ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αναπαραγωγή της δομής του κώδικα λογισμικών συστημάτων μέσω επεξεργασίας ακολουθιακών διαγραμμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ορέστη Γκόργκα

Επιβλέπων : Κωνσταντίνος Κοντογιάννης
Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Αναπαραγωγή της δομής του κώδικα λογισμικών συστημάτων μέσω επεξεργασίας ακολουθιακών διαγραμμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ορέστη Γκόργκα

Επιβλέπων : Κωνσταντίνος Κοντογιάννης
Αναπλ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2009

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

Αναπαραγωγή της δομής του κώδικα λογισμικών συστημάτων μέσω επεξεργασίας ακολουθιακών διαγραμμάτων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Ορέστη Γκόργκα

Επιβλέπων : Κωνσταντίνος Κοντογιάννης
Αναπληρωτής Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Μαρτίου 2009.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Κωσταντίνος Κοντογιάννης	Ιωάννης Βασιλείου	Τιμολέων Σελλής
Αναπλ.Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2009

Copyright @ Ορέστης Γ. Γκόργκας, 2009.
Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

Περίληψη

Ένα από τα κυριότερα προβλήματα των σύγχρονων συστημάτων λογισμικού είναι ότι, λόγω του μεγέθους και της πολυπλοκότητας τους καθώς και των συνεχών αναβαθμίσεων τους, γίνεται ολοένα και πιο δύσκολο να αναλυθούν και να συντηρηθούν. Συνέπεια αυτού είναι να αυξάνεται το κόστος και ο χρόνος της συντήρησης των συστημάτων και να καθιστούν οποιαδήποτε τέτοια εργασία συντήρησης και αναβάθμισης ασύμφορη. Επίσης, παρατηρείται συχνά το φαινόμενο, οι λειτουργίες που επιτελεί ένα σύστημα λογισμικού να είναι διαφορετικές από αυτές που αρχικά είχαν προδιαγραφεί. Η μοντελοκεντρική προσέγγιση αντιμετωπίζει αφαιρετικά μέσω ενός πλήθους μοντέλων τα συστήματα λογισμικού, με σκοπό να γίνει ευκολότερη η σχεδίαση, η υλοποίηση και η συντήρηση των σύγχρονων συστημάτων λογισμικού.

Στόχος αυτός της εργασίας είναι η ανάπτυξη και υλοποίηση ενός περιβάλλοντος πλαισίου μετασχηματισμών, το οποίο θα εφαρμόζεται στα ακολουθιακά διαγράμματα ενός συστήματος και θα παράγει έναν σκελετό του κώδικα του συστήματος. Ο κώδικας αυτός θα είναι συνεπής ως προς τα αντίστοιχα ακολουθιακά διαγράμματα. Κατά τις φάσεις της σχεδίασης και υλοποίησης, το προτεινόμενο περιβάλλον θα παρέχει στους σχεδιαστές των συστημάτων μία πρώτη μορφή του κώδικα η οποία, με τις απαραίτητες προσθήκες, θα μετασχηματίζεται σε ολοκληρωμένο σύστημα. Το περιβάλλον μπορεί να χρησιμοποιηθεί κατά τη φάση της συντήρησης όπου ο σκελετός του κώδικα θα συγκρίνεται με τον πραγματικό κώδικα του συστήματος, θα εντοπίζονται οι ασυνέπειες μεταξύ του κώδικα του συστήματος και των ακολουθιακών διαγραμμάτων και έπειτα από μια ενημέρωση των ακολουθιακών διαγραμμάτων θα εντοπίζονται τα σημεία που πρέπει να ενημερωθεί και ο κώδικας και αντίστροφα.

Ο μετασχηματισμός των ακολουθιακών διαγραμμάτων καθώς και η σύγκριση του σκελετού του κώδικα με τον πραγματικό κώδικα επιδεικνύονται με τη χρήση ενός υποδειγματικού συστήματος στο οποίο εφαρμόζονται οι τεχνικές που αναλύονται σε αυτήν την εργασία

«**λέξεις-κλειδιά:** τεχνολογία λογισμικού μοντελοκεντρική αρχιτεκτονική μοντέλα λογισμικού, ακολουθιακά διαγράμματα, μετασχηματισμός διαγραμμάτων, συνέπεια διαγραμμάτων »

Abstract

One of the major problems of the modern software systems is that due to their size, their complexity and the constant upgrades they are subject to, it has become very hard to afford the time,the money and the effort to analyze and maintain them. Additionally, the tasks that a system performs are frequently quite different from those the system is intended to complete. The model-driven approach deals with the software systems through an abstractive view using a variety of models, in order to make the design, the implementation and the maintenance of the software systems easier.

The aim of this project is the design and the development of a transformation, that will be applied to the sequence diagrams of a software system and will produce an abstract framework of the code. This code will be consistent with the sequence diagrams. The code will provide the software engineers, during the face of design and development, with an abstract view of the code which with the necessary insertion of extra code will be transformed into a runnable system. During the phase of maintenance the framework will be compared with actual code to trace inconsistencies between the code and the sequence diagrams. Thus it will be easier ,after an update of the sequence diagrams to spot the areas where the code has to be updated to and vice versa

The transformation of the sequence diagrams and the comparison of the code framework with the actual code are demonstrated through an example system to which the techniques proposed in this paper are applied.

« **Keywords:** software engineering, model driven architecture, software models, sequence diagrams, model transformation, model consistency »

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Κώστα Κοντογιάννη για το ενδιαφέρον του, τις συμβουλές του και τον χρόνο που αφιέρωσε καθ' όλη τη διάρκεια συγγραφής της διπλωματικής μου εργασίας. Επίσης θα ήθελα να ευχαριστήσω θερμά τους καθηγητές κ. Τίμο Σελλή και κ. Γιάννη Βασιλείου για την που μου έκαναν να διατελέσουν μέλη της τριμελής επιτροπής για την εργασία αυτή.

Στο σημείο αυτό θέλω να πω ένα μεγάλο ευχαριστώ στον συμφοιτητή μου και φίλο Βασίλη Κουτσούμπα, για την πολύτιμη βοήθεια, τις παρατηρήσεις και διορθώσεις του πάνω στη διπλωματική μου εργασία.

Με την ευκαιρία αυτή θα ήθελα να ευχαριστήσω από τα βάθη της καρδίας μου, τους γονείς μου Γιώργο και Ουρανία, για την στήριξη τους όλα αυτά τα χρόνια και για την ανεξάντλητη υπομονή τους.

Δεν θα μπορούσα να παραλείψω όλους τους φίλους και συγγενείς μου που με υποστήριξαν κατά τη διάρκεια της φοιτητικής μου ζωής, ο καθένας με τον τρόπο του.

Περιεχόμενα

Περίληψη.....5

Abstract.....7

Ευχαριστίες.....9

Περιεχόμενα.....11

Κατάλογος Σχημάτων.....15

1 Εισαγωγή.....19

1.1 Περιγραφή Προβλήματος.....19

1.1.1 Αυτοματισμός στον συγχρονισμό διαγραμμάτων και κώδικα...20

1.1.2 Εύρεση της δομής ενός προγράμματος από ακολουθιακά διαγράμματα και παραγωγή κώδικα.....21

1.2 Ανάγκη για έρευνα.....22

1.2.1 Υλοποίηση της Μοντελοκεντρικής Αρχιτεκτονικής.....22

1.2.2 Έλλειψη χρησιμοποίησης των συνολικών δυνατοτήτων αναπαράστασης των διαγραμμάτων.....24

1.2.3 Εύρεση ασυνεπειών μεταξύ διαγραμμάτων και υλοποίησης.....25

1.3 Συμβολή της διπλωματικής εργασίας.....26

1.4 Η δομή της διπλωματικής εργασίας.....28

2 Σχετική Έρευνα.....29

2.1 Μοντελοποίηση των συστημάτων.....29

2.1.1 Το πρότυπο Unified Modeling Language.....29

2.1.2 Το πρότυπο Meta Object Facility.....38

2.1.3 Το Eclipse Modeling Framework.....43

2.2 Γλώσσες Μετατροπής Μοντέλων.....45

2.2.1 Atlas Transformation Language.....45

2.2.2 Model Transformation Framework.....47

2.3 Εργαλεία ανοιχτού κώδικά ανάλυσης συστημάτων.....47

2.4 Ανάλυση UML διαγραμμάτων.....49

2.5 Εύρεση ασυνεπειών μεταξύ των διαγραμμάτων.....54

3 Μοντελοποίηση-Σχεδίαση.....57

3.1 Σκοπός της υλοποίησης.....57

3.2 Περιγραφή της διαδικασίας ορισμού μοντέλων.....59

3.3 Θεωρητικό υπόβαθρο της υλοποίησης.....60

3.4 Περιγραφή της αρχιτεκτονικής της υλοποίησης.....64

3.5 Ορισμός του μοντέλου του ακολουθιακού διαγράμματος.....67

3.6 Ορισμός του μοντέλου του κώδικα.....72

3.7 Τεχνική ανάλυσης των ακολουθιακών διαγραμμάτων.....75

3.8 Περίληψη.....93

4 Πρακτική Εφαρμογή.....95

4.1 Η εφαρμογή ATM System.....95

4.2 Εφαρμογή των κανόνων μετασχηματισμού.....98

4.2.1 Απλή περίπτωση.....98

4.2.2 Περίπτωση δομών loop.....110

4.2.3 Περίπτωση δομών alternative.....115

4.2.4 Εύρεση της κληρονομικότητας.....121

4.3 Σύγκριση του παραγόμενου και του πραγματικού κώδικα.....123

4.4 Περίληψη.....135

5 Επίλογος.....137

5.1 Αποτίμηση.....137

5.2 Μελλοντική έρευνα.....139

5.2.1 Μελλοντικές επεκτάσεις.....139

5.2.2 Μελλοντικές κατευθύνσεις.....141

Βιβλιογραφία.....143

Κατάλογος σχημάτων

2.1 Το πλαίσιο ενός ακολουθιακού διαγράμματος.....32

2.2 Παράδειγμα γραμμής ετοιμότητας.....33

2.3 Σύγχρονο και ασύγχρονο μήνυμα 34

2.4 Παράδειγμα ενός απλού ακολουθιακού διαγράμματος.....34

2.5 Παράδειγμα ενός διαγράμματος με alternative.....35

2.6 Παράδειγμα διαγράμματος με χρήση δομής option.....36

2.7 Παράδειγμα διαγράμματος με δομή loop.....37

2.8 Η αρχιτεκτονική του MOF.....39

2.9 Προδιαγραφή κλάσεων MOF.....41

2.10 Προδιαγραφή των σχέσων μεταξύ των κλάσεων.....42

2.11 Συνοπτικό σχήμα παρουσίασης των βασικότερων στοιχείων του CMOF.....42

2.12 Η αρχιτεκτονική του Eclipse Modeling Framework.....43

2.13 Ο πυρήνας του Ecore.....44

2.14 Η διαδικασία μετασχηματισμού μοντέλων.....46

2.15 Μετασχηματισμός μοντέλου σε ATL 46

2.16 Το scenario diagram μεταμοντέλο της επιστημονικής εργασίας των Brian Labiche,Leduc.....52

2.17 Μετασχηματισμός μεταμοντέλων..... 53

3.1 Η διαδικασία ορισμού μοντέλων.....59

3.2 Η αρχιτεκτονική του συστήματος.....64

3.3. Μοντέλο ακολουθιακού διαγράμματος.....68

3.4 Μοντέλο κώδικα.....74

3.5 Ο κανόνας ορισμού των κλάσεων.....76

3.6 Ο κανόνας ορισμού μεθόδων.....77

3.7 Διάγραμμα διαδικασίας επεξεργασίας των μηνυμάτων κλήσης μεθόδων.....78

3.8 Ο κανόνας ορισμού πεδίων.....82

3.9 Ο κανόνας προσδιορισμού του σώματος των μεθόδων.....84

3.10 1ο παράδειγμα ορισμού δομών ελέγχου.....86

3.11 2ο παράδειγμα ορισμού δομών ελέγχου. Ακολουθιακό διάγραμμα 86

3.12 2ο παράδειγμα. Το αντίστοιχο διάγραμμα κώδικα για το διάγραμμα του σχήματος 3.11.....87

3.13 Κανόνας ορισμού των διαπροσωπειών.....89

3.14 Κανόνας συγχώνευσης των διαπροσωπειών.....90

4.1 Διάγραμμα σεναρίων χρήσης του ATM.....96

4.2 Το διάγραμμα κλάσεων του ATM.....97

4.3 Το ακολουθιακό διάγραμμα εκκίνησης συστήματος 99

4.4 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 99

4.5 Το ακολουθιακό διάγραμμα παύσης συστήματος.....100

4.6 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 100

4.7 Το συνεργατικό διάγραμμα της κατάθεσης στο ΑΤΜ.....101

4.8 Το ακολουθιακό διάγραμμα της κατάθεσης στο ΑΤΜ.....102

4.9 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 103

4.10 Το συνεργατικό διάγραμμα της λειτουργίας μεταφοράς χρημάτων του ΑΤΜ.....104

4.11 Το ακολουθιακό διάγραμμα της λειτουργίας μεταφοράς χρημάτων του ΑΤΜ.....105

4.12 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 106

4.13 Το συνεργατικό διάγραμμα της ερώτησης στο ΑΤΜ.....107

4.14 Το ακολουθιακό διάγραμμα της ερώτησης στο ΑΤΜ.....108

4.15 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 109

4.16 Το ακολουθιακό διάγραμμα συνεδρίας πελάτη με το ΑΤΜ.....111

4.17 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 112

4.18 Το συνεργατικό διάγραμμα ανάληψης χρημάτων.....113

4.19 Το ακολουθιακό διάγραμμα ανάληψης χρημάτων.....114

4.20 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 115

4.21 Το ακολουθιακό διάγραμμα συναλλαγής πελάτη με το ATM.....116

4.22 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 117

4.23 Το συνεργατικό διάγραμμα λανθασμένου PIN.....118

4.24 Το ακολουθιακό διάγραμμα λανθασμένου PIN.....119

4.25 Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 119

4.26 Συνολικό διάγραμμα κώδικα.....120

4.27 Η κληρονομικότητα των κλάσεων του ATM όπως προκύπτει από την εφαρμογή του κανόνα B1.....122

4.28 Η κληρονομικότητα των κλάσεων του ATM όπως προκύπτει από την εφαρμογή του κανόνα B3α και B3β.....122

5.1 Μοτίβο υλοποίησης φίλτρου.....142

Κεφάλαιο 1

Εισαγωγή

Τις τελευταίες 2 δεκαετίες η εξέλιξη του υλικού των υπολογιστών έδωσε τη δυνατότητα στους προγραμματιστές να αναπτύσσουν ολοένα μεγαλύτερα και πολυπλοκότερα προγράμματα για να καλύψουν τις ανάγκες του επιστημονικού, επιχειρηματικού και εμπορικού κόσμου σε λογισμικό. Το αυξανόμενο μέγεθος και η πολυπλοκότητα των εφαρμογών οδήγησε από την σύνταξη απλών προγραμμάτων στην σχεδίαση και ανάπτυξη συστημάτων λογισμικού τα οποία δομούνται από μικρότερα υποσυστήματα που επικοινωνούν μεταξύ τους αλλά ταυτόχρονα είναι ανεξάρτητα, κατά το δυνατόν, το ένα από το άλλο.
Μέσα από την υλοποίηση και κυρίως την συντήρηση των συστημάτων λογισμικού, φάνηκε η ανάγκη δημιουργίας ενός συστηματικού τρόπου σχεδίασης και μοντελοποίησης των συστημάτων λογισμικού μέσω του οποίου θα απεικονίζεται η δομή και οι λειτουργίες του συστήματος. Το κενό αυτό προσπαθεί να καλύψει ο οργανισμός Object Management Group (OMG) μέσω της Μοντελοκεντρικής Αρχιτεκτονικής (Model Driven Engineering-MDA). Σύμφωνα με αυτήν την προσέγγιση, το υπό υλοποίηση σύστημα λογισμικού πρέπει αρχικά να σχεδιαστεί και να μοντελοποιηθεί με έναν συστηματικό τρόπο. Η γλώσσα που χρησιμοποιείται για την μοντελοποίηση των λογισμικών συστημάτων είναι η UML (Unified Modeling Language). Η UML παίζει ρόλο ανάλογο με τα αρχιτεκτονικά σχέδια ενός κτηρίου. Δίνει τη δυνατότητα στους μηχανικούς λογισμικού και τους προγραμματιστές να μπορούν να ανταλλάσσουν πληροφορίες με έναν σύντομο,εποπτικό,σαφή και ταυτόχρονα ακριβή τρόπο. Η σημαντικότητα του ρόλου της UML είναι ιδιαίτερα εμφανής στις φάσεις της σχεδίασης και της συντήρησης ενός συστήματος λογισμικού.

Στη φάση της σχεδίασης οι συντελεστές του έργου πρέπει αρχικά να σχεδιάσουν τη δομή και τις λειτουργίες του συστήματος έτσι ώστε κατά τη φάση της υλοποίησης να είναι σαφώς καθορισμένο όχι μόνο ποιες λειτουργικές μονάδες θα υλοποιηθούν αλλά και ο τρόπος που θα υλοποιηθούν. Η UML με την πληθώρα των διαφορετικών διαγραμμάτων

που διαθέτει, δίνει τη δυνατότητα στους σχεδιαστές να απεικονίσουν σαφώς και με κάθε λεπτομέρεια κάθε πτυχή της εφαρμογής.
Κατά τη φάση της συντήρησης οι μηχανικοί λογισμικού και οι προγραμματιστές καλούνται να τροποποιήσουν ένα κομμάτι του συστήματος ελαχιστοποιώντας τις παρενέργειες σε άλλα κομμάτια του συστήματος. Είναι πιθανό οι συντηρητές ενός συστήματος να μη συμμετείχαν στην υλοποίηση του συστήματος ή λόγω μεγάλης χρονικής απόστασης από την υλοποίηση και προηγούμενων εργασιών συντήρησης να μην είναι σε θέση να γνωρίζουν την τρέχουσα δομή του συστήματος. Η μεγάλη ποικιλία των διαγραμμάτων της UML καθώς και οι πολυάριθμες δυνατότητες αυτών των διαγραμμάτων καθιστούν ευκολότερη την κατανόηση της δομής ενός συστήματος απο τους συντηρητές. Είναι όμως αναγκαίο τα σχέδια του συστήματος να είναι συνεχώς ενημερωμένα ώστε να απεικονίζουν την τρέχουσα κατάσταση του συστήματος.

1.1 Περιγραφή Προβλήματος

1.1.1 Αυτοματισμός στο συγχρονισμό διαγραμμάτων και κώδικα

Η μεγάλη εκφραστικότητα που έχει η UML μέσω της πληθώρας των διαγραμμάτων είναι ταυτόχρονα το δυνατό της σημείο αλλά και η αχίλλειος φτέρνα της. Για να είναι χρήσιμα τα διαγράμματα, θα πρέπει ανά πάσα στιγμή να αναπαριστούν την τρέχουσα υλοποίηση, πράγμα που σημαίνει ότι σε κάθε συντήρηση ή αναβάθμιση του συστήματος θα πρέπει να ενημερώνονται και τα διαγράμματα στο σύνολο τους έτσι ώστε να μην έχουν ασυνέπειες ως προς τον κώδικα του συστήματος αλλά και να μην υπάρχουν ασυνέπειες και συγκρούσεις μεταξύ τους. Η εργασία αυτή είναι εξαιρετικά επίπονη και χρονοβόρα λόγω της πληθώρας των διαγραμμάτων και της απαίτησης τα διαγράμματα να είναι συγχρονισμένα μεταξύ τους αλλά και με τον κώδικα.

Εξαιρετικά χρονοβόρα και επίπονη είναι και η αντίστροφη διαδικασία, η παραγωγή δηλαδή κώδικα από τα UML διαγράμματα. Η μεγάλη σπατάλη χρόνου αλλά και κόπου εστιάζεται στην παραγωγή όλων των απαραίτητων κλάσεων αλλά και των αναφορών των κλάσεων μεταξύ τους καθώς και στην κατανόηση της δομής του συστήματος. Ειδικά στην περίπτωση που δεν είναι διαθέσιμα στο σύνολο τους διαγράμματα που περιγράφουν το σύστημα,

είναι εξαιρετικά δύσκολο για τους προγραμματιστές να συλλάβουν τη δομή που το σύστημα πρέπει να έχει. Ακόμα όμως και αν έχουμε την ιδανική περίπτωση και έχουμε στη διάθεση μας όλα τα διαγράμματα τα σύγχρονα λογισμικά συστήματα είναι τόσο πολύπλοκα που η κατανόηση των διαγραμμάτων και η παραγωγή κώδικα σύμφωνα με αυτά μοιάζει με άθλο. Σε όλες αυτές τις δυσκολίες πρέπει επίσης να προσθέσουμε τα λάθη και τις ασυνέπειες λόγω του ανθρώπινου παράγοντα που μοιραία έχουν τα διαγράμματα τόσο πολύπλοκων συστημάτων. Επίσης κατά τη φάση της σχεδίασης και υλοποίησης οι δοκιμαστικές εκδόσεις και οι αλλαγές τόσο στον κώδικα όσο και στα διαγράμματα είναι πολλές και με δεδομένο το μέγεθος των σύγχρονων λογισμικών συστημάτων, τον αυξανόμενο ρυθμό παραγωγής τους και φυσικά την αναγκαιότητα της ορθότητας των διαγραμμάτων και του συγχρονισμού μεταξύ τους και με τον κώδικα, ο αγώνας με τον χρόνο που δίνουν οι προγραμματιστές και οι μηχανικοί λογισμικού είναι άνισος. Είναι λοιπόν βασικό να αυτοματοποιηθεί η παραγωγή κώδικα (ή τουλάχιστον ενός σκελετού του συστήματος) από τα διαγράμματα έτσι ώστε να ελαχιστοποιηθεί η "βρώμικη" δουλειά των προγραμματιστών και να επικεντρωθούν στα κρίσιμα σημεία της υλοποίησης.

1.1.2 Εύρεση της δομής ενός προγράμματος από ακολουθιακά διαγράμματα και παραγωγή κώδικα

Ένα από τα βασικότερα διαγράμματα που χρησιμοποιούνται κατά τη φάση της σχεδίασης αλλά και της συντήρησης, είναι τα ακολουθιακά διαγράμματα (sequence diagrams). Τα διαγράμματα αυτά απεικονίζουν τις κλήσεις μεταξύ των κλάσεων του συστήματος και καταδεικνύουν τη "διαδρομή" μιας ή περισσότερων λειτουργιών του συστήματος ανάμεσα στις ψηφίδες και τις κλάσεις του συστήματος. Η πληροφορία που περιέχουν είναι εξαιρετικά χρήσιμη καθώς απεικονίζουν τον ακριβή τρόπο επικοινωνίας των διαφόρων υποσυστημάτων του κύριου συστήματος. Δυστυχώς όμως είναι όσο μεγάλη είναι η χρησιμότητα τους τόσο δύσκολα είναι στην ανάγνωση, και η δυσκολία ανάγνωσής τους αυξάνεται δυσανάλογα με την πολυπλοκότητα του συστήματος και με την λεπτομέρεια που αυτό αναπαρίσταται στο ακολουθιακό διάγραμμα.

Γεννάται λοιπόν το πρόβλημα υλοποίησης του συστήματος με τέτοιον τρόπο ώστε η αρχιτεκτονική δομή του συστήματος να μην έρχεται σε σύγκρουση με τα ακολουθιακά ή άλλα διαγράμματα και ταυτόχρονα οι λειτουργίες του

συστήματος να υλοποιούνται με τον τρόπο που προδιαγράφεται απο τα ακολουθιακά διαγράμματα. Προκύπτει κατά συνέπεια η ανάγκη της αυτοματοποιημένης ανάγνωσης, επεξεργασίας και σύνθεσης των ακολουθιακών διαγραμμάτων μέσω της οποίας θα παράγεται μια αρχική δομή του συστήματος η οποία θα είναι συνεπής με τα ακολουθιακά διαγράμματα καθώς επίσης και ένας σκελετός του κώδικα του συστήματος ο οποίος θα σκιαγραφεί τις λειτουργίες κάθε μονάδας του συστήματος όπως αυτές έχουν μοντελοποιηθεί από το σύνολο των ακολουθιακών διαγραμμάτων

1.2 Ανάγκη για έρευνα

1.2.1 Υλοποίηση της Μοντελοκεντρικής Αρχιτεκτονικής

Ένας από τους βασικούς στόχους του OMG είναι ο σαφής διαχωρισμός της σχεδίασης από την υλοποίηση του συστήματος. Αυτός είναι και ο βασικός άξονας της φιλοσοφίας της Μοντελοκεντρικής Αρχιτεκτονικής. Πάνω σε αυτόν τον άξονα έχει αναπτυχθεί ένα υπόβαθρο και οι αντίστοιχες τεχνολογίες που στόχο έχουν να προσφέρουν τις εξής δυνατότητες:

- Την περιγραφή ενός συστήματος ανεξάρτητα από την πλατφόρμα υλοποίησης του
- Την προδιαγραφή πλατφορμών υλοποίησης
- Την επιλογή συγκεκριμένης πλατφόρμας υλοποίησης
- Την δυνατότητα μεταφοράς ενός συστήματός από μία πλατφόρμα υλοποίησης σε μία άλλη

Μέσω αυτών των δυνατοτήτων επιτυγχάνονται οι 3 βασικοί στόχοι της Μοντελοκεντρικής Αρχιτεκτονικής οι οποίοι είναι μεταφερσιμότητα,η διαλειτουργικότητα,και η επαναχρησιμοποίηση μέσω του διαχωρισμού των εννοιών του λογισμικού συστήματος.

Όπως περιγράφεται και στο [1] η μεθοδολογία της Μοντελοκεντρικής Αρχιτεκτονικής για την ανάπτυξη λογισμικών συστημάτων αποτελείται από τα εξής βήματα:

1. Την μοντελοποίηση των λειτουργικών απαιτήσεων του συστήματος. Το μοντέλο που παράγεται περιγράφει το σύστημα χωρίς να εξετάζει τη δομή ή τον τρόπο υλοποίησης του. Το μοντέλο αυτό ονομάζεται Computation Independent Model(CIM).

2. Την μοντελοποίηση της λειτουργίας και της δομής του συστήματος ανεξάρτητα από την πλατφόρμα υλοποίησης του συστήματος (Platform Independent Model - PIM).

3. Την περιγραφή των κανόνων μετατροπής του PIM σε ένα μοντέλο συγκεκριμένης πλατφόρμας (Platform Specific Model-PSM).

4. Την παραγωγή του PSM από το PIM και τους κανόνες μετατροπής. Η παραγωγή του PSM περιλαμβάνει και κάποιες σχεδιαστικές αποφάσεις που σχετίζονται με την χρησιμοποιούμενη πλατφόρμα υλοποίησης.

Αυτά τα βήματα εξασφαλίζουν τον διαχωρισμό σχεδίασης και υλοποίησης. Δυστυχώς όμως ο διαχωρισμός αυτός δεν συμβαίνει πάντα στην πράξη και αυτό οφείλεται στο γεγονός ότι πολύ συχνά οι μετατροπές από PIM σε PSM γίνονται χειρωνακτικά και όχι με αυτοματοποιημένο τρόπο. Στην πράξη δηλαδή η ακολουθούμενη διαδικασία δεν διαφέρει και πολύ από τον "παραδοσιακό" τρόπο προγραμματισμού. Στην καλύτερη περίπτωση θα δημιουργηθεί ένα PIM μοντέλο από το οποίο θα προκύψει ένα PSM μοντέλο στο οποίο θα διαπιστωθούν οι σχεδιαστικές αδυναμίες και ελλείψει χρόνου όλες οι μετέπειτα σχεδιαστικές αλλαγές θα γίνουν πάνω στο PSM μοντέλο. Όταν αυτό θα φθάσει στην τελική του μορφή τότε θα ενημερωθεί και το PIM μοντέλο. Στην χειρότερη περίπτωση και σύμφωνα με τον OMG στο [1] πιο συνηθισμένη περίπτωση, ο σχεδιασμός του μοντέλου δεν φθάνει ποτέ σε κάποια απεικόνιση παρά υπάρχει το μοντέλο για κάποια δευτερόλεπτα στο μυαλό του προγραμματιστή και ύστερα χάνεται για πάντα. Για να μπορέσει να γίνει πραγματικότητα η ουσία της Μοντελοκεντρικής Αρχιτεκτονικής θα πρέπει να απαλλαχθούν οι προγραμματιστές από το βάρος της προσαρμογής του PSM μοντέλου από το PIM μοντέλο. Θα πρέπει δηλαδή να αυτοματοποιηθεί ο μετασχηματισμός του PIM σε PSM με βάση συγκεκριμένους κανόνες μετατροπής τους οποίους θα προσφέρει κάθε πλατφόρμα υλοποίησης.

1.2.2 Έλλειψη χρησιμοποίησης των συνολικών δυνατοτήτων αναπαράστασης των διαγραμμάτων.

Ένας απο τους σπουδαιότερους λόγους μη εφαρμογής της μεθόδου της Μοντελοκεντρικής Αρχιτεκτονικής είναι η έλλειψη τεχνολογικής υποστήριξης των σχεδιαστών λογισμικών συστημάτων. Η UML, η γλώσσα προδιαγραφής συστημάτων είναι όπως αναφέρεται στο [2] μια γραφική, εποπτική γλώσσα προδιαγραφής συστημάτων με σκοπό την περιγραφή, κατασκευή, και τη συγγραφή των φύλλων προδιαγραφής μιας εφαρμογής ανεξάρτητα από την πλατφόρμα υλοποίησης. Τα πρώτα εργαλεία που είχαν στη διάθεση τους οι σχεδιαστές λογισμικού αρκούνταν στο να προσφέρουν τη δυνατότητα γραφικής απεικόνισης των διαγραμμάτων της UML. Πρόσφατα όμως αναπτύχθηκαν πολλά εργαλεία τα οποία εκτός από τον σχεδιασμό γραφημάτων έδιναν και τη δυνατότητα αυτοματοποιημένης επεξεργασίας των διαγραμμάτων και παραγωγής κώδικα. Ακόμα όμως και σήμερα που υπάρχει πληθώρα λογισμικών πακέτων σχεδιασμού σε UML (Poseidon,Eclipse, StarUML κλπ) κανένα σχεδόν από αυτά δεν υλοποιεί όλες τις σχεδιαστικές ικανότητες της UML. Χαρακτηριστικό παράδειγμα είναι ότι κανένα σχεδόν πρόγραμμα δεν μπορεί να απεικονίσει βρόγχους (loops) ή ενναλακτικές (alternatives) των ακολουθιακών διαγραμμάτων. Επιπλέον κάθε εργαλείο ακολουθεί το δικό του μοντέλο απεικόνισης, και αποθήκευσης των διαγραμμάτων και συνεπώς είναι δύσκολη η μεταφορά διαγραμμάτων από τη μία πλατφόρμα στην άλλη. Επίσης τα περισσότερα μοντέλα απεικόνισης των διαγραμμάτων εστιάζουν στις πληροφορίες που έχουν σχέση με την παρουσίαση και όχι με την ουσία των διαγραμμάτων και συνεπώς παρουσιάζουν πολύ μεγάλο πλεονασμό πληροφορίας. Είναι αξιοσημείωτο ότι πολλοί ερευνητές όταν χρειάζεται να χρησιμοποιήσουν κάποιο μοντέλο ακολουθιακού διαγράμματος, ορίζουν το δικό τους ανεξάρτητο μοντέλο και συνήθως ορίζουν μόνο εκείνα τα στοιχεία που τους ενδιαφέρουν. Χαρακτηριστικά παραδείγματα αποτελούν τα μοντέλα ακολουθιακών διαγραμμάτων στα [3] και [4]. Είναι επίσης χαρακτηριστικό ότι οι περισσότερες εφαρμογές επιλέγουν για την ανάλυση της δομής ενός συστήματος και την παραγωγή του κώδικα τα διαγράμματα κλάσεων (Class diagrams) τα οποία είναι και τα πιο κοντινά και εύκολα στη μετατροπή σε αντικειμενοστραφείς γλώσσες προγραμματισμού, ενώ αποφεύγουν την επεξεργασία άλλων μορφών διαγραμμάτων,όπως για παράδειγμα το Eclipse Modeling Framework (EMF) όπως αυτό φαίνεται στο [5].

1.2.3 Εύρεση ασυνεπειών μεταξύ διαγραμμάτων και υλοποίησης.

Μια ακόμα σημαντική έλλειψη, που προκύπτει από τις ελλείψεις ανάλυσης των UML διαγραμμάτων, είναι η σύγκριση των διαφόρων UML διαγραμμάτων για την εύρεση ασυνεπειών μεταξύ τους και μεταξύ αυτών και του κώδικα της υλοποίησης. Καθώς τα διαγράμματα αποτελούν μια διαφορετική απεικόνιση του ίδιου συστήματος θα πρέπει να είναι σε συμφωνία μεταξύ τους και να μην καταργεί το ένα διάγραμμα το άλλο. Κατά την σχεδίαση ενός λογισμικού συστήματος είναι εξαιρετικά σημαντικό τα διαγράμματα που παρέχονται στους προγραμματιστές να είναι συνεπή μεταξύ τους έτσι ώστε να είναι ξεκάθαρο όχι μόνο το τι ακριβώς θα πρέπει να υλοποιηθεί αλλά και ο τρόπος που θα γίνει η υλοποίηση. Η εύρεση αυτών των ασυνεπειών είναι αρκετά χρονοβόρα διαδικασία και θα πρέπει να επαναλαμβάνεται συχνά κατά τη διάρκεια επανασχεδίασης του συστήματος αλλά και κατά την διάρκεια της υλοποίησης για την εύρεση ασυνεπειών μεταξύ διαγραμμάτων και κώδικα. Μια αυτοματοποιημένη διαδικασία εύρεσης ασυνεπειών μεταξύ των μοντέλων και του κώδικα θα κατεδείκνυε άμεσα πολλά από τα σχεδιαστικά και προγραμματιστικά λάθη και θα εξοικονομούσε πολύτιμο χρόνο στις ομάδες σχεδιασμού και υλοποίησης ενός συστήματος ενώ ταυτόχρονα θα μπορούσε να βοηθήσει σημαντικά στην αποσφαλμάτωση της υλοποίησης.

Η εύρεση ασυνεπειών μεταξύ των διαγραμμάτων και του κώδικα είναι σημαντική και στη συντήρηση και αναβάθμιση ενός συστήματος καθώς μετά από τις αλλαγές στον κώδικα θα πρέπει να ενημερωθούν και τα διαγράμματα του συστήματος έτσι ώστε να παραμείνουν συνεπή ως προς το σύστημα που μοντελοποιούν. Η αυτοματοποιημένη εύρεση ασυνεπειών θα μπορούσε να καταδείξει σε ελάχιστο χρόνο τα σημεία στα οποία υπάρχουν ασυνέπειες ή και να διορθώσει κάποιες από αυτές. Επίσης θα μπορούσε να εφαρμοστεί και η αντίθετη πορεία να γίνουν δηλαδή οι αλλαγές στα διαγράμματα και να ευρεθούν τα σημεία του κώδικα που πρέπει να αλλαχθούν ώστε να υλοποιηθούν οι αλλαγές που απεικονίζονται στα διαγράμματα.

1.3 Συμβολή διπλωματικής εργασίας

Σκοπός της εργασίας αυτής είναι η συμβολή στην επεξεργασία των ακολουθιακών διαγραμμάτων τα οποία αποτελούν σημαντικό εργαλείο στην ανάλυση των συστημάτων λογισμικού με στόχο την εξαγωγή συμπερασμάτων όσον αφορά τη δομή του συστήματος και την παραγωγή κώδικα με βάση αυτά τα διαγράμματα. Πιο αναλυτικά οι στόχοι της εργασίας είναι οι εξής:

i) Ορισμός ενός μινιμαλιστικού μεταμοντέλου ακολουθιακού διαγράμματος με στόχο την εύκολη αυτοματοποιημένη επεξεργασία του, το οποίο θα μοντελοποιεί τις πληροφορίες που μπορούν να απεικονιστούν σε ένα ακολουθιακό διάγραμμα. Το μοντέλο θα υπακούει σε ευρέως διαδεδομένο πρότυπο ορισμού μεταμοντέλων, όπως για παράδειγμα το MOF, ενώ η απεικόνιση του ακολουθιακού διαγράμματος θα γίνει σε XML αρχείο το οποίο θα υπακούει στο πρότυπο XMI. Στόχος είναι η σχεδίαση και η υλοποίηση ενός μοντέλου ακολουθιακού διαγράμματος το οποίο θα είναι πλήρες, και επαναχρησιμοποιήσιμο από οποιαδήποτε εφαρμογή.

ii) Ορισμός ενός μεταμοντέλου κώδικα το οποίο θα μοντελοποιεί τη δομή του κώδικα του συστήματος παρουσιάζοντας τις κλάσεις του κώδικα, τις μεθόδους και τα πεδία των κλάσεων καθώς επίσης και έναν σκελετό του σώματος κάθε μεθόδου μέσω του οποίου θα απεικονίζονται οι κλήσεις μεθόδων, η δημιουργία αντικειμένων αλλά και η σειρά που αυτά πραγματοποιούνται κατά την εκτέλεση κάθε μεθόδου. Επιπρόσθετα θα μοντελοποιούνται σημαντικά στοιχεία της υλοποίησης όπως βρόγχοι και διακλαδώσεις που τυχόν παρουσιάζονται στα ακολουθιακά διαγράμματα και επηρεάζουν την ροή των κλήσεων των διαφόρων υποσυστήματων και κλάσεων της εφαρμογής. Με αυτόν τον τρόπο σκιαγραφείται η λειτουργία που επιτελεί κάθε μέθοδος, η συνολική λειτουργία κάθε κλάσης καθώς και η επικοινωνία των λειτουργικών μονάδων του συστήματος. Το μεταμοντέλο αυτό αποτελεί έναν συστηματικό τρόπο απεικόνισης του κώδικα ο οποίος είναι ανεξάρτητος από την πλατφόρμα υλοποίησης του. Το μεταμοντέλο αυτό όπως και το μεταμοντέλο ακολουθιακού διαγράμματος θα υπακούει σε ευρέως διαδεδομένο

πρότυπο ορισμού μεταμοντέλων, ενώ η απεικόνιση του διαγράμματος κώδικα θα γίνει σε XML αρχείο το οποίο θα υπακούει στο πρότυπο XMI.

iii) Ορισμός κανόνων μετασχηματισμού των ακολουθιακών διαγραμμάτων σε ένα ή περισσότερα διαγράμματα κώδικα. Οι κανόνες αυτοί θα επεξεργάζονται και θα συνθέτουν τις πληροφορίες ενός συνόλου ακολουθιακών διαγραμμάτων και θα παράγουν ένα ενιαίο διάγραμμα κώδικα το οποίο θα απεικονίζει την δομή του κώδικα του συστήματος ο οποίος θα επιτελεί τις λειτουργίες και τη συμπεριφορά του συστήματος που μοντελοποιούν τα ακολουθιακά διαγράμματα.

iv) Τελικός στόχος θα είναι η σύγκριση μέσω ενός συνόλου κριτηρίων, του παραγόμενου κώδικα από τα ακολουθιακά διαγράμματα με δεδομένο κώδικα ο οποίος θα θεωρείται εν γένει μη συνεπής με τα ακολουθιακά διαγράμματα με απώτερο στόχο την εύρεση ασυνεπειών μεταξύ δεδομένου κώδικα και ακολουθιακών διαγραμμάτων.

