περιέχει τους φοιτητές δεν σχετίζεται με τον αρχικό πίνακα. Γι` αυτό το λόγο ενεργοποιείται η λειτουργία *LoadFoitites* η οποία περιέχει ένα ερώτημα που θα επιστρέψει όλους τους φοιτητές. Για να λειτουργήσει όμως ενεργοποιείται η λειτουργία *OpenSession* η οποία κάνει την σύνδεση με τη βάση δεδομένων. Αφού επιστραφούν οι φοιτητές η *RefreshAvailiableFoitites* με βάση το επιλεγμένο μάθημα του δεύτερου πεδίου μας επιστρέφει ποιοι φοιτητές δεν έχουν γραφτεί σε καμία ομάδα του. Η δεύτερη υποκλάση που ενεργοποιείται είναι η *RefreshOccupiedFoitites* η οποία γεμίζει με δεδομένα το έκτο πεδίο της φόρμας (*Καταχωρημένοι Φοιτητές*). Ο τρόπος που παίρνει τους φοιτητές είναι ίδιος με της προηγούμενης υποκλάσης αλλά αυτή χρησιμοποιεί την επιλεγμένη εργαστηριακή ομάδα του τρίτου πεδίου και μας επιστρέφει τους φοιτητές που είναι γραμμένοι σε αυτή. Μετά και από αυτές τις δυο υποκλάσεις ενεργοποιεί η *ComboBox3_SelectedIndexChanged* την δεύτερη υποκλάση της. Η *InitializeComboBox4* γεμίζει με δεδομένα το τέταρτο πεδίο της φόρμας (*Εξάμηνο Φοίτησης*). Στηριζόμενοι στην βάση δεδομένων γνωρίζουμε ότι ο πίνακας *ErgastiriakoMathima* σχετίζεται με τον πίνακα *EksaminoFoitisis* οπότε παίρνουμε από αυτόν τα δεδομένα του πίνακα *EksaminoFoitisis* και με βάση το επιλεγμένο μάθημα του δεύτερου πεδίου εμφανίζουμε στην φόρμα το όνομα του εξαμήνου φοίτησης στο οποίο ανήκει το μάθημα.

Σε αυτό το σημείο στην φόρμα μας έχουμε επιλεγμένο ένα μάθημα μια από τις εργαστηριακές του ομάδες τους φοιτητές που δεν είναι γραμμένοι σε

καμία από τις ομάδες και τους φοιτητές που έχουν γραφτεί στην επιλεγμένη ομάδα. Αν θελήσουμε να επιλέξουμε άλλο μάθημα, ή άλλη ομάδα δεν χρειάζεται να ξανασυνδεθούμε στην βάση καθώς έχουμε τα δεδομένα που χρειαζόμαστε στις κλάσεις που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων.

Αν θέλουμε να γράψουμε στην ήδη επιλεγμένη ομάδα κάποιον από τους φοιτητές που δεν έχουν γραφτεί, επιλέγουμε τον φοιτητή και πατάμε το κουμπί > όπου ενεργοποιεί την υποκλάση *btnInsertStudents_Click* η οποία θα γράψει τον φοιτητή στην ομάδα (θα τον μεταφέρει από το πέμπτο στο έκτο πεδίο) και θα κρατήσει τις αλλαγές προσωρινά στην μνήμη μέχρι να γίνει αποθήκευση ή να τερματίσουμε την εφαρμογή. Αν θελήσουμε να διαγράψουμε κάποιον φοιτητή που είναι ήδη γραμμένος τον επιλέγουμε και πατάμε το κουμπί < όπου ενεργοποιεί την υποκλάση *btnRemoveStudents_Click* η οποία θα διαγράψει τον φοιτητή από την ομάδα (θα τον μεταφέρει από το έκτο στο πέμπτο πεδίο) και θα κρατήσει τις αλλαγές προσωρινά στην μνήμη μέχρι να γίνει αποθήκευση ή να τερματίσουμε την εφαρμογή.