Συνοπτικά ο γενικότερος σκοπός αυτής της εργασίας είναι η ανάλυση και επεξεργασία των ακολουθιακών διαγραμμάτων έτσι ώστε να παραχθεί η δομή του συστήματος και ο αντίστοιχος αντικειμενοστραφής κώδικας. Στόχος είναι η συνεισφορά στην σχεδίαση και ανάπτυξη ενός λογισμικού συστήματος παράγοντας αυτοματοποιημένα κώδικα ο οποίος θα είναι συνεπής ώς προς τα ακολουθιακά διαγράμματα αλλά ταυτόχρονα θα περιέχει όσο το δυνατόν περισσότερες πληροφορίες για την δομή και υλοποίηση του συστήματος. Επίσης, η εύρεση ασυνεπειών μεταξύ του κώδικα και των ακολουθιακών διαγραμμάτων στόχο έχει να συνεισφέρει στην ενημέρωση των ακολουθιακών διαγραμμάτων ύστερα από εργασίες συντήρησης του συστήματος ή στον προσδιορισμό συγκεκριμένου σημείου του κώδικα της υλοποίησης όπου πρέπει να γίνει η βελτίωση του κώδικα όπως αυτή έχει διαπιστωθεί από το ακολουθιακό διάγραμμα.

1.4 Η δομή της διπλωματικής εργασίας

Στο κεφάλαιο 2 γίνεται μια παρουσίαση των τεχνικών και των τεχνολογιών που υπάρχουν στο χώρο της ανάλυσης συστημάτων λογισμικού και εστιάζουμε κυρίως στις τεχνικές δυναμικής ανάλυσης. Περιγράφουμε τις τεχνικές και τη σχετική έρευνα που υπάρχει και εφαρμόζεται στην επεξεργασία και την ανάλυση UML διαγραμμάτων που προδιαγράφουν τη δομή και τη λειτουργία των συστημάτων λογισμικού. Επίσης παρουσιάζουμε μια συλλογή από υπάρχοντα εργαλεία και σχετικές έρευνες που σχετίζονται με το συγκεκριμένο κλάδο που πραγματεύεται η εργασία αυτή και κάνουμε μια σύγκριση μεταξύ αυτών και των στόχων αυτής της εργασίας.

Στο κεφάλαιο 3 παρουσιάζουμε αναλυτικά το πρόβλημα της ανάλυσης και επεξεργασίας των ακολουθιακών διαγραμμάτων που προσπαθούμε να επιλύσουμε σε αυτήν την διπλωματική εργασία. Ορίζουμε με μαθηματικό τρόπο το συνολικό πρόβλημα αλλά και τα επιμέρους υποπροβλήματα που το απαρτίζουν. Περιγράφουμε τα μοντέλα δεδομένων που χρησιμοποιούμε αλλά και τους αλγορίθμους που υλοποιούν την μεθοδολογία επίλυσης του προβλήματος που πραγματεύεται η εργασία.

Στο κεφάλαιο 4 παρουσιάζουμε την εφαρμογή της προτεινόμενης μεθόδου ανάλυσης των ακολουθιακών διαγραμμάτων σε ένα σύστημα λογισμικού ανοιχτού κώδικα για το οποίο παρέχονται στα φύλλα προδιαγραφών του και τα ακολουθιακά διαγράμματα που περιγράφουν τη συμπεριφορά και τις λειτουργίες του συστήματος. Από τα ακολουθιακά διαγράμματα παράγουμε έναν σκελετό της δομής και του κώδικα του συστήματος. Στη συνέχεια ορίζουμε ένα σύνολο κριτηρίων σύγκρισης του σκελετού που παράγαμε με τον πραγματικό κώδικα του συστήματος. Με την χρήση αυτών των κριτηρίων εξετάζουμε την αποτελεσματικότητα της μεθόδου που ακολουθήσαμε, εντοπίζουμε τα αδύνατα σημεία της μεθοδολογίας μας και ταυτόχρονα προσδιορίζουμε τα σημεία ασυνέπειας των ακολουθιακών διαγραμμάτων με τον πραγματικό κώδικα του συστήματος.

Στο κεφάλαιο 5 κάνουμε αποτίμηση της δουλειάς που παρουσιάστηκε στην διπλωματική αυτή και εξηγούμε τις πιθανές μελλοντικές ερευνητικές κατευθύνσεις που προκύπτουν από την εργασία αυτή καθώς και τις μελλοντικές επεκτάσεις που επιδέχεται η εργασία αυτή.

Κεφάλαιο 2

Σχετική έρευνα

Πριν την ανάπτυξη του αντικειμένου αυτής της εργασίας κρίνεται σκόπιμο να γίνει μια συνοπτική παρουσίαση του θεωρητικού υποβάθρου και των τεχνολογιών πάνω στα οποία βασίζεται η εργασία αυτή.

2.1 Μοντελοποίηση συστημάτων.

Όπως αναφέρθηκε και πιο πάνω η δραματική αύξηση της πολυπλοκότητας των λογισμικών συστημάτων ώθησε τους σχεδιαστές συστημάτων στη χρήση γλωσσών σχεδιασμού των συστημάτων. Η στροφή του ερευνητικού ενδιαφέροντος προς τις γλώσσες μοντελοποίησης συστημάτων οδήγησε στην δημιουργία μιας πληθώρας τεχνικών και τεχνολογιών μοντελοποίησης τις πιο σημαντικές από τις οποίες αναφέρουμε παρακάτω

2.1.1 Το πρότυπο Unified Modeling Language

Σήμερα, κυρίαρχο ρόλο στη σχεδίαση λογισμικών συστημάτων παίζει η Unified Modeling Language (UML). Η UML είναι μια γραφική γλώσσα μοντελοποίησης γενικού σκοπού η οποία έχει σχεδιαστεί έτσι ώστε να περιγράφει και να προδιαγράφει τη δομή και τη λειτουργία ενός συστήματος για την εκπλήρωση ενός συγκεκριμένου σκοπού.

Για την μοντελοποίηση και την περιγραφή των συστημάτων λογισμικού η UML διαθέτει μια πληθώρα διαγραμμάτων, τα οποία περιγράφουν το σύστημα το καθένα από διαφορετική σκοπιά και προδιαγράφουν συγκεκριμένα χαρακτηριστικά του συστήματος. Πιο αναλυτικά η UML χρησιμοποιείται για να μοντελοποιήσει τις εξής πτυχές ενός λογισμικού συστήματος:

α) Λειτουργικές απαιτήσεις.

- Διαγράμματα χρήσης

Με την χρήση των διαγραμμάτων χρήσης μοντελοποιούνται συνήθως οι λειτουργικές απαιτήσεις του συστήματος διαφόρων σεναρίων αλληλεπίδρασης των χρηστών με το σύστημα. Περιγράφουν δηλαδή τι προβλέπεται να κάνει το σύστημα. Οι έννοιες που χρησιμοποιούνται στα διαγράμματα χρήσης είναι οι δράστες, τα σενάρια χρήσης και το υποκείμενο σύστημα. Το υποκείμενο σύστημα είναι το σύστημα που περιγράφουμε τη συμπεριφορά του σε κάθε σενάριο που εξετάζουμε. Οι δράστες είναι οι χρήστες του συστήματος ή άλλα συστήματα τα οποία είναι εξωτερικά του υποκείμενου συστήματος και αλληλεπιδρούν με αυτό.

β) Στατική δομή του συστήματος.

- Διαγράμματα κλάσεων.
- Διαγράμματα αντικειμένων
- Ψηφιδικά διαγράμματα
- Παραταξιακά διαγράμματα

Με τη χρήση αυτών των διαγραμμάτων απεικονίζουμε στην UML τη στατική δομή του συστήματος και τις σχέσεις που έχουν οι διάφορες δομικές μονάδες μεταξύ τους.

γ) Δυναμική συμπεριφορά του συστήματος

- ακολουθιακά διαγράμματα
- συνεργατικά διαγράμματα

Με τη χρήση αυτών των διαγραμμάτων μοντελοποιούνται τα δυναμικά χαρακτηριστικά του συστήματος. Τα διαγράμματα αυτά απεικονίζουν τον τρόπο επικοινωνίας μεταξύ των διαφόρων δομικών στοιχείων του συστήματος.

δ) Συμπεριφορά συγκεκριμένων αντικειμένων.

- διαγράμματα κατάστασης

Τα διαγράμματα κατάστασης περιγράφουν τις καταστάσεις στις οποίες μπορεί να βρεθεί ένα αντικείμενο σε σχέση με τα μηνύματα που λαμβάνει ή τα γεγονότα που παρατηρεί. Πρόκειται στην ουσία για ένα αυτόματο περιορισμένων ή άπειρων καταστάσεων το οποίο μοντελοποιεί τη συμπεριφορά ενός συγκεκριμένου αντικειμένου.

ε) Στάδια εκτέλεσης μιας διαδικασίας

- διαγράμματα δραστηριότητας

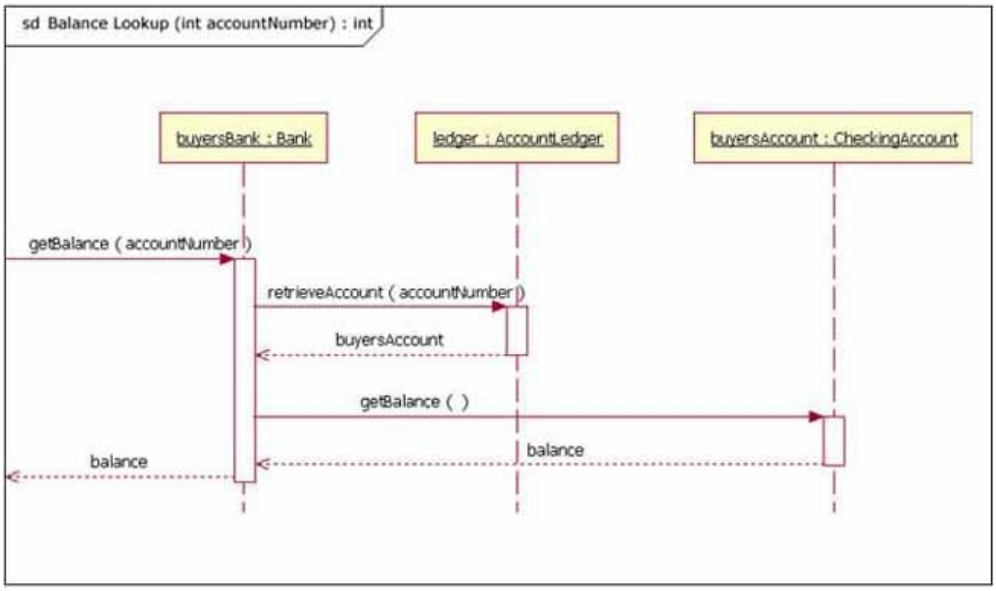
Τα διαγράμματα αυτά μοντελοποιούν στάδια διαδικασιών σε εφαρμογές ανταλλαγής δεδομένων ή σε εφαρμογές που έχουν πολύπλοκη λογική ροή.

Αν και τα UML διαγράμματα περιγράφονται αναλυτικά στα φύλλα προδιαγραφής [2] και [8] της OMG, κρίνεται απαραίτητο να δώσουμε την σημειογραφία των ακολουθιακών διαγραμμάτων η επεξεργασία των οποίων είναι το αντικείμενο της εργασίας αυτής. Τα ακολουθιακά διαγράμματα χρησιμοποιούνται όπως αναφέρεται στο [7] κυρίως για να απεικονίσουν τις αλληλεπιδράσεις μεταξύ των στοιχείων του συστήματος με την χρονική σειρά που αυτές συμβαίνουν. Κατά τη διάρκεια της φάσης της σχεδίασης ενός συστήματος οι μηχανικοί και οι προγραμματιστές χρησιμοποιούν τα ακολουθιακά διαγράμματα για να καθορίσουν και να απεικονίσουν τις αλληλεπιδράσεις ενός αντικειμένου και με αυτόν τον τρόπο να καταστήσουν πιο ολοκληρωμένη την περιγραφή ενός συστήματος.

Μια από τις κυριότερες χρήσεις των ακολουθιακών διαγραμμάτων είναι η μετάβαση από τις λειτουργικές απαιτήσεις, οι οποίες μοντελοποιούνται μέσω των διαγραμμάτων χρήσης, σε ένα επόμενο και πιο φορμαλιστικό επίπεδο προδιαγραφής. Τα σενάρια χρήσης μετατρέπονται σε ένα ή περισσότερα ακολουθιακά διαγράμματα. Εκτός από τη φάση της σχεδίασης τα ακολουθιακά διαγράμματα μπορούν να έχουν και έναν πρόσθετο ρόλο. Μπορούν να χρησιμοποιηθούν ως βοηθητικά δεδομένα για την κατανόηση του τρόπου αλληλεπίδρασης των διαφόρων δομικών στοιχείων ενός ήδη υπάρχοντος συστήματος. Σε αυτήν την περίπτωση τα ακολουθιακά διαγράμματα είναι πολύ χρήσιμα σε περιπτώσεις συντήρησης ενός

λογισμικού συστήματος ή σε περίπτωση μεταφοράς του από ένα πρόσωπο ή οργανισμό σε έναν άλλο.

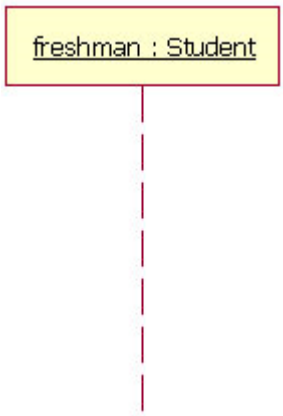
Στην εργασία αυτή θα ακολουθήσουμε τις αρχές της UML 2 όπως αυτές ορίζονται από τον OMG στα κείμενα προδιαγραφών της UML 2 [8] και περιγράφονται στο [9]. Όπως σε κάθε UML διάγραμμα στην UML 2 αρχικά ορίζουμε το πλαίσιο του διαγράμματος. Το πλαίσιο χρησιμεύει πρωταρχικά για να ορίσουμε τον χώρο του διαγράμματος. Επίσης το πλαίσιο χρησιμεύει για να προσφέρει ένα συγκεκριμένο χώρο ονοματολογίας του διαγράμματος. Αν και δεν είναι απαραίτητο το πλαίσιο περιέχει το στοιχείο "ετικέτα" το οποίο είναι το όνομα του διαγράμματος και χρησιμεύει στην αναφορά του διαγράμματος από άλλα διαγράμματα ή κείμενα. Το πλαίσιο έχει έναν επιπλέον σημαντικό ρόλο στα διαγράμματα τα οποία απεικονίζουν αλληλεπιδράσεις, όπως τα ακολουθιακά διαγράμματα. Στα ακολουθιακά διαγράμματα τα εισερχόμενα ή τα εξερχόμενα από το σύστημα μηνύματα μπορούν να συνδεθούν με το πλαίσιο.



Σχήμα 2.1:Το πλαίσιο ενός ακολουθιακού διαγράμματος. Παρατηρούμε το εισερχόμενο στο σύστημα μήνυμα getBalance(accountNumber), καθώς και το εξερχόμενο balance.

Γραμμές ετοιμότητας

Όταν σχεδιάζουμε ένα ακολουθιακό διάγραμμα κάθε αντικείμενο που συμμετέχει στην διαδικασία που μοντελοποιούμε παριστάνεται με μία γραμμή ετοιμότητας. Η γραμμή ετοιμότητας (lifeline) σχεδιάζεται ως ένα παραλληλόγραμμο με μία γραμμή κάθετη στην κάτω βάση του. Το όνομα της γραμμής ετοιμότητας τοποθετείται μέσα στο παραλληλόγραμμο. Η τυπική ονομασία κάθε αντικειμένου σε ένα ακολουθιακό διάγραμμα είναι η εξής: Όνομα Αντικειμένου : Όνομα Κλάσης.

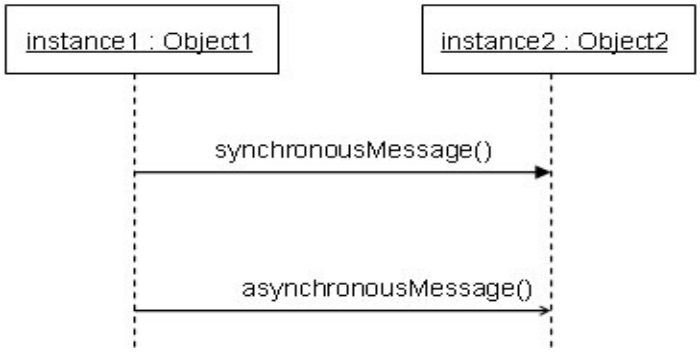


Σχήμα 2.2:Παράδειγμα γραμμής ετοιμότητας

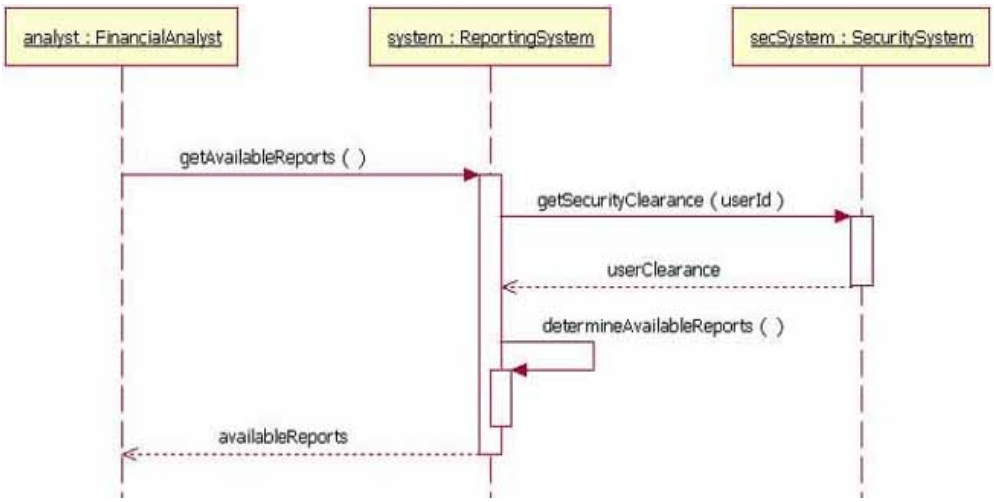
Μηνύματα

Τα μηνύματα στα ακολουθιακά διαγράμματα μοντελοποιούν την επικοινωνία και τις αλληλεπιδράσεις μεταξύ των στοιχείων του συστήματος. Τα μηνύματα μπορούν να αντιπροσωπεύουν κλήση κάποιας συνάρτησης, δημιουργία κάποιου αντικειμένου ή καταστροφή του. Για να δείξουμε ένα αντικείμενο το οποίο στέλνει μήνυμα σε ένα άλλο αντικείμενο σχεδιάζουμε ένα βέλος του οποίου η αρχή είναι στον αποστολέα (source) και το τέλος στον παραλήπτη (target). Το όνομα του μηνύματος ή της μεθόδου που καλείται τοποθετείται πάνω από το βέλος. Στην περίπτωση σύγχρονου μηνύματος η μύτη του βέλους είναι συμπαγής ενώ σε περίπτωση ασύγχρονου μηνύματος η μύτη του βέλους είναι μη συμπαγής. Η απόκριση

ενός αντικείμενου σε μήνυμα παριστάνεται με διακεκομμένη γραμμή.



Σχήμα 2.3:Σύγχρονο και ασύγχρονο μήνυμα.



Σχήμα 2.4: Παράδειγμα ενός απλού ακολουθιακού διαγράμματος.

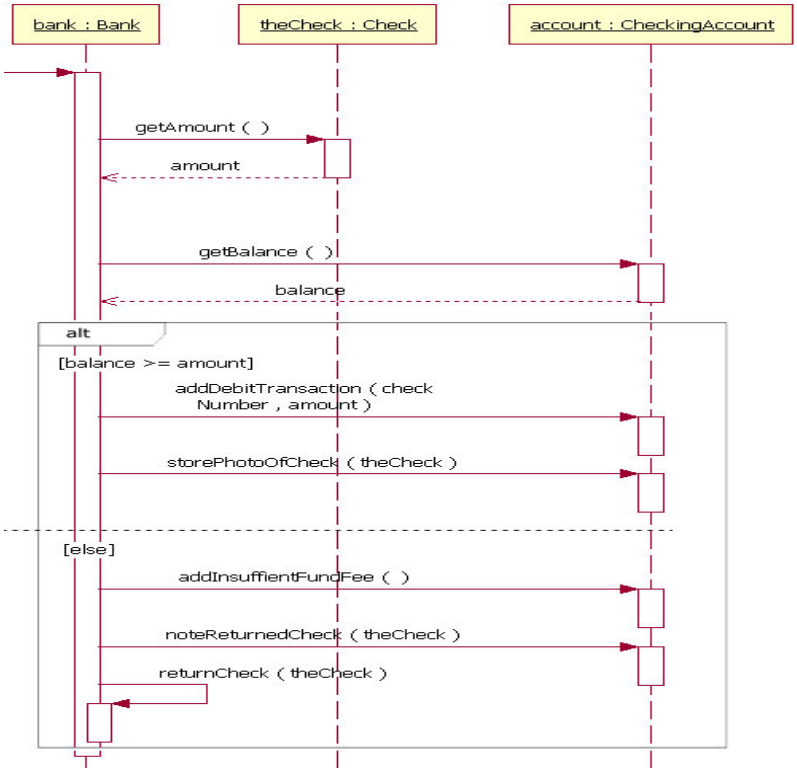
Σύνθετα τμήματα

Όταν μοντελοποιούμε κάποιες αλληλεπιδράσεις μεταξύ αντικειμένων υπάρχουν φορές που κάποια μηνύματα αποστέλλονται μόνο όταν ικανοποιούνται μία ή περισσότερες συνθήκες. Επίσης είναι πιθανό να υπάρχουν βρόγχοι επανάληψης αποστολής και λήψης μηνυμάτων μέχρι να ικανοποιηθεί κάποια συνθήκη. Για την μοντελοποίηση της ροής ελέγχου μιας διαδικασίας χρησιμοποιούνται στην UML 2 τα σύνθετα τμήματα

(combined fragments) τα οποία απεικονίζουν στα διαγράμματα τον έλεγχο ροής.

Εναλλακτικές (alternatives)

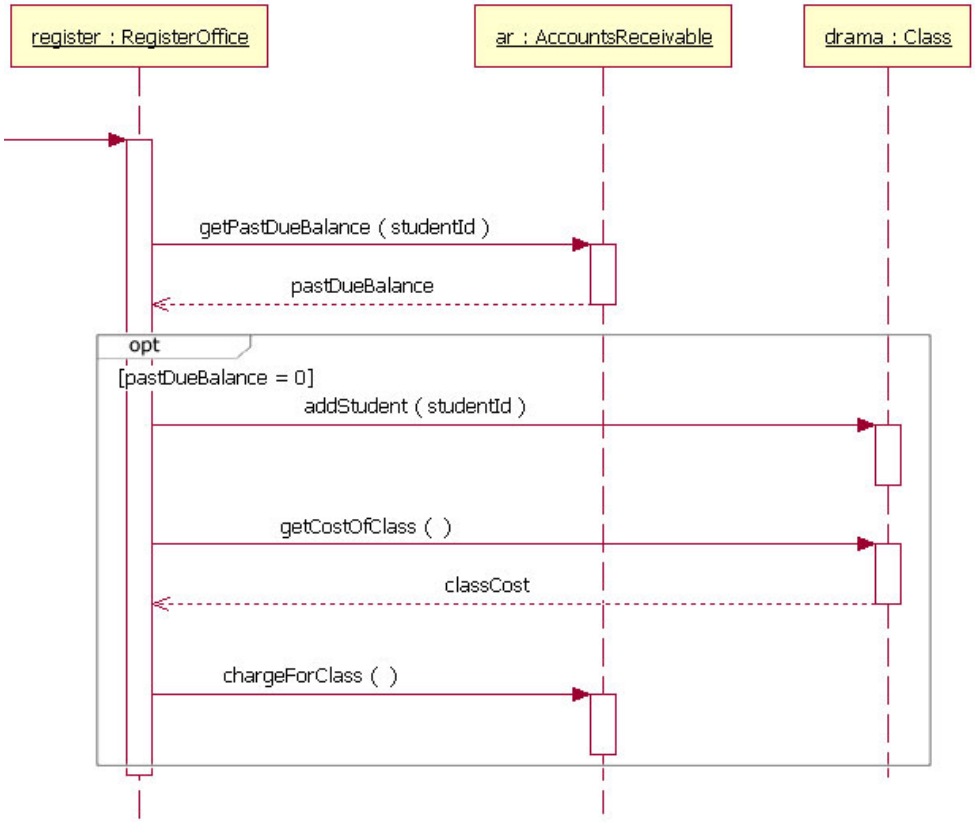
Τα alternatives χρησιμοποιούνται για να μοντελοποιήσουν τη συνήθη δομή if (condition) then ... else. Η δομή του alternative αποτελείται από δύο τμήματα. Τα δύο τμήματα ορίζονται στο διάγραμμα μέσα σε ένα πλαίσιο το οποίο χωρίζεται σε δύο υποπλαίσια. Σαν πρώτο τμήμα, ορίζεται το πρώτο υποπλαίσιο με ετικέτα alt και την συνθήκη σύγκρισης η οποία τοποθετείται πάνω στην γραμμή ετοιμότητας (lifeline) του αντικειμένου με το οποίο σχετίζεται η σύγκριση. Περιέχει την συμπεριφορά του συστήματος αν ικανοποιείται η συνθήκη σύγκρισης. Το δεύτερο τμήμα ορίζεται από το δεύτερο υποπλαίσιο με ετικέτα else και περιέχει την συμπεριφορά του συστήματος όταν δεν ικανοποιείται η συνθήκη.



Σχήμα 2.5: Παράδειγμα διαγράμματος με alternative

Επιλογές (Options)

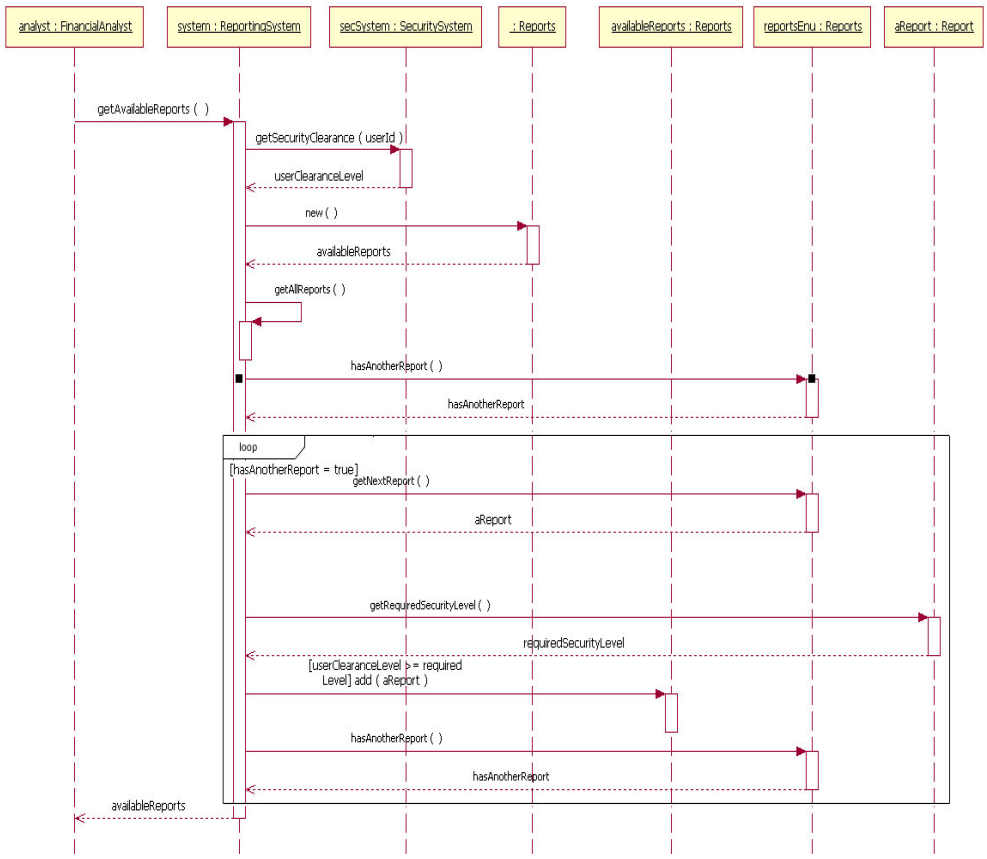
Το option χρησιμοποιείται για να μοντελοποιηθεί μια δομή if (condition) then, χωρίς την ύπαρξη του τμήματος else. Η δομή option ορίζεται από ένα πλαίσιο το οποίο περιέχει την ετικέτα opt και την συνθήκη σύγκρισης η οποία τοποθετείται πάνω στην γραμμή ετοιμότητας (lifeline) του αντικειμένου με το οποίο σχετίζεται η σύγκριση. Το πλαίσιο περιέχει την συμπεριφορά του συστήματος στην περίπτωση που ικανοποιείται η συνθήκη. Αν δεν ικανοποιείται η συνθήκη το πλαίσιο παραλείπεται και θεωρούμε ότι η ροή ελέγχου του συστήματος συνεχίζεται αμέσως μετά την δομή option.



Σχήμα 2.6: Παράδειγμα διαγράμματος με χρήση δομής option.

Βρόγχοι (Loops)

Για να μοντελοποιήσουμε βρόγχους επαναλαμβανόμενων μηνυμάτων χρησιμοποιούμε τη δομή loop. Η δομή του loop είναι παρόμοια με αυτή του option. Και εδώ η δομή loop ορίζεται από ένα πλαίσιο το οποίο έχει την ετικέτα loop. Πάνω στη γραμμή ετοιμότητας του αντικειμένου που σχετίζεται με τον βρόγχο ορίζεται η συνθήκη του βρόγχου. Το πλαίσιο περιέχει τις αλληλεπιδράσεις των αντικειμένων που επαναλαμβάνονται κατά τη διάρκεια του βρόγχου.



Σχήμα 2.7: Παράδειγμα διαγράμματος με δομή loop.

2.1.2 Το πρότυπο Meta Object Facility (MOF)

Το MOF αποτελεί μια προσπάθεια να συστηματοποιηθεί ο τρόπος μοντελοποίησης των συστημάτων λογισμικού. Σύμφωνα με την Μοντελοκεντρική Αρχιτεκτονική που προτείνει ο OMG, κάθε μοντέλο πρέπει να υπακούει σε κάποιο μεταμοντέλο. Το μεταμοντέλο είναι, σύμφωνα με το [10], ένα μοντέλο το οποίο μοντελοποιεί την υλοποίηση ενός άλλου μοντέλου. Το MOF ορίζει σύμφωνα με το [9] μια αφηρημένη γλώσσα και ένα περιβάλλον πλαίσιο για τον ορισμό και την διαχείριση τεχνολογικά ανεξάρτητων μεταμοντέλων. Η προσέγγιση του MOF στοχεύει στην επεκτασιμότητα των δυνατοτήτων των μοντέλων των λογισμικών συστημάτων. Ο σκοπός δηλαδή είναι να παρέχει ένα πλαίσιο το οποίο να υποστηρίζει κάθε είδος μεταδεδομένων και να επιτρέπει νέα είδη μεταδεδομένων να προστίθενται όποτε αυτό είναι αναγκαίο. Για να επιτευχθεί αυτός σκοπός, το MOF βασίζεται σε μια διαστρωματωμένη αρχιτεκτονική 4 επιπέδων. Η αρχιτεκτονική αυτή είναι αρκετά διαδεδομένη σε διαδικασίες και οργανισμούς προτυποποίησης όπως ο ISO και ο CDIF. Το κλειδί αυτής της αρχιτεκτονικής είναι το επίπεδο του μετα-μεταμοντέλου το οποίο συνδέει τα μοντέλα και τα αντίστοιχα σε αυτά μεταμοντέλα.

Η δομή αυτής της αρχιτεκτονικής των 4 επιπέδων είναι η εξής:

Στη βάση της ιεραρχίας των 4 επιπέδων βρίσκεται το επίπεδο των πληροφοριών ή επίπεδο M0. Αποτελείται δηλαδή αυτό το επίπεδο από τις πληροφορίες που θέλουμε να περιγράψουμε.

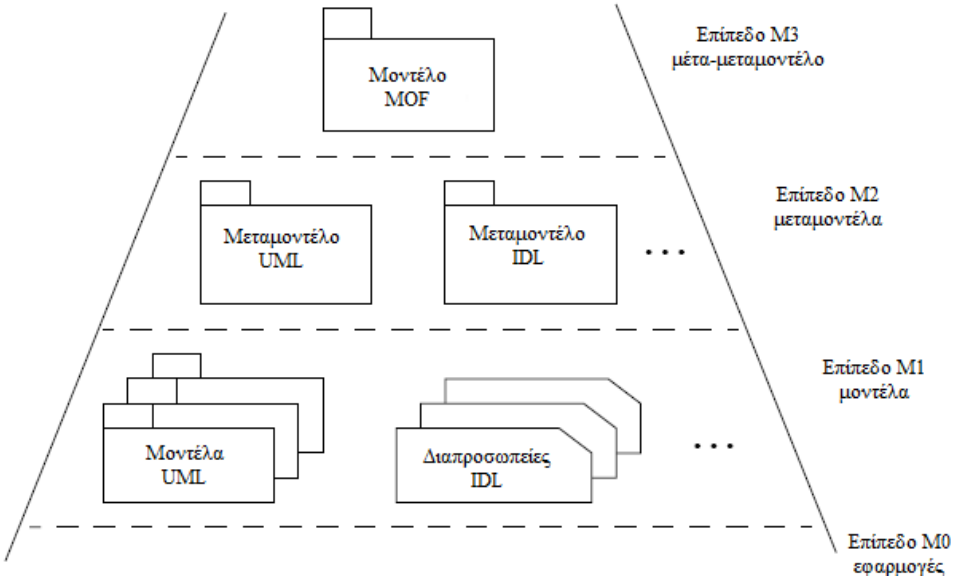
Το αμέσως ανώτερο επίπεδο είναι το επίπεδο του μοντέλου ή M1. Το επίπεδο αυτό αποτελείται από τα μεταδεδομένα τα οποία περιγράφουν τα δεδομένα του επιπέδου πληροφοριών.

Το τρίτο επίπεδο είναι το επίπεδο του μεταμοντέλου ή M2. Το επίπεδο αυτό αποτελείται από την περιγραφή της δομής και της σημασιολογίας των μεταδεδομένων. Πρόκειται δηλαδή για μια συλλογή μέτα-μεταδεδομένων. Η συλλογή αυτή των μέτα-μεταδεδομένων αποτελεί ένα μεταμοντέλο.

Στην κορυφή της ιεραρχίας είναι το επίπεδο του μέτα-μεταμοντέλου ή M3. Αποτελείται από την περιγραφή της δομής και της σημασιολογίας των μέτα-μεταδεδομένων. Με άλλα λόγια είναι μια αφηρημένη γλώσσα ορισμού διαφορετικών ειδών μεταδεδομένων.

Κύριο χαρακτηριστικό αυτής της αρχιτεκτονικής είναι ότι κάθε επίπεδο αποτελεί ένα στιγμιότυπο του αμέσως ανώτερου επιπέδου. Το επίπεδο M3 ορίζεται αναδρομικά από το ίδιο το επίπεδο. Με αυτόν τον τρόπο αποφεύγουμε τον ορισμό άπειρων επιπέδων.

Η αρχιτεκτονική του MOF βασίζεται στην αρχιτεκτονική των 4 επιπέδων που μόλις περιγράψαμε. Το παρακάτω παράδειγμα είναι ένα τυπικό στιγμιότυπο της αρχιτεκτονικής του MOF με μεταμοντέλα που αντιπροσωπεύουν τα UML διαγράμματα και την OMG IDL.



Σχήμα 2.8: Η αρχιτεκτονική του MOF

Η αρχιτεκτονική του MOF παρουσιάζει κάποια σημαντικά πλεονεκτήματα που την κάνουν να υπερέχει από προηγούμενες αρχιτεκτονικές μεταμοντέλων.

• Το μοντέλο MOF είναι αντικειμενοστραφές και οι δομές που χρησιμοποιεί είναι εναρμονισμένες με την UML και τις αντικειμενοστραφείς γλώσσες προγραμματισμού.
• Τα επίπεδα του MOF δεν είναι σταθερά. Αν και τα επίπεδα είναι συνήθως 4, θα μπορούσαν να να είναι περισσότερα ή λιγότερα, ανάλογα με το πως

αναπτύσσεται η μοντελοποίηση. Το MOF δεν απαιτεί να υπάρχουν ξεχωριστά επίπεδα μοντελοποίησης. Τα επίπεδα του MOF είναι μια σύμβαση για την κατανόηση των σχέσεων μεταξύ δεδομένων και μεταδεδομένων.

• Το μοντέλο MOF περιγράφει τον εαυτό του. Μπορούμε να το ορίσουμε πλήρως χρησιμοποιώντας τις δικές του δομές μοντελοποίησης.

Η δομή του MOF

Η τρέχουσα έκδοση του MOF είναι το MOF2. Το MOF2 έχει επηρεάσει και είναι ταυτόχρονα επηρεασμένο από την δομή της UML2 λόγω της θέλησης του OMG να δημιουργηθεί ένας κοινός πυρήνας εννοιών μοντελοποίησης μεταξύ της UML, του MOF, αλλά και των υπολοίπων προτύπων του OMG. Το σημαντικό είναι ότι η UML2 και το MOF2 χρησιμοποιούν μια επαναχρησιμοποιήσιμη βιβλιοθήκη υποδομής της UML 2 και αυτό έχει πολλά πλεονεκτήματα από τα οποία τα κυριότερα είναι τα εξής:

• Οι κανόνες μοντελοποίησης μεταδεδομένων είναι πιο απλοί από την προηγούμενη έκδοση του MOF.

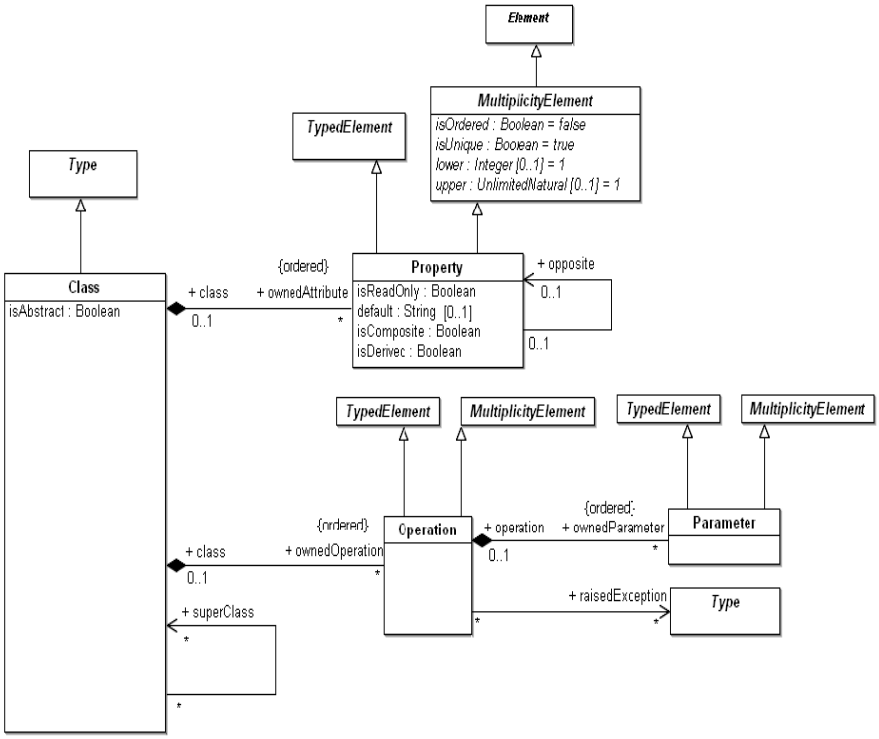
• Οι διάφορες τεχνολογικές απεικονίσεις του MOF έχουν εφαρμογή σε μεγαλύτερο εύρος UML μοντέλων

• Λόγω των κοινών εννοιών UML 2 και MOF 2, μπορεί να χρησιμοποιηθεί οποιοδήποτε εργαλείο μοντελοποίησης σε UML για την σχεδίαση ενός MOF μεταμοντέλου.

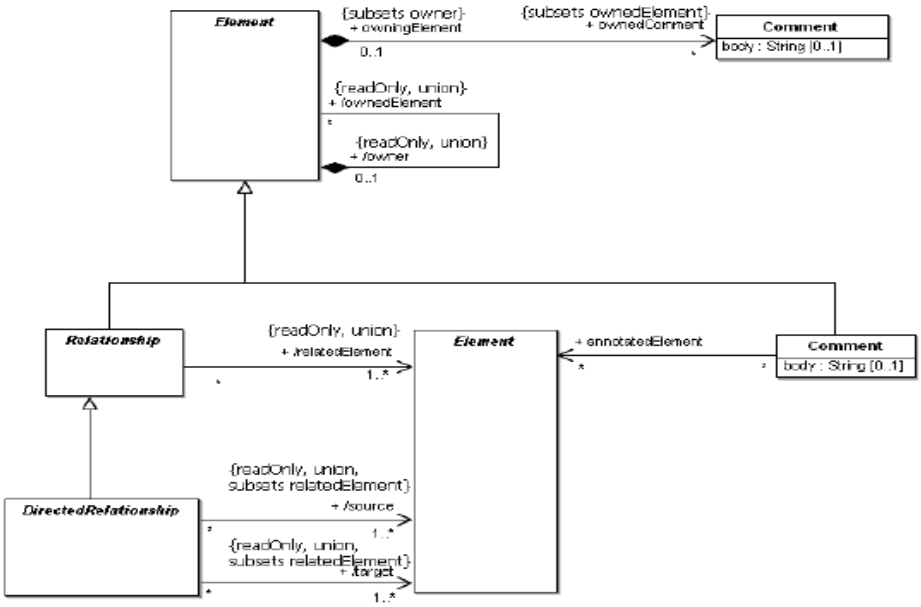
Στην βελτιωμένη έκδοση του MOF προβλέπονται δύο εκδόσεις του MOF. Το βασικό MOF (Essential MOF-EMOF) και το πλήρες MOF (Complete MOF-CMOF). Το EMOF μοντέλο αποτελεί ένα υποσύνολο του MOF το οποίο αντιστοιχεί στις έννοιες των αντικειμενοστραφών γλωσσών προγραμματισμού και του XML. Η σημασία του EMOF είναι το γεγονός ότι παρέχει ένα πλαίσιο για την άμεσες απεικονίσεις μοντέλων MOF σε εφαρμογές όπως η JMI και η XMI για απλά μεταμοντέλα. Ένας πρωτεύων στόχος του EMOF είναι ο ορισμός απλών μεταμοντέλων ενώ ταυτόχρονα θα

υπάρχει υποστήριξη επεκτάσεων των απλών αυτών μεταμοντέλων ώστε να προκύψουν πιο πολύπλοκα μεταμοντέλα χρησιμοποιώντας το CMOF. Χαρακτηριστικό του EMOF είναι ότι ενώ ορίζεται σαν ένα CMOF μοντέλο μπορεί να οριστεί χωρίς τη χρήση του CMOF αλλά αναδρομικά χρησιμοποιώντας τον εαυτό του.

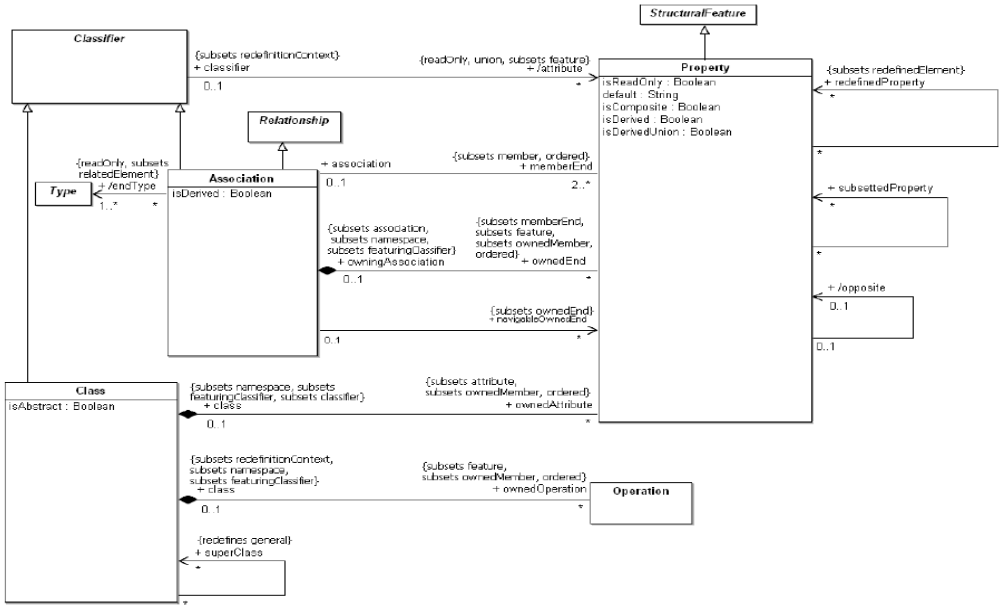
Το CMOF είναι το μεταμοντέλο που χρησιμοποιείται για να ορίσει άλλα μεταμοντέλα όπως τη UML 2. Το CMOF παράγεται από το EMOF και από το πακέτο Core::Constructs της UML 2. Όπως το EMOF, έτσι και το CMOF μπορεί να οριστεί χωρίς τη χρήση του EMOF, αλλά αναδρομικά χρησιμοποιώντας τον εαυτό του. Παρατηρούμε τη στενή σχέση μεταξύ του MOF 2 και της UML 2 αλλά και την ευελιξία του MOF, καθώς τόσο το EMOF όσο και το CMOF ενώ μπορούν να σταθούν σαν ανεξάρτητα μεταμοντέλα, ταυτόχρονα προκύπτουν το ένα από το άλλο. Η πλήρης προδιαγραφή του CMOF μπορεί να βρεθεί στα [9] και [10]. Παραθέτουμε όμως τα σχήματα προδιαγραφής βασικών στοιχείων του CMOF.



Σχήμα 2.9: Προδιαγραφή κλάσεων MOF



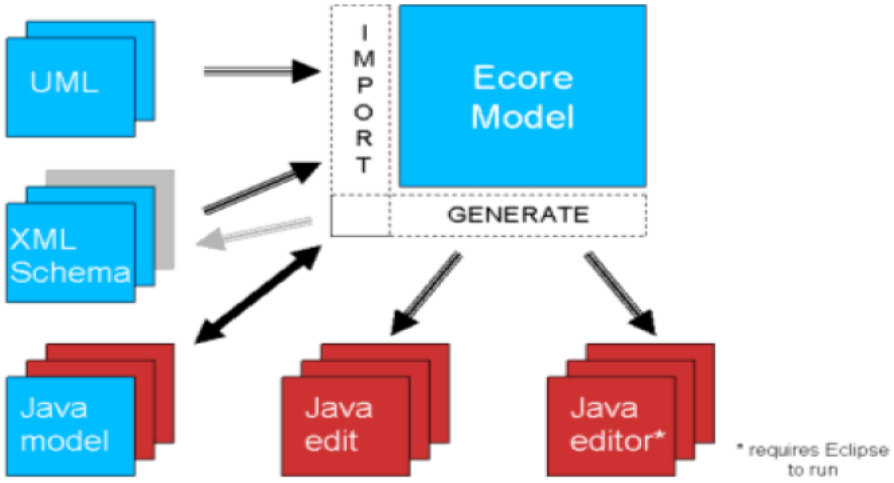
Σχήμα 2.10: Προδιαγραφή των σχέσεων μεταξύ των κλάσεων



Σχήμα 2.11: Συνοπτικό σχήμα παρουσίασης των βασικότερων στοιχείων του CMOF.

2.1.3 Το Eclipse Modeling Framework

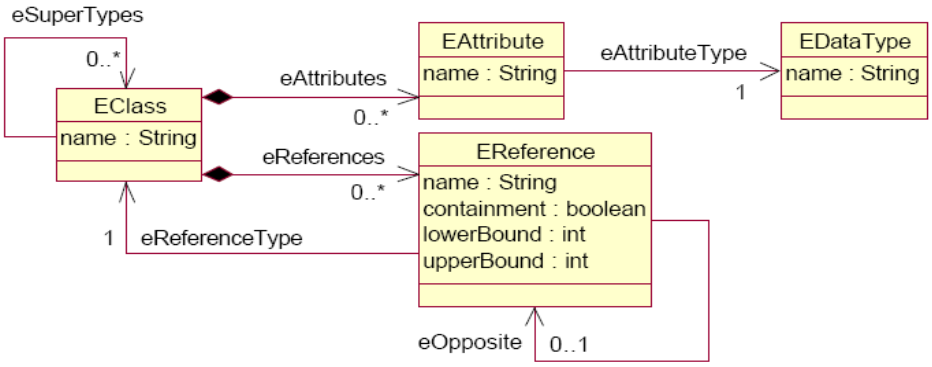
Το Eclipse Modeling Framework (EMF) αποτελεί μια προσπάθεια υλοποίησης της Μοντελοκεντρικής Αρχιτεκτονικής από την IBM και βασίζεται στην πλατφόρμα Eclipse. Το EMF δίνει στον σχεδιαστή την δυνατότητα να σχεδιάσει μοντέλα που περιγράφουν ένα σύστημα λογισμικού. Δεδομένου ότι ο πηγαίος κώδικας είναι ένα μοντέλο περιγραφής του συστήματος, τα UML διαγράμματα είναι επίσης μοντέλα περιγραφής του συστήματος όπως επίσης και τα XML σχήματα, το EMF προσπαθεί να ενοποιήσει τις υπάρχουσες τεχνικές μοντελοποίησης και να δώσει στον μηχανικό λογισμικού την δυνατότητα να ορίσει το σύστημα στο μοντέλο με το οποίο αυτός είναι πιο εξοικειωμένος αλλά ταυτόχρονα να έχει και την περιγραφή των άλλων μοντέλων. Σχηματικά η αρχιτεκτονική του EMF φαίνεται στο επόμενο σχήμα.



Σχήμα 2.12: Η αρχιτεκτονική του Eclipse Modeling Framework (EMF).

Το μεταμοντέλο το οποίο χρησιμοποιείται για την μοντελοποίηση των συστημάτων είναι το Ecore. Το Ecore είναι εμπνευσμένο και παραπλήσιο του MOF. Το Ecore είναι ένα υποσύνολο της UML όπως ακριβώς και το EMOF. Η αντιστοιχία των τύπων του MOF με το Ecore είναι σχεδόν 1-1 και οι διαφορές τους είναι κυρίως λεξικογραφικού χαρακτήρα. Συνεπώς είναι

αρκετά εύκολη η μετατροπή ενός μοντέλου το οποίο υπακούει στο EMOF ή στο CMOF σε ένα μοντέλο το οποίο να υπακούει στο Ecore και αντίστροφα. Η ομοιότητα των δύο μεταμοντέλων φαίνεται και στο παρακάτω σχήμα στο οποίο απεικονίζεται ο πυρήνας του Ecore (Ecore kernel)



Σχήμα 2.13: Ο πυρήνας του Ecore.

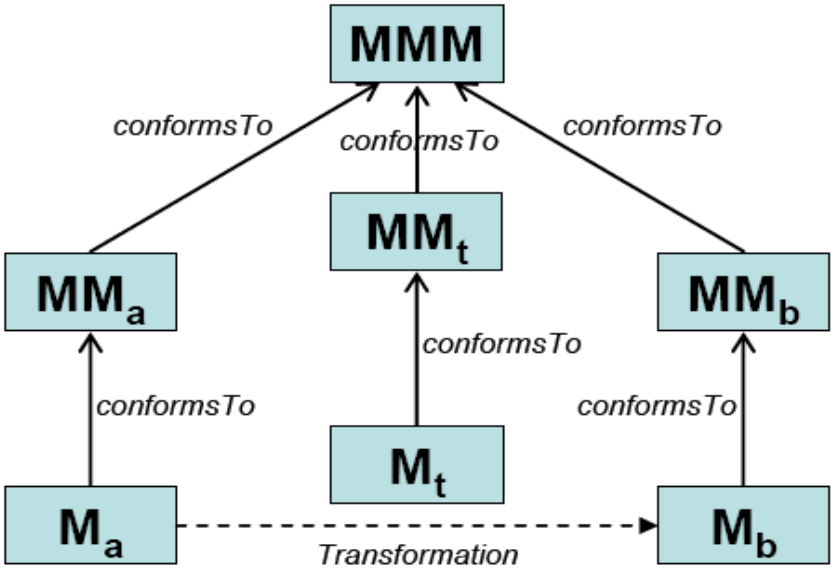
2.2. Γλώσσες μετατροπής μοντέλων

Η ύπαρξη διαφορετικών προτύπων μοντελοποίησης, αλλά και η πληθώρα των διαγραμμάτων που χρειάζεται για την απεικόνιση όλων των πλευρών ενός λογισμικού συστήματος οδήγησε στην ανάπτυξη εργαλείων μετασχηματισμού μοντέλων έτσι ώστε να είναι δυνατή η σύνθεση πληροφοριών από ένα ή περισσότερα μοντέλα για την παραγωγή νέων μοντέλων και διαγραμμάτων

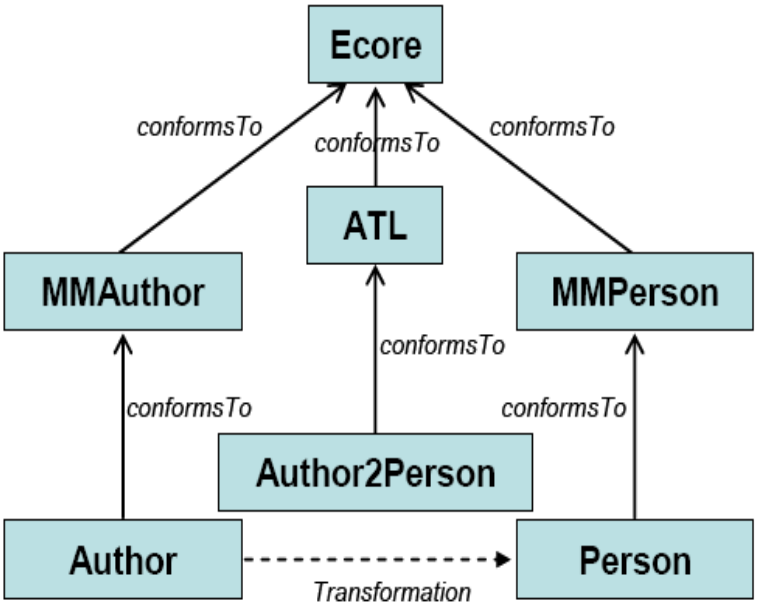
2.2.1 Atlas Transformation Language

Η Atlas Transformation Language (ATL) αποτελεί ένα έργο των οργανισμών Laboratoire d'Informatique de Nantes Atlantique (LINA) και του Institut national de recherche en informatique et en automatique (INRIA) . Όπως αναφέρεται και στο [11] σκοπός της ATL είναι να προσφέρει στους σχεδιαστές λογισμικών συστημάτων ένα μέσο για την παραγωγή ενός συνόλου μοντέλων από ένα άλλο σύνολο μοντέλων.

Ένα πρόγραμμα ATL αποτελείται από ένα σύνολο κανόνων μετασχηματισμού οι οποίοι ορίζουν πως τα στοιχεία των αρχικών μοντέλων συνδυάζονται και χρησιμοποιούνται για την αρχικοποίηση και παραγωγή των στοιχείων των τελικών μοντέλων. Η ATL βασίζεται πάνω στην πλατφόρμα του Eclipse και προσφέρει μια μεγάλη ποικιλία εργαλείων για τον σχεδιασμό και την υλοποίηση των κανόνων μετασχηματισμού. Η ATL είναι ένας συνδυασμός δηλωτικού και προστακτικού προγραμματισμού. Υπακούει στο Ecore μεταμοντέλο και ενώ τα στοιχεία και οι τύποι που χρησιμοποιεί βασίζονται στην OCL (Object Constraint Language) του OMG. Στην ATL αλλά και γενικότερα στην Μοντελοκεντρική Αρχιτεκτονική ο μετασχηματισμός ενός μοντέλου σε ένα άλλο μοντέλο αποτελεί και αυτός ένα μοντέλο το οποίο πρέπει να υπακούει σε ένα μεταμοντέλο. Επίσης τα μεταμοντέλα που ορίζουν τα μοντέλα που λαμβάνουν μέρος στο μετασχηματισμό θα πρέπει να υπακούν στο ίδιο μετα-μεταμοντέλο. Στο σχήμα 2.14 συνοψίζεται η πλήρης διαδικασία του μετασχηματισμού των μοντέλων ενώ στο σχήμα 2.15 περιγράφεται ο μετασχηματισμός μοντέλων χρησιμοποιώντας την ATL.



Σχήμα 2.14: Η διαδικασία μετασχηματισμού μοντέλων.



Σχήμα 2.15: Μετασχηματισμός μοντέλου σε ATL.

2.2.2 Model Transformation Framework

Το Model Transformation Framework είναι ένα έργο της IBM και αποτελείται από ένα σύνολο εργαλείων τα οποία βοηθούν τους σχεδιαστές λογισμικού να μεταβαίνουν από ένα EMF μοντέλο σε ένα άλλο EMF μοντέλο. Προφανώς το MTF βασίζεται πάνω στην πλατφόρμα του Eclipse και στο Eclipse Modeling Framework. Τα μοντέλα αυτά μπορούν να υπακούν στο ίδιο μεταμοντέλο ή και σε διαφορετικά μεταμοντέλα. Το MTF προσφέρει μια γλώσσα ορισμού κανόνων οι οποίοι ορίζουν την μορφή του τελικού μοντέλου έτσι ώστε να είναι το τελικό μοντέλο συνεπές ως προς το αρχικό μοντέλο. Το MTF δεν είναι τόσο αυστηρό ως προς τον ορισμό των αρχικών μοντέλων και μεταμοντέλων. Μπορούμε να ορίσουμε τους κανόνες μετασχηματισμού και να καθορίσουμε το αρχικό και τελικό μοντέλο κατά την εκτέλεση του μετασχηματισμού. Στο [28] αναλύεται η λειτουργία του MTF και παρατίθεται ένα παράδειγμα μετασχηματισμού EMF μοντέλων.

2.3 Εργαλεία ανοιχτού κώδικα ανάλυσης συστημάτων.

Τα πρότυπα που αναφέρθηκαν πιο πάνω βρίσκουν πρακτική εφαρμογή μέσα από μια πληθώρα εργαλείων ανάλυσης συστημάτων είτε ανοιχτού κώδικα είτε ολοκληρωμένων εμπορικών εφαρμογών.
Στην εργασία αυτή θα αναφερθούμε μόνο σε εργαλεία ανοιχτού κώδικα.

Ένα από τα σημαντικότερα και πιο ολοκληρωμένα εργαλεία στο χώρο της σχεδίασης,ανάπτυξης και ανάλυσης συστημάτων είναι το Eclipse Modeling Framework (EMF). Το EMF είναι ένα περιβάλλον-πλαίσιο σχεδίασης και ανάπτυξης συστημάτων Java. Το EMF χρησιμοποιεί όπως προαναφέραμε το μεταμοντέλο Ecore για την μοντελοποίηση των συστημάτων, το οποίο είναι αρκετά παραπλήσιο του EMOF και είναι και αυτό υποσύνολο της UML. Το EMF δίνει τη δυνατότητα στον μηχανικό λογισμικού να σχεδιάσει το σύστημα του χρησιμοποιώντας το πρότυπο της επιλογής του και το EMF αναλαμβάνει να δημιουργήσει το μοντέλο του συστήματος σύμφωνα και με τα υπόλοιπα πρότυπα που υποστηρίζει το Eclipse. Για παράδειγμα ο σχεδιαστής ενός συστήματος μπορεί να ορίσει το μοντέλο του χρησιμοποιώντας Annotated Java και το EMF θα κατασκευάσει σύμφωνα με το μοντέλο αυτό τα Ecore και UML διαγράμματα καθώς επίσης και τα XMI κείμενα που θα περιγράφουν τα διαγράμματα αυτά. Τέλος από τα

παραγόμενα αυτά μοντέλα το EMF παράγει τον κώδικα του συστήματος ο οποίος είναι συνεπής ως προς τα μοντέλα και συντακτικά σωστός.

Το Eclipse έχει επίσης τη δυνατότητα να ενσωματώνει εργαλεία (plugins) και να τα χρησιμοποιεί σε συνδυασμό με το EMF ή άλλα εργαλεία. Με αυτόν τον τρόπο αυξάνονται σημαντικά οι δυνατότητες του Eclipse και του EMF με βασικότερη δυνατότητα αυτή της εύκολης επέκτασης των λειτουργιών του περιβάλλοντος-πλαισίου. Χρησιμοποιώντας αυτήν την ιδιότητα του Eclipse έχουν αναπτυχθεί πολλές εφαρμογές οι οποίες βασίζονται πάνω στην πλατφόρμα του Eclipse και παρέχουν σημαντικές δυνατότητες σχεδίασης και ανάλυσης λογισμικών συστημάτων. Μια βασική εφαρμογή η οποία έχει υλοποιηθεί σαν ενσωματωμένο εργαλείο του Eclipse είναι το ATL tool το οποίο είναι η πρακτική εφαρμογή της ATL γλώσσας. Δίνει τη δυνατότητα στον χρήστη της πλατφόρμας του Eclipse και του EMF να παράξει αυτοματοποιημένα ένα Ecore μοντέλο από ένα άλλο Ecore μοντέλο με τη χρήση ενός συνόλου κανόνων μετασχηματισμού.

Παρόμοιες δυνατότητες παρέχει στον μηχανικό λογισμικού το Metadata Repository (MDR) το οποίο βασίζεται στην πλατφόρμα NetBeans της εταιρίας Sun. Το MDR ,όπως αναφέρεται και στο [17] είναι μια υλοποίηση των προτύπων MOF,XMI και JMI. Παρέχει πρόσβαση σε μεταδεδομένα μέσω ενός συνόλου διεπαφών προγραμματισμού εφαρμογών (API). Κύρια διαφοροποίηση του MDR από το EMF είναι το γεγονός ότι το MDR υλοποιεί το πρότυπο MOF χωρίς καμία τροποποίηση του προτύπου σε αντίθεση με το EMF το οποίο υλοποιεί το Ecore το οποίο διαφοροποιείται λίγο από το MOF.

Σε αυτό το σημείο αξίζει να αναφέρουμε και την AndroMDA το οποίο αποτελεί ένα περιβάλλον πλαίσιο ανοιχτού κώδικα το οποίο υλοποιεί την μοντελοκεντρική αρχιτεκτονική στην ανάπτυξη εφαρμογών. Όπως αναφέρεται και στο [18],ο χρήστης μπορεί να σχεδιάσει ένα μοντέλο της εφαρμογής το οποίο θα είναι ανεξάρτητο από την πλατφόρμα υλοποίησης (PIM-μοντέλο) και να εισάγει αυτό το μοντέλο στην AndroMDA. Η AndroMDA αναλαμβάνει να μετασχηματίσει το PIM-μοντέλο σε ένα μοντέλο συγκεκριμένης πλατφόρμας (PSM-μοντέλο) και να συνεισφέρει στην υλοποίηση της εφαρμογής με την παραγωγή κώδικα. Ο χρήστης της AndroMDA έχει τη δυνατότητα να παράξει κώδικα σε σχεδόν οποιαδήποτε γλώσσα προγραμματισμού επιθυμεί. Επίσης ο κώδικας που παράγεται από την AndroMDA μπορεί να χρησιμοποιηθεί και σε άλλα περιβάλλοντα

πλαίσια όπως το Hibernate, το Spring και το Struts. Τέλος είναι σημαντικό να προσθέσουμε ότι στόχος της ομάδας ανάπτυξης της AndroMDA είναι η ενσωμάτωση της AndroMDA με το Eclipse.

2.4 Ανάλυση UML διαγραμμάτων.

Κατά την φάση της υλοποίησης του συστήματος τα μοντέλα που παρήχθησαν στη φάση της σχεδίασης θα πρέπει να μετατραπούν σε κώδικα. Αρχικά τα διαγράμματα που χρησιμοποιήθηκαν για την αυτόματη παραγωγή κώδικα ήταν τα διαγράμματα κλάσεων λόγω της παρόμοιας δομής των διαγραμμάτων αυτών με την δομή των αντικειμενοστραφών γλωσσών. Στη συνέχεια όμως, το ερευνητικό ενδιαφέρον στράφηκε στην αυτοματοποιημένη επεξεργασία και άλλων τύπων διαγραμμάτων και την εξαγωγή πληροφοριών και συμπερασμάτων από αυτά. Τα δεδομένα που προκύπτουν από την επεξεργασία των ποικίλων διαγραμμάτων χρησιμοποιούνται στον συγχρονισμό των διαγραμμάτων μεταξύ τους καθώς και την ανάλυση με μεγαλύτερη λεπτομέρεια της δομής, των λειτουργιών αλλά και τον κώδικα του συστήματος.

Στο [21] οι Blech,Glesner και Leitner ασχολούνται με την μελέτη διαγραμμάτων κατάστασης και την επαλήθευση της διαδικασίας της παραγωγής Java κώδικα από τα διαγράμματα κατάστασης. Στο [22] οι San gal, Farrell, Lieberman και Lorenz επεξεργάζονται ακολουθιακά διαγράμματα σε συνδυασμό με διαγράμματα κλάσεων τα οποία θεωρούν ότι είναι συνεπή μεταξύ τους. Από την επεξεργασία των διαγραμμάτων κλάσεων προκύπτουν τα βασικά στοιχεία των κλάσεων του κώδικα του συστήματος που μελετούν, όπως τα πεδία και οι μέθοδοι κάθε κλάσης, ενώ από τα ακολουθιακά διαγράμματα προκύπτουν πληροφορίες σχετικές με τις αλληλουχίες των κλήσεων των μεθόδων που πραγματοποιούνται κατά την εκτέλεση μιας λειτουργίας του συστήματος. Κατασκευάζεται επίσης και ένας σκελετός του σώματος των μεθόδων των κλάσεων του συστήματος.

Στο [14] οι Mathematics Thongmak και Pornsiri Muenchaisri χρησιμοποιούν ακολουθιακά διαγράμματα και διαγράμματα κλάσης για να παράγουν κώδικα. Ορίζουν και χρησιμοποιούν ένα μεταμοντέλο ακολουθιακών διαγραμμάτων το οποίο βασίζεται στο μεταμοντέλο της UML όπως αυτό περιγράφεται στο [8]. Όπως και στο [3] ορίζουν ένα πολύ πιο απλό

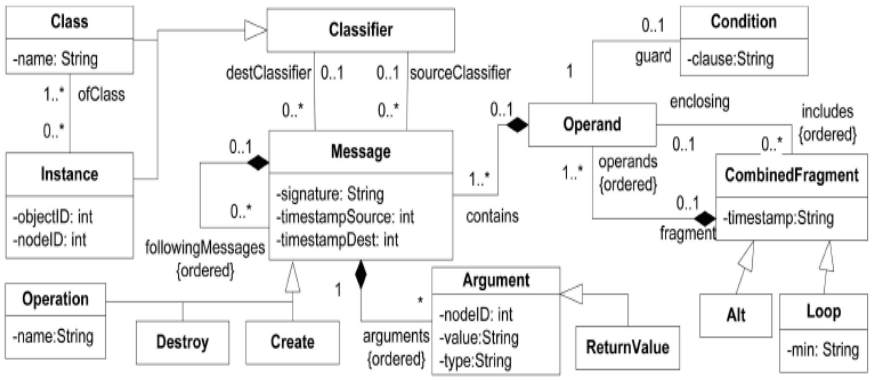
μεταμοντέλο ακολουθιακών διαγραμμάτων από αυτό που προδιαγράφεται στη UML και περιγράφουν τον μετασχηματισμό ακολουθιακών διαγραμμάτων που υπακούν στο μεταμοντέλο της UML σε ακολουθιακά διαγράμματα που υπακούν στο δικό τους μοντέλο. Το σημαντικότερο στοιχείο της εργασίας τους αποτελεί ένα σύνολο κανόνων που αναγνωρίζουν μοτίβα ακολουθιακών διαγραμμάτων, διαγραμμάτων κλάσης ή και συνδυασμό των δύο. Το αποτέλεσμα που προκύπτει είναι κώδικας σε Java ο οποίος περιέχει τους ορισμούς των μεθόδων και των πεδίων των κλάσεων όπως προκύπτουν από τα διαγράμματα κλάσης τις αρχικοποιήσεις των πεδίων και τις λοιπές κλήσεις μεθόδων όπως προκύπτουν από τα ακολουθιακά διαγράμματα. Παρατηρούμε ότι όπως και στις προηγούμενες εργασίες η εξαγωγή της αρχιτεκτονικής του συστήματος βασίζεται στην ανάγνωση διαγραμμάτων κλάσης και τα ακολουθιακά διαγράμματα χρησιμοποιούνται για να συμπληρώσουν τα συμπεράσματα που έχουν προκύψει από τα διαγράμματα κλάσης.

Στο [13] οι Engels,Hücking,Sauer,Wagner βασίζονται σε συνεργατικά διαγράμματα και διαγράμματα κλάσεων για να παράγουν κώδικα Java ο οποίος θα είναι συντακτικά ορθός. Σε αυτήν την επιστημονική εργασία αρχικά θεωρείται ότι είναι διαθέσιμα τόσο τα διαγράμματα κλάσεων όσο και τα συνεργατικά διαγράμματα και ότι τα διαγράμματα αυτά είναι συντακτικά σωστά και ότι είναι συνεπή μεταξύ τους. Η ανάλυση του συστήματος γίνεται χρησιμοποιώντας ένα σύνολο κανόνων αναγνώρισης μοτίβων στα διαγράμματα και αποτελείται από δύο βήματα. Στο πρώτο βήμα αναλύεται η δομή του συστήματος από τα διαγράμματα κλάσεων. Από την επεξεργασία των διαγραμμάτων κλάσεων παράγονται οι κλάσεις του συστήματος, οι μέθοδοι και τα πεδία της κάθε κλάσης. Επίσης καθίσταται δυνατό να προσδιοριστούν οι τύποι των δεδομένων των πεδίων των κλάσεων, των παραμέτρων των μεθόδων καθώς και των δεδομένων που επιστρέφουν οι μέθοδοι. Στο δεύτερο βήμα προστίθενται πληροφορίες που εξάγονται από τα συνεργατικά διαγράμματα. Με βάση τις πληροφορίες αυτές εξάγονται συμπεράσματα για τις αναθέσεις τιμών των πεδίων των κλάσεων, δηλαδή ποιες μέθοδοι καλούνται για τον προσδιορισμό των τιμών των πεδίων των κλάσεων. Το αποτέλεσμα που παράγεται είναι κώδικας Java ο οποίος αποτελείται από τις κλάσεις,τις μεθόδους και τα πεδία που παρήχθησαν από το πρώτο βήμα της ανάλυσης και τις κλήσεις των συναρτήσεων και τις αναθέσεις τιμών που παρήχθησαν από το δεύτερο βήμα της ανάλυσης.

Ερευνητικές προσπάθειες έχουμε επίσης και για την αντίστροφη διαδικασία,

την παραγωγή δηλαδή διαγραμμάτων από τον κώδικα του συστήματος ή από την καταγραφή της λειτουργίας του. Στο [23] οι Jackson και Waingold αναλύουν το WOMBLE, ένα εργαλείο παραγωγής διαγραμμάτων αντικειμένων από το bytecode Java διαγραμμάτων ενώ στο [24] περιγράφεται η διαδικασία παραγωγής διαγραμμάτων κλάσης και ακολουθιακών διαγραμμάτων μέσω του IBM Rational Software. Στο [25] οι Niere, Wadsack και Zündorf παράγουν από τον κώδικα του συστήματος που εξετάζουν διαγράμματα της στατικής δομής αλλά και της δυναμικής συμπεριφοράς του συστήματος.

Στο [3] οι Brian,Labiche και Leduc προσπαθούν να παράγουν sequence diagrams από την ανάλυση του κώδικα της εφαρμογής καθώς και την δυναμική εκτέλεση της εφαρμογής. Συγκεκριμένα τρέχουν την εφαρμογή σύμφωνα με κάποια σενάρια χρήσης ώστε να μπορέσουν να απεικονίσουν και να μοντελοποιήσουν τη συμπεριφορά του συστήματος και την επικοινωνία μεταξύ των διαφόρων στοιχείων του συστήματος. Για την μοντελοποίηση της λειτουργίας του συστήματος προτείνουν ένα μεταμοντέλο για την αναπαράσταση των ακολουθιακών διαγραμμάτων το οποίο είναι εμπνευσμένο από το μεταμοντέλο της UML όπως αυτό περιγράφεται στο [8] . Το μοντέλο αυτό είναι αρκετά πιο απλό και λιτό όπως φαίνεται και στο σχήμα 2.16. Παρά τον μινιμαλισμό του, το μοντέλο διατηρεί σε μεγάλο βαθμό την περιγραφικότητα του μεταμοντέλου των ακολουθιακών διαγραμμάτων της UML. Ονομάζουν τα διαγράμματα που υπακούν σε αυτό το μεταμοντέλο scenario diagrams για να τα διαχωρίσουν από τα ακολουθιακά διαγράμματα αλλά και για να τονίσουν ότι πρόκειται για διαγράμματα που απεικονίζουν την επικοινωνία μεταξύ των διαφόρων μερών του συστήματος όπως αυτή καταγράφηκε από την εκτέλεση ενός συγκεκριμένου σεναρίου και όχι για την συνολική απεικόνιση της επικοινωνίας των λειτουργικών μερών του συστήματος γενικότερα. Μετά την εκτέλεση των διαφόρων σεναρίων και την ανάλυση του κώδικα του συστήματος συγχωνεύουν τα scenario diagrams για να προκύψει ένα τελικό scenario diagram το οποίο θα δίνει μια όσο το δυνατό πληρέστερη μοντελοποίηση της επικοινωνίας των μονάδων του συστήματος καθώς και της λειτουργίας του. Το τελικό μοντέλο μπορεί να συγκριθεί με τα ακολουθιακά διαγράμματα τω προδιαγραφών του συστήματος.



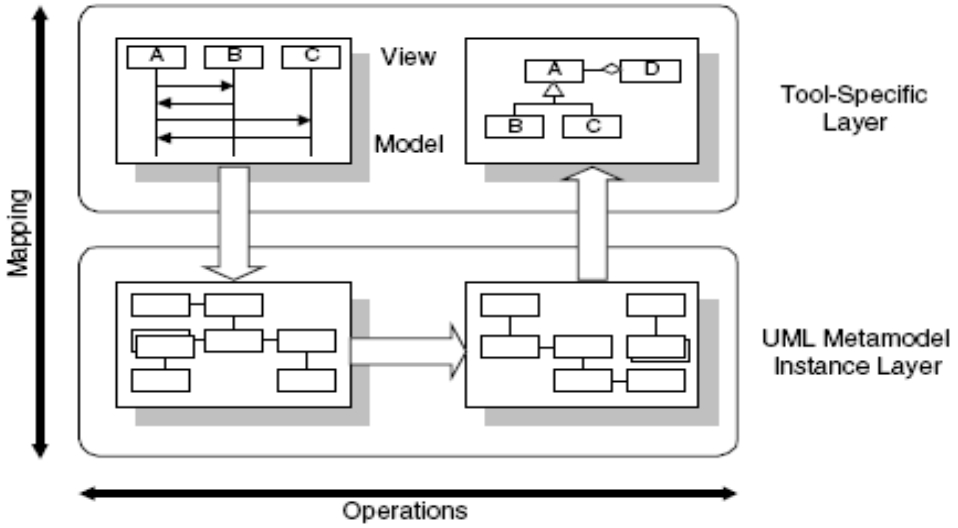
Σχήμα 2.16: Το scenario diagram μεταμοντέλο της επιστημονικής εργασίας των Brian,Labiche και Leduc.

Σημαντική προσπάθεια έχει καταβληθεί στον μετασχηματισμό διαγραμμάτων από τον έναν τύπο στον άλλο, στον συγχρονισμό των διαγραμμάτων ώστε να είναι μεταξύ τους συνεπή, καθώς επίσης και στον συγχρονισμό των διαγραμμάτων με τα μεταμοντέλα τους.

Στο [26] οι Ivkovic και Kontogiannis μελετούν τον συγχρονισμό των UML διαγραμμάτων όταν ένα ή περισσότερα από αυτά δέχονται αλλαγές λόγω τροποποίησης ή συντήρησης ενός λογισμικού συστήματος. Στο [20] αναλύονται οι σχέσεις εξάρτησης των UML διαγραμμάτων κατά τη διάρκεια της σχεδίασης και υλοποίησης ενός συστήματος ενώ στο [27] οι Hößler, Soden και Eichler ασχολούνται με τον συγχρονισμό των διαγραμμάτων και των μεταμοντέλων στα οποία τα διαγράμματα υπακούν.

Στο [4] οι Selonen,Koskimies και Sakkinen ασχολούνται με τον μετασχηματισμό UML διαγραμμάτων σε UML διαγράμματα άλλου τύπου. Αναφέρουν ότι τα UML διαγράμματα αποτελούν ένα σύνολο διαγραμμάτων τα οποία απεικονίζουν ένα σύστημα από διαφορετικές πτυχές και συνεπώς τα διάφορα μοντέλα του συστήματος είναι εξαρτώμενα μεταξύ τους και επικαλυπτόμενα. Βασιζόμενοι σε αυτό το γεγονός προτείνουν ένα σύνολο μετασχηματισμών πάνω στα μοντέλα ενός συστήματος. Οι μετασχηματισμοί αυτοί είναι ο έλεγχος συνέπειας μεταξύ των διαγραμμάτων, ο εμπλουτισμός της πληροφορίας που περιέχει ένα διάγραμμα με τη χρήση άλλων διαγραμμάτων, η προβολή μιας πτυχής μόνο του συστήματος και τέλος η

δημιουργία ενός διαγράμματος με βάση τις πληροφορίες που προκύπτουν από τα διαγράμματα άλλου τύπου. Η τεχνική που χρησιμοποιούν στηρίζεται σε ένα σύνολο κανόνων μετασχηματισμών μοντέλων. Οι κανόνες αυτοί βασίζονται στην ύπαρξη μεταμοντέλων τα οποία προδιαγράφουν τη δομή των UML διαγραμμάτων και περιγράφουν τη μετάβαση από ένα στιγμιότυπο ενός μεταμοντέλου σε ένα στιγμιότυπο ενός άλλου μεταμοντέλου. Ο τρόπος λειτουργίας των κανόνων μετασχηματισμού απεικονίζεται γραφικά και στο παρακάτω σχήμα.