Όταν ολοκληρώσουμε τις αλλαγές που θέλουμε να κάνουμε πατάμε το κουμπί *Αποθήκευση* το οποίο ενεργοποιεί την υποκλάση *cmdSave_Click* η οποία χρησιμοποιώντας την υποκλάση *save* αποθήκευει στην βάση δεδομένων τις αλλαγες που εχουν γίνει στη φόρμα.

Συμπεράσματα

Με αυτό τον τρόπο υλοποίησης αντιμετωπίσαμε προβλήματα τα οποία δεν υπολογίζαμε να έχουμε. Λόγο του ότι το NHibernate είναι τύπου Open Source το Visual Studio που χρησιμοποιούμε για την ανάπτυξη των εφαρμογών μας δεν το περιέχει. Γι` αυτό το λόγο χρειάστηκε να ψάξουμε να το βρούμε στο internet για να το κατεβάσουμε και να το εγκαταστήσουμε στο Visual Studio. Λόγο των λίγων γνώσεων μας στο συγκεκριμένο θέμα δυσκολευτήκαμε αρκετά, σε αντίθεση με τα άλλα πειράματα που ότι χρειαστήκαμε το περιείχε το Visual Studio. Μετά την εγκατάσταση του χρειάστηκε να εισάγουμε και κάποια ακόμα αρχεία (dlls) στην εφαρμογή μας όπου πάλι δεν ξέραμε πια ακριβώς χρειαζόμασταν. Παρότι το internet περιέχει τα πάντα γι` αυτό το θέμα δεν βρήκαμε αρκετή βοήθεια. Κάποιος που γνωρίζει καλά το NHibernate βρίσκει αρκετή βοήθεια αλλά κάποιος αρχάριος όχι. Όποιος εξοικειωθεί με την χρήση του Nhibernate κάνει την ζωή του πιο εύκολη καθώς δημιουργεί πιο καθαρό και κατανοητό κώδικα και κάνει πράγματα, σε αντίθεση με την Ado.NET, με απλούστερο τρόπο.

Για να κάνουμε την χαρτογράφηση των πινάκων της βάσης δεδομένων βρήκαμε αρκετά προγράμματα που κάνουν αυτή τη δουλεία για τον

προγραμματιστή. Το πρόβλημα μας ήταν ότι όλα έβγαζαν αποτελέσματα σε C# πράγμα που μας δείχνει ότι η επιλογή μας να γράψουμε σε VB.NET να ήταν λανθασμένη, καθότι τις εξελίξεις τις ακολουθεί η C#.Για να διορθώσουμε αυτό το πρόβλημα βρήκαμε ένα άλλο πρόγραμμα το οποίο έκανε την μετατροπή από C# σε VB.NET. Παρ` όλα αυτά ο χρόνος που χρειαστήκαμε για να κάνουμε την χαρτογράφηση και την μετατροπή δεν συγκρίνεται με τον χρόνο που θα χρειαζόμασταν για να κάνουμε την χαρτογράφηση με το χέρι.

Με την χρήση του NHibernate για να γεμίσουμε τις κλάσεις που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων καθώς και να πάρουμε αποτελέσματα στην φόρμα μας, χρειάστηκε να κάνουμε μόνο δυο ερωτήματα στην βάση δεδομένων. Με αυτό τον τρόπο μειώνεται κατά πολύ η κατανάλωση μνήμης, μειώνεται ο χρόνος που χρειάζεται για να δουλέψει η εφαρμογή μας και μειώνεται η πιθανότητα εμφάνισης κάποιου απροσδόκητου προβλήματος όπως για παράδειγμα να χαθεί η επικοινωνία με την βάση δεδομένων. Ο λόγος που χρειαστήκαμε δυο ερωτήματα δεν οφείλεται σε κακό προγραμματισμό της εφαρμογής ή στο ίδιο το NHibernate αλλά οφείλεται στην βάση δεδομένων. Το Nhibernate μπορεί και ακολουθεί τις σχέσεις που έχουν μεταξύ τους οι πινάκες και έτσι μπορεί περνώντας από πίνακα σε πίνακα να διαβάσει όλη την βάση δεδομένων σε ένα μόνο ερώτημα. Το μειονέκτημα που έχει αυτό το χαρακτηριστικό του NHibernate είναι ότι ο προγραμματιστής πρέπει να μπει στην λογική με την οποία διαβάζει το NHibernate δεδομένα για να μπορέσει αυτός να εξάγει πληροφορία. Αυτό το πρόβλημα αυξάνει το χρόνο υλοποίησης της εφαρμογής Όταν δεν μπορεί να γίνει αυτό οφείλεται καθαρά στον σχεδιασμό της βάσης. Λόγο του ότι και σε μια πραγματική εφαρμογή η οποία στήνεται με σκοπό να χρησιμοποιήσει μια ήδη υπάρχουσα βάση δεδομένων οι προγραμματιστές δεν πειράζουν την βάση αλλά δουλεύουν γύρω από τα πιθανά προβλήματα της έτσι και εμείς δεν πειράξαμε την δίκη μας βάση δεδομένων.