Σχήμα 2.17: Μετασχηματισμός μεταμοντέλων.

Στην εργασία αυτή προτείνεται επίσης και ένα σύνολο κανόνων μετασχηματισμού για την δημιουργία διαγραμμάτων κλάσεων από ακολουθιακά διαγράμματα. Οι κανόνες αυτοί είναι ευριστικοί με την έννοια ότι σκοπό δεν έχουν να εξασφαλίσουν την πιστή αναπαραγωγή της πραγματικής αρχιτεκτονικής ενός συστήματος από τα ακολουθιακά διαγράμματα αλλά να βρουν την πιθανότερη δομή του συστήματος ώστε είτε το παραγόμενο διάγραμμα κλάσεων να αποτελεί μια βάση-πρόταση για την σχεδίαση της αρχιτεκτονικής του συστήματος ή να γίνει σύγκριση με το ήδη υπάρχον διάγραμμα κλάσεων και να εντοπιστούν τυχόν ασυνέπειες μεταξύ του διαγράμματος κλάσεων και αντίστοιχου ακολουθιακού διαγράμματος.

2.5 Εύρεση ασυνεπειών μεταξύ των διαγραμμάτων.

Εξίσου σημαντική με την ανάλυση της δομής ενός συστήματος με βάση τα UML διαγράμματα και την παραγωγή κώδικα ή άλλων διαγραμμάτων, είναι και η σύγκριση διαγραμμάτων για την εύρεση ασυνεπειών. Αν και η σύγκριση διαγραμμάτων έχει θιγεί στις προηγούμενες εργασίες που αναφέρθηκαν, στις περισσότερες θεωρείται ότι η συνεπεία των διαγραμμάτων μεταξύ τους και με τον κώδικα είναι δεδομένη.

Αντίθετα, στο [39] οι Malgouyres και Motet ελέγχουν την συνέπεια των UML διαγραμμάτων μεταξύ τους βασιζόμενοι στα μεταμοντέλα των διαγραμμάτων, ενώ στο [40] οι Liu, Easterbrook και Mylopoulos ορίζουν ένα σύνολο κανόνων ανίχνευσης ασυνεπειών όχι μόνο μεταξύ διαγραμμάτων UML, αλλά και ασυνέπειες μέσα σε ένα διάγραμμα UML ή και ασυνέπειες μεταξύ του διαγράμματος και του μεταμοντέλου του.

Στο [37] οι Holger και Wehrheim καθώς επίσης και ο Yeung στο [38] πραγματεύονται τον έλεγχο της συνέπειας μεταξύ των διαγραμμάτων κλάσεων και των διαγραμμάτων κατάστασης. Στο [15] ο Ragnhild Van Der Straeten μελετάει ακολουθιακά διαγράμματα και διαγράμματα κλάσεων έτσι ώστε να εντοπίσει ασυνέπειες μεταξύ των διαγραμμάτων αυτών. Οι ασυνέπειες που εντοπίζει εστιάζονται κυρίως στους βαθμούς των σχέσεων των κλάσεων των διαγραμμάτων κλάσεων, την κατευθυντικότητα των σχέσεων και αν αυτά τα στοιχεία επιβεβαιώνονται στα ακολουθιακά διαγράμματα καθώς και αν η συμπεριφορά ενός αντικειμένου ή μιας κλάσης σε ένα ακολουθιακό διάγραμμα, είναι συνεπής με τη συμπεριφορά της κλάσης όπως αυτή προβλέπεται από το διάγραμμα κλάσης.

Στο [16] οι Litvak, Tyszberowicz και Yehudai προσπαθούν να εντοπίσουν ασυνέπειες μεταξύ διαγραμμάτων κατάστασης και ακολουθιακών διαγραμμάτων. Η μέθοδος τους βασίζεται σε ένα υβριδικό διάγραμμα το οποίο αποτελεί συνδυασμό ακολουθιακού διαγράμματος και διαγράμματος κατάστασης. Το διάγραμμα αυτό απεικονίζει καταστάσεις που σχετίζονται με μια γραμμή ετοιμότητας στο αρχικό ακολουθιακό διάγραμμα. Ένα μήνυμα στο ακολουθιακό διάγραμμα ερμηνεύεται στο διάγραμμα κατάστασης με δύο τρόπους:

• Σαν πυροδότης (trigger), όταν το μήνυμα είναι ένα εισερχόμενο μήνυμα στο επιλεγμένο αντικείμενο του ακολουθιακού διαγράμματος.

• Σαν ενέργεια(action), όταν το μήνυμα είναι ένα εξερχόμενο μήνυμα στο επιλεγμένο αντικείμενο του ακολουθιακού διαγράμματος.

Ένα μήνυμα μπορεί να αλλάξει την κατάσταση του υπό εξέταση(τρέχοντος) αντικειμένου στο διάγραμμα κατάστασης. Κάθε αλλαγή κατάστασης γίνεται σύμφωνα με τις προηγούμενες μεταβάσεις καταστάσεων. Για κάθε μετάβαση από μια αρχική κατάσταση σε μια τελική που πυροδοτείται από το ακολουθιακό διάγραμμα, ο αλγόριθμος που προτείνουν οι Litvak, Tyszberowicz και Yehudai ελέγχει αν προβλέπεται η μετάβαση αυτή από το διάγραμμα κατάστασης. Αν αυτή η μετάβαση είναι δυνατή τότε σε αυτό το σημείο τα 2 διαγράμματα είναι συνεπή. Διαφορετικά ο προτεινόμενος αλγόριθμος εντοπίζει σε αυτό το σημείο μια ασυνέπεια μεταξύ των διαγραμμάτων.

Κεφάλαιο 3

Μοντελοποίηση-Σχεδίαση

3.1 Σκοπός της υλοποίησης

Σε αυτό το σημείο, έχοντας αναλύσει τα πρότυπα που χρησιμοποιούνται στην σχεδίαση,ανάπτυξη και ανάλυση συστημάτων καθώς και την έρευνα που έχει γίνει σε αυτόν τον τομέα, μπορούμε να αναπτύξουμε τον σκοπό της εφαρμογής αυτής της εργασίας.

• Αρχικός μας στόχος είναι να ορίσουμε ένα μοντέλο ακολουθιακού διαγράμματος το οποίο θα χρησιμοποιήσουμε καθ' όλη τη διάρκεια της εργασίας αυτής, για να ορίσουμε και να σχεδιάσουμε τα ακολουθιακά διαγράμματα που θα επεξεργαστούμε σε αυτήν την εργασία. Το μοντέλο αυτό βασίζεται στα μοντέλα που αναφέρονται στα [3],[4] και [14], ενώ ακολουθεί τις προδιαγραφές των ακολουθιακών διαγραμμάτων όπως αυτές αναφέρονται στα [7] και [8]. Απώτερος στόχος είναι να έχουμε ένα λιτό και εύχρηστο μοντέλο ακολουθιακού διαγράμματος το οποίο όμως ταυτόχρονα θα διατηρεί κατά το δυνατόν τις απεικονιστικές δυνατότητες του μοντέλου του ακολουθιακού διαγράμματος όπως αυτό ορίζεται από τον OMG στο [8]. Ιδιαίτερη έμφαση έχει δοθεί σε εκείνα τα χαρακτηριστικά των ακολουθιακών διαγραμμάτων τα οποία αναφέρονται στο κεφάλαιο 2 και στα οποία επικεντρώνεται το ενδιαφέρον μας. Σημαντικό είναι επίσης το μοντέλο που θα χρησιμοποιήσουμε να είναι επαναχρησιμοποιήσιμο. Για τον λόγο αυτό επιλέξαμε να χρησιμοποιήσουμε την πλατφόρμα του Eclipse διότι διαθέτει πληθώρα εργαλείων ανάλυσης και σχεδίασης συστημάτων ενώ ταυτόχρονα δεν περιορίζει τον μηχανικό-λογισμικού σε PSM-μοντέλα αλλά του δίνει τη δυνατότητα να σχεδιάσει PIM-μοντέλα έτσι ώστε να μην περιορίζεται σε κάποια πλατφόρμα. Ο μηχανικός λογισμικού δεν περιορίζεται ακόμα και αν επιλέξει να ορίσει το μοντέλο του χρησιμοποιώντας το μοντέλο Ecore, καθώς το MOF και το Ecore είναι και τα δύο υποσύνολα της UML και παραπλήσια μεταξύ τους, έτσι ώστε η μηχανιστική μετάβαση από το ένα στο άλλο να είναι σχετικά απλή

διαδικασία.

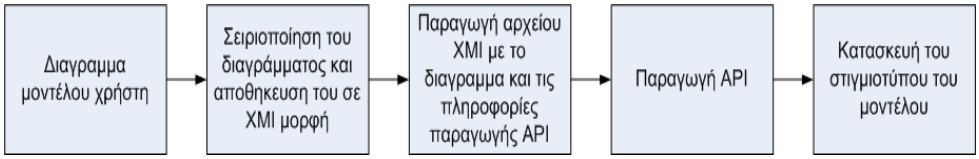
• Έχοντας ορίσει το μοντέλο των ακολουθιακών διαγραμμάτων που θα χρησιμοποιήσουμε, δεύτερος στόχος είναι ο ορισμός ενός μοντέλου κώδικα. Ένα διάγραμμα κώδικα που αποτελεί στιγμιότυπο του μοντέλου κώδικα θα περιγράφει τη δομή και τη λειτουργία ενός υπό εξέταση συστήματος. Πιο συγκεκριμένα κάθε διάγραμμα κώδικα θα περιέχει τις κλάσεις του συστήματος και για κάθε κλάση θα παρέχει πληροφορίες για τις μεθόδους και τα πεδία που η κλάση περιέχει. Επιπρόσθετα θα παρέχει έναν σκελετό του κώδικα του σώματος των μεθόδων υπό τη μορφή κλήσης μεθόδων και δημιουργίας αντικειμένων. Τέλος θα παρέχει και πληροφορίες σχετικά με την κληρονομικότητα των κλάσεων. Η αναπαράσταση του κώδικα μέσω αυτού του μοντέλου έχει το πλεονέκτημα της ανεξαρτησίας από την πλατφόρμα και την γλώσσα υλοποίησης της υπό εξέταση εφαρμογής αλλά και της πλατφόρμας υλοποίησης των κανόνων μετασχηματισμού που αναφέρουμε παρακάτω.

• Τρίτος στόχος είναι ο μετασχηματισμός του συνόλου των ακολουθιακών διαγραμμάτων σε ένα ενιαίο διάγραμμα κώδικα. Ο μετασχηματισμός αυτός υλοποιείται μέσω ενός συνόλου αλγορίθμων οι οποίοι επεξεργάζονται τα δεδομένα των ακολουθιακών διαγραμμάτων και τα συνθέτουν δημιουργώντας το διάγραμμα κώδικα.

• Τέταρτος στόχος είναι η σύγκριση του κώδικα του συστήματος με τον κώδικα που παρήχθη από τα ακολουθιακά διαγράμματα. Μπορούμε επίσης, λόγω της ομοιότητας των αντικειμενοστραφών γλωσσών με τα διαγράμματα κλάσης, να συγκρίνουμε τα διαγράμματα κλάσης, εφόσον αυτά υπάρχουν με τα διαγράμματα κώδικα και να εντοπίσουμε με αυτόν τον τρόπο τις ασυνέπειες μεταξύ ακολουθιακών διαγραμμάτων και διαγραμμάτων κλάσεων.

3.2 Περιγραφή της διαδικασίας ορισμού μοντέλων

Το περιβάλλον πλαίσιο που χρησιμοποιήθηκε σε αυτήν την εργασία ήταν το Eclipse Modelling Framework (EMF), το οποίο στηρίζεται στην πλατφόρμα του Eclipse. Το Eclipse με τη χρήση του EMF δίνει τη δυνατότητα στον χρήστη να ορίσει μοντέλα τα οποία υπακούν στο μεταμοντέλο του EMF, το Ecore. Με τη χρήση των εργαλείων που προσφέρει το Eclipse, ο σχεδιαστής έχει τη δυνατότητα, χρησιμοποιώντας το γραφικό περιβάλλον του Eclipse, να σχεδιάσει το μοντέλο του όπως θα το σχεδίαζε και στο χέρι ή με οποιοδήποτε άλλο σχεδιαστικό εργαλείο. Μέσω μιας αυτοματοποιημένης διαδικασίας , με αρχική πηγή πληροφοριών μόνο το γράφημα του μοντέλου, μπορεί να παράξει μια ολοκληρωμένη διεπαφή προγραμματισμού εφαρμογών (API) για την παραγωγή στιγμιοτύπων του μοντέλου καθώς και μια λειτουργική εφαρμογή παραγωγής στιγμιοτύπων.



Σχήμα 3.1: Η διαδικασία ορισμού μοντέλων

Όπως φαίνεται και στο σχήμα 3.1 ο σχεδιαστής δημιουργεί ένα αρχείο ecorediag το οποίο περιέχει την γραφική απεικόνιση του μοντέλου του σχεδιαστή. Καθώς ο χρήστης σχεδιάζει το διάγραμμα του μοντέλου του , το Eclipse παράγει ταυτόχρονα το αρχείο .ecore το οποίο είναι το μοντέλο του συστήματος σειριοποιημένο σε ένα αρχείο. Το αρχείο ecore είναι απόλυτα συμβατο με το πρότυπο XMI της OMG. Χρησιμοποιώντας τη βιβλιοθήκη εργαλείων του EMF μπορεί ο σχεδιαστής να παράξει ένα αρχείο .genmodel, το οποίο είναι και αυτό συμβατό με το πρότυπο XMI και περιέχει όλες τις πληροφορίες που περιέχει το αρχείο ecore, έχει όμως επιπρόσθετα, πληροφορίες για την παραγωγή του API. Από το αρχείο αυτό, παράγεται με τη βοήθεια της βιβλιοθήκης εργαλείων του EMF το API του μοντέλου του σχεδιαστή. Ο σχεδιαστής μετά την παραγωγή του API, έχει τη δυνατότητα να παράξει στιγμιότυπα του μοντέλου του, τα οποία αποθηκεύονται αυτόματα σε σειριοποιημένη μορφή σε αρχεια XMI. Τα στιγμιότυπα του μοντέλου, μπορούν να παραχθούν είτε προγραμματιστικά με τη χρήση του

API, είτε με την χρήση του editor που κατασκευάζει το Eclipse, αποκλειστικά για τα στιγμιότυπα του μοντέλου που όρισε ο σχεδιαστής. Με αυτόν τον τρόπο ο σχεδιαστής λογισμικού από ένα απλό διάγραμμα του μοντέλου του παράγει ένα εργαλείο, το οποίο του επιτρέπει να υλοποιήσει το μοντέλο του χωρίς να χρειαστεί να εμπλακεί με τη διαχείριση των αρχείων XMI.

3.3 Θεωρητικό Υπόβαθρο της Υλοποίησης

Αφού περιγράψαμε στην προηγούμενη ενότητα περιληπτικά την διαδικασία μετασχηματισμού των ακολουθιακών διαγραμμάτων σε διαγράμματα κώδικα θα αναπτύξουμε σε αυτό το σημείο αναλυτικά με τη χρήση μαθηματικού φορμαλισμού την διαδικασία βήμα προς βήμα.

Έστω λοιπόν ότι έχουμε ένα σύνολο ακολουθιακών διαγραμμάτων $S=\{s_1,s_2,s_3,...s_i,...,s_n\}$ όπου s_i κάποιο ακολουθιακό διάγραμμα του συστήματος που μελετάμε. Το πρόβλημα που προσπαθούμε να επιλύσουμε σε αυτήν την διπλωματική εργασία είναι ο ορισμός ενός μετασχηματισμού T του συνόλου των ακολουθιακών διαγραμμάτων S σε ένα διάγραμμα κώδικα έστω c. Το διάγραμμα c θα περιέχει τη σύνθεση των πληροφοριών που περιέχονται στα ακολουθιακά διαγράμματα του συνόλου S, αλλά και πληροφορίες που προέρχονται από την δευτερογενή επεξεργασία των πληροφοριών που εξάγονται από τα ακολουθιακά διαγράμματα. Για να το επιτύχουμε αυτό, σχηματίζουμε ένα αρχικό διάγραμμα κώδικα, έστω c_{temp}, εφαρμόζοντας ένα μετασχηματισμό στα ακολουθιακά διαγράμματα και ύστερα το επεξεργαζόμαστε ώστε να το συμπληρώσουμε με πληροφορίες που εξάγονται από την επεξεργασία αυτού του αρχικού διαγράμματος κώδικα. Σύμφωνα με την προσέγγιση αυτή, ο μετασχηματισμός T αναλύεται σε 2 μετασχηματισμούς έστω P και Q ,όπου ο P εφαρμόζεται πάνω στο σύνολο των ακολουθιακών διαγραμμάτων S, ενώ ο Q εφαρμόζεται πάνω στο διάγραμμα κώδικα c_{temp} που παράγεται από την εφαρμογή του P μετασχηματισμού παράγοντας το τελικό διάγραμμα κώδικα c_{final}.

Για να ορίσουμε τον μετασχηματισμό T, πρέπει πρώτα να ορίσουμε τους μετασχηματισμούς P και Q. Θεωρούμε τον χώρο των ακολουθιακών διαγραμμάτων $\mathfrak{S}$ καθώς και το δυναμοσύνολο του $\mathfrak{S}$, $\mathcal{P}(\mathfrak{S})$ το οποίο περιέχει

όλα τα δυνατά υποσύνολα του $\mathcal{S}$. Θεωρούμε επίσης το σύνολο των διαγραμμάτων κώδικα $\mathcal{C}$. Τότε ο μετασχηματισμός P που επιζητούμε να βρούμε έχει την εξής μορφή:

$$P: \mathcal{P}(\mathcal{S}) \rightarrow \mathcal{C} \qquad (1)$$

Επειδή είναι αδύνατο να γίνει παράλληλα η επεξεργασία όλων των ακολουθιακών διαγραμμάτων, επεξεργαζόμαστε τα διαγράμματα του συνόλου S σειριακά και σχηματίζουμε αυξητικά το διάγραμμα c_{temp} . Για να το επιτύχουμε αυτό θεωρούμε τον μετασχηματισμό P'. Ο μετασχηματισμός P' εφαρμόζεται διαδοχικά σε όλα τα ακολουθιακά διαγράμματα παράγοντας το τελικό διάγραμμα κώδικα όταν εφαρμοστεί σε όλα τα ακολουθιακά διαγράμματα. Ο μετασχηματισμός P' παίρνει σαν ορίσματα το υπό επεξεργασία ακολουθιακό διάγραμμα και το διάγραμμα κώδικα που παρήχθη από την προηγούμενη εφαρμογή του μετασχηματισμού. Σύμφωνα με τα προηγούμενα, όταν εφαρμόσουμε τον μετασχηματισμό P' στα στοιχεία ενός συνόλου ακολουθιακών διαγραμμάτων $S=\{s_1,s_2,s_3,...s_i,...,s_n\}$, ισχύουν τα εξής:

$$P': \mathcal{S} \times \mathcal{C} \rightarrow \mathcal{C}$$
$$c_i = P'(s_i, c_{i-1}) \quad i > 1$$
$$c_1 = P'(s_1) \qquad (2)$$

Το διάγραμμα κώδικα c_{temp} είναι το διάγραμμα κώδικα c_n όπου n είναι το πλήθος των ακολουθιακών διαγραμμάτων του συνόλου S.

Πιο αναλυτικά ο μετασχηματισμός P' παράγει από το διάγραμμα s_i ένα διάγραμμα c_i' το οποίο συντίθεται με το διάγραμμα c_{i-1} για να παραχθεί το διάγραμμα c_i. Κάθε διάγραμμα κώδικα περιέχει ένα σύνολο στοιχείων E τα οποία συνδέονται μεταξύ τους μέσω ενός συνόλου σχέσεων R. Ισχύει δηλαδή για κάθε διάγραμμα το εξής:

$$c=\{E,R\} \qquad (3)$$

Αν θεωρήσουμε ότι το διάγραμμα κώδικα προκύπτει από την εφαρμογή του μετασχηματισμού P' σε κάποιο ακολουθιακό διάγραμμα s_i, τότε ο παραπάνω τύπος γράφεται ως εξής:

$$c_i=\{E_i , R_i\} \qquad (4)$$

Το διάγραμμα κώδικα c_i' το οποίο παράγεται κατά τη διάρκεια του μετασχηματισμού P' από το διάγραμμα s_i γράφεται σύμφωνα με τον τύπο (3) ως εξής:

$$c_i'=\{E_i',R_i'\} \qquad (5)$$

Αν θεωρήσουμε μια τυχαία εφαρμογή του μετασχηματισμού P' τότε θα έχουμε σύμφωνα με τον τύπο (2) ότι

$$c_i =P'(s_i, c_{i-1}) \qquad (6)$$

Τα στοιχεία που περιέχονται στο c_i είναι τα κοινά και μη κοινά στοιχεία των c_i και c_i'. Ισχύει δηλαδή ότι:

$$E_i = E_{i-1} \cup E_i' \qquad (7)$$

Για να προσδιορίσουμε το περιεχόμενο του συνόλου R_i, θα πρέπει να εξετάσουμε την φύση των σχέσεων, τον τρόπο δηλαδή που απεικονίζονται οι σχέσεις στο διάγραμμα. Όπως θα φανεί στις επόμενες ενότητες της περιγραφής της αρχιτεκτονικής της υλοποίησης οι σχέσεις μεταξύ των στοιχείων των διαγραμμάτων κώδικα είναι και αυτές στοιχεία, με μόνη διαφορά ότι τα δεδομένα που περιέχουν είναι στην ουσία διευθύνσεις σε άλλα στοιχεία. Αν θεωρήσουμε ότι τα διαγράμματα είναι συντακτικά σωστά και δεν περιέχουν αντιφάσεις και συγκρούσεις μεταξύ τους, τότε και για τις σχέσεις ισχύουν τα ίδια με τα στοιχεία των διαγραμμάτων. Έχουμε δηλαδή ότι:

$$R_i = R_{i-1} \cup R_i' \qquad (8)$$

Εφαρμόζοντας διαδοχικά τον μετασχηματισμό P' στα στοιχεία του συνόλου $S=\{s_1,s_2,s_3,...s_i,...,s_n\}$ έχουμε σαν αποτέλεσμα το διάγραμμα κώδικα $c_n=c_{temp}$ για το οποίο με εφαρμογή της επαγωγής ισχύει ότι

$$c_{temp} = (\cup E_i', \cup R_i') \qquad (8)$$

Την διαδοχική εφαρμογή του μετασχηματισμού P' στα στοιχεία του συνόλου

S την γράφουμε συνοπτικά ως έναν ενιαίο μετασχηματισμό P που έχει την μορφή του τύπου (1) και για τον οποίο ισχύει ότι:

$$c_{temp}=P(S) \qquad (9)$$

Το τελικό διάγραμμα προκύπτει από την εφαρμογή του μετασχηματισμού Q στο διάγραμμα κώδικα c_{temp}. Ο μετασχηματισμός Q είναι της μορφής

$$Q: \mathcal{C} \rightarrow \mathcal{C} \qquad (10)$$

Άρα για το τελικό διάγραμμα κώδικα c_{final} ισχύει ότι :

$$c_{final}=Q(c_{temp}) \qquad (11)$$

Από τις σχέσεις (9) και (11) προκύπτει ότι

$$c_{final}=Q(P(S)) \qquad (12)$$

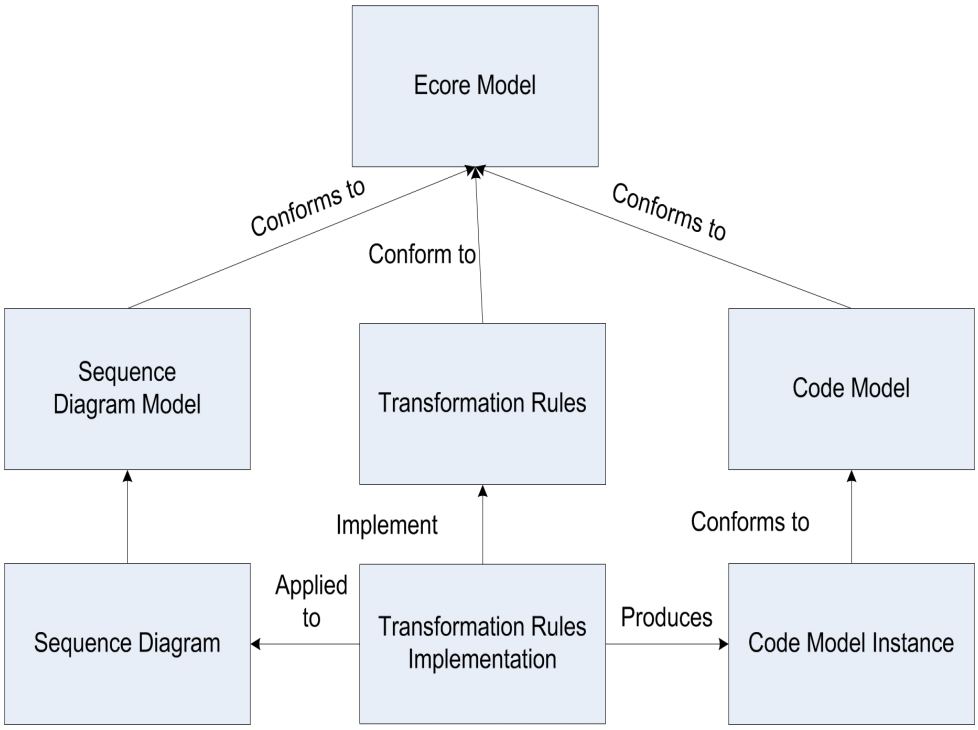
Η σύνθεση των δύο μετασχηματισμών που γίνεται στη σχέση (12) αποτελεί και το μετασχηματισμό T που επιζητούμε. Η μορφή του T προκύπτει από τις προηγούμενες σχέσεις και είναι η εξής:

$$T: \mathcal{P}(\mathcal{S}) \rightarrow \mathcal{C}$$
$$c_{final}=T(S) \qquad (13)$$

όπου $c_{final} \in \mathcal{C}$ και $s \in \mathcal{S}$

3.4 Περιγραφή της αρχιτεκτονικής της υλοποίησης.

Πριν την ανάλυση των των δομικών στοιχείων της εφαρμογής της εργασίας αυτής κρίνεται σκόπιμο να προηγηθεί μια συνοπτική περιγραφή της αρχιτεκτονικής της υλοποίησης του συστήματος όπως αυτή παρουσιάζεται στο σχήμα 3.2



Σχήμα 3.2: Η αρχιτεκτονική του συστήματος

Αρχικά έχουμε ορίσει το μοντέλο των ακολουθιακών διαγραμμάτων που θα επεξεργαστούμε σε αυτήν την εργασία. Το μοντέλο αυτό προδιαγράφει τα βασικότερα στοιχεία ενός ακολουθιακού διαγράμματος παραλείποντας μόνο στοιχεία που σπάνια χρησιμοποιούνται σε ακολουθιακά διαγράμματα ή συγχωνεύοντας παραπλήσιες δομές, όπως οι δομές alternative και option.

Γενικότερα τα χαρακτηριστικά που μοντελοποιούνται σε αυτό το μοντέλο

είναι τα εξής:

• οι γραμμές ετοιμότητας

• η κλάση του αντικειμένου της κάθε γραμμής ετοιμότητας

• το όνομα του αντικειμένου-στιγμιοτύπου της κάθε γραμμής ετοιμότητας

• τα μηνύματα που ανταλλάσσονται μεταξύ των στιγμιοτύπων τα οποία χωρίζονται σε 3 κατηγορίες:

- κλήση μεθόδου
- δημιουργία αντικειμένου
- καταστροφή αντικειμένου.

• τα πεδία κάθε κλάσης που επηρεάζονται από την ανταλλαγή των μηνυμάτων

• οι παράμετροι που εμπεριέχουν τα μηνύματα και επηρεάζουν τα πεδία των κλάσεων

• τα σύνθετα τμήματα που μοντελοποιούν την ροή των μηνυμάτων τα οποία χωρίζονται στις εξής κατηγορίες:

- τμήματα alternative τα οποία είναι προαιρετικά συνδεδεμένα με τμήματα else
- τμήματα else συνδεδεμένα υποχρεωτικά με ακριβώς ένα alternative
- τμήματα βρόγχων(loop)

Με τα τμήματα alternative, μοντελοποιούμε τις δομές alternative και option του μοντέλου ακολουθιακών διαγραμμάτων της OMG, ενώ με το συνδυασμό τμημάτων alternative και else μοντελοποιούμε τις δομές alternative-else.
Με τα τμήματα loop μοντελοποιούμε κάθε είδους βρόγχο.

Στη συνέχεια έχουμε ορίσει το μοντέλο του κώδικα το οποίο περιγράφει τις πληροφορίες που θα περιέχει ο παραγόμενος κώδικας. Στο μοντέλο αυτό προδιαγράφεται η δομή του κώδικα των διαφόρων δομικών μονάδων του

συστήματος. Ο λόγος χρησιμοποίησης του μοντέλου του κώδικα αντί της απευθείας παραγωγής κώδικα σε συγκεκριμένη γλώσσα προγραμματισμού, είναι το γεγονός ότι θέλουμε να περιγράψουμε τη δομή του κώδικα του συστήματος ανεξάρτητα από την τελική γλώσσα υλοποίησης. Η μόνη υπόθεση που κάνουμε, είναι ότι η γλώσσα υλοποίησης είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού.

Το μοντέλο του κώδικα, παρουσιάζει αρκετές ομοιότητες με το μοντέλο των διαγραμμάτων κλάσεων και αυτό οφείλεται στο γεγονός ότι η δομή των διαγραμμάτων κλάσεων είναι παραπλήσια της δομής του κώδικα των αντικειμενοστραφών γλωσσών προγραμματισμού. Δεν πρέπει όμως να συγχέονται τα δύο μοντέλα μεταξύ τους διότι το διάγραμμα κλάσεων έχει σκοπό να περιγράψει τη δομή ενός συστήματος, ενώ ένα στιγμιότυπο του μοντέλου που παρουσιάζουμε, έχει ως σκοπό να περιγράψει την δομή του κώδικα του συστήματος.

Με τη χρήση του μοντέλου αυτού περιγράφουμε τα σημαντικότερα στοιχεία του κώδικα τα οποία μπορούν να εξαχθούν από την επεξεργασία του ακολουθιακού διαγράμματος.

Τα στοιχεία του κώδικα που μοντελοποιούμε είναι τα εξής:

• οι κλάσεις του συστήματος.

• τη κληρονομικότητα μεταξύ των κλάσεων (οι υπερκλάσεις και οι υποκλάσεις του συστήματος καθώς και τα interfaces).

• τα πεδία κάθε κλάσης και ο τύπος δεδομένων τους.

• οι μέθοδοι κάθε κλάσης οι παράμετροι και ο τύπος τους καθώς και ο τύπος των δεδομένων που επιστρέφουν.

• οι βρόγχοι, η συνθήκη ελέγχου του βρόγχου, η μέθοδος που τους περιέχει καθώς η θέση τους μέσα στο σώμα της μεθόδου.

• οι διακλαδώσεις, οι συνθήκες διακλάδωσης, η μέθοδος που περιέχει την διακλάδωση καθώς και η θέση τους μέσα στο σώμα της μεθόδου.

• τις κλήσεις των μεθόδων και τη δημιουργία αντικειμένων που περιέχονται

στο σώμα κάθε μεθόδου καθώς και η σειρά με την οποία συμβαίνουν.
Για να μεταβούμε από τα ακολουθιακά διαγράμματα σε στιγμιότυπα του μοντέλου κώδικα χρησιμοποιούμε ένα σύνολο κανόνων συμπερασμού της δομής του κώδικα. Οι κανόνες αυτοί υλοποιούνται από αντίστοιχους αλγορίθμους οι οποίοι συλλέγουν πληροφορίες από τα ακολουθιακά διαγράμματα και μορφοποιούν ένα ή περισσότερα στιγμιότυπα του μοντέλου κώδικα.

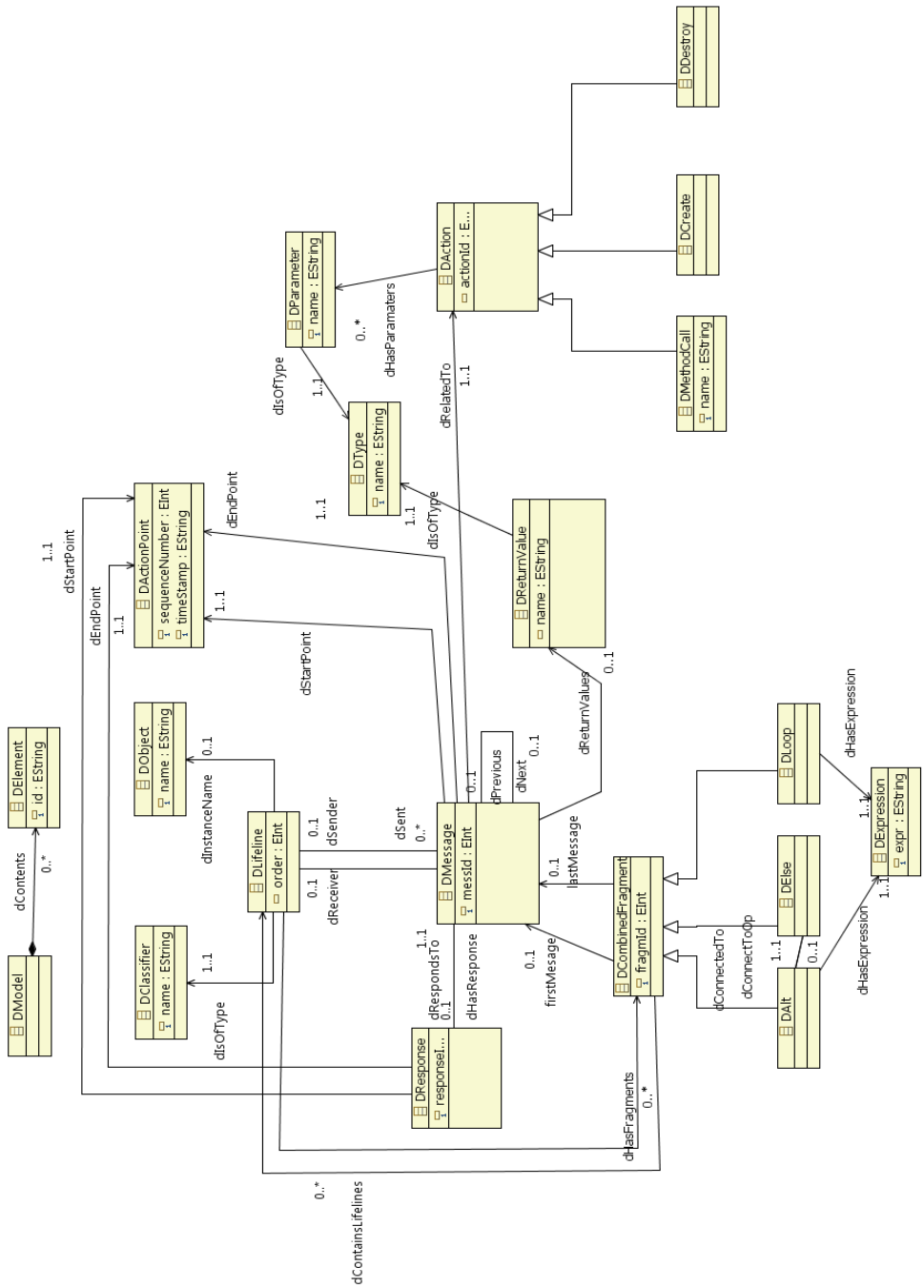
Οι κανόνες που εφαρμόζουμε προσδιορίζουν τα εξής στοιχεία:

- τις κλάσεις του συστήματος.
- τους τύπους δεδομένων που χρησιμοποιούνται απο το σύστημα
- τις μεθόδους των κλάσεων και των παραμέτρων τους.
- τα πεδία των κλάσεων.
- την κληρονομικότητα μεταξύ των κλάσεων
- τους βρόγχους και την μέθοδο στην οποία ανήκουν
- τις διακλαδώσεις και την μέθοδο στην οποία ανήκουν

Οι κανόνες συμπερασμού της δομής του κώδικα αποτελούν στην ουσία ένα σύνολο κανόνων μετασχηματισμού ενός συνόλου ακολουθιακών διαγραμμάτων σε ένα διάγραμμα κώδικα. Οι κανόνες αυτοί χρησιμοποιούν το μοντέλο ακολουθιακών διαγραμμάτων και το μοντέλο κώδικα για να παράξουν ένα αρχείο XML το οποίο θα αποτελεί στιγμιότυπο του μοντέλου κώδικα που έχουμε ορίσει.

3.5 Ορισμός του μοντέλου του ακολουθιακού διαγράμματος.

Αφού αναλύσαμε την γενικότερη αρχιτεκτονική της εργασίας αυτής μπορούμε να περιγράψουμε λεπτομερώς τα επιμέρους δομικά και λειτουργικά στοιχεία της εργασίας. Στο σχήμα 3.3 φαίνεται το μοντέλο του ακολουθιακού διαγράμματος που χρησιμοποιούμε σε αυτήν την εργασία. Το μοντέλο αυτό δείχνει τις πληροφορίες που αναμένουμε να παρέχει



Σχήμα 3.3: Μοντέλο ακολουθιακού διαγράμματος

ένα ακολουθιακό διάγραμμα αλλά και τη δομή αυτών των πληροφοριών. Ταυτόχρονα, με το σχήμα αυτό καθορίζουμε τη μορφή που θα έχει το XML κείμενο στο οποίο θα είναι αποθηκευμένο το ακολουθιακό διάγραμμα. Το αρχείο XML που παράγεται αυτόματα από την πλατφόρμα του Eclipse είναι συμβατό με το μοντέλο XMI του οργανισμού OMG.

Κύριο στοιχείο του μοντέλου μας είναι το στοιχείο DModel. Το στοιχείο DModel έχει μια σχέση συσσώρευσης με κάθε στοιχείο του διαγράμματος (DElement). Αυτό σημαίνει ότι το DModel αποτελεί την ρίζα του XML κειμένου στο οποίο είναι αποθηκευμένο το ακολουθιακό διάγραμμα. Επίσης η σχέση συσσώρευσης υπονοεί ότι κάθε DElement εμπεριέχεται στο στοιχείο DModel. Αυτό αντικατοπτρίζεται άμεσα στο XML αρχείο. Κάθε XML στοιχείο (XML element) DElement, ορίζεται μέσα στο XML στοιχείο DModel.

DElement είναι κάθε στοιχείο του ακολουθιακού διαγράμματος εκτός από το DModel. Κάθε στοιχείο του ακολουθιακού διαγράμματος έχει δηλαδή μία σχέση γενίκευσης (ISA) με το DElement (π.χ DMessage is a DElement). Η σχέση αυτή της γενίκευσης σε συνδυασμό με τη σχέση συσσώρευσης που έχει το DModel με το DElement συνεπάγεται ότι όλα τα στοιχεία του ακολουθιακού διαγράμματος βρίσκονται στο ίδιο βάθος μέσα στο XML αρχείο. Είναι δηλαδή όλα παιδιά του DModel.

Πριν περιγράψουμε την σημασία κάθε στοιχείου του διαγράμματος είναι σκόπιμο να αναφέρουμε ότι κατά κανόνα, κάθε κατευθυνόμενη σχέση στο διάγραμμα απεικονίζεται στο XML αρχείο ως ένα ή περισσότερα παιδιά του XML στοιχείου που αποτελεί το στοιχείο πηγή και κάθε τέτοιο παιδί περιέχει μια αναφορά στο στοιχείο στόχο. Η αναφορά στο στοιχείο στόχο γίνεται με βάση ένα πεδίο κάθε στοιχείου το οποίο αποτελεί την ταυτότητα κάθε στοιχείου και είναι μοναδικό μέσα στο κείμενο XML. Η σημαντική διαφορά μιας απλής κατευθυνόμενης σχέσης και μιας σχέσης συσσώρευσης είναι ότι στην σχέση συσσώρευσης το στοιχείο-παιδί ορίζεται μέσα στο στοιχείο-πρόγονο που το περιέχει. Αντίθετα σε μία απλή κατευθυνόμενη σχέση το στοιχείο-παιδί αποτελεί στην ουσία δείκτη σε άλλο στοιχείο.

Κάθε αμφίδρομη σχέση μπορεί να αναλυθεί σε δύο κατευθυνόμενες σχέσεις. Στη μία σχέση τα στοιχεία στόχοι αποτελούν παιδιά του στοιχείου στόχου ενώ στην αντίθετη σχέση τα στοιχεία στόχοι είναι χαρακτηριστικά (XML attributes) του στοιχείου πηγής. Ο λόγος που ακολουθείται αυτή η

στρατηγική από το Eclipse είναι για να αποφεύγονται οι ατέρμονες βρόγχοι κατά την επεξεργασία και ανάγνωση ενός αρχείου XML από το Eclipse. Ακόμα και αν δεν υπάρχει αμφίδρομη σχέση αλλά οι κατευθυνόμενες σχέσεις σχηματίζουν βρόγχο θα πρέπει για μία από αυτές τις σχέσεις να ορίσουμε ότι τα στοιχεία στόχοι αποτελούν χαρακτηριστικά του στοιχείου-πηγή και όχι παιδιά.

Αφού δείξαμε τη συσχέτιση του μοντέλου και της δομής του XML αρχείου θα αναλύσουμε τα στοιχεία του μοντέλου του ακολουθιακού διαγράμματος και θα περιγράψουμε την πληροφορία που αυτά περιέχουν. Κάθε στοιχείο DElement χαρακτηρίζεται μοναδικά από ένα χαρακτηριστικό με όνομα id. Δεν υπάρχει κανένας περιορισμός για τον τύπο αυτού του χαρακτηριστικού ούτε για τον τρόπο απόδοσης τιμής σε αυτό. Σε αυτήν την εργασία χρησιμοποιήσαμε id τύπου String για λόγους ευχρηστίας.

Κάθε λειτουργική μονάδα του συστήματος που δέχεται και λαμβάνει μηνύματα μοντελοποιείται στο ακολουθιακό διάγραμμα με μία γραμμή ετοιμότητας(DLifeline). Κάθε γραμμή ετοιμότητας χαρακτηρίζεται από έναν αύξοντα αριθμό. Κάθε γραμμή ετοιμότητας σχετίζεται με ακριβώς μία κλάση(DClassifier) και με μηδέν ή ένα αντικείμενο (DObject). Κάθε κλάση χαρακτηρίζεται από ένα όνομα. Κάθε αντικείμενο χαρακτηρίζεται επίσης από ένα όνομα.

Κάθε γραμμή ετοιμότητας σχετίζεται με δύο σύνολα μηνυμάτων (DMessage). Τα σύνολα αυτά είναι τα μηνύματα που έστειλε η συγκεκριμένη γραμμή ετοιμότητας και τα μηνύματα που έλαβε. Κάθε μήνυμα (DMessage) σχετίζεται με δύο σημεία δραστηριότητας (DActionPoint), ένα σημείο αναχώρησης (dStartPoint) και ένα σημείο άφιξης (dEndPoint). Στα σημεία δραστηριότητας καταχωρείται η θέση της δραστηριότητας στο ακολουθιακό διάγραμμα και η χρονική στιγμή που αυτή συνέβη. Κάθε μήνυμα σχετίζεται επίσης και με μία δραστηριότητα (DAction) η οποία περιγράφει τον ρόλο του μηνύματος. Οι δραστηριότητες που μοντελοποιούνται σε αυτό το μοντέλο είναι η κλήση μεθόδου (DMethodCall), η δημιουργία νέου αντικειμένου (DCreate) και η καταστροφή αντικειμένου (DDestroy). Εάν υπάρχει μεταβλητή η οποία επηρεάζεται από τα δεδομένα που επιστρέφει κάποια δραστηριότητα, αυτή μοντελοποιείται μέσω του στοιχείου DReturnValue. Υπάρχει επιπρόσθετα η δυνατότητα να ορίσουμε και το γεγονός της επιστροφή ενός μηνύματος μέσω ενός στοιχείου επιστροφής DResponse. Το στοιχείο DResponse

σηματοδοτεί απλά πότε γίνεται η επιστροφή από το μήνυμα που εστάλη. Επίσης κάθε μήνυμα σχετίζεται με μία τιμή που επιστέφεται στην γραμμή ετοιμότητας που αποστέλλει το μήνυμα. Τέλος κάθε μήνυμα σχετίζεται με δύο άλλα μηνύματα, το προηγούμενο κατά σειρά μήνυμα του διαγράμματος και το επόμενο κατά σειρά μήνυμα του διαγράμματος.

Με αυτόν τον τρόπο μοντελοποίησης έχουμε μια πλήρη περιγραφή των μηνυμάτων όπως αυτά αναπαρίστανται στο ακολουθιακό διάγραμμα. Τα σημεία δραστηριότητας μαζί με τον συσχετισμό επόμενο-προηγούμενο μεταξύ των μηνυμάτων προσδιορίζουν ακριβώς την θέση των μηνυμάτων στο ακολουθιακό διάγραμμα. Η συσχετιζόμενη με το μήνυμα δραστηριότητα περιγράφει τον ρόλο του μηνύματος ενώ η απάντηση περιγράφει τα δεδομένα που επιστρέφονται στην καλούσα από την καλούμενη γραμμή ετοιμότητας.

Κάθε δραστηριότητα που είναι κλήση μεθόδου(DMethodCall) περιγράφεται με το όνομα της μεθόδου και τις παραμέτρους που δέχεται η μέθοδος, ενώ η επιστρεφόμενη τιμή περιγράφεται ανεξάρτητα μέσω ενός στοιχείου DReturnValue, σε συσχέτιση όμως με το μήνυμα με το οποίο σχετίζεται η μέθοδος που καλείται. Στα πλαίσια αυτής της εργασίας κάθε μέθοδος θεωρείται ότι χαρακτηρίζεται μοναδικά από το χαρακτηριστικό id που περιέχει ως στοιχείο DElement. Η παραδοχή αυτή γίνεται για να μην υπάρχει περιορισμός ως προς τον ορισμό της υπογραφής μιας μεθόδου. Μπορούμε με αυτόν τον τρόπο να ορίσουμε ότι οι μέθοδοι προσδιορίζονται είτε μόνο από το όνομα τους, είτε από το όνομα και τις παραμέτρους που δέχονται ή/και από τον τύπο των δεδομένων που επιστρέφουν.

Εκτός από την κλήση της μεθόδου έχουμε επίσης και δύο άλλες δραστηριότητες, την δημιουργία (DCreate) και την καταστροφή (DDestroy) αντικειμένου. Δεν κρίθηκε χρήσιμο να ενσωματώνουν οποιαδήποτε άλλη πληροφορία πέραν του μηνύματος της δημιουργίας ή καταστροφής και των χαρακτηριστικών που κληρονομούν από την υπερκλάση DAction.

Τέλος με το στοιχείο σύνθετο τμήμα (DCombinedFragment) περιγράφονται τα σύνθετα τμήματα ενός ακολουθιακού διαγράμματος. Τα σύνθετα τμήματα μπορούν να είναι το alternative (DAlt), το else (DElse),ή loop (DLoop). Κάθε σύνθετο τμήμα συσχετίζεται με μία συνθήκη (DExpression). Ένα στοιχείο DAlt μπορεί να συνδέεται προαιρετικά με ένα DElse αλλά κάθε DElse συσχετίζεται υποχρεωτικά με ένα DAlt. Ένα DAlt χωρίς DElse

μοντελοποιεί τη δομή option των ακολουθιακών διαγραμμάτων κατά OMG, ενώ ένα ζεύγος DAlt-DElse μοντελοποιεί το ζευγάρι alternative-else. Κάθε σύνθετο τμήμα συσχετίζεται με δύο μηνύματα, το πρώτο και το τελευταίο κατά σειρά μέσα στο σύνθετο τμήμα. Τέλος συσχετίζεται με ένα σύνολο γραμμών ετοιμότητας και αυτή η σχέση δηλώνει τις γραμμές ετοιμότητας που περιλαμβάνει το σύνθετο τμήμα.

3.6 Ορισμός του μοντέλου του κώδικα.

Στο σχήμα 3.4 φαίνεται το μοντέλο κώδικα που ορίσαμε και χρησιμοποιούμε σε αυτήν την εργασία. Με το μοντέλο αυτό προσπαθούμε να περιγράψουμε τη δομή του κώδικα του συστήματος υπο μελέτη. Όπως και στο ακολουθιακό διάγραμμα η δομή του μοντέλου καθορίζει και τη δομή του XML κειμένου στο οποίο αποθηκεύεται το στιγμιότυπο του μοντέλου της δομής του κώδικα. Κατ' αυτόν τον τρόπο κάθε κείμενο XML έχει ως στοιχείο-ρίζα το στοιχείο CDModel μέσα στο οποίο περιέχονται τα στοιχεία του διαγράμματος του κώδικα(CDElement). Κάθε στοιχείο του διαγράμματος κώδικα (στιγμιότυπο μοντέλου κώδικα) έχει μια σχέση γενίκευσης (ISA) με το στοιχείο CDElement. Το στοιχειο CDModel έχει μια σχέση συσσώρευσης με όλα τα στοιχεία CDElement. Κατά συνέπεια κάθε στοιχείο (CDElement) του διαγράμματος κώδικα είναι παιδί του CDModel. Κάθε στοιχείο CDElement χαρακτηρίζεται μοναδικά απο το χαρακτηριστικό(attribute) elemId. Όπως και με το χαρακτηριστικό id του στοιχείου DElement του ακολουθιακού διαγράμματος δεν υπάρχει περιορισμός ώς προς τον τύπο του χαρακτηριστικού αυτού ή του τρόπου ανάθεσης της τιμής του.

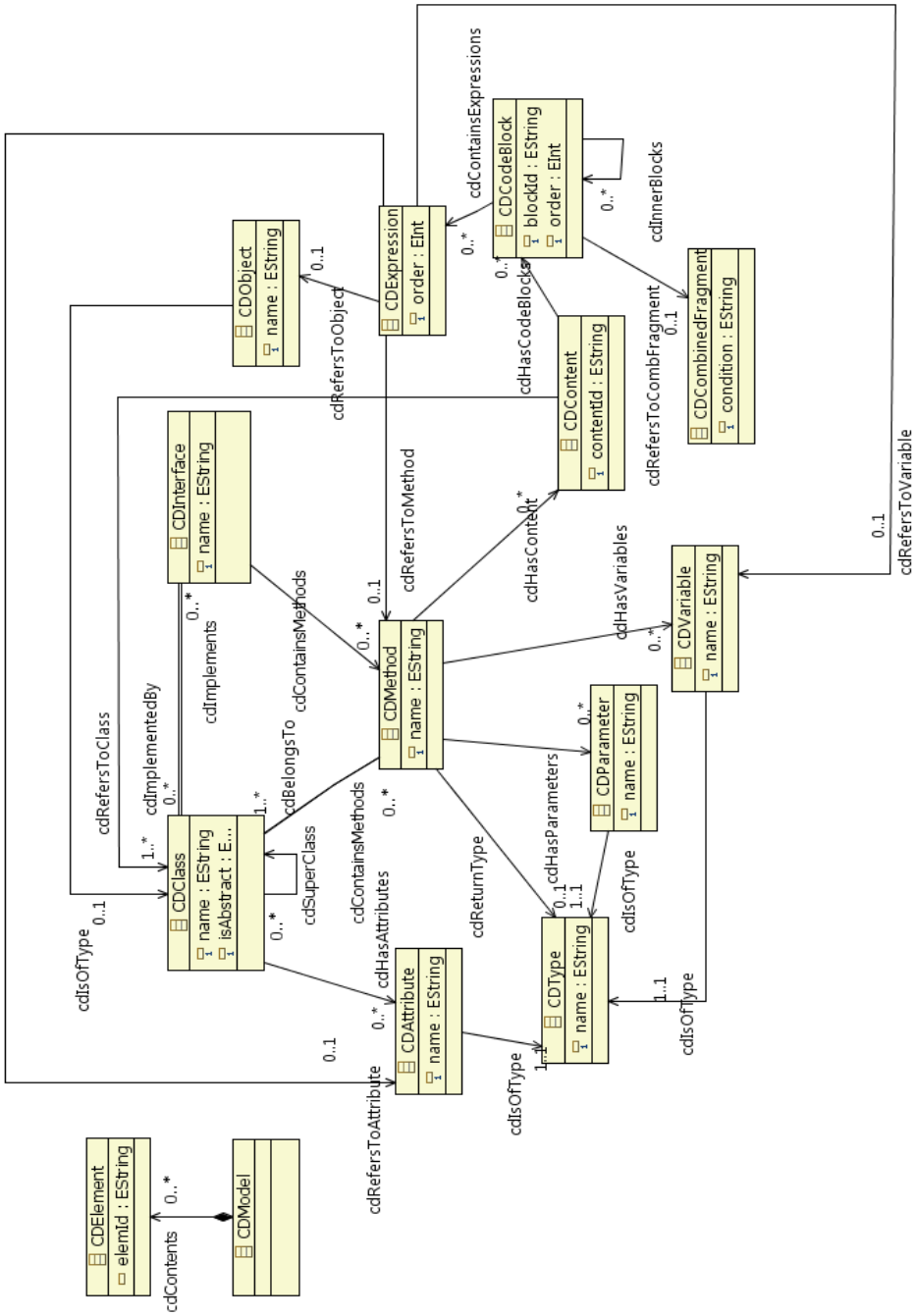
Η δομή του μοντέλου αυτού ακολουθεί τη δομή των αντικειμενοστραφών γλωσσών προγραμματισμού. Κατ' αυτόν τον τρόπο έχουμε τις κλάσεις (CDClass) οι οποίες χαρακτηρίζονται μοναδικά από το όνομα τους.Κάθε κλάση συσχετίζεται με ένα σύνολο πεδίων(CDAttribute). Κάθε πεδίο ορίζεται απο το όνομα του και σχετίζεται με έναν τύπο δεδομένων. Κάθε κλάση συσχετίζεται με ένα σύνολο μεθόδων(CDMethod). Σε αυτό το μοντέλο, οι μέθοδοι που υλοποιούνται στις κλάσεις του συστήματος δεν περιέχονται μέσα στις μεθόδους αλλά αποτελούν ανεξάρτητες οντότητες που συσχετίζονται με τις κλάσεις. Κάθε μέθοδος μπορεί προαιρετικά να συσχετιστεί και με μια διαπροσωπεία (CDInterface). Σε μία μέθοδο

μπορούμε επίσης να συσχετίσουμε ένα σύνολο παραμέτρων (CDParameter) όπου κάθε παράμετρος ορίζεται από το όνομα της και συσχετίζεται με έναν τύπο δεδομένων.

Κάθε κλάση επίσης μπορεί να έχει μία ή περισσότερες υπερκλάσεις και να υλοποιεί ένα ή περισσότερα interfaces. Αν και αυτή η μορφή κληρονομικότητας (πολλές διαπροσωπείες) δεν υποστηρίζεται σε μερικές αντικειμενοστραφείς γλώσσες, όπως η Java, στόχος μας είναι να εντοπίσουμε κοινές συμπεριφορές μεταξύ των κλάσεων. Τα μοντέλα που χρησιμοποιούνται και οι αλγόριθμοι που εφαρμόζονται έχουν σαν σκοπό την εφαρμογή τους σε όσο το δυνατόν περισσότερες αντικειμενοστραφείς γλώσσες προγραμματισμού και για αυτόν τον λόγο δεν ακολουθούμε τις ιδιαιτερότητες κάποιας συγκεκριμένης γλώσσας αλλά σχηματίζουμε δομές που μπορούν άμεσα να αντιστοιχιστούν με δομές οποιασδήποτε αντικειμενοστραφούς γλώσσας.

Κάθε μέθοδος επίσης σχετίζεται με ένα ή περισσότερα στοιχεία που μοντελοποιούν το περιεχόμενο της (CDContent). Αν μία μέθοδος περιέχεται σε περισσότερες της μιας κλάσεις, τότε μπορούμε να συσχετίσουμε κάθε διαφορετική υλοποίηση της μεθόδου με διαφορετικό CDContent. Κάθε στοιχείο CDContent συσχετίζεται με κομμάτια κώδικα (CDCodeBlock) τα οποία περιέχουν τις εντολές που καλούνται από μια μέθοδο. Επίσης κάθε στοιχείο CDContent σχετίζεται με μία ή περισσότερες κλάσεις (CDClass) που περιέχουν το συγκεκριμένο κώδικα.

Κάθε κομμάτι κώδικα (CDCodeBlock) μπορεί να περιλαμβάνει μια δομή ελέγχου (CDCombinedFragment). Αν ένα κομμάτι κώδικα περιλαμβάνει μια τέτοια δομή, αυτή η δομή χαρακτηρίζει όλο το κομμάτι κώδικα. Αν η μέθοδος που μοντελοποιείται, έχει κλήσεις που δεν επηρεάζονται από την συνθήκη αυτή, τότε το περιεχόμενό της μοντελοποιείται με περισσότερα του ενός κομματιών κώδικα. Κάθε συνθήκη ελέγχου μπορεί να συσχετιστεί με ένα αντικείμενο (CDObject) το οποίο συμμετέχει στην συνθήκη ελέγχου.



Σχήμα 3.4: Μοντέλο κώδικα

3.7 Τεχνική ανάλυσης των ακολουθιακών διαγραμμάτων

Ο μετασχηματισμός των ακολουθιακών διαγραμμάτων σε διαγράμματα κώδικα, γίνεται με τη χρήση ενός συνόλου κανόνων-αλγορίθμων οι οποίοι εφαρμόζονται διαδοχικά πάνω στα 2 μοντέλα με σκοπό την παραγωγή του κώδικα. Επειδή κάθε ακολουθιακό διάγραμμα δεν μοντελοποιεί το σύνολο ενός συστήματος, αλλά την επικοινωνία εκείνων μόνο των λειτουργικών μονάδων που συμμετέχουν σε ένα σενάριο χρήσης, θεωρήθηκε χρήσιμο να δοθεί η δυνατότητα ένα διάγραμμα κώδικα να μπορεί να προκύψει από την επεξεργασία πολλών ακολουθιακών διαγραμμάτων. Με αυτήν την προσέγγιση έχουμε τη δυνατότητα να συνθέσουμε τις διαφορετικές λειτουργίες μιας λειτουργικής μονάδας του συστήματος οι οποίες εμφανίζονται σε διαφορετικά σενάρια χρήσης και να έχουμε μια όσο το δυνατόν πληρέστερη εικόνα της λειτουργικής αυτής μονάδας και όχι την δημιουργία πολλών μικρών υπομονάδων με λίγες και ελλιπείς λειτουργίες .

Για την εφαρμογή της παραπάνω μεθοδολογίας οι κανόνες συμπερασμού της δομής του κώδικα χωρίζονται σε 2 κατηγορίες. Στην πρώτη κατηγορία ανήκουν οι κανόνες που μετασχηματίζουν στοιχεία του ακολουθιακού διαγράμματος σε στοιχεία του διαγράμματος κώδικα. Στην δεύτερη κατηγορία ανήκουν οι κανόνες που επεξεργάζονται αποκλειστικά τα δεδομένα που περιέχονται στο διάγραμμα κώδικα και μετασχηματίζουν το διάγραμμα κώδικα. Οι κανόνες της δεύτερης κατηγορίας έχουν περισσότερο σχέση με τον συμπερασμό της κληρονομικότητας μεταξύ των κλάσεων ενώ οι κανόνες της πρώτης κατηγορίας έχουν περισσότερο σχέση με την εσωτερική δομή των κλάσεων.

<u>Κατηγορία Α</u>

Όπως αναφέρθηκε και στην προηγούμενη παράγραφο οι κανόνες αυτής της κατηγορίας επεξεργάζονται τα δεδομένα των ακολουθιακών διαγραμμάτων και κατασκευάζουν το διάγραμμα κώδικα.

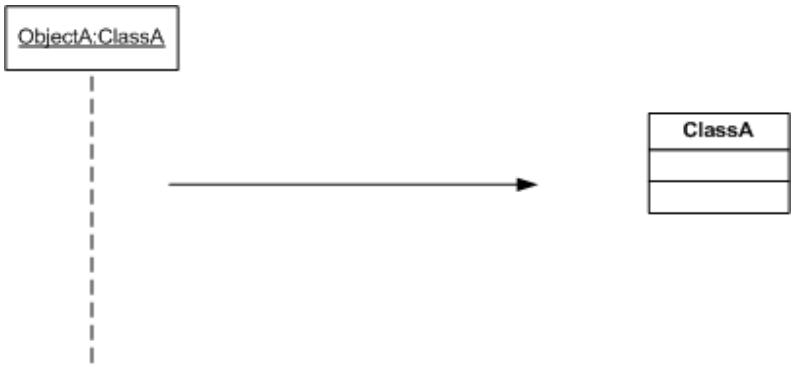
Κανόνας 1ος : Ορισμός κλάσεων.

Για κάθε γραμμή ετοιμότητας δημιουργούμε μια κλάση με όνομα το όνομα της κλάσης με την οποία συσχετίζεται η γραμμή ετοιμότητας. Αν η κλάση υπάρχει ήδη τότε δεν προσθέτουμε τίποτα.

Σε ψευδοκώδικα ο κάνονας γράφεται ως εξής:

Αλγόριθμος Α1

for each lifeline in sequence diagram
  do class=getClassifier(lifeline);
  if class not added in codeDiagram
  add(class,codeDiagram);
  endif
endfor

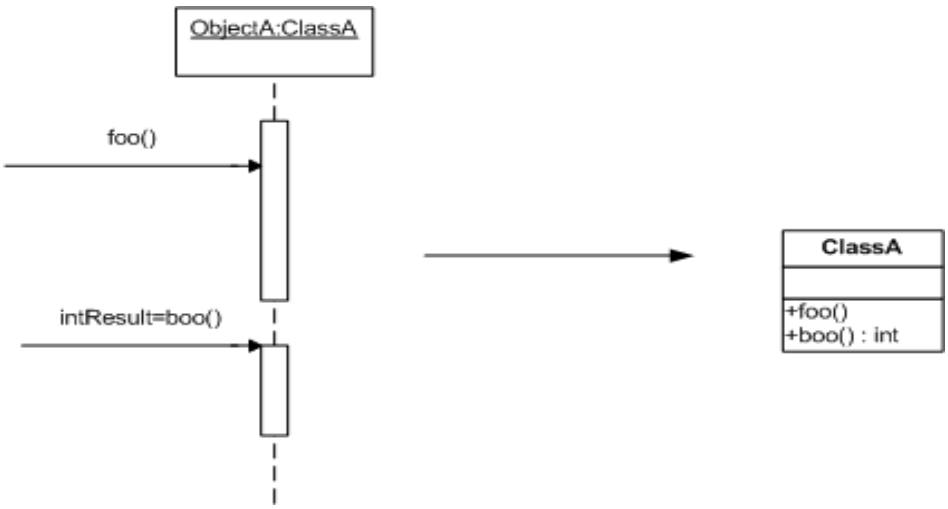


Σχήμα 3.5: Ο κανόνας ορισμού κλάσεων

Κανόνας 2ος : Ορισμός μεθόδων

Σε κάθε γραμμή ετοιμότητας διατρέχουμε το σύνολο των ληφθέντων μηνυμάτων. Για κάθε ληφθέν μήνυμα το οποίο είναι κλήση μεθόδου, προσθέτουμε στην κλάση την καλούμενη μέθοδο, εφόσον αυτή δεν έχει ήδη προστεθεί. Επειδή οι μέθοδοι στα διαγράμματα κώδικα είναι ανεξάρτητα στοιχεία από τις κλάσεις, στην πράξη ακολουθείται η διαδικασία που

φαίνεται στο σχήμα 3.7.

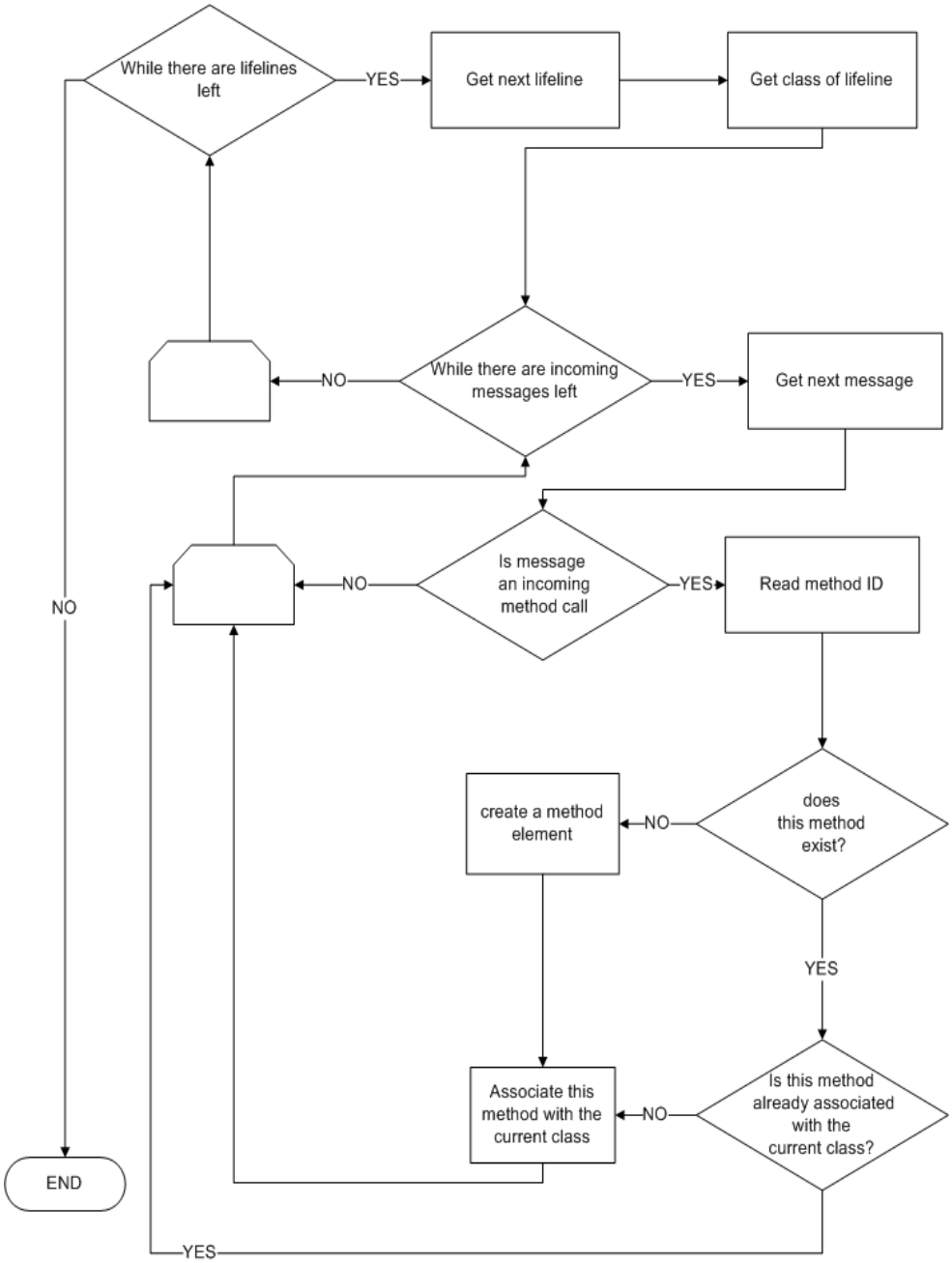


Σχήμα 3.6: Ο κανόνας ορισμού μεθόδων

Σε ψευδοκώδικα ο δεύτερος κανόνας γράφεται ως εξής:

Αλγόριθμος Α2

for each lifeline in sequenceDiagram
  do class=getClassifier(lifeline);
  for each incomingMessage of lifeline
  do if incomingMessage is MethodCall
  do methodName=getmethodName(incomingMessage);
  method=getMethod(codeDiagram, methodName);
  endif
  if method does not exist in classDiagram
  do create new method;
  endif
  if method not associated with class
  do associate(method,class);
  endif
  endfor
endfor



Σχήμα 3.7: Διάγραμμα διαδικασίας επεξεργασίας των μηνυμάτων κλήσης μεθόδων.

Κανόνας 3ος: Ορισμός πεδίων

βήμα 1ο :

Σε κάθε γραμμή ετοιμότητας διατρέχουμε τα μηνύματα που λαμβάνει η γραμμή ετοιμότητας και αν βρούμε κλήση μεθόδου με όνομα που ξεκινά από set ή get (setters, getters) προσθέτουμε αν δεν υπάρχει ήδη ένα πεδίο με την όνομα αυτό που ακολουθεί την λέξη set ή get. Ο τύπος του πεδίου είναι στην περίπτωση του set, ο τύπος της παραμέτρου που δέχεται η μέθοδος και στην περίπτωση του get ο τύπος δεδομένων που επιστρέφει η μέθοδος. Για παράδειγμα αν έχουμε μια κλήση της μεθόδου setAttributeA((int)10) τότε ορίζουμε το πεδίο attributeA με τύπο δεδομένων int. Ο αλγόριθμος για αυτό το σκέλος του κανόνα μπορεί να ενσωματωθεί στον αλγόριθμο υλοποίησης του δεύτερου κανόνα. Το βήμα αυτό είναι προαιρετικό και μπορούμε να επιλέξουμε αν θέλουμε να χρησιμοποιήσουμε μόνο τις εντολές set, τις εντολές get, ή και τις 2.

Αν το μήνυμα που λαμβάνεται είναι μήνυμα δημιουργίας, τότε οι παράμετροι που ενδεχομένως έχει το μήνυμα δημιουργίας είναι πεδία της κλάσης που αντιστοιχεί στην γραμμή ετοιμότητας που λαμβάνει το μήνυμα δημιουργίας.

Αλγόριθμος Α3α

```
for each lifeline in sequenceDiagram
    do class=getClassifier(lifeline);
    for each incomingMessage of lifeline
        do if incomingMessage is MethodCall
            do if incomingMessage is setter (or getter)
                do createAttribute(class,incomingMessage);
            endif
        else if incomingMesssage is createMessage
            do parameterList=getParameterList(createMessage);
            for each parameter in parameterList
                do createAttribute(class,parameter);
            endfor
        endif
    endfor
endfor
```

βήμα 2ο:

Σε κάθε γραμμή ετοιμότητας διατρέχουμε τα μηνύματα που στέλνει η γραμμή ετοιμότητας και εντοπίζουμε τις δημιουργίες αντικειμένων («create»). Αν δεν υπάρχει μέθοδος του αντικειμένου της γραμμής ετοιμότητας που δεν έχει επιστρέψει έως την αποστολή του μηνύματος της δημιουργίας αντικειμένου τότε δημιουργούμε ένα πεδίο με τύπο δεδομένων την κλάση του αντικειμένου που έλαβε το μήνυμα δημιουργίας.

Ο αλγόριθμος του δευτερου βήματος σε ψευδοκώδικα είναι ο εξής:

Αλγόριθμος Α3β

```
for each lifeline in sequenceDiagram
    do class=getClassifier(lifeline);
    boolean allRetruned=true;
    for each outgoingMessage in lifeline
        do if outgoingMessage is createMessage
            do incomingMessages= all incoming Messages of lifeline;
            while(incomingMessages.notEmpty())
                do lastMessage=incomingMessages.popLastMessage();
                if lastMessage notReturned
                    do allReturned=false;
                    incomingMessages.empty();
                endif
            endwhile
        endif
    endfor
endfor
```

βήμα 3ο :

Σε κάθε γραμμή ετοιμότητας διατρέχουμε τα μηνύματα που στέλνει η γραμμή ετοιμότητας και εντοπίζουμε τις κλήσεις μεθόδων. Από κάθε κλήση μεθόδου εντοπίζουμε, αν υπάρχει, την μεταβλητή που η τιμή της ορίζεται

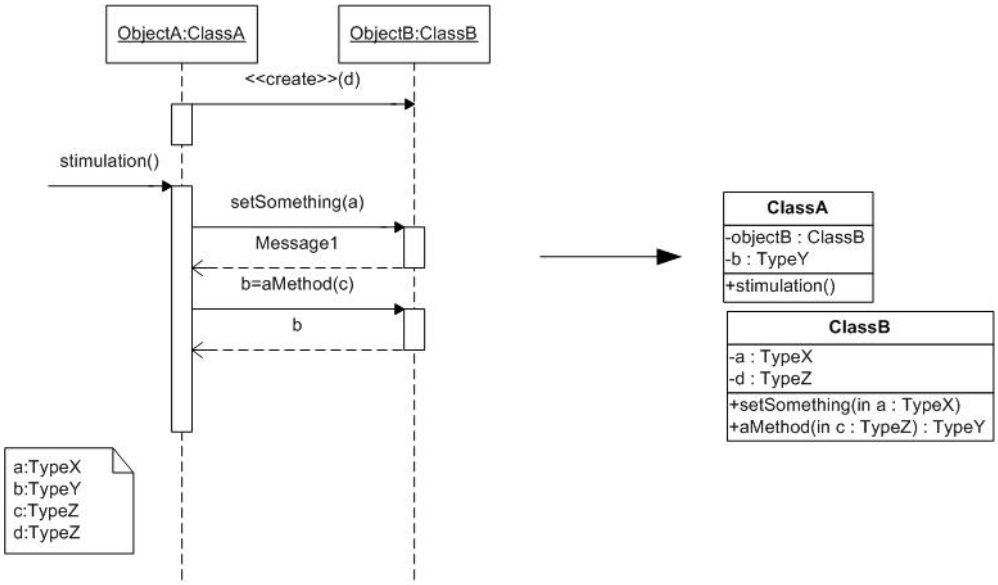
από την τιμή επιστροφής της μεθόδου. Ορίζουμε πεδίο με όνομα το όνομα της μεταβλητής και τύπο δεδομένων τον τύπο δεδομένων που επιστρέφει η μέθοδος που καλείται.

Το σκεπτικό πίσω από αυτόν τον κανόνα είναι ότι στα ακολουθιακά διαγράμματα αναφέρονται κυρίως οι ανταλλαγές των δεδομένων που έχουν σχέση με τα πεδία των κλάσεων ή τουλάχιστον παίζουν σημαντικό ρόλο στην επικοινωνία των κλάσεων που συμμετέχουν στο ακολουθιακό διάγραμμα και δεν αναφέρονται συχνά μεταβλητές που έχουν τοπικό χαρακτήρα και ο ρόλος τους είναι κυρίως αυτός της προσωρινής αποθήκευσης.

Ο αλγόριθμος του τρίτου βήματος σε ψευδοκώδικα είναι ο εξής:

Αλγόριθμος Α3γ

```
for each lifeline in sequenceDiagram
    do class=getClassifier(lifeline);
    for each outgoingMessage in lifeline
        do if outgoingMessage is MethodCall
            do fieldName=outgoingMessage.getReturnValue().getName();
            fieldType=outgoingMessage.getReturnValue().getType();
        endif
        field=new Field(fieldName,fieldType);
        if field does not exist in codeDiagram
            do add(codeDiagram,field);
        endif
        if field not associated with class
            do associate (class,field);
        endif
    endfor
endfor
```



Σχήμα 3.8: Ο κανόνας ορισμού πεδίων.

Κανόνας 4ος: Ορισμός του περιεχομένου μιας μεθόδου.

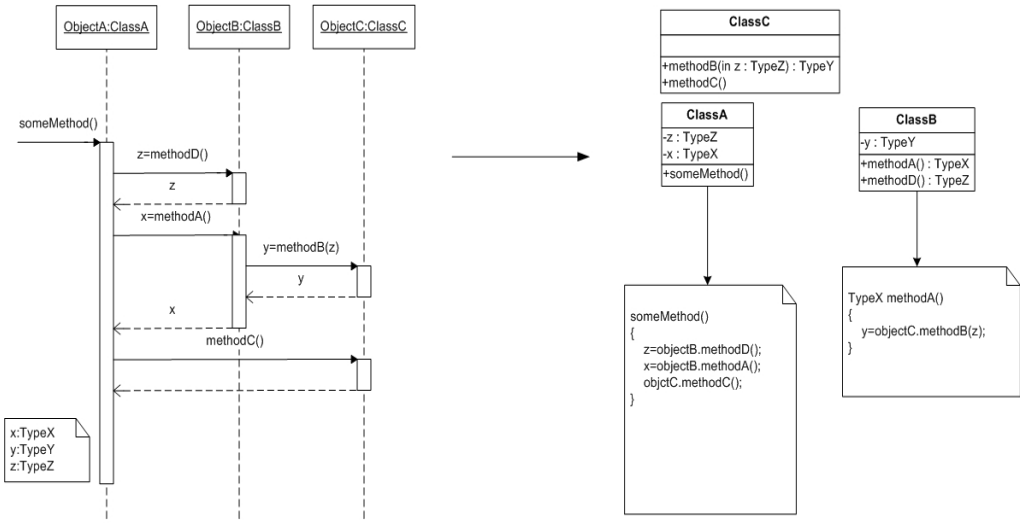
Εκτός από τον προσδιορισμό των μεθόδων που περιέχει μία κλάση, επιθυμούμε να προσδιορίσουμε επίσης και τις μεθόδους που καλούνται από το σώμα κάθε μεθόδου όπου αυτό είναι δυνατόν. Επιθυμούμε επίσης να προσδιορίσουμε κατά το δυνατόν τις τοπικές μεταβλητές που εμπεριέχονται στο σώμα της μεθόδου. Για να ορίσουμε το περιεχόμενο του σώματος της μεθόδου εντοπίζουμε το σημείο κλήσης της μεθόδου στην γραμμή ετοιμότητας και εξετάζουμε τα μηνύματα που στέλνονται από την γραμμή ετοιμότητας από εκείνο το σημείο έως το σημείο επιστροφής της μεθόδου. Οι μέθοδοι που καλούνται σε αυτό το διάστημα από την γραμμή ετοιμότητας που έχει δεχθεί το μήνυμα της κλήσης της μεθόδου, καλούνται μέσα από το σώμα της κληθείσης μεθόδου. Για κάθε μήνυμα δημιουργίας αντικειμένου («create») που αποστέλλεται από μία γραμμή ετοιμότητας, εντοπίζουμε την τελευταία μέθοδο της γραμμής ετοιμότητας που κλήθηκε και δεν έχει επιστρέψει. Σε αυτήν την μέθοδο θα προσθέσουμε μία μεταβλητή με τύπο δεδομένων, την κλάση του δημιουργούμενου αντικειμένου. Εάν δεν υπάρχει μέθοδος που κλήθηκε και δεν έχει επιστρέψει το δημιουργούμενο αντικείμενο είναι πεδίο της κλάσης της γραμμής ετοιμότητας και έχει προστεθεί από τον κανόνα Α3.