Παρά τις διευκολύνσεις που μας παρέχει αν εξετάσουμε και τα προβλήματα που αντιμετωπίζουμε δεν υπάρχει κάτι που να κάνει το NHibernate να ξεχωρίζει τόσο από τους άλλους τρόπους υλοποίησης.

Επίλογος

Με βάση την θεωρία μας υλοποιήσαμε τρία πειράματα (δημιουργήσαμε μια εφαρμογή με τρεις διαφορετικούς τρόπους).

- **Στο πρώτο πείραμα κάναμε χρήση Ado.NET και πολυεπίπεδης αρχιτεκτονικής.** Με τον συγκεκριμένο τρόπο υλοποίησης, την χρήση των αρχιτεκτονικών του πολυεπίπεδου προγραμματισμού και ADO.NET δημιουργούμε σχετικά γρήγορα μια εφαρμογή. Αυτό βέβαια οφείλεται στο ότι τα εργαλεία που χρησιμοποιούμε έχουν μεγάλη συμβατότητα μεταξύ τους. Δημιουργούμε πάρα πολλά sql ερωτήματα που στην πλειοψηφία τους είναι αρκετά πολύπλοκα. Τόσο η υλοποίηση τους αλλά και η κατανόηση τους σε κάποια συντήρηση η επέκταση της εφαρμογής θα είναι πολλή δύσκολη. Η εσωτερική λίστα που έχουμε φτιάξει ώστε να κρατάμε τις αλλαγές των δεδομένων μέχρι να αποφασίσουμε να τις αποθηκεύσουμε στην βάση, αν τοποθετηθεί σε μια μεγαλύτερης κλίμακας εφαρμογή θα δεσμεύει πάρα πολύ μνήμη με αποτέλεσμα να είναι η εφαρμογή πιο αργή και λιγότερο αξιόπιστη λόγο του ότι υπάρχει και ο κίνδυνος να “μπουχτίσει” η λίστα με αποτέλεσμα να σταματήσει η λειτουργία της εφαρμογής. Όλες οι μέθοδοι της εφαρμογής που χρησιμοποιούν κάποιο sql ερώτημα κάνουν, κάθε φόρα που ενεργοποιούνται, σύνδεση με την βάση δεδομένων. Αυτό έχει σαν αποτέλεσμα να υπάρχει και εδώ κίνδυνος να σταματήσει η λειτουργία της εφαρμογής, λόγο των πολλαπλών προσπαθειών της να συνδεθεί στη βάση δεδομένων (μπορεί κάποια φορά να αποτύχει). Με αυτό τον τρόπο υλοποίησης παρ` ότι το πείραμα μας υλοποιήθηκε πολύ γρήγορα βλέπουμε ότι δεν είναι κατάλληλος να χρησιμοποιηθεί για την υλοποίηση μεγάλης κλίμακας εφαρμογών αλλά ταιριάζει περισσότερο σε μικροεφαρμογές οι οποίες χρησιμοποιούνται για απλά πράγματα και δεν τους είναι και τόσο απαραίτητη η χρήση πολυεπίπεδης αρχιτεκτονικής.
- **Στο δεύτερο πείραμα χρησιμοποιήσαμε πολυεπίπεδη αρχιτεκτονική με ένα μείγμα στοιχείων από ADO.NET και Nhibernate(κάναμε χρήση του domain model)**. Με αυτό τον τρόπο υλοποίησης της εφαρμογής αντιμετωπίσαμε σημαντικά προβλήματα. Για να γίνει η χαρτογράφηση των πινάκων της βάσης δεδομένων στην εφαρμογή χρειαστήκαμε αρκετό χρόνο καθώς φτιάξαμε ένα αρκετά μεγάλο κομμάτι κώδικα με το χέρι. Το πρόβλημα που έχουμε είναι ότι σε αυτό το κομμάτι χρειάστηκε να γράψουμε πολλές φορές τα ίδια πράγματα με πάρα πολύ προσοχή γιατί η διόρθωση τους είναι πολύ δύσκολη καθώς περιέχουν ονόματα και

μεταβλητές που προέρχονται από την βάση και αν γίνει κάτι λάθος βρίσκεται δύσκολα. Αν η εφαρμογή μας απαιτούσε την χρήση εκατό πινάκων ο χρόνος υλοποίησης της θα ήταν τεράστιος γιατί θα έπρεπε να κάνουμε την χαρτογράφηση με το χέρι. Αυτή η διαδικασία δεν θα έπρεπε να είναι το κύριο μέλημα ενός προγραμματιστή γιατί απαιτεί την σπάταλη χρόνου που κάποιες φορές δεν έχουμε. Θα έπρεπε να υπάρχει κάποιο εργαλείο για να κάνει αυτή τη δουλειά ώστε ο προγραμματιστής να συγκεντρώνεται σε πιο σημαντικά πράγματα. Άλλο ένα σημαντικό πρόβλημα που συναντήσαμε οφειλόταν στην προσπάθεια μας να ξεφύγουμε από την λογική μιας εφαρμογής που χρησιμοποιεί Ενότητα πίνακα(table module) και να εισάγουμε κάποια στοιχεία της λογικής Μοντέλο τομέα (Domain model). Όλες οι ευκολίες που είχαμε χάθηκαν, δεν χρησιμοποιούμε DataTable οπότε αμέσως ακυρώνεται η χρήση του DataGridView στο οποίο γίνονται οι σημαντικές διεργασίες της εφαρμογής μας. Αυτή η δυσκολία μας οδήγησε στην χρήση του DataGrid και του CurrencyManager. Ο CurrencyManager αντίθετα με τα πολύπλοκα sql ερωτήματα που η δημιουργία τους απαιτεί υπερβολικά πολύ χρόνο, κάνει την ίδια δουλειά με λιγότερο και πιο κατανοητό κώδικα. Η εφαρμογή μας χρησιμοποιεί πάλι sql ερωτήματα αλλά δεν είναι το ίδιο πολύπλοκα και δεν βρίσκονται πίσω από σχεδόν από κάθε διεργασία, όπως προηγουμένως. Όλα τα πεδία της φόρμας στην προηγούμενη εφαρμογή έπαιρναν τα δεδομένα τους επικοινωνώντας απευθείας με την βάση δεδομένων. Η εφαρμογή τώρα επικοινωνεί με την βάση στην αρχή της λειτουργίας της όπου είναι απαραίτητο για να γεμίζει τους πίνακες της και να μπορεί να πάρει πληροφορίες αργότερα από αυτούς και στο τέλος αν υπάρχουν αλλαγές που πρέπει να δοθούν στη βάση δεδομένων. Η λιγότερη άμεση επικοινωνία με την βάση σημαίνει λιγότερη κατανάλωση μνήμης, λιγότερος χρόνος για να γίνουν κάποιες διεργασίες και λιγοστεύει η πιθανότητα εμφάνισης κάποιου προβλήματος όπως να χαθεί η επικοινωνία με την βάση. Με αυτό τον τρόπο υλοποίησης (την εισαγωγή κάποιων στοιχείων της λογικής Μοντέλο τομέα) μπορούμε να γράψουμε πιο κατανοητό κώδικα σε σημεία που πριν δεν μπορούσαμε, τροποποιούμε κάποιο κομμάτι της εφαρμογής μας πιο εύκολα καθώς έχουμε σε κάποια σημαντικά σημεία πολύ πιο απλό και κατανοητό κώδικα και η εφαρμογή μας αποκτά πιο αξιόπιστο κώδικα. Με αυτό τον τρόπο υλοποίησης παρ` ότι το πείραμα μας άργησε να υλοποιηθεί σε σχέση με το προηγούμενο υποστηρίζεται περισσότερο η πολυεπίπεδη αρχιτεκτονική και η χρήση του domain model. μας δείχνει ότι μπορεί να χρησιμοποιηθεί σε αρκετά μεγάλης κλίμακας εφαρμογές αλλά η υλοποίηση απαιτεί αρκετό χρόνο.