Σε περίπτωση που χρησιμοποιηθούν περισσότερα του ενός ακολουθιακά διαγράμματα για την παραγωγή του διαγράμματος κώδικα και προκύψουν διαφορετικές εκδοχές του σώματος μιας μεθόδου, τότε οι εκδοχές αυτές μπορούν να ενωθούν με τη χρήση μιας επιπρόσθετης συνθήκης ή να απορριφθεί κάποια εκδοχή και να δηλωθεί με αυτόν τον τρόπο η ασυνέπεια μεταξύ των ακολουθιακών διαγραμμάτων.

Ο αλγόριθμός υλοποίησης του κανόνα σε ψευδοκώδικα είναι ο εξής:

Αλγόριθμος Α4

```
for each lifeline in sequenceDiagram
    do for each messageReceived in lifeline
    if messageReceived is MethodCall
        do method=getMethod(messageReceived);
        methodBody= getMethodBody(method);
        while messageReceived has not returned
            do for each messageSent from lifeline after messageReceived
                do if messageSent is MethodCall
                    do expressionToAdd=getExpression(messageSent);
                    addExpression(methodBody, expressionToAdd);
                endif
                if messageSent is createMessage
                    do addVariable(methodBody,messageSent);
                endif
            endfor
        endwhile
    endif
endfor
```



Σχήμα 3.9: Ο κανόνας προσδιορισμού του σώματος των μεθόδων

Κανόνας 5ος: Ορισμός των δομών ελέγχου.

Κάθε σύνθετο τμήμα αντιστοιχεί στον κώδικα, σε μια δομή ελέγχου της ροής του προγράμματος. Θεωρούμε ότι για κάθε σύνθετο τμήμα δηλώνεται η γραμμή ετοιμότητας στην οποία αντιστοιχεί το σύνθετο τμήμα καθώς και η συνθήκη ελέγχου της ροής. Για κάθε δομή ελέγχου τα στοιχεία που μας ενδιαφέρουν είναι τα εξής:

- Η έκφραση της δομής ελέγχου
- Η κλάση η οποία περιέχει τη δομή ελέγχου
- Η μέθοδος η οποία περιέχει τη δομή ελέγχου
- Οι μέθοδοι που περιέχονται στην δομή ελέγχου και η σειρά τους

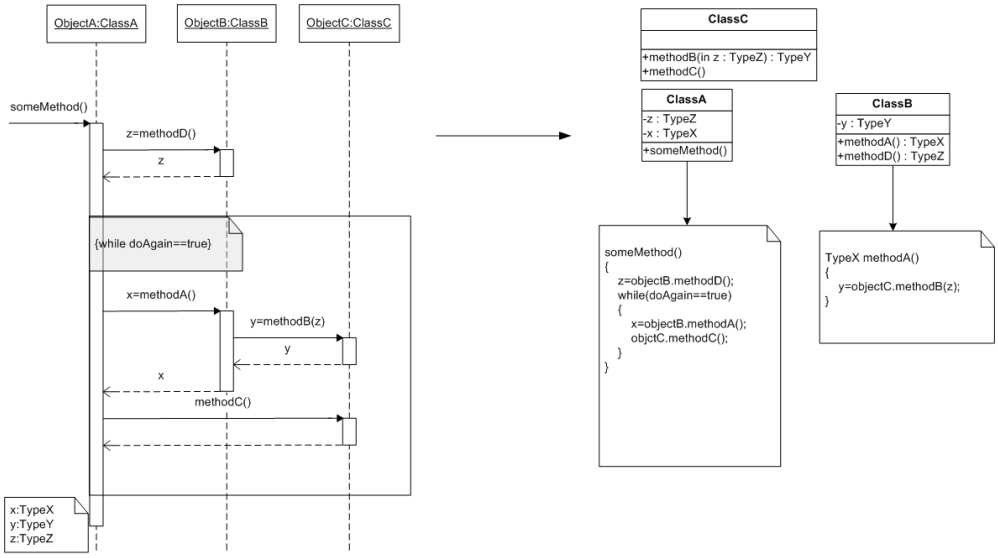
Έστω μια γραμμή ετοιμότητας στην οποία είναι δηλωμένο ένα σύνθετο τμήμα. Η κλάση που ανήκει το σύνθετο τμήμα είναι κλάση του διαγράμματος κώδικα που αντιστοιχεί στην γραμμή ετοιμότητας. Η μέθοδος που περιέχει το σύνθετο τμήμα είναι η μέθοδος που αντιστοιχεί στο τελευταίο μήνυμα που δέχτηκε η γραμμή ετοιμότητας και δεν έχει επιστρέψει πριν το σύνθετο τμήμα.
Οι μέθοδοι που περιέχονται στη δομή ελέγχου είναι οι μέθοδοι που αντιστοιχούν στα μηνύματα που στέλνει η γραμμή ετοιμότητας αλλά όχι οι

μέθοδοι που περιέχονται σε μηνύματα που στέλνονται από άλλες γραμμές ετοιμότητας.

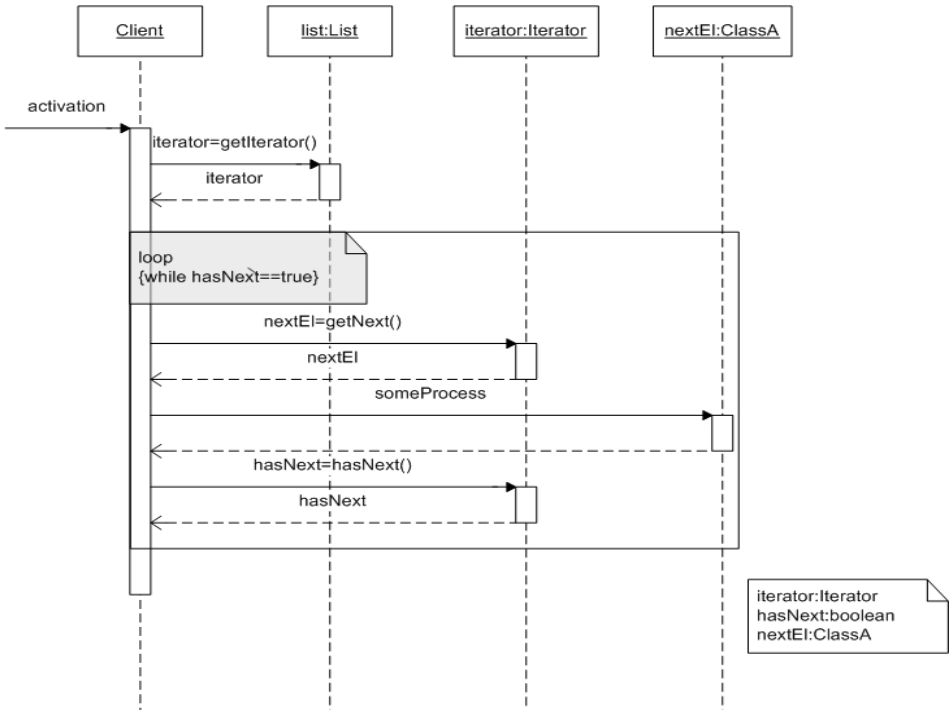
Ο αλγόριθμος Α5 αποτελεί υλοποίηση του κανόνα για την περίπτωση των βρόγχων. Παρόμοια υλοποιούνται και οι υπόλοιπες περιπτώσεις σύνθετων τμημάτων.

Αλγόριθμος Α5

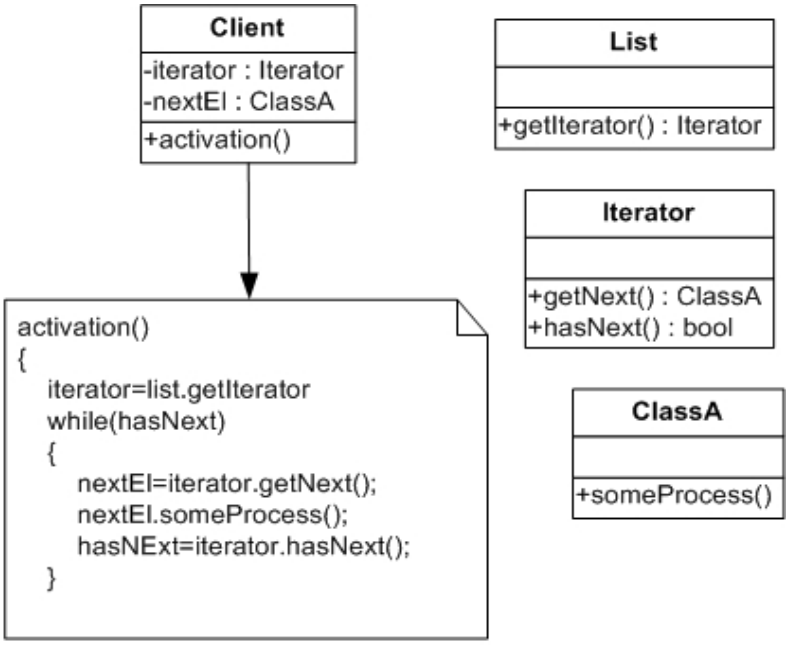
```
for each lifeline in sequenceDiagram
    class=getClassifier(lifeline);
    do for each loop in sequenceDiagram
        do expression=getExpression(loop);
        messageList=getContainedMessages(Loop);
        firstMessage=getFirstMessage(loop);
        currentMessage=previous(firstMessage);
        containerFound=false;
        while containerFound==false
            do if  not (currentMessage returned before loop)
                do containerFound=true;
                break;
            endif
            currentMessage=previous(currentMessage);
        endwhile
        if containerFound==true
            do containerMethod=getMethod(currentMessage);
            loopMethods=getMethodList(messageList);
            loopExpresion=loop.getExpresion();
            addLoop(containerMethod,loopMethods,loopExpresion);
        else containerMethod is Class Contructor;
        endif
    enfor
endfor
```



Σχήμα 3.10: 1ο παράδειγμα ορισμού δομών ελέγχου.



Σχήμα 3.11: 2ο Παράδειγμα ορισμού δομών ελέγχου. Ακολουθιακό διάγραμμα



Σχήμα 3.12: 2ο Παράδειγμα. Το αντίστοιχο διάγραμμα κώδικα για το διάγραμμα του σχήματος 3.11

Κατηγορία Β:

Στην κατηγορία αυτή ανήκουν οι κανόνες που επεξεργάζονται αποκλειστικά τα δεδομένα που υπάρχουν στο διάγραμμα κώδικα, εξάγουν νέα δεδομένα και συμπληρώνουν το διάγραμμα. Οι κανόνες αυτοί έχουν σχέση με την ιεραρχική δομή του συστήματος και δεν προκύπτουν άμεσα από τα δεδομένα των ακολουθιακών διαγραμμάτων αλλά υλοποιούνται από ευριστικούς αλγορίθμους και αποτελούν την εφαρμογή βασικών αρχών της σχεδίασης λογισμικών συστημάτων. Οι κανόνες αυτοί αποκτούν περισσότερο νόημα αν θεωρήσουμε ότι εφαρμόζονται μετά την επεξεργασία πολλών ακολουθιακών διαγραμμάτων μέσω των οποίων παράγεται το τελικό διάγραμμα κώδικα.

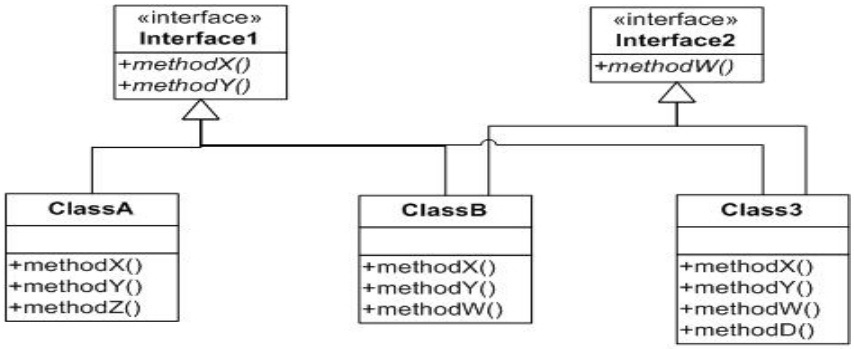
Κανόνας 1ος: Ορισμός διαπροσωπειών

Αν υπάρχει ένα κοινό σύνολο μεθόδων μεταξύ διαφορετικών κλάσεων τότε ορίζεται μία διαπροσωπεία με μεθόδους το κοινό σύνολο μεθόδων αυτών

των κλάσεων. Το σύνολο των μεθόδων κάθε διαπροσωπείας είναι ξένο με κάθε άλλο σύνολο μεθόδων διαφορετικής διαπροσωπείας.
Ο αλγόριθμός υλοποίησης του κανόνα σε ψευδοκώδικα είναι ο εξής:

Αλγόριθμος Β1

```
for each method in codeDiagram
    do interfacesList= list of interfaces containing method,initially empty;
    do if method contained in more than one classes
        do for each interface in codeDiagram
            //εαν το interface και η method έχουν κοινές κλάσεις
            do if (sameClasses(interface,method))
                do add(interfacesList);
                endif
        endfor
        if interfacesList not empty
            do for each interface in interfacesList
                if interface does not contain method
                    do add(method,interface)
                endif
            endfor
        else
            do create new interface(classesList(method))
            add(method,interface);
        endif
    endif
endfor
```



Σχήμα 3.13: Κανόνας ορισμού των διαπροσωπειών.

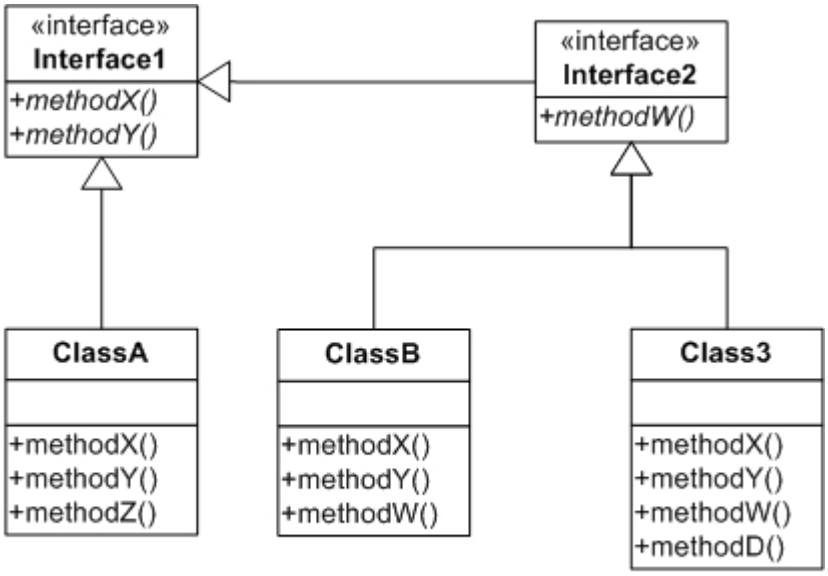
Κανόνας 2ος: Συγχώνευση διαπροσωπειών

Εάν το σύνολο των κλάσεων που υλοποιούν μία διαπροσωπεία IntefaceA είναι γνήσιο υποσύνολο του συνόλου των κλάσεων μιας διαφορετικής διαπροσωπείας InterfaceB τότε οι δύο διαπροσωπείες έχουν μια σχέση γενίκευσης μεταξύ τους και πιο συγκεκριμένα η διαπροσωπεία InterfaceB αποτελεί γενίκευση της διαπροσωπείας InterfaceA (InterfaceA is a InterfaceB).

Ο αλγόριθμος υλοποίησης του δεύτερου κανόνα είναι ο εξής:

Αλγόριθμος B3

interfacesList = list of all interfaces;
for each pair (InterfaceA,InterfaceB) of intefacesList
  do if (interfaceB.getClassesList().contains(interfaceA.getClassesList()
  and
interfaceB.getClassesList().size()>interfaceA.getClassesList.size())
  do interfaceA.implements(interfaceB);
  for each class implementing both interfaces
  do class.removeInterface(interfaceB);
  endfor
  interfaceB.removeClasses(interfaceA.getClassesList());
  endif
endfor



Σχήμα 3.14: Ο κανόνας συγχώνευσης των διαπροσωπειών

Κανόνας 3ος: Ορισμός υπερκλάσεων

Ο ορισμός των διαπροσωπειών καθορίζει την δομή του συστήματος βάσει μόνο των μεθόδων των κλάσεων. Η χρήση μόνο αυτών των κανόνων αν και μπορεί να δώσει μια καλή σχετικά προσέγγιση της δομής του συστήματος, αφήνει αναξιοποίητη την πληροφορία που μπορούμε να εξάγουμε εξετάζοντας τα πεδία των κλάσεων. Βασιζόμενοι στην ήδη διαθέσιμη πληροφορία των κοινών μεθόδων μπορούμε να ελέγξουμε το διάγραμμα κώδικα για κλάσεις που έχουν κοινές μεθόδους αλλά και κοινά πεδία. Με αυτόν τον τρόπο μπορούμε να εντοπίσουμε σχέσεις υπερκλάσεων-υποκλάσεων και να προσδιορίσουμε με μεγαλύτερη ακρίβεια την ιεραρχική δομή του συστήματος.

Έστω λοιπόν, classA μία κλάση η οποια υλοποιεί κάποια διαπροσωπεία Interface. Εάν κάθε άλλη κλάση που υλοποιεί την διαπροσωπεία Interface περιέχει τα πεδία της classA, τότε αυτή η κλάση είναι υπερκλάση των κλάσεων του Interface και συνεπώς δεν υλοποιούν το Interface αλλά είναι υποκλάσεις της classA.

Ο αλγόριθμος υλοποίησης του κανόνα σε ψευδοκώδικα είναι ο εξής:

Αλγόριθμος Β3α

interfacesList = list of all interfaces;
boolean flag=false;
for each interface in interfacesList
  do classesList = list of classes implementing interface;
  for each class in classesList
  //ελεγξε εαν κάθε κλάση της classesList περιέχει τα πεδια της class.
  do flag=checkFields(class,classesList);
  if (flag==true)
  do remove(classesList,class);
  //αφαιρεσε ολες τις κλάσεις από το Interface εκτός απο την class
  remove(interface,classesList);
  makeSuperClass(class);
  addSubClasses(class,classesList);//κάνε όλες τις κλάσεις
  //παιδια της class
  endif
  endfor
endfor

Ένας δεύτερος τρόπος εντοπισμού σχέσεων υπερκλάσεων-υποκλάσεων είναι ο εξής:

Έστω ένα σύνολο κλάσεων C για το οποίο ισχύει ότι κάθε μία απο τις κλάσεις του συνόλου υλοποιούν μια συγκεκριμένη διαπροσωπεία, την Interface1. Εάν υπάρχει μια κλάση, έστω classA που ανήκει στο σύνολο C τέτοια ώστε να ισχύουν οι παρακάτω προϋποθέσεις τότε θεωρούμε ότι η κλάση classA είναι υπερκλάση των κλάσεων του συνόλου C. Οι προϋποθέσεις είναι οι εξής:

- Υπάρχει έστω μία μέθοδος μιας κλάσης εκτός του συνόλου C, η οποία καλεί μια μέθοδο της classA, ή δημιουργεί αντικείμενο της classA.

-Δεν υπάρχει καμία μέθοδος κλάσης εκτός του συνόλου C, η οποία να καλεί μέθοδο κάποιας κλάσης του συνόλου C εκτός της classA.

Το σκεπτικό πίσω από αυτόν τον κανόνα είναι ότι εφόσον οι κλάσεις που υλοποιούν μια διαπροσωπεία υπάρχουν στα ακολουθιακά διαγράμματα, τότε προφανώς συμμετέχουν σε κάποιες λειτουργίες του συστήματος. Αφού μόνο μία δέχεται μηνύματα από το υπόλοιπο σύστημα συμπεραίνουμε ότι η κοινή τους συμπεριφορά εκφράζεται μέσω αυτής της μίας κλάσης, έστω classA, και ότι από το υπόλοιπο σύστημα, οι κλάσεις της διαπροσωπείας, είναι ορατές ως classA.

Ο αλγόριθμος που υλοποιεί τον παραπάνω κανόνα είναι ο εξής:

Αλγόριθμος Β3β

interfacesList = list of all interfaces;
for each interface in interfacesList
  do for each class in interface
  do if class has a method called outside interface
  do candidateClass=class;
  candidates=candidates+1;
  endif
  endfor
 	
  //εαν μόνο μία κλάση καλείται από το υπόλοιπο σύστημα
  if (candidates==1)
  do superClass=candidateClass;
  endif
  classesList= list of classes of interface;
  remove(classesList,superClass);
  removeClasses(interface, classesList);
  addClasses(superClass,classesList)
endfor

3.8 Περίληψη

Στο κεφάλαιο αυτό περιγράψαμε αναλυτικά την μεθοδολογία επεξεργασίας των ακολουθιακών διαγραμμάτων. Αρχικά ορίσαμε με μαθηματικό τρόπο τον μετασχηματισμό που εφαρμόζουμε πάνω στα ακολουθιακά διαγράμματα με σκοπό την παραγωγή του διαγράμματος κώδικα. Ορίσαμε τα μοντέλα των διαγραμμάτων που χρησιμοποιήσαμε και περιγράψαμε τον ρόλο κάθε στοιχείου του μοντέλου. Τέλος παρουσιάσαμε τους κανόνες και αλγορίθμους που υλοποιούν τον μετασχηματισμό των ακολουθιακών διαγραμμάτων σε διαγράμματα κώδικα.

Κεφάλαιο 4

Πρακτική Εφαρμογή

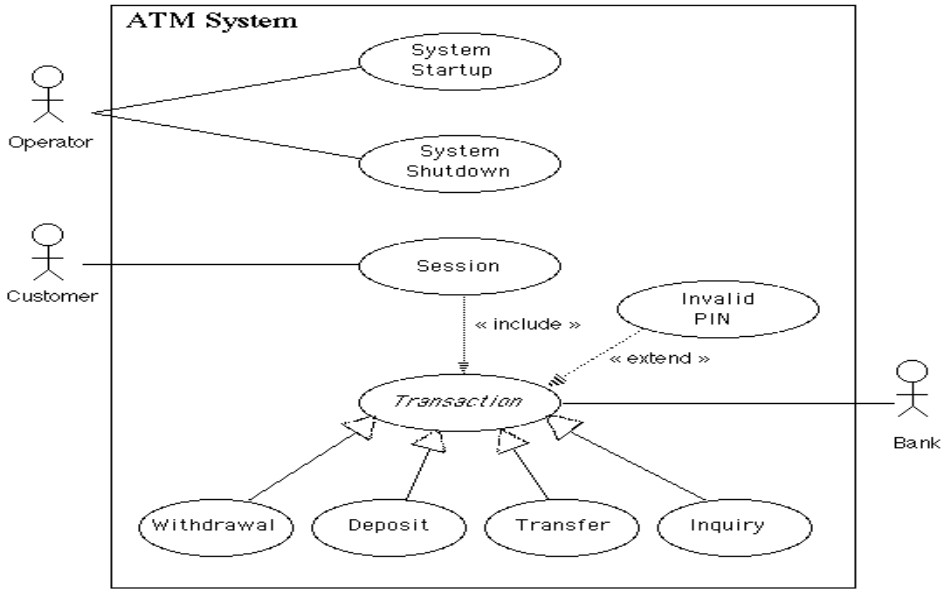
Σε αυτό το κεφάλαιο θα χρησιμοποιήσουμε μία πραγματική εφαρμογή μεσαίου μεγέθους για να εφαρμόσουμε τα μοντέλα και τους αλγορίθμους που περιγράψαμε στο προηγούμενο κεφάλαιο. Η εφαρμογή που θα χρησιμοποιήσουμε είναι μια πειραματική εφαρμογή η οποία προσομοιώνει την λειτουργία μιας Αυτόματης Ταμειακής Μηχανής μιας Τράπεζας (ATM). Η εφαρμογή αυτή αναπτύχθηκε από το εκπαιδευτικό ίδρυμα Gordon College στην Μασσαχουσέτη των Η.Π.Α. Οι προδιαγραφές του συστήματος, τα διαγράμματα καθώς και ο κώδικας είναι διαθέσιμα στο κοινό μέσω του δικτυακού ιστοτόπου του ιδρύματος [15].

4.1 Η εφαρμογή ATM System

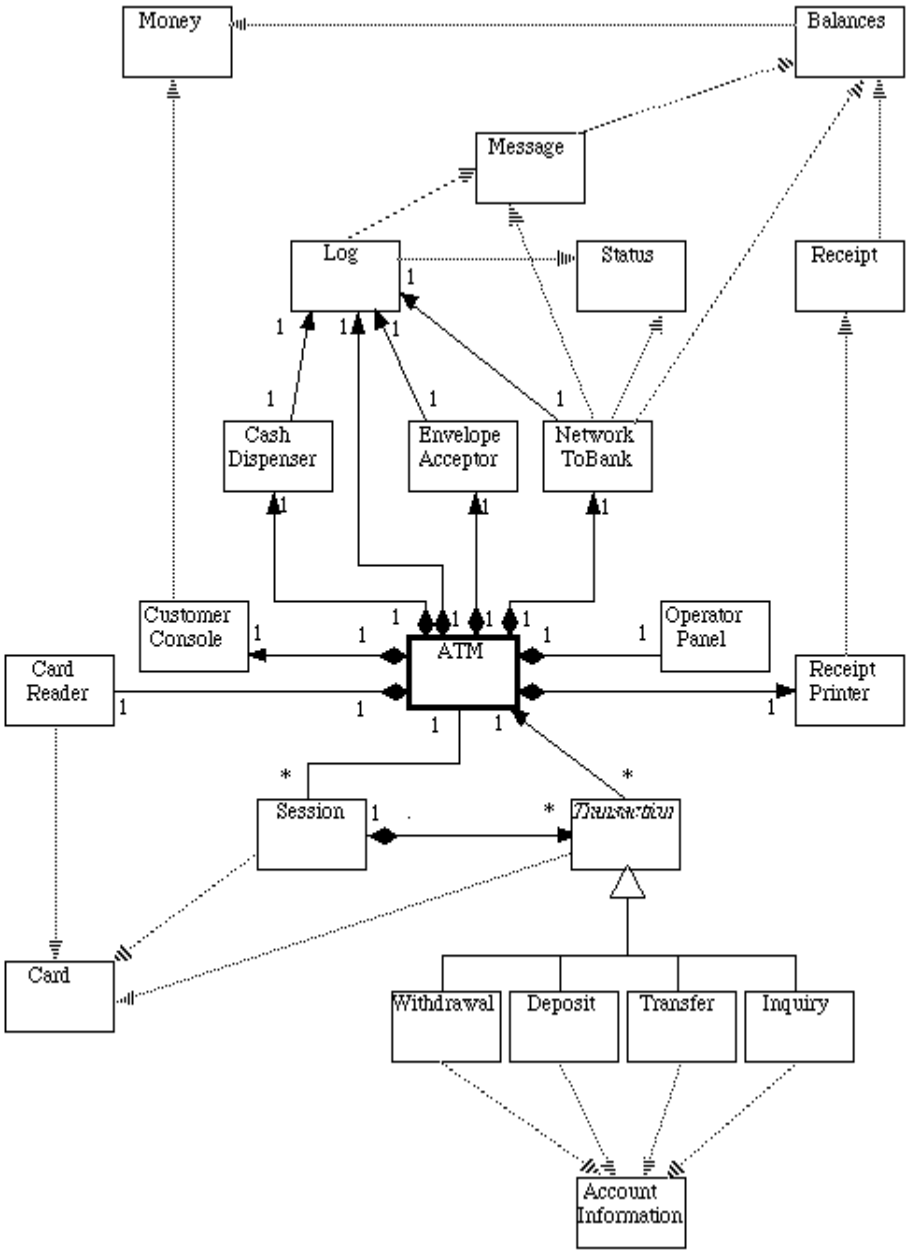
Όπως αναφέρεται και στο [15], το λογισμικό αυτό αναπτύχθηκε με σκοπό την επίδειξη της διαδικασίας της σχεδίασης αντικειμενοστραφών λογισμικών συστημάτων και συνεπώς είναι κατάλληλο για την εφαρμογή των μοντέλων και των κανόνων ανάλυσης της δομής του συστήματος και της παραγωγής κώδικα, καθώς έχει ακολουθηθεί η πλήρως η διαδικασία σχεδίασης και ανάπτυξης λογισμικών συστημάτων. Για την σχεδίαση της εφαρμογής έχουν τεθεί οι λειτουργικές απαιτήσεις του συστήματος και αυτές μοντελοποιούνται μέσω των διαγραμμάτων χρήσης του συστήματος. Η δομή του συστήματος έχει μοντελοποιηθεί με τη χρήση διαγραμμάτων κλάσης ενώ η επικοινωνία των λειτουργικών μονάδων του συστήματος έχει μοντελοποιηθεί με διαγράμματα κατάστασης και διαγράμματα αλληλεπίδρασης. Σε αυτήν την εργασία θα παράξουμε κώδικα από τα διαθέσιμα ακολουθιακά διαγράμματα και θα συγκρίνουμε τον αρχικό κώδικα με αυτόν που παρήχθη από την εφαρμογή των κανόνων πάνω στα ακολουθιακά διαγράμματα. Η δομή του συστήματος που θα χρησιμοποιηθεί στην εργασία περιγράφεται από το διάγραμμα κλάσεων του σχήματος 4.2. Για λόγους οικονομίας χώρου στο διάγραμμα κλάσεων δεν απεικονίζονται

τα πεδία και οι μέθοδοι κάθε κλάσης. Τα στοιχεία αυτά θα δίνονται για κάθε κλάση ξεχωριστά όταν αυτό είναι αναγκαίο.

Το σύστημα αυτό ικανοποιεί κάποιες βασικές λειτουργικές απαιτήσεις που μοντελοποιούνται μέσω ενός διαγράμματος σεναρίων χρήσης όπως φαίνεται και στο σχήμα 4.1. Για κάθε σενάριο χρήσης υπάρχει και ένα διάγραμμα αλληλεπίδρασης το οποίο μοντελοποιεί την επικοινωνία μεταξύ των λειτουργικών μονάδων. Τα διαγράμματα αυτά είναι είτε ακολουθιακά διαγράμματα οπότε και θα τα επεξεργαστούμε ως έχουν, είτε είναι συνεργατικά διαγράμματα. Στις περιπτώσεις των συνεργατικών διαγραμμάτων θα μετατρέψουμε τα συνεργατικά διαγράμματα σε ακολουθιακά διαγράμματα. Όπως αναφέρεται και στο [4] ο μετασχηματισμός συνεργατικών διαγραμμάτων σε ακολουθιακά και αντίστροφα, είναι πλήρης δηλαδή δεν χάνεται πληροφορία κατά το μετασχηματισμό των διαγραμμάτων από τον ένα τύπο στον άλλο. Αναφέρεται επίσης ότι δεν γίνεται διαχωρισμός των δύο αυτών διαγραμμάτων στο μεταμοντέλο της UML.



Σχήμα 4.1: Διάγραμμα σεναρίων χρήσης του ATM.



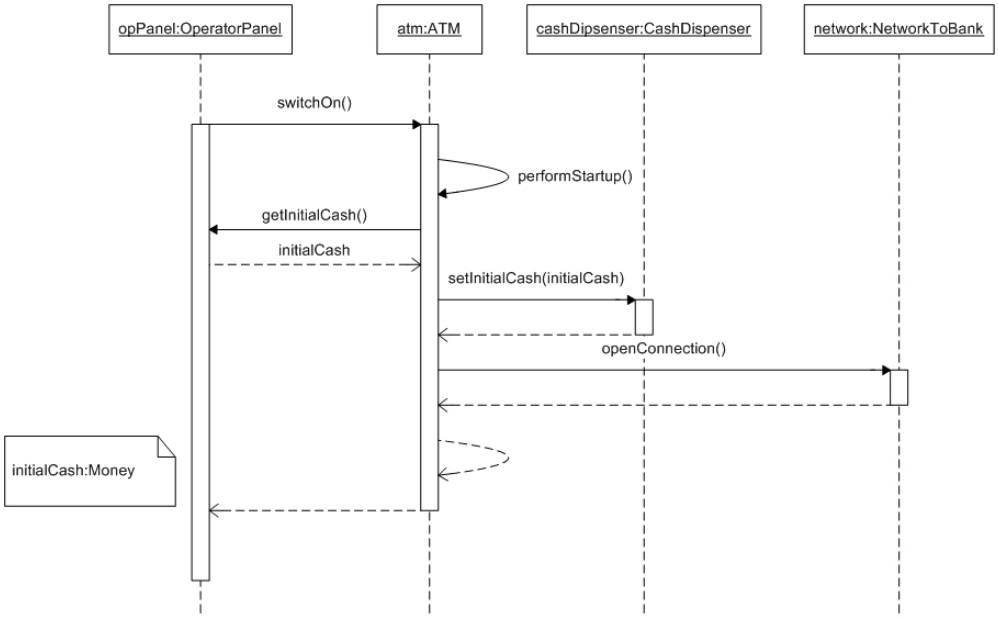
Σχήμα 4.2: Το διάγραμμα κλάσεων του ATM.

4.2 Εφαρμογή κανόνων μετασχηματισμού

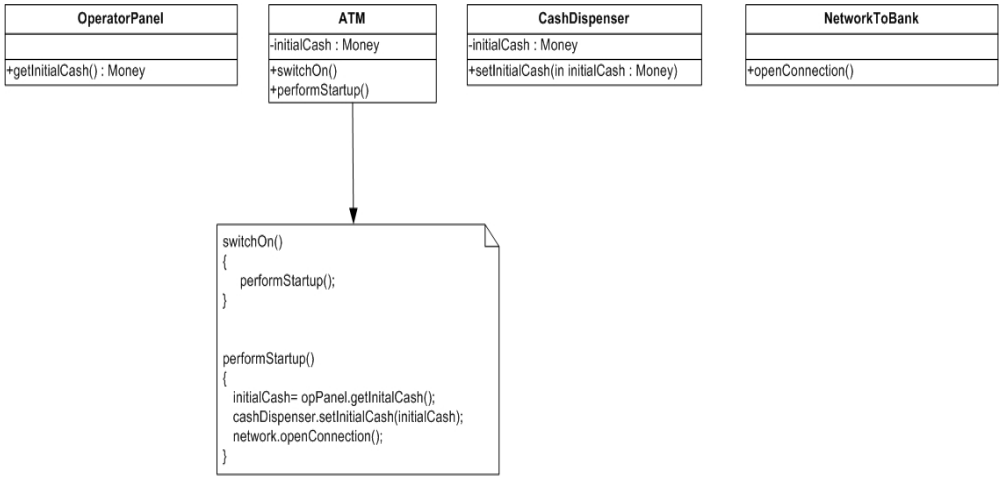
4.2.1 Απλή περίπτωση

Αρχικά θα εφαρμόσουμε τους κανόνες μετασχηματισμού στα ακολουθιακά διαγράμματα τα οποία δεν περιέχουν σύνθετα τμήματα. Αυτό σημαίνει ότι ο 5ος κανόνας της Α' κατηγορίας δεν θα εφαρμόζεται εκτός και αν αναφέρεται ρητά. Το στιγμιότυπο του μοντέλου κώδικα που θα προκύψει θα είναι η σύνθεση των αποτελεσμάτων όλων των ακολουθιακών διαγραμμάτων που θα επεξεργαστούμε σε αυτήν την πρώτη φάση. Με την επεξεργασία κάθε ακολουθιακού διαγράμματος θα ενημερώνεται και το στιγμιότυπο του μοντέλου κώδικα. Έτσι σε κάθε βήμα θα έχουμε την επεξεργασία ενός ακολουθιακού διαγράμματος και το παραγόμενο διάγραμμα κώδικα που προκύπτει από τη σύνθεση του τελευταίου διαγράμματος κώδικα και των δεδομένων του ακολουθιακού διαγράμματος που επεξεργαστήκαμε σε αυτό το βήμα.

Το πρώτο ακολουθιακό διάγραμμα που θα αναλυθεί είναι το διάγραμμα του σχήματος 4.3 το οποίο είναι το διάγραμμα εκκίνησης του ATM (System Startup Sequence Diagram). Στο διάγραμμα του σχήματος 4.3 εφαρμόζουμε διαδοχικά τους κανόνες της ομάδας Α(εξαιρουμένου του 5ου κανόνα).Από την εφαρμογή των κανόνων προκύπτει το στιγμιότυπο του μοντέλου κώδικα του σχήματος 4.4. Από τα μηνύματα που λαμβάνει κάθε γραμμή ετοιμότητας προκύπτουν οι μέθοδοι κάθε κλάσης, ενώ οι μέθοδοι που καλούνται από κάθε μέθοδο προκύπτουν με βάση το σημείο επιστροφής της κάθε μεθόδου. Κάθε μέθοδος Χ, που καλείται πριν από την επιστροφή μιας μεθόδου Ψ που έχει ήδη κληθεί, θεωρούμε ότι καλείται μέσα από το σώμα της μεθόδου Ψ. Η σειρά με την οποία καλούνται οι μέθοδοι μέσα στο σώμα της μεθόδου Ψ είναι η ίδια με τη σειρά με την οποία καλούνται στο ακολουθιακό διάγραμμα. Το σημείο επιστροφής κάθε μεθόδου σηματοδοτείται από το τέλος ενός ενεργού τμήματος πάνω στην γραμμή ετοιμότητας. Το τέλος ενός ενεργού τμήματος συνοδεύεται ορισμένες φορές και από ένα βέλος επιστροφής όταν μια καλούμενη μέθοδος επιστρέφει κάποια τιμή στην καλούσα μέθοδο.