- **Στο τρίτο πείραμα κάναμε χρήση NHibernate και πολυεπίπεδης αρχιτεκτονικής.** Με αυτό τον τρόπο υλοποίησης αντιμετωπίσαμε προβλήματα τα οποία δεν υπολογίζαμε να έχουμε. Λόγο του ότι το NHibernate είναι τύπου Open Source το Visual Studio που χρησιμοποιούμε για την ανάπτυξη των εφαρμογών μας δεν το περιέχει. Γι` αυτό το λόγο χρειάστηκε να ψάξουμε να το βρούμε στο internet για να το κατεβάσουμε και να το εγκαταστήσουμε στο Visual Studio. Λόγο των λίγων γνώσεων μας στο συγκεκριμένο θέμα δυσκολευτήκαμε αρκετά, σε αντίθεση με τα άλλα πειράματα που ότι χρειαστήκαμε το περιείχε το Visual Studio. Μετά την εγκατάσταση του χρειάστηκε να εισάγουμε και κάποια ακόμα αρχεία (dlls) στην εφαρμογή μας όπου πάλι δεν ξέραμε πια ακριβώς χρειαζόμασταν. Παρότι το internet περιέχει τα πάντα γι` αυτό το θέμα δεν βρήκαμε αρκετή βοήθεια. Κάποιος που γνωρίζει καλά το NHibernate βρίσκει αρκετή βοήθεια αλλά κάποιος αρχάριος όχι. Όποιος εξοικειωθεί με την χρήση του Nhibernate κάνει την ζωή του πιο εύκολη καθώς δημιουργεί πιο καθαρό και κατανοητό κώδικα και κάνει πράγματα, σε αντίθεση με την Ado.NET, με απλούστερο τρόπο. Για να κάνουμε την χαρτογράφηση των πινάκων της βάσης δεδομένων βρήκαμε αρκετά προγράμματα που κάνουν αυτή τη δουλεία για τον προγραμματιστή. Το πρόβλημα μας ήταν ότι όλα έβγαζαν αποτελέσματα σε C# πράγμα που μας δείχνει ότι η επιλογή μας να γράψουμε σε VB.NET να ήταν λανθασμένη, καθότι τις εξελίξεις τις ακολουθεί η C#.Για να διορθώσουμε αυτό το πρόβλημα βρήκαμε ένα άλλο πρόγραμμα το οποίο έκανε την μετατροπή από C# σε VB.NET. Παρ` όλα αυτά ο χρόνος που χρειαστήκαμε για να κάνουμε την χαρτογράφηση και την μετατροπή δεν συγκρίνεται με τον χρόνο που θα χρειαζόμασταν για να κάνουμε την χαρτογράφηση με το χέρι. Με την χρήση του NHibernate για να γεμίσουμε τις κλάσεις που αντιπροσωπεύουν τους πίνακες της βάσης δεδομένων καθώς και να πάρουμε αποτελέσματα στην φόρμα μας, χρειάστηκε να κάνουμε μόνο δυο ερωτήματα στην βάση δεδομένων. Με αυτό τον τρόπο μειώνεται κατά πολύ η κατανάλωση μνήμης, μειώνεται ο χρόνος που χρειάζεται για να δουλέψει η εφαρμογή μας και μειώνεται η πιθανότητα εμφάνισης κάποιου απροσδόκητου προβλήματος όπως για παράδειγμα να χαθεί η επικοινωνία με την βάση δεδομένων. Ο λόγος που χρειαστήκαμε δυο ερωτήματα δεν οφείλεται σε κακό προγραμματισμό της εφαρμογής ή στο ίδιο το NHibernate αλλά οφείλεται στην βάση δεδομένων. Το Nhibernate μπορεί και ακολουθεί τις σχέσεις που έχουν μεταξύ τους οι πινάκες και έτσι μπορεί περνώντας από πίνακα σε πίνακα να διαβάσει όλη την βάση δεδομένων σε ένα μόνο ερώτημα. Το

μειονέκτημα που έχει αυτό το χαρακτηριστικό του NHibernate είναι ότι ο προγραμματιστής πρέπει να μπει στην λογική με την οποία διαβάζει το NHibernate δεδομένα για να μπορέσει αυτός να εξάγει πληροφορία. Αυτό το πρόβλημα αυξάνει το χρόνο υλοποίησης της εφαρμογής Όταν δεν μπορεί να γίνει αυτό οφείλεται καθαρά στον σχεδιασμό της βάσης. Λόγο του ότι και σε μια πραγματική εφαρμογή η οποία στήνεται με σκοπό να χρησιμοποιήσει μια ήδη υπάρχουσα βάση δεδομένων οι προγραμματιστές δεν πειράζουν την βάση αλλά δουλεύουν γύρω από τα πιθανά προβλήματα της έτσι και εμείς δεν πειράξαμε την δίκη μας βάση δεδομένων. Με αυτό τον τρόπο υλοποίησης το πείραμα μας υλοποιήθηκε σε ικανοποιητικό χρόνο και υποστηρίζει πλήρως πολυεπίπεδη αρχιτεκτονική και είναι κατάλληλο να χρησιμοποιηθεί σε μεγάλης κλίμακας εφαρμογές και ο προγραμματιστής αντιμετωπίζει λιγότερες δυσκολίες κατά την υλοποίηση σε αντίθεση με τον δεύτερο τρόπο υλοποίησης.

Η άποψη μας είναι πως ο πιο σωστός τρόπος για την υλοποίηση μιας εφαρμογής είναι η χρήση Nhibernate. Τα πλεονεκτήματα που έχει είναι πολλά αλλά και σημαντικότερα από τα μειονεκτήματα του. Επίσης απ` ότι είδαμε μετά την χρήση του, για να μας δώσει το πλήρες των δυνατοτήτων του απαιτεί μια σωστά σχεδιασμένη βάση δεδομένων και πιθανότατα την χρήση της γλώσσας προγραμματισμού C# και όχι της VB,NET.

Με βάση την θεωρία μας οι τρείς αυτοί τρόποι ενώ διαφέρουν μεταξύ τους δεν ξεχωρίζει κάποιος ώστε να είναι στον προγραμματιστή ξεκάθαρο το τι να χρησιμοποιήσει. Η θεωρία σε συνδυασμό με την υλοποίηση των πειραμάτων μας κάνει να πιστεύουμε ότι παρότι το NHibernate χρειάζεται πολύ μεγάλη προσπάθεια για να γίνει κατανοητό από τους αρχάριους, μόλις κάποιος εξοικειωθεί με αυτό είναι η καλύτερη επιλογή για την υλοποίηση μιας εφαρμογής.

Βιβλιογραφία

1. **Chartier, Robert.** 15 Seconds : Application Architecture: An N-Tier Approach. [Online] [Cited: 07 10, 2009.] http://www.15seconds.com/issue/011023.htm.

2. **Sheriff, Paul D.** Building an N-Tier Application in .NET. [Online] [Cited: 07 10, 2009.] http://msdn.microsoft.com/en-us/library/ms973279.aspx.