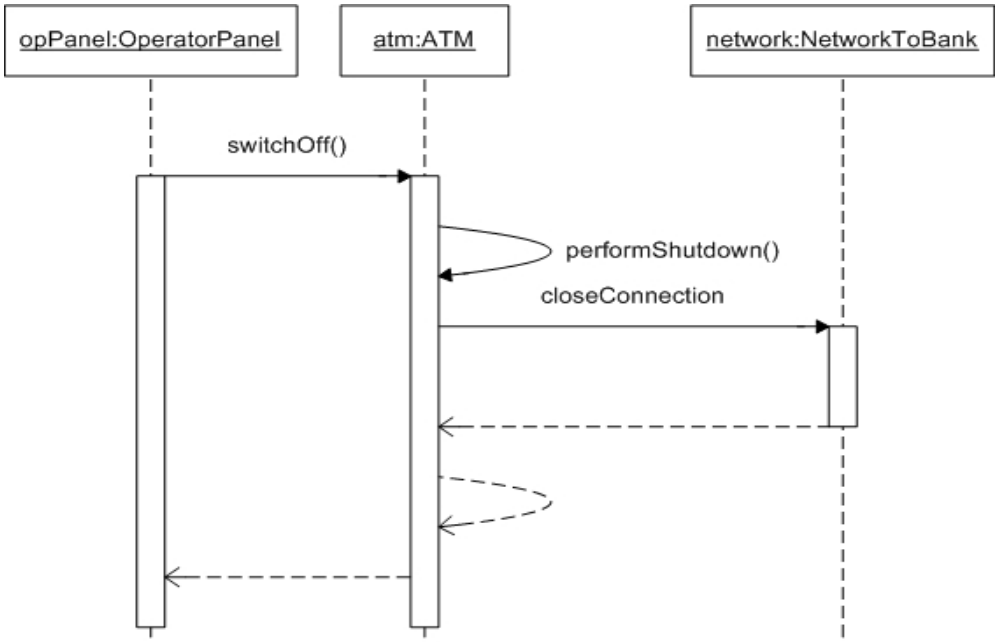


Σχήμα 4.3: Το ακολουθιακό διάγραμμα εκκίνησης συστήματος

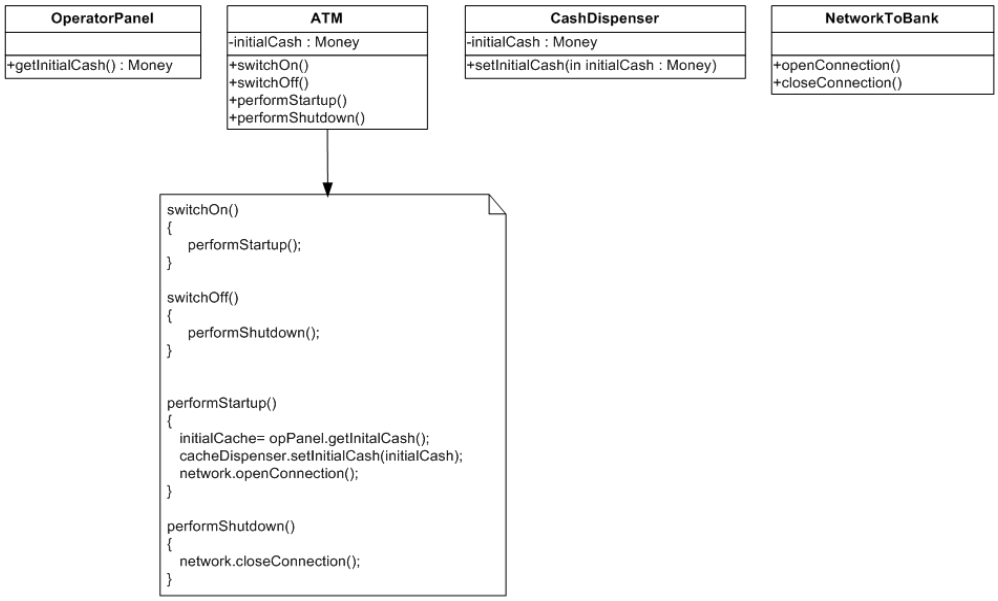


Σχήμα 4.4: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.3.

Στη συνέχεια επεξεργαζόμαστε με τον ίδιο τρόπο το ακολουθιακό διάγραμμα της παύσης του ATM το οποίο φαίνεται στο σχήμα 4.5. Το



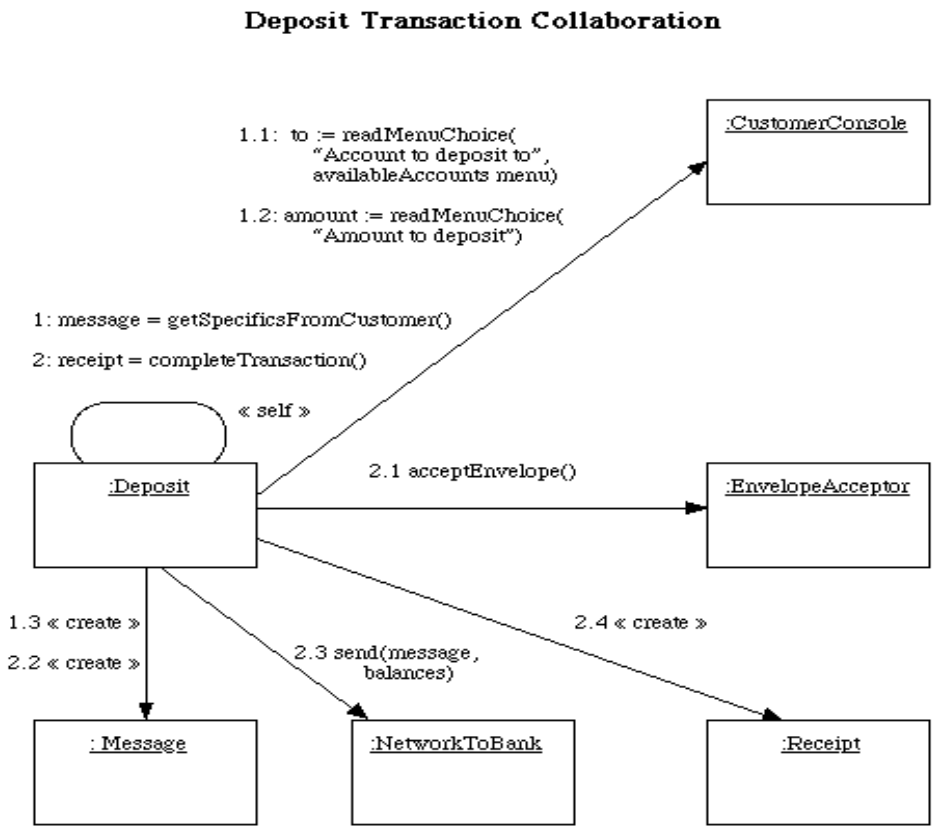
Σχήμα 4.5: Το ακολουθιακό διάγραμμα παύσης συστήματος



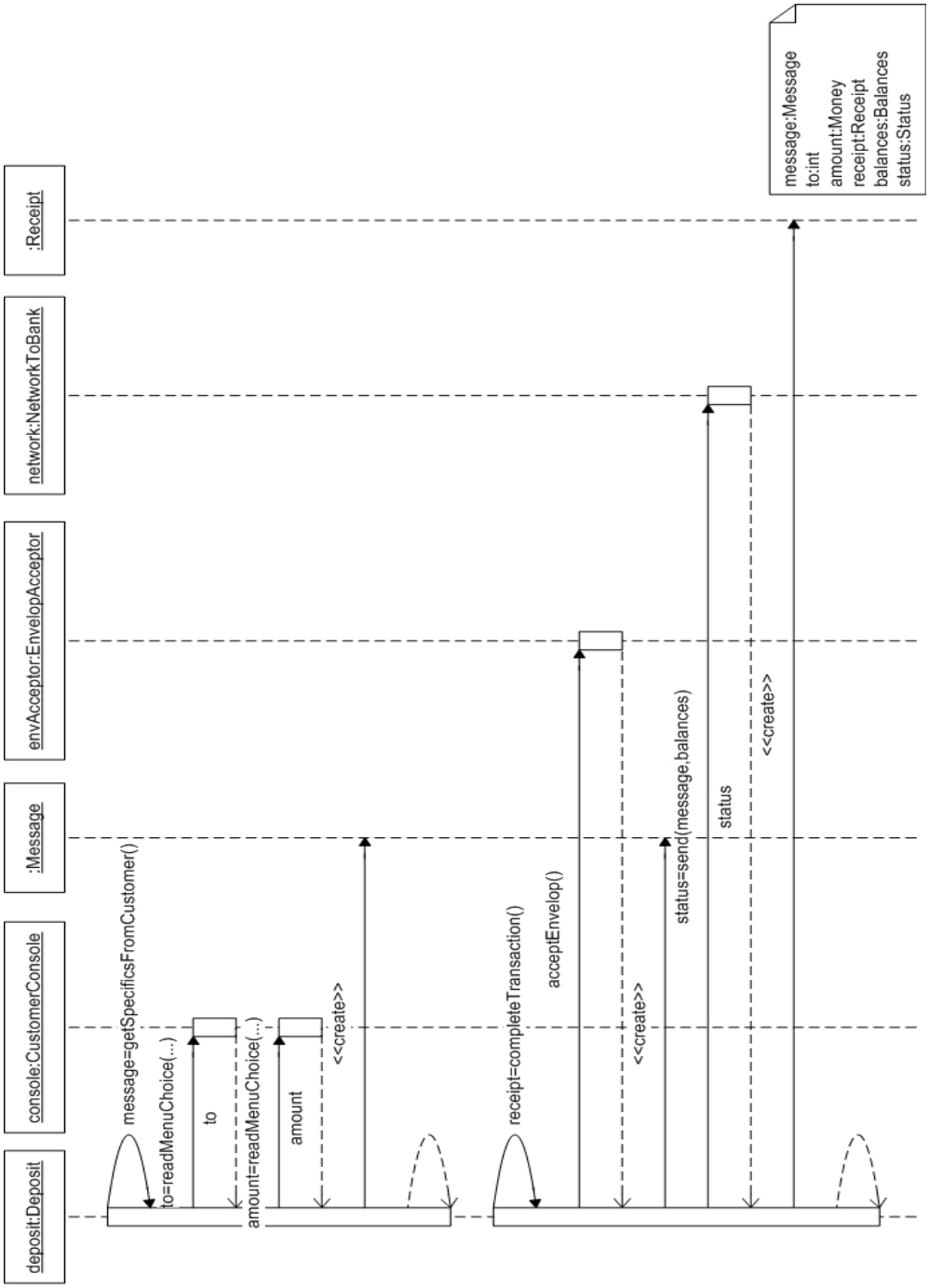
Σχήμα 4.6: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.5.

στιγμιότυπο κώδικα που προκύπτει, αποτελεί σύνθεση των αποτελεσμάτων των δύο μετασχηματισμών, αυτού του διαγράμματος έναρξης και αυτού του διαγράμματος παύσης

Στη συνέχεια επεξεργαζόμαστε το συνεργατικό διάγραμμα του σχήματος 4.7 το οποίο μοντελοποιεί την συναλλαγή της κατάθεσης. Για να μπορέσουμε να το επεξεργαστούμε, το μετατρέπουμε στο ακολουθιακό διάγραμμα του σχήματος 4.8.

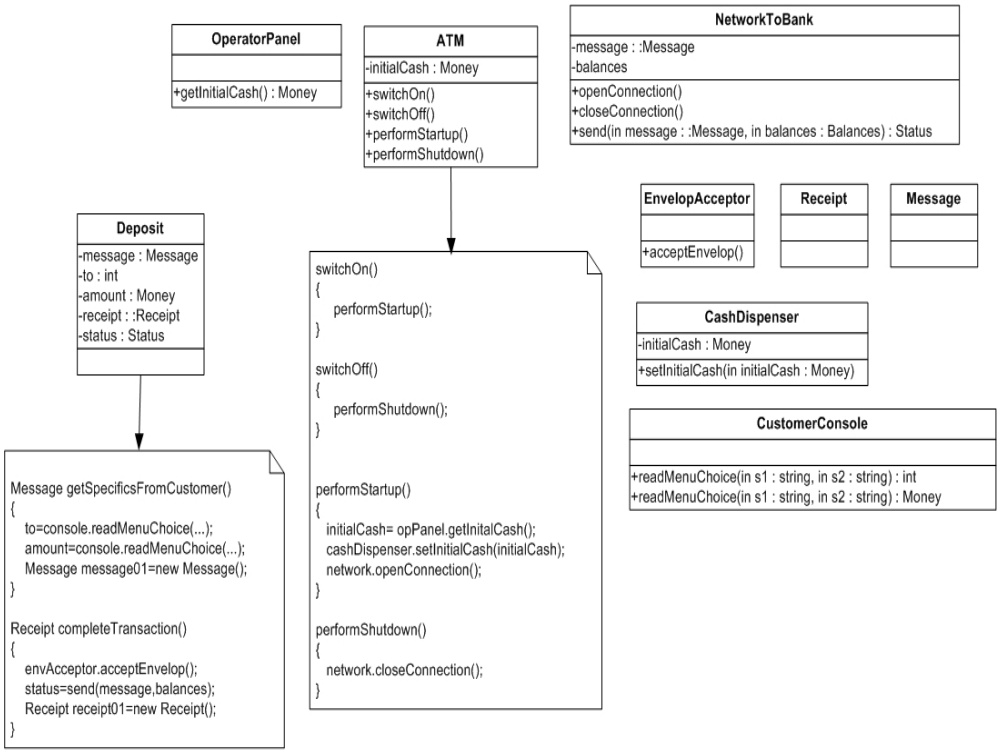


Σχήμα 4.7: Το συνεργατικό διάγραμμα της κατάθεσης στο ATM.



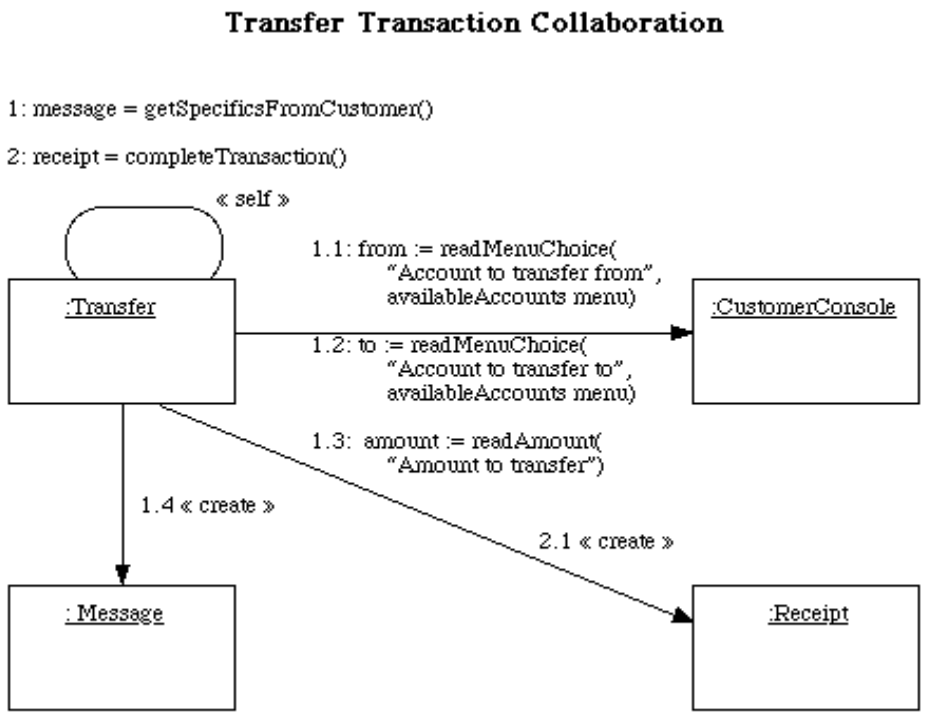
Σχήμα 4.8: Το ακολουθιακό διάγραμμα της κατάθεσης στο ATM.

Από την επεξεργασία του ακολουθιακού διαγράμματος του σχήματος 4.8 και του διαγράμματος κώδικα του σχήματος 4.6 προκύπτει το νέο διάγραμμα κώδικα του σχήματος 4.9.

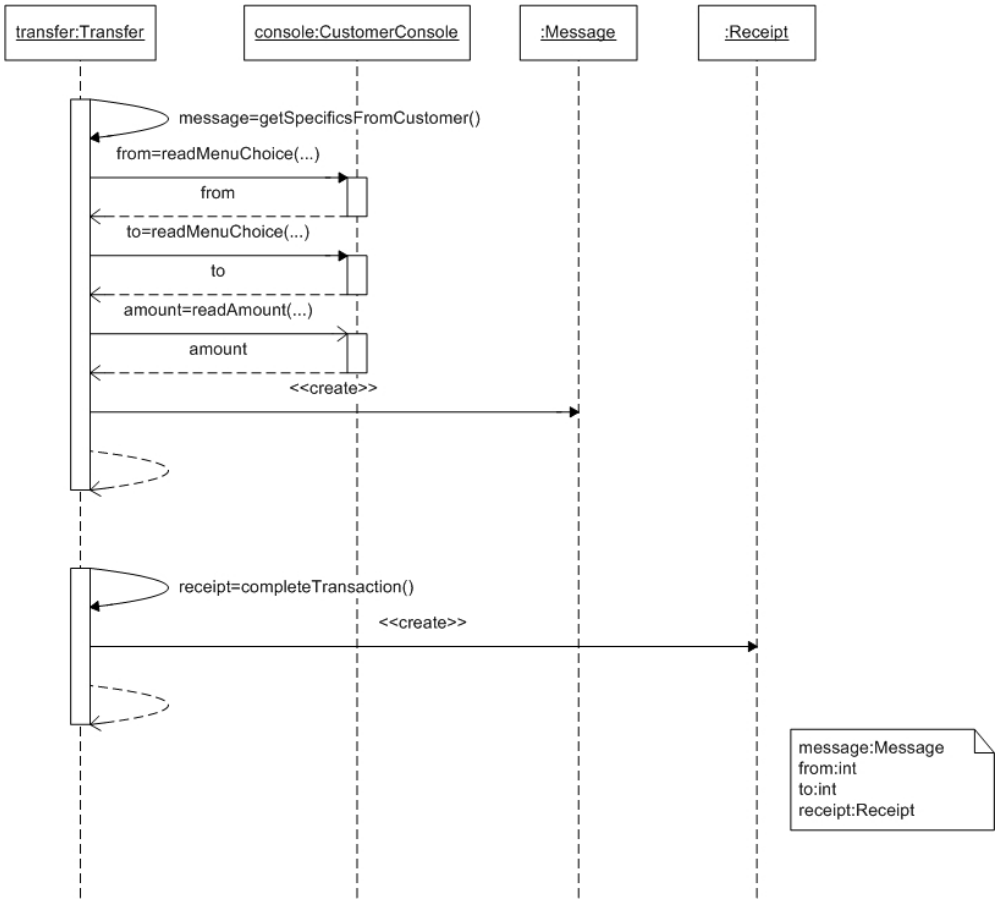


Σχήμα 4.9: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.8

Το επόμενο διάγραμμα προς επεξεργασία είναι το διάγραμμα του σχήματος 4.10 το οποίο είναι το συνεργατικό διάγραμμα της συναλλαγής της μεταφοράς χρημάτων. Το συνεργατικό αυτό διάγραμμα μετασχηματίζεται στο ακολουθιακό διάγραμμα του σχήματος 4.11

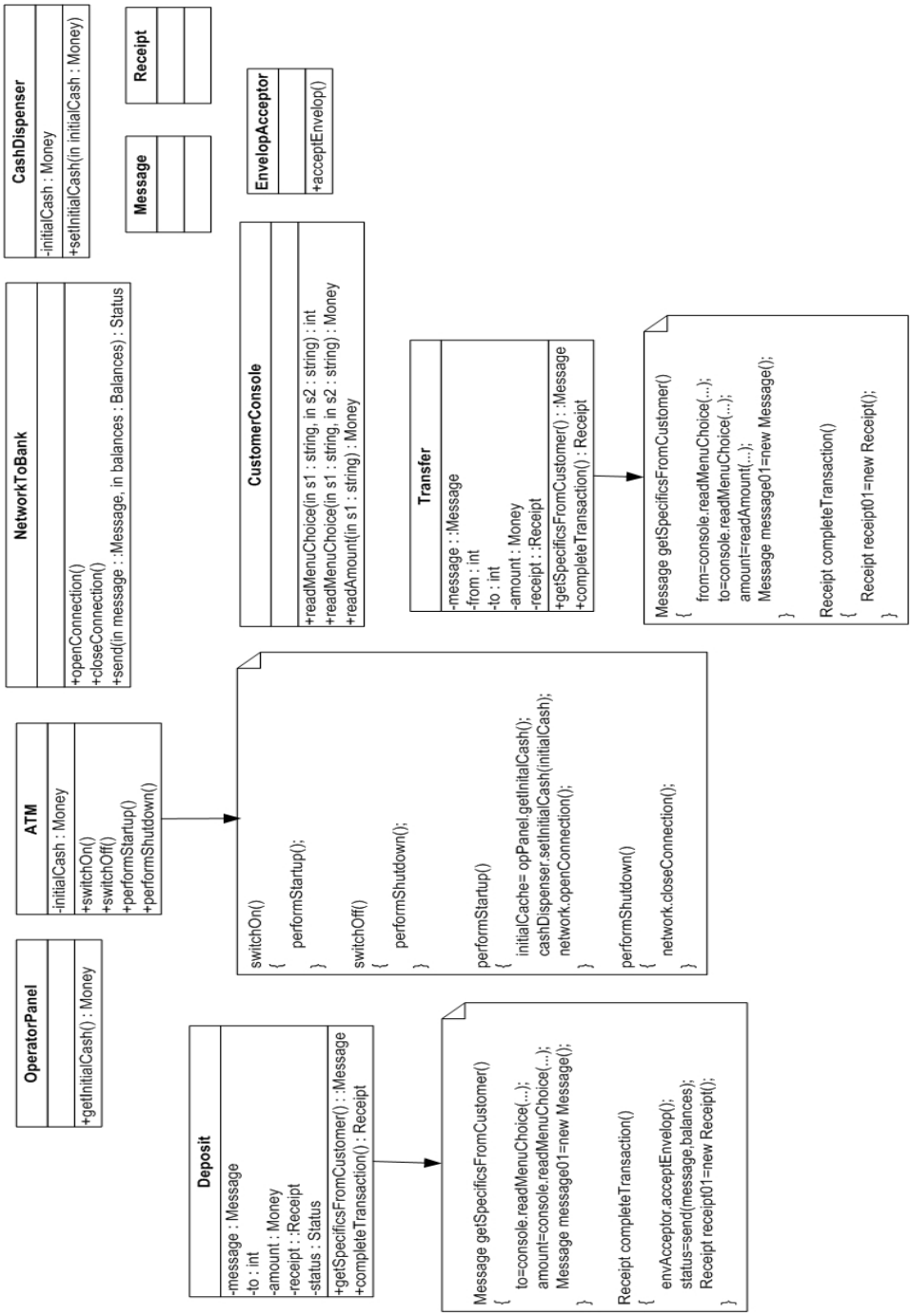


Σχήμα 4.10: Το συνεργατικό διάγραμμα της λειτουργίας μεταφοράς χρημάτων του ATM



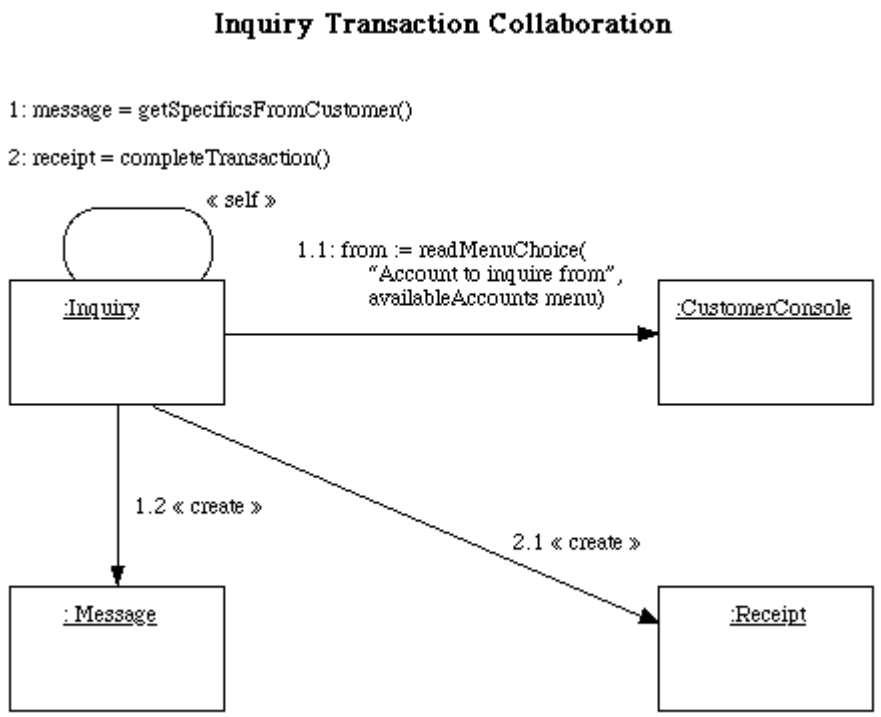
Σχήμα 4.11: Το ακολουθιακό διάγραμμα της λειτουργίας μεταφοράς χρημάτων του ATM

Μετά την επεξεργασία του ακολουθιακού διαγράμματος του σχήματος 4.11 και σε συνδυασμό με το διάγραμμα κώδικα του σχήματος 4.9 προκύπτει το ενημερωμένο διάγραμμα κώδικα του σχήματος 4.12

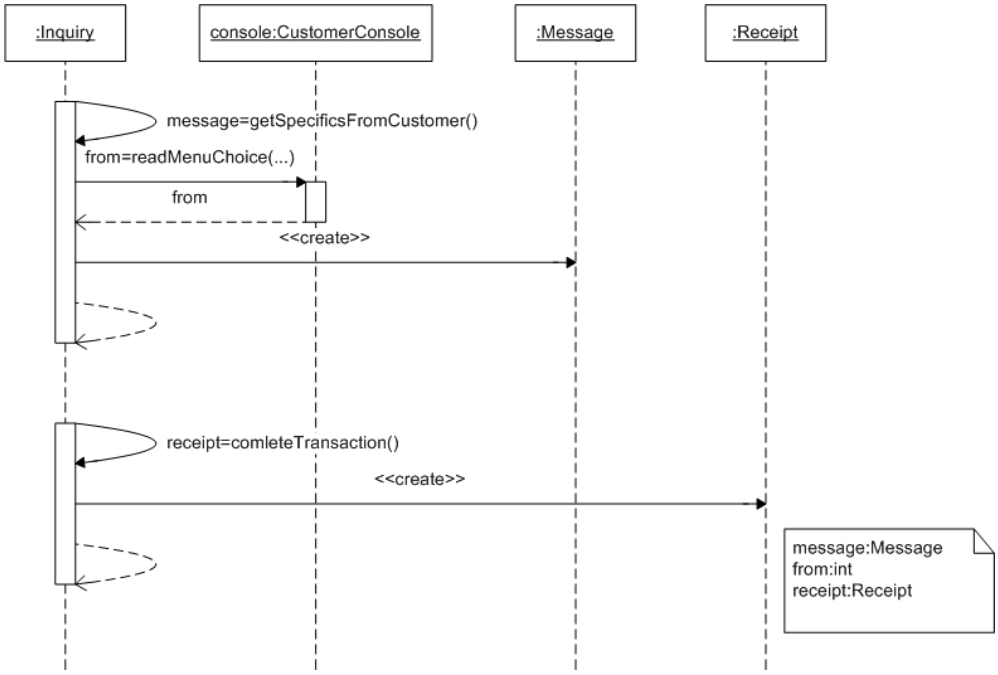


Σχήμα 4.12: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.11 .

Τέλος επεξεργαζόμαστε το ακολουθιακό διάγραμμα της ερώτησης στο ATM το οποίο απεικονίζεται στο σχήμα 4.14 το οποίο προκύπτει από το συνεργατικό διάγραμμα του σχήματος 4.13.

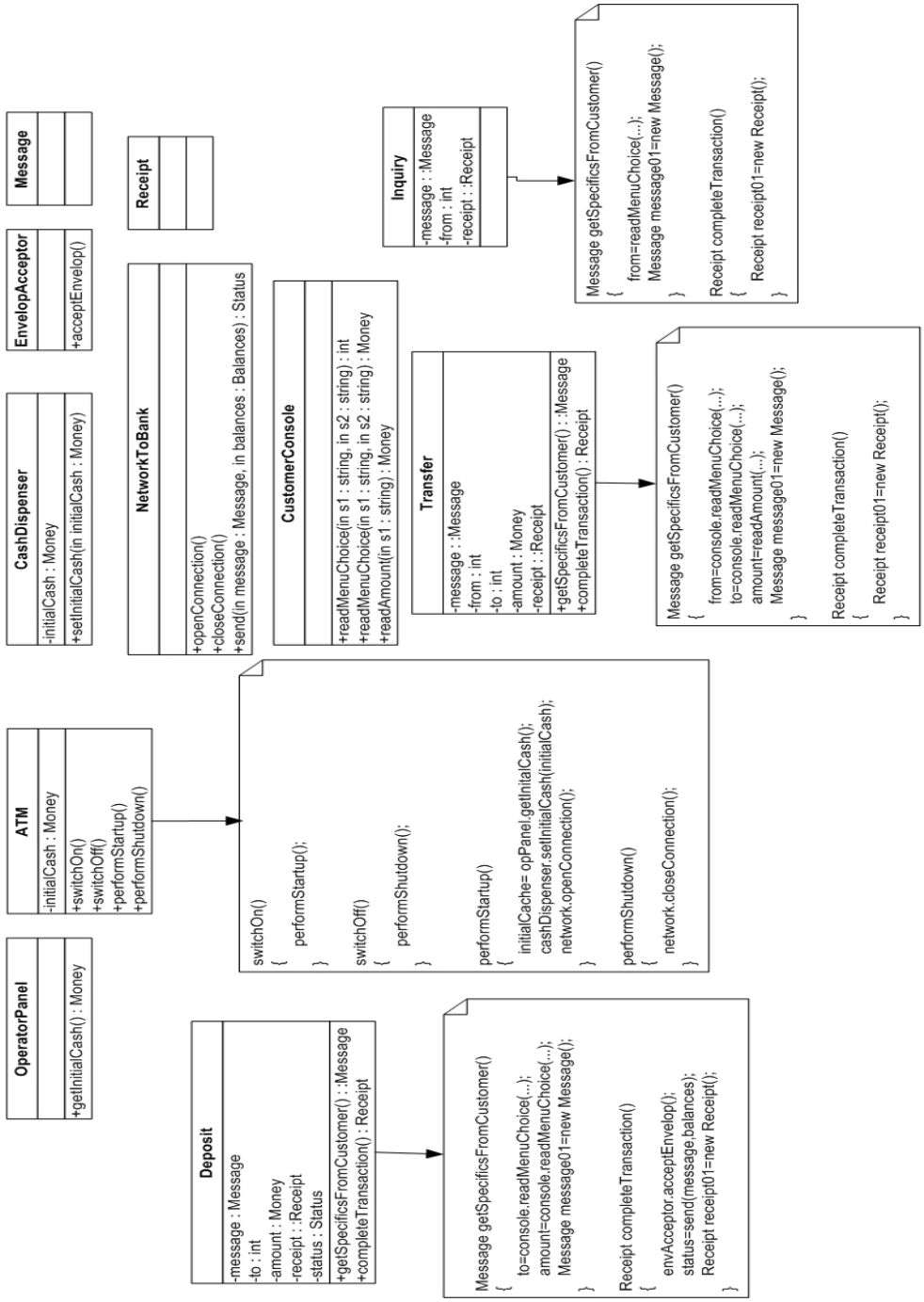


Σχήμα 4.13: Το συνεργατικό διάγραμμα ερώτησης στο ATM.



Σχήμα 4.14: Το ακολουθιακό διάγραμμα ερώτησης στο ATM.

Από την επεξεργασία του πιο πάνω ακολουθιακού διαγράμματος και το διάγραμμα κώδικα του σχήματος 4.12 προκύπτει το καινούργιο διάγραμμα κώδικα του σχήματος 4.15. Το διάγραμμα αυτό του κώδικα αποτελεί σύνθεση την επεξεργασίας όλων των ακολουθιακών διαγραμμάτων που εξετάστηκαν έως αυτό το σημείο. Στη συνέχεια θα εξετάσουμε και άλλα διαγράμματα με πιο πολύπλοκες δομές και θα συνεχίσουμε να ενημερώνουμε μετά την επεξεργασία κάθε ακολουθιακού διαγράμματος το διάγραμμα του σχήματος 4.15.



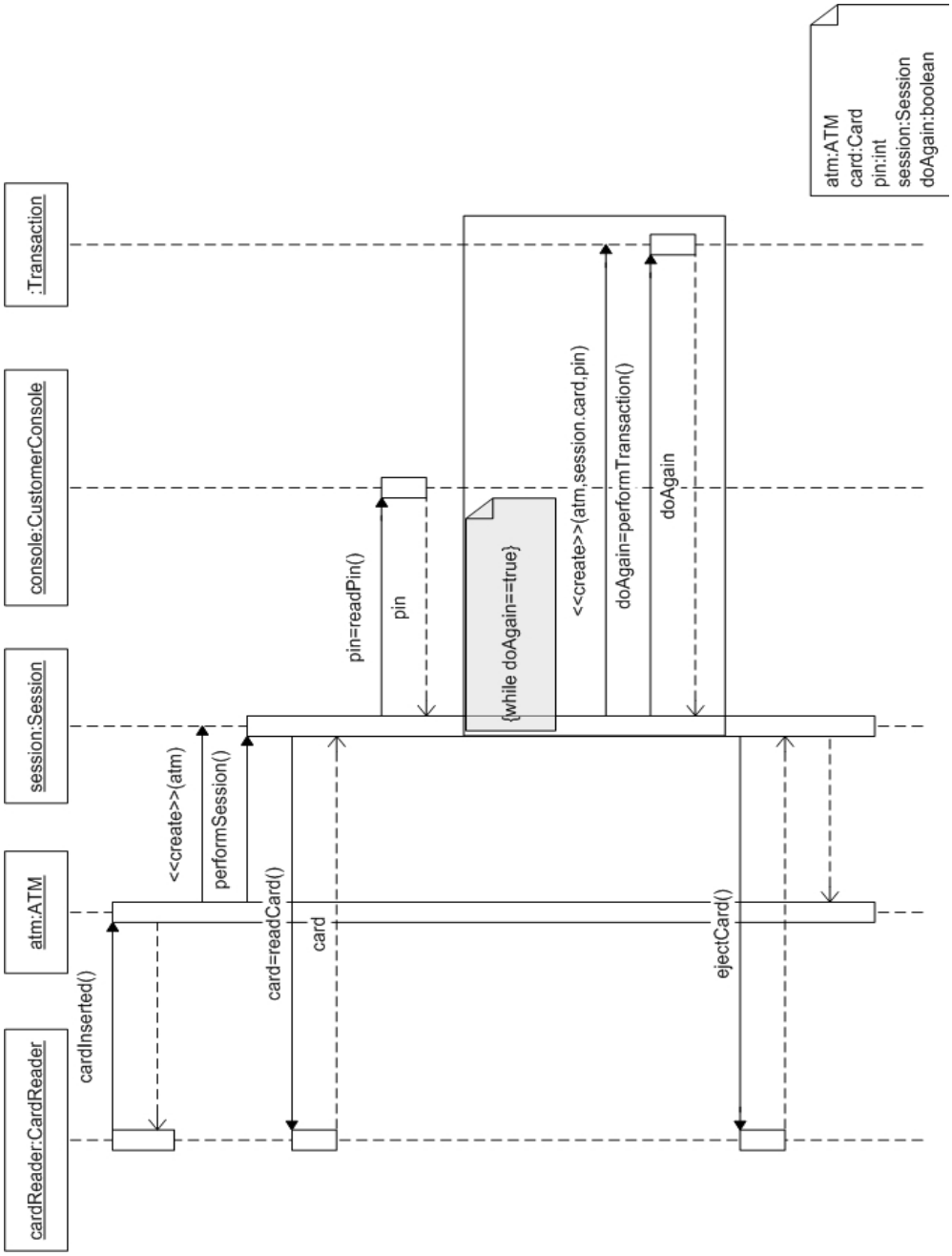
Σχήμα 4.15: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.14 .

4.2.2 Περίπτωση δομών loop

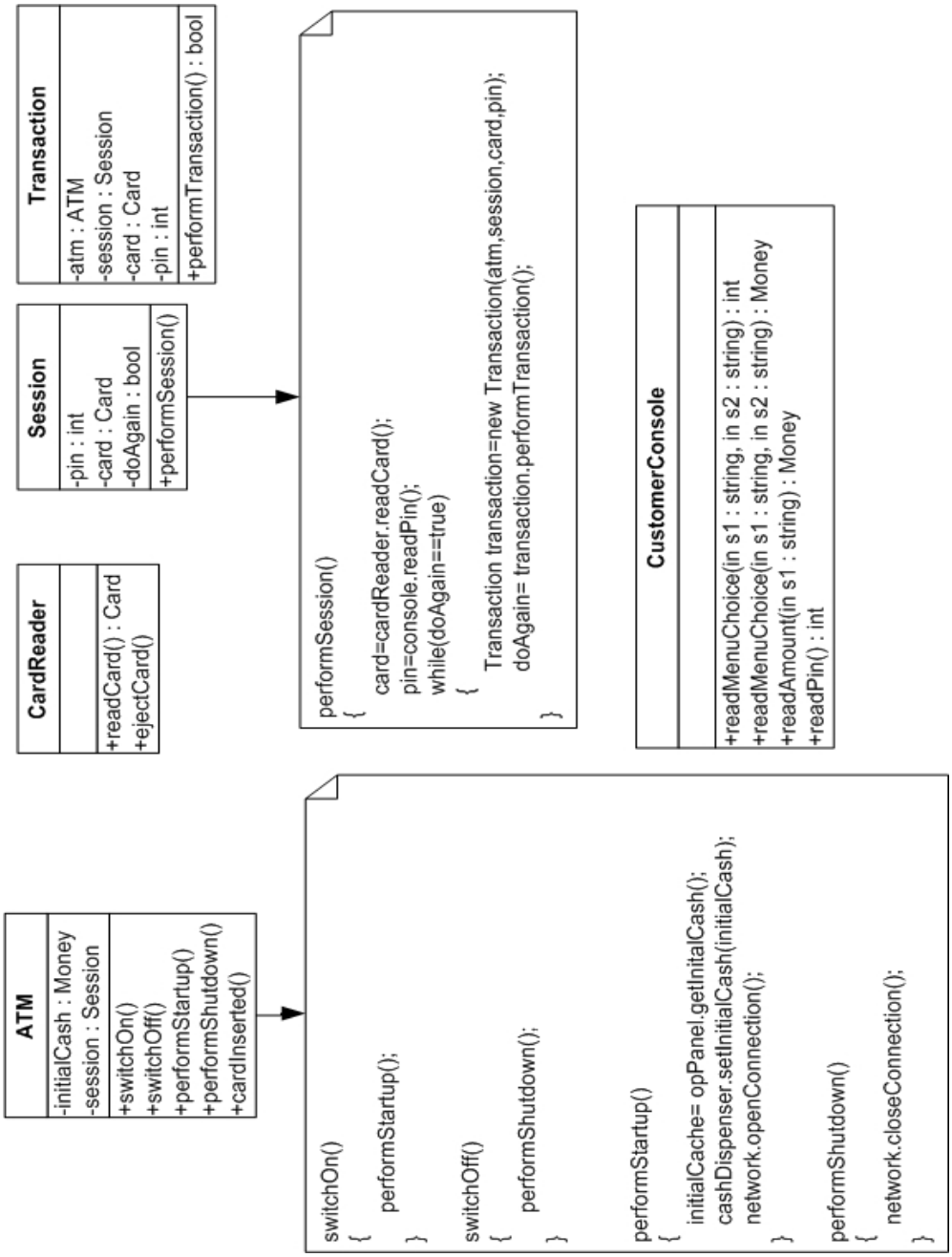
Έπειτα από την επεξεργασία των ακολουθιακών διαγραμμάτων χωρίς σύνθετα τμήματα θα εξετάσουμε σε αυτό το σημείο τα ακολουθιακά διαγράμματα με βρόγχους. Η επεξεργασία των σύνθετων τμημάτων γίνεται από τον 5^{ο} κανόνα της ομάδας Α. Στα επόμενα δηλαδή διαγράμματα αυτής της ενότητας θα χρησιμοποιούμε όλους τους κανόνες της Α ομάδας. Με αυτόν τον τρόπο γίνεται η επεξεργασία των ακολουθιακών διαγραμμάτων όπως γινόταν στα προηγούμενα διαγράμματα, αλλά ταυτόχρονα γίνεται και η επεξεργασία των βρόγχων. Σε αυτό το σημείο θα πρέπει να επισημάνουμε ότι δεν έχει σημασία η σειρά με την οποία επεξεργαζόμαστε τα ακολουθιακά διαγράμματα. Επιλέχθηκαν πρώτα τα απλά διαγράμματα με σκοπό να παρουσιαστούν προοδευτικά οι κανόνες μετασχηματισμού των ακολουθιακών διαγραμμάτων έτσι ώστε να τονίζονται οι ιδιαιτερότητες κάθε περίπτωσης χωρίς να δημιουργούνται ασάφειες για τα υπόλοιπα σκέλη του μετασχηματισμού των ακολουθιακών διαγραμμάτων.

Το διάγραμμα του σχήματος 4.16 απεικονίζει την επικοινωνία μεταξύ των λειτουργικών μονάδων του ATM στην περίπτωση μιας συνεδρίας του ATM με τον πελάτη. Παρατηρούμε ότι υπάρχει ένας βρόγχος στον οποίο συμμετέχουν τρεις γραμμές ετοιμότητας,η Session, η CustomerConsole και η Transaction. Ο βρόγχος αυτός θεωρούμε ότι ανήκει στην τελευταία μέθοδο της γραμμής ετοιμότητας στη οποία έχει δηλωθεί και δεν έχει επιστρέψει έως το σημείο που ξεκινάει ο βρόγχος. Στη συγκεκριμένη περίπτωση θεωρούμε ότι ο βρόγχος ανήκει στη μέθοδο performSession().

Το διάγραμμα του σχήματος 4.17 είναι η σύνθεση του διαγράμματος κώδικα του σχήματος 4.15 και των δεδομένων που εξάγονται από την εφαρμογή των κανόνων μετασχηματισμού στο ακολουθιακό διάγραμμα του σχήματος 4.16. Σε αυτό το διάγραμμα, αλλά και στα υπόλοιπα διαγράμματα κώδικα που θα παρουσιάσουμε, θα απεικονίζονται μόνο τα νέα στοιχεία που προστίθενται στο διάγραμμα που δημιουργούνται ή αυτά που τροποποιούνται. Εάν κάποιο στοιχείο διαγραφεί από το διάγραμμα κώδικα τότε αυτό θα αναφέρεται ρητά.

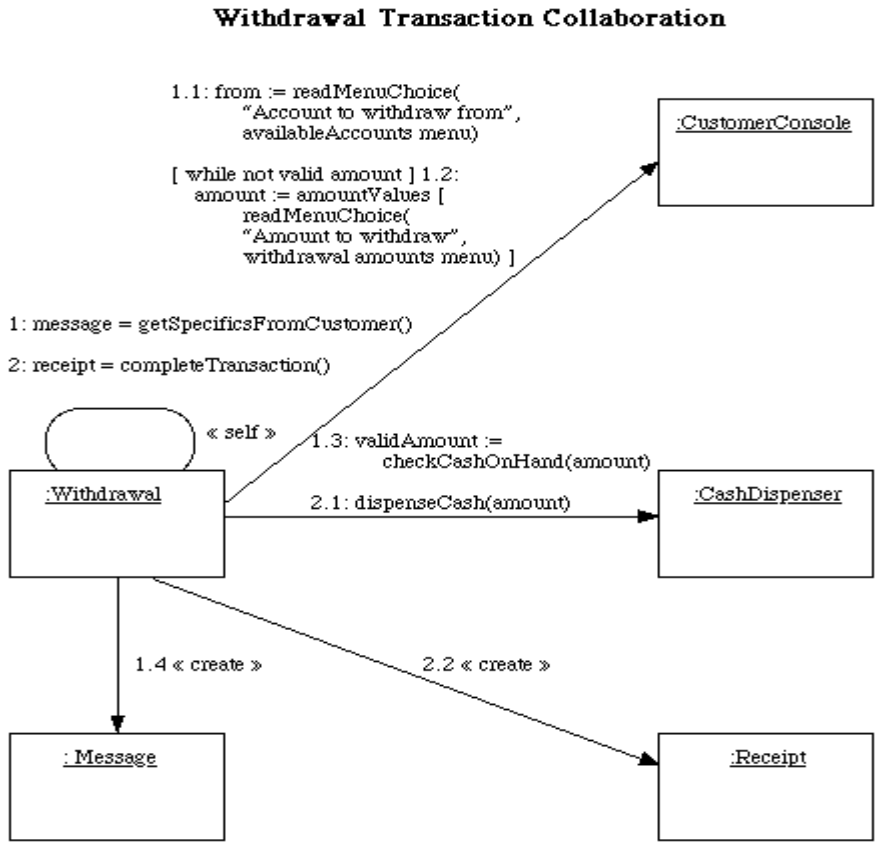


Σχήμα 4.16: Το ακολουθιακό διάγραμμα συνεδρίας πελάτη με το ATM.

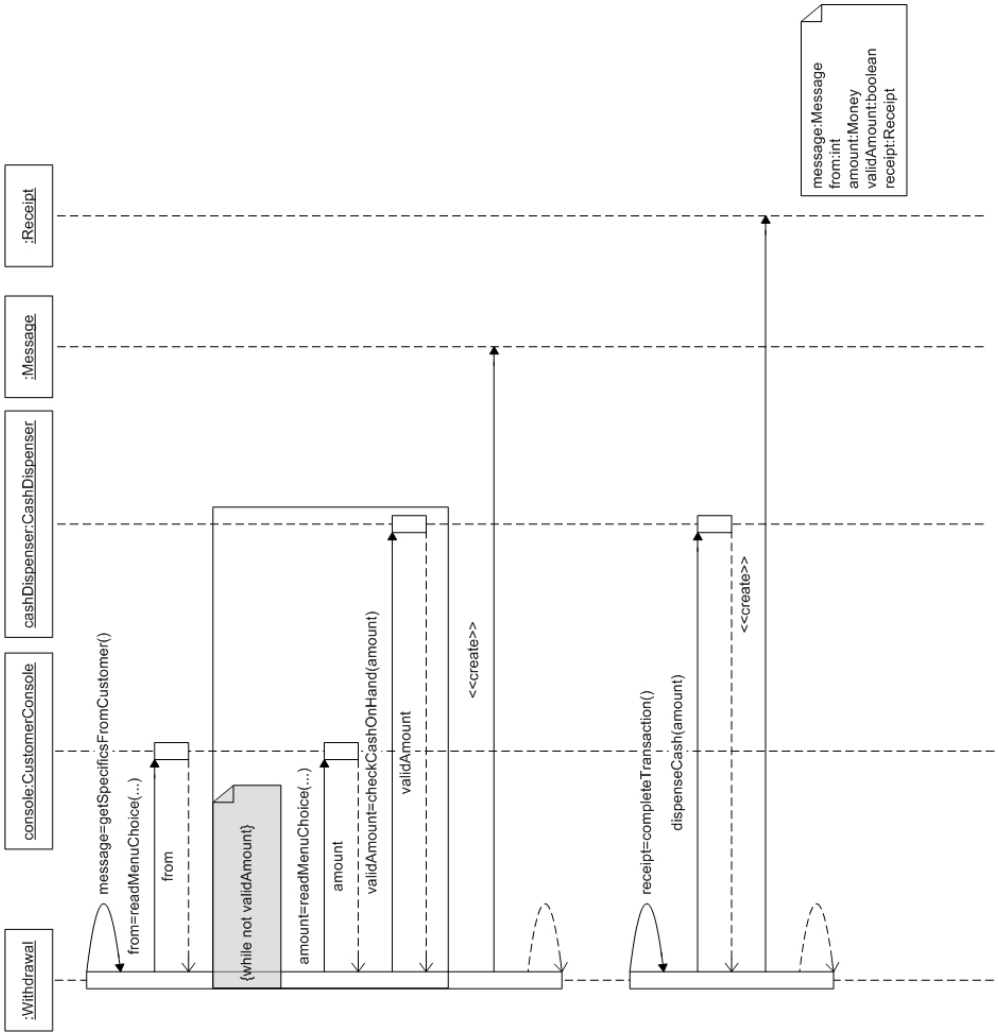


Σχήμα 4.17: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.16 .

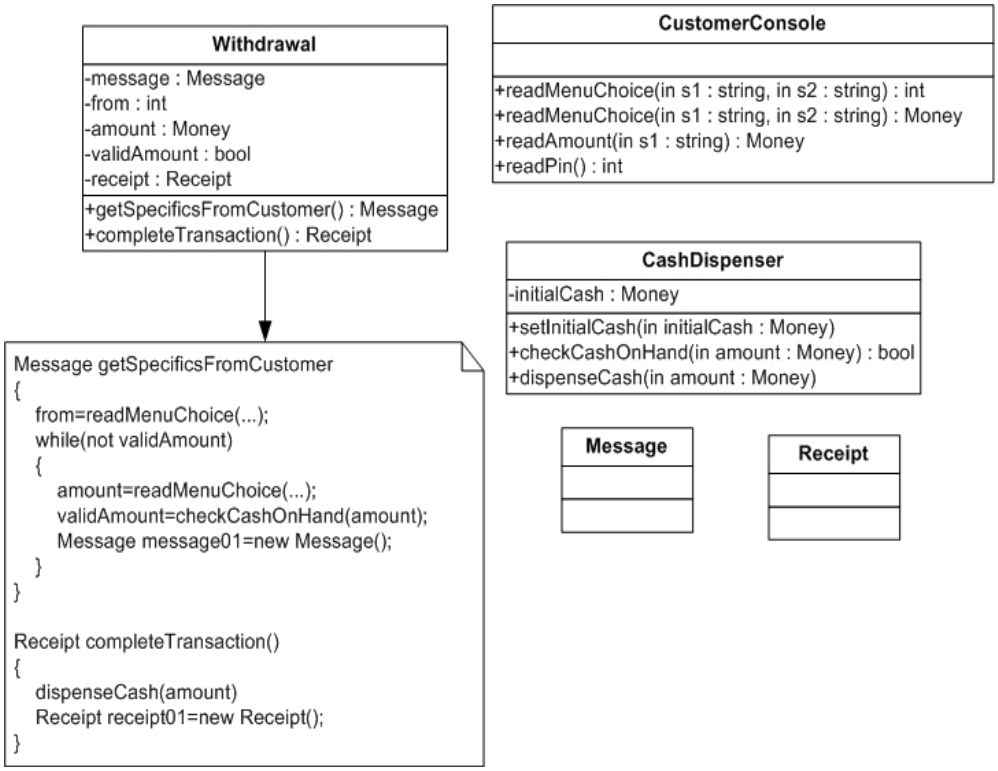
Στη συνέχεια επεξεργαζόμαστε το ακολουθιακό διάγραμμα του σχήματος 4.19 το οποίο μοντελοποιεί την συναλλαγή της ανάληψης χρημάτων. Το ακολουθιακό αυτό διάγραμμα προκύπτει από το ισοδύναμο συνεργατικό διάγραμμα του σχήματος 4.19. Σε αυτό το διάγραμμα υπάρχει ένας βρόγχος επανάληψης ο οποίος δηλώνεται στην γραμμή ετοιμότητας Withdrawal και θεωρούμε ότι ανήκει στην μέθοδο getSpecificsFromCustomer() αφού είναι η τελευταία μέθοδος η οποία δεν έχει επιστρέψει πριν την έναρξη του βρόγχου. Το διάγραμμα του σχήματος 4.20 είναι το διάγραμμα κώδικα που προκύπτει από τον μετασχηματισμό του ακολουθιακού διαγράμματος του σχήματος 4.18



Σχήμα 4.18: Το συνεργατικό διάγραμμα ανάληψης χρημάτων.



Σχήμα 4.19: Το ακολουθιακό διάγραμμα ανάληψης χρημάτων.



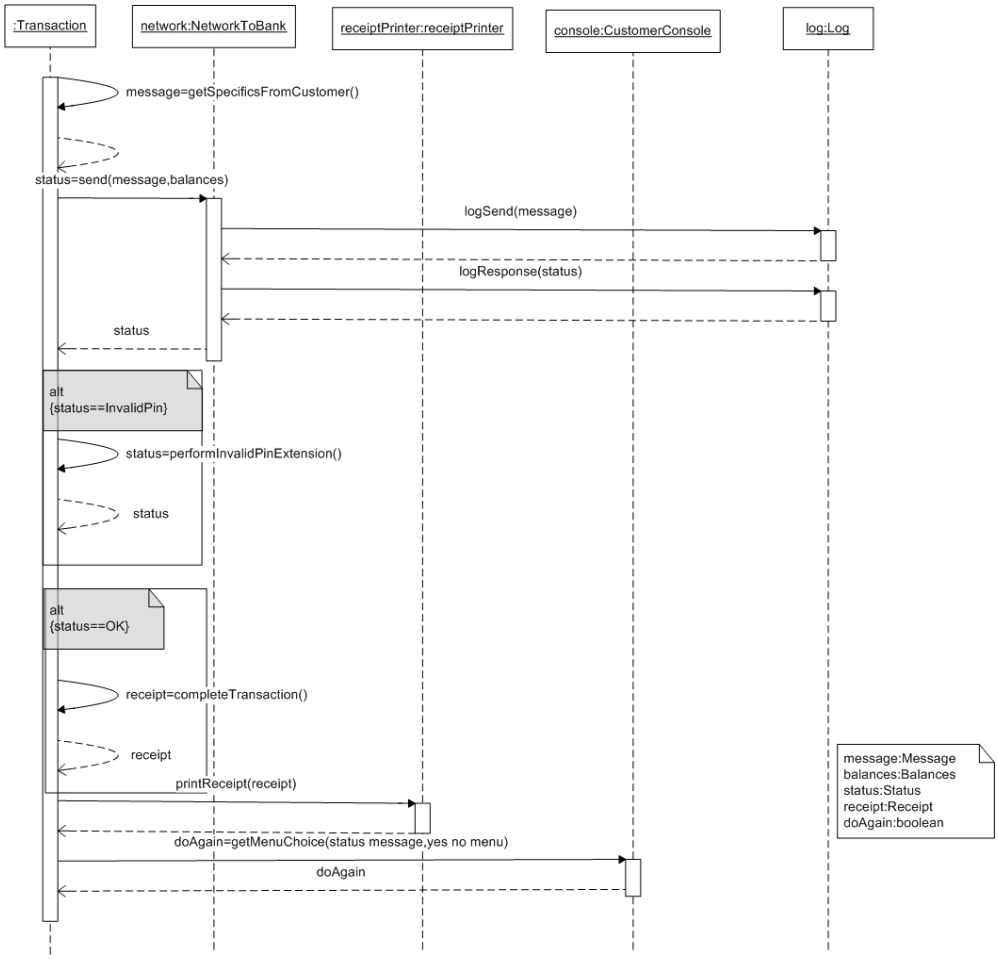
Σχήμα 4.20: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.19 .

4.2.3 Περίπτωση δομών alternative

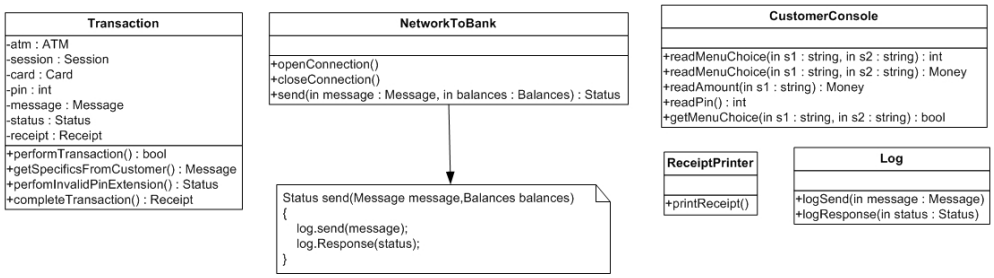
Ο δεύτερος τύπος σύνθετων τμημάτων που θα επεξεργαστούμε είναι οι δομές alternative. Οι alternatives αντιστοιχούν στην δομή ελέγχου ροής if... then ... else. Εκτός από τις alternatives, τα ακολουθιακά διαγράμματα μπορεί να περιέχουν φρουρούς και switches. Στο συγκεκριμένο παράδειγμα που εξετάζουμε σε αυτήν την ενότητα περιέχονται φρουροί οι οποίοι αντιστοιχούν σε alternatives χωρίς το τμήμα else.

Στο σχήμα 4.21 απεικονίζεται το ακολουθιακό διάγραμμα μιας συναλλαγής του πελάτη με το ATM. Στο ακολουθιακό αυτό διάγραμμα υπάρχουν δύο φρουροί οι οποίοι μετατρέπονται σε ισοδύναμα σύνθετα τμήματα alternative τα οποία περιλαμβάνουν μόνο ένα μήνυμα. Κάθε τμήμα alternative το επεξεργαζόμαστε όμοια με τα τμήματα βρόγχου. Θεωρούμε δηλαδή ότι

ανήκει στην τελευταία μέθοδο της γραμμής ετοιμότητας που έχει δηλωθεί και η οποία μέθοδος δεν έχει επιστρέψει. Το αποτέλεσμα της εφαρμογής των κανόνων μετασχηματισμού είναι το διάγραμμα κώδικα του σχήματος 4.22.



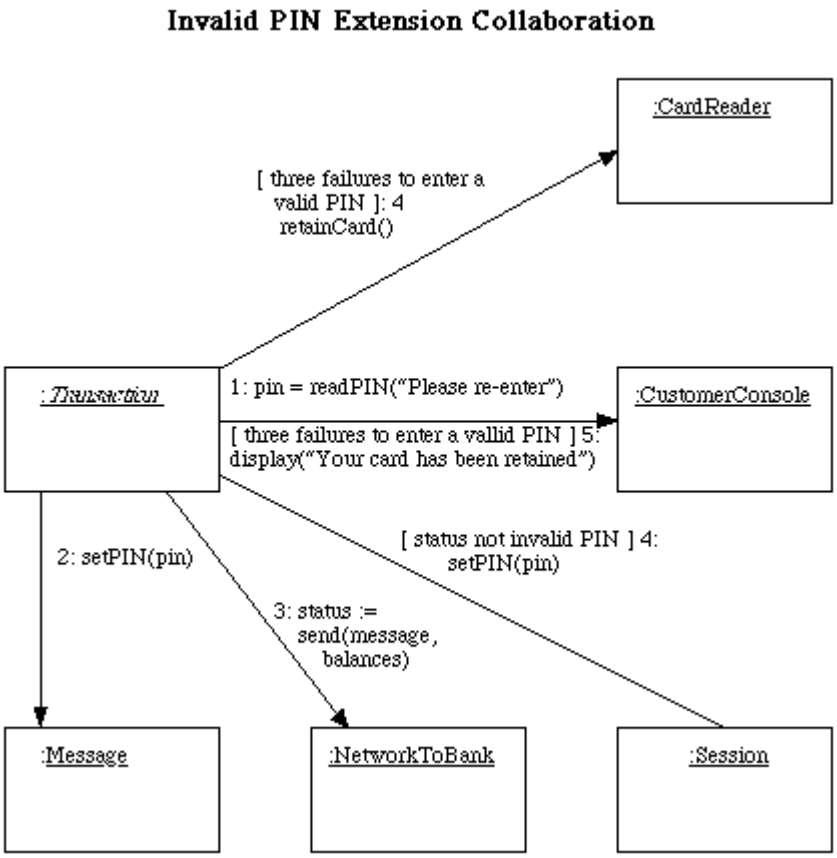
Σχήμα 4.21: Το ακολουθιακό διάγραμμα συναλλαγής πελάτη με ATM.



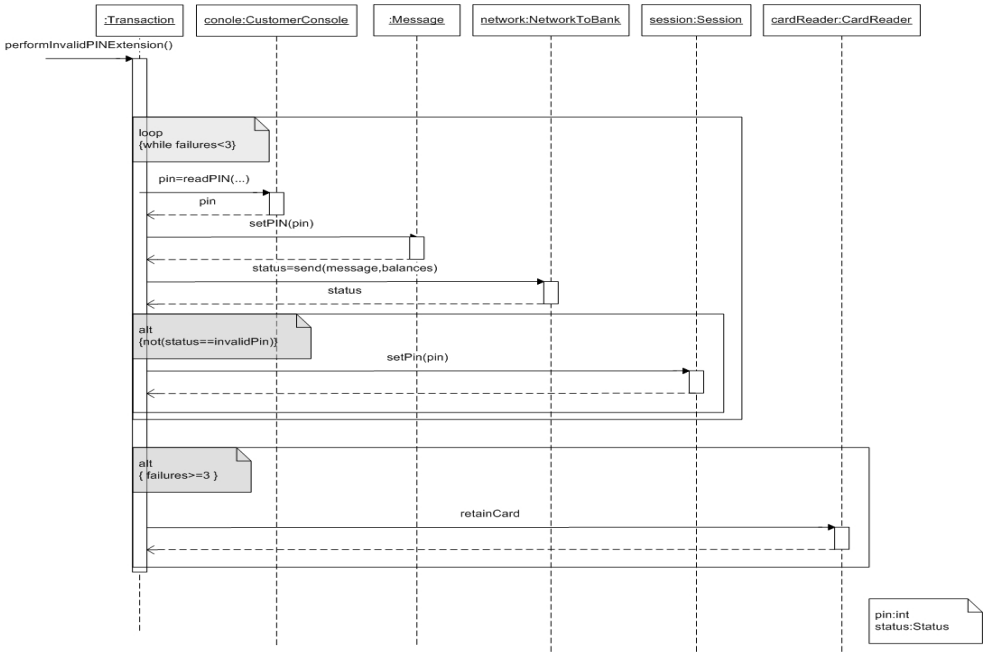
Σχήμα 4.22: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.21 .

Τέλος επεξεργαζόμαστε το ακολουθιακό διάγραμμα του σχήματος 4.24 το οποίο μοντελοποιεί την επικοινωνία των μονάδων του συστήματος στην περίπτωση εισαγωγής λανθασμένου PIN. Το ακολουθιακό αυτό διάγραμμα προκύπτει από το αντίστοιχο συνεργατικό διάγραμμα του σχήματος 4.23. Στο ακολουθιακό διάγραμμα της περίπτωσης λανθασμένου PIN παρατηρούμε την ύπαρξη φωλιασμένου σύνθετου τμήματος. Στη συγκεκριμένη περίπτωση δημιουργείται ένα τμήμα κώδικα (CDCodeBlock) για τον εξωτερικό βρόγχο το οποίο περιέχει δύο τμήματα κώδικα, ένα για τις κλήσεις πριν τον εσωτερικό βρόγχο και ένα για τον εσωτερικό βρόγχο. Αν υπήρχαν κλήσεις και μετά τον εσωτερικό βρόγχο θα δημιουργούταν και ένα τμήμα κώδικα για τις κλήσεις μετά τον βρόγχο. Δημιουργείται επίσης ένα τμήμα κώδικα για τη δομή alternative που έπεται του βρόγχου.

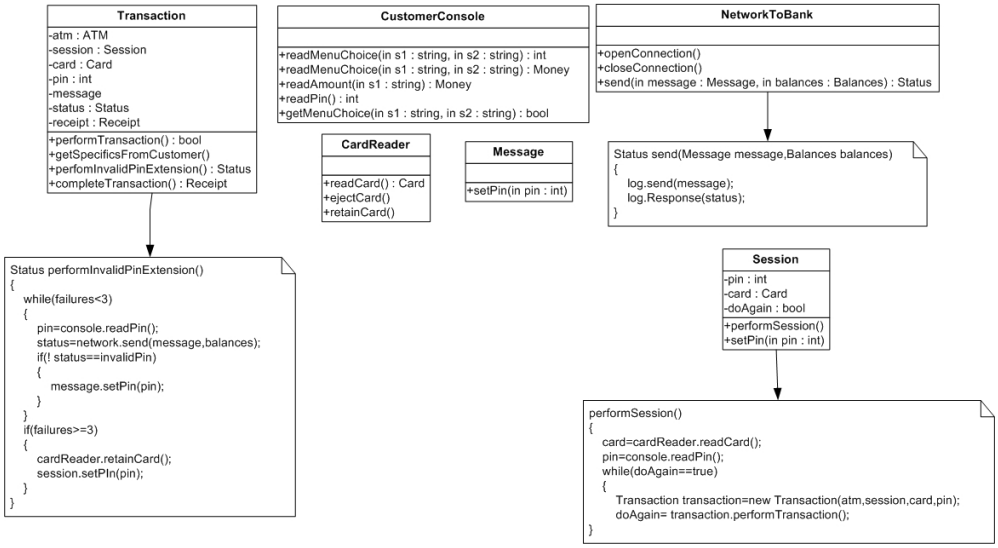
Το αποτέλεσμα της επεξεργασίας όλων των ακολουθιακών διαγραμμάτων είναι το ενιαίο διάγραμμα κώδικα του σχήματος 4.26 το οποίο περιλαμβάνει όλες τις πληροφορίες που εξήχθησαν από τα διαθέσιμα ακολουθιακά διαγράμματα.



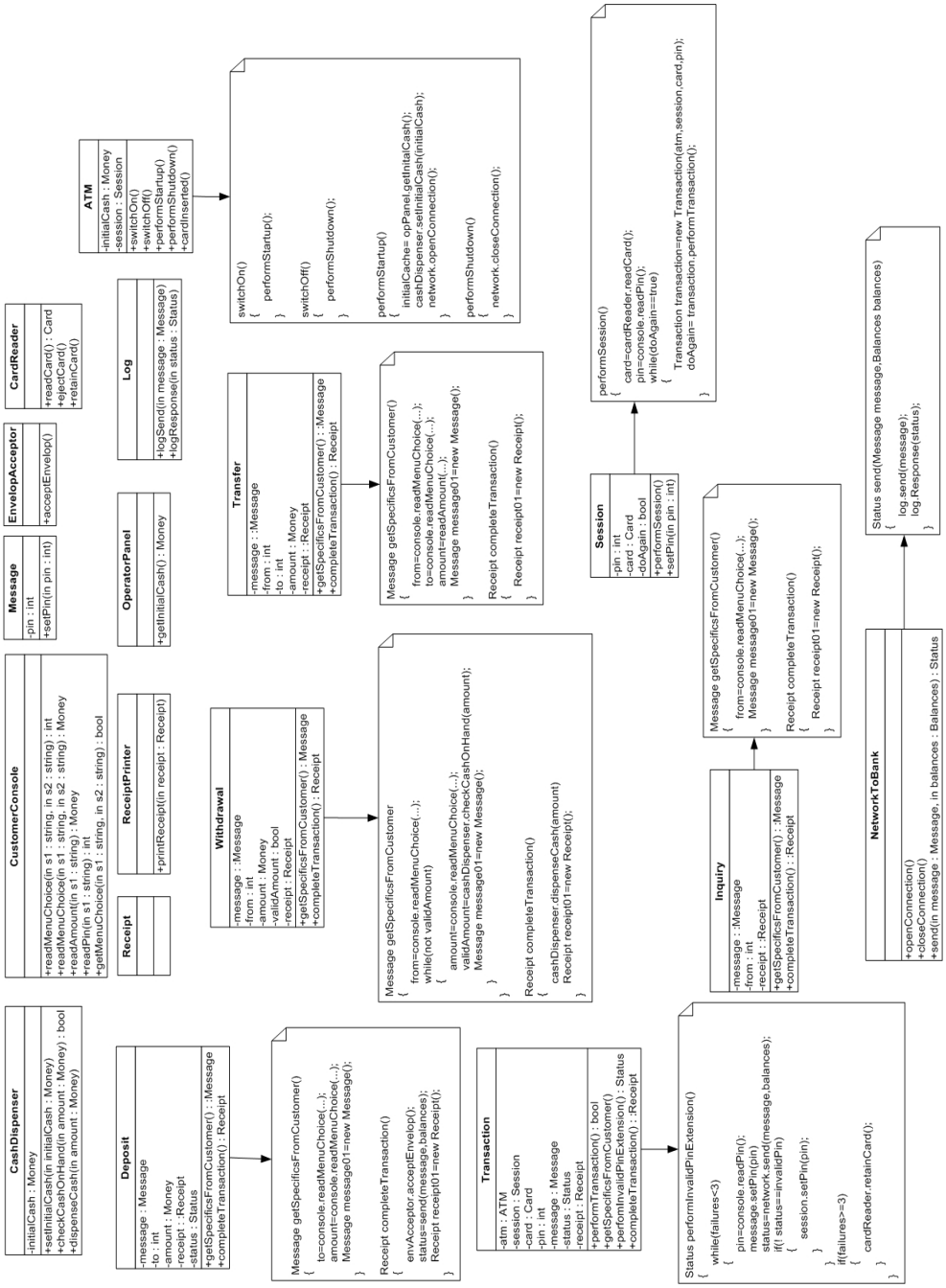
Σχήμα 4.23: Το συνεργατικό διάγραμμα λανθασμένου PIN



Σχήμα 4.24: Το ακολουθιακό διάγραμμα λανθασμένου PIN.



Σχήμα 4.25: Το προκύπτον διάγραμμα κώδικα από το διάγραμμα του σχήματος 4.24 .



Σχήμα 4.26: Συνολικό διάγραμμα κώδικα.

4.2.4 Εύρεση κληρονομικότητας

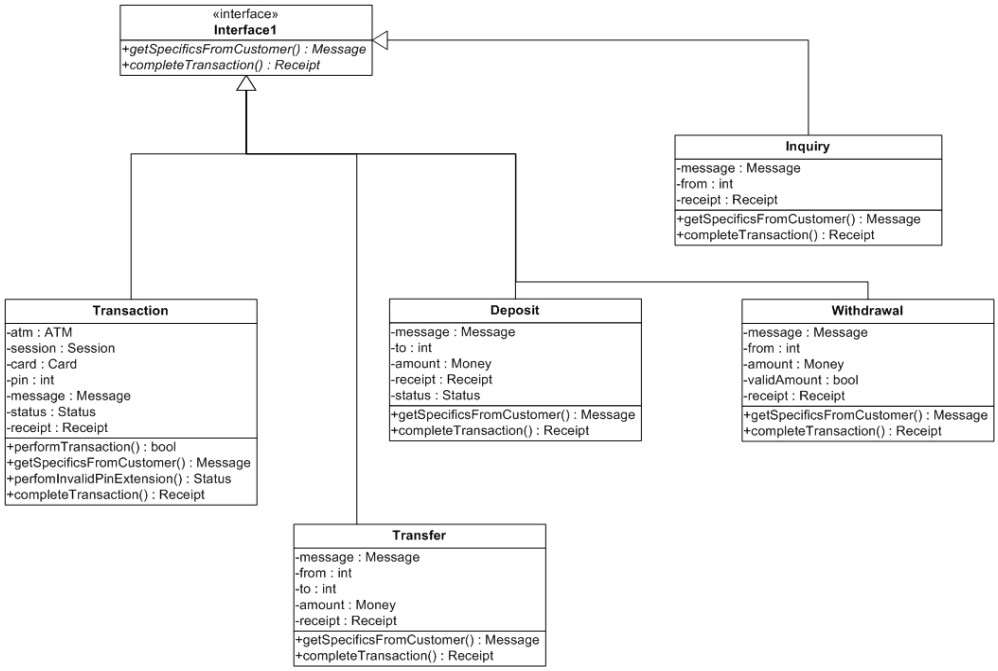
Μετά την επεξεργασία όλων των ακολουθιακών διαγραμμάτων εφαρμόζουμε τους κανόνες της ομάδας Β, οι οποίοι επεξεργάζονται αποκλειστικά δεδομένα του συνολικού διαγράμματος κώδικα ώστε να εξάγουν πληροφορίες για τη δομή του συστήματος, να συμπληρώσουν το διάγραμμα κώδικα και να παράξουν το τελικό μοντέλο του κώδικα του συστήματος.

Εύρεση διαπροσωπειών

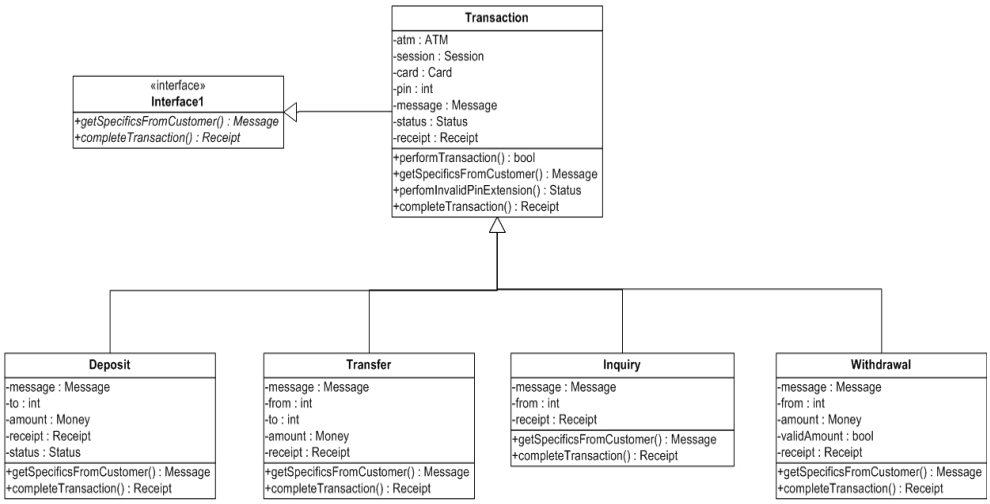
Εφαρμόζουμε τον κανόνα Β1 πάνω στο διάγραμμα κώδικα του σχήματος 4.26. Σύμφωνα με τον συγκεκριμένο κανόνα, οι κλάσεις που έχουν κοινές μεθόδους μεταξύ τους υλοποιούν μια διαπροσωπεία. Όπως φαίνεται και στο σχήμα 4.27, προκύπτει ότι οι κλάσεις Deposit, Trasfer, Withdrawal,Inquiry και Transaction υλοποιούν μια διαπροσωπεία η οποία παίρνει το όνομα Interface01. Η διαπροσωπεία Interface01 περιέχει τις μεθόδους getSpecificsFromCustomer() και completeTransaction(). Η εφαρμογή του κανόνα Β2 δεν θα επιφέρει καμία αλλαγή στο διάγραμμα κώδικα λόγω της υπαρξης μιας μόνο διαπροσωπείας.

Εύρεση υπερκλάσεων.

Μετά την εφαρμογή του κανόνα Β3α δεν εντοπίζουμε κάποια υπερκλάση.
Εάν όμως εφαρμόσουμε τον κανόνα Β3β τότε βρίσκουμε ότι η Transaction είναι υπερκλάση των Deposit,Transfer,Inquiry,και Withdrawal.



Σχήμα 4.27 Η κληρονομικότητα των κλάσεων του ATM όπως προκύπτει από την εφαρμογή του κανόνα Β1.



Σχήμα 4.28 κληρονομικότητα των κλάσεων του ATM όπως προκύπτει από την εφαρμογή του κανόνων Β3α και Β3β

4.3 Σύγκριση του παραγόμενου και του πραγματικού κώδικα.

Μετά την ολοκλήρωση του μετασχηματισμού των ακολουθιακών διαγραμμάτων σε ένα ενιαίο διάγραμμα κώδικα, ακολουθεί η σύγκριση του διαγράμματος κώδικα με τον πραγματικό κώδικα. Ο σκοπός της σύγκρισης του παραγόμενου κώδικα και του πραγματικού κώδικα είναι διπλός. Πρώτον, θέλουμε να ελέγξουμε την ορθότητα του μετασχηματισμού, αν δηλαδή ανταποκρίνεται ο παραγόμενος κώδικας στην δομή της εφαρμογής και την απόκλιση που έχει από αυτή. Δεύτερον, θέλουμε να επαληθεύσουμε την συνέπεια των ακολουθιακών διαγραμμάτων ως προς τον κώδικα. Να ελέγξουμε δηλαδή αν τα ακολουθιακά διαγράμματα,και κατά συνέπεια και ο παραγόμενος κώδικας, αποκλίνουν από την πραγματική συμπεριφορά του συστήματος. Θα πρέπει να τονίσουμε ότι ο μετασχηματισμός των ακολουθιακών διαγραμμάτων σε διαγράμματα κώδικα δεν αποσκοπεί στην πλήρη περιγραφή και μοντελοποίηση της δομής του συστήματος αλλά στην σκιαγράφηση των βασικών χαρακτηριστικών του καθώς και στην δημιουργία ενός βασικού σκελετού του κώδικα του συστήματος ώστε να γίνει πιο εύκολη η σύγκριση του κώδικα με τα ακολουθιακά διαγράμματα.
Για να συγκρίνουμε τον διάγραμμα κώδικα με τον πραγματικό κώδικα του συστήματος εισάγουμε σε αυτό το σημείο ένα σύνολο κριτηρίων. Η σειρά με την οποία έχουν δηλωθεί τα κριτήρια υποδηλώνει και έναν αυξανόμενο βαθμό συνέπειας του διαγράμματος κώδικα με τον πραγματικό κώδικα. Ξεκινάμε δηλαδή τη σύγκριση ελέγχοντας βασικά στοιχεία όπως οι κλάσεις και στο τέλος συγκρίνουμε το σώμα των μεθόδων του διαγράμματος κώδικα με τον πραγματικό κώδικα. Τα κριτήρια που θα χρησιμοποιήσουμε είναι σχεδόν αντίστοιχα με τους κανόνες μετασχηματισμού.

<u>Κριτήριο 1ο: Επαλήθευση των κλάσεων</u>

Επαληθεύουμε ότι οι κλάσεις που παρήχθησαν από το μετασχηματισμό όντως υπάρχουν και στο πραγματικό σύστημα

<u>Κριτήριο 2ο: Επαλήθευση των μεθόδων</u>

Επαληθεύουμε ότι οι μέθοδοι που έχουν οριστεί σε κάθε κλάση του

διαγράμματος κώδικα, έχουν οριστεί και στις αντίστοιχες κλάσεις του πραγματικού συστήματος. Η επαλήθευση γίνεται ως προς τις παραμέτρους κάθε μεθόδου, τον τύπο επιστροφής της και το όνομά της.

Κριτήριο 3ο: Επαλήθευση των πεδίων

Επαληθεύουμε ότι τα πεδία που έχουν οριστεί σε κάθε κλάση του διαγράμματος κώδικα, έχουν οριστεί και στις αντίστοιχες κλάσεις του πραγματικού συστήματος.

Κριτήριο 4ο: Επαλήθευση του σώματος των μεθόδων

Επαληθεύουμε την δομή του σώματος κάθε μεθόδου. Το κριτήριο αυτό αποτελείται από τρία σκέλη

α) επαλήθευση των κλήσεων των μεθόδων

Σε αυτό το σκέλος επαληθεύουμε τις κλήσεις των μεθόδων που πραγματοποιούνται μέσα στο σώμα