3. **Wikipedia contributors.** Multitier architecture. [Online] Wikipedia, The Free Encyclopedia. [Cited: 10 6, 2009.] http://en.wikipedia.org/w/index.php?title=Multitier_architecture.

4. **Poremsky, Diane, et al.** Introduction. *Beginning Visual Basic 6 Application Development.* s.l. : Peer Information Inc., 2000.

5. **Petersen, Jeremy.** Benefits of using the n-tiered approach for web applications. [Online] [Cited: 10 6, 2009.] http://www.adobe.com/devnet/coldfusion/articles/ntier.html.

6. **Rose India.** Struts Guide. [Online] [Cited: 11 10, 2009.] http://www.roseindia.net/struts/strutsguide.shtml.

7. **Wikipedia contributors.** Model–view–controller. [Online] Wikipedia, the free encyclopedia. [Cited: 11 10, 2009.] http://en.wikipedia.org/wiki/Model-view-controller.

8. **Dagum, Diego.** MVC VS N-Tiers. *MSDN Architecture Center - Architecture Forums.* [Online] 11 5, 2006. [Cited: 11 10, 2009.] http://social.msdn.microsoft.com/Forums/en-US/architecturegeneral/thread/97562908-f75b-4afb-9906-91a6b19a28bb.

9. **Wikipedia contributors.** ADO.NET. [Online] [Cited: 10 9, 2009.] http://en.wikipedia.org/wiki/ADO.NET.

10. **MSDN Visual Studio Developer Center.** Benefits of ADO.NET. [Online] [Cited: 7 8, 2009.] http://msdn.microsoft.com/en-us/library/3y0bb1zd%28VS.71%29.aspx.

11. **Nagarajan, Ramesh Kumar.** Past, Present, and Future of ADO.NET. [Online] [Cited: 7 8, 2009.] http://www.codeguru.com/csharp/.net/net_data/datagrid/article.php/c7321__4/.

12. **Sandeep, Avh and Yerra, Rajendra Kumar.** Disadvantages of ADO.NET. *DotNetSpider.com Forums.* [Online] [Cited: 7 8, 2009.]

13. **Rose India.** Hibernate Architecture. [Online] [Cited: 7 8, 2009.] http://www.roseindia.net/hibernate/hibernate_architecture.shtml.

14. **Wikipedia contributors.** Hibernate Framework. [Online] Wikipedia, the free encyclopedia. [Cited: 11 10, 2009.] http://el.wikipedia.org/wiki/Hibernate_Framework.

15. **Linwood, Jeff and Minter, Dave.** *Beginning Hibernate: From Novice to Professional.* s.l. : Apress, 2006.

16. **Phutela, Dipti.** *Hibernate Vs JDBC.* [White Paper] s.l. : Mindfire Solutions, 2007.

17. **Kuaté, Pierre Henri, et al.** *NHibernate in Action.* s.l. : Manning Publications, 2009.

18. **NHibernate Reference Documentation.** Architecture. [Online] [Cited: 11 20, 2009.] https://www.hibernate.org/hib_docs/nhibernate/1.2/reference/en/html/architecture.html.

19. **NHibernate.** NHibernate for .NET. [Online] [Cited: 11 20, 2009.] https://www.hibernate.org/343.html.

20. **Wikipedia contributors.** POCO. [Online] Wikipedia, the free encyclopedia. [Cited: 11 20, 2009.] http://en.wikipedia.org/wiki/POCO.

21. **Willems, Bert.** Hello NHibernate. *Dev@Work.* [Online] [Cited: 11 20, 2009.] http://www.devatwork.nl/index.php/2007/06/26/hello-nhibernate/.

22. **Stack Overflow.** Advantages and Disadvantages of NHibernate. [Online] [Cited: 12 5, 2009.] http://stackoverflow.com/questions/1278094/advantages-and-disadvantages-of-nhibernate.

23. **Fowler, Martin, et al.** *Patterns of Enterprise Application Architecture.* s.l. : Addison-Wesley Professional, 2002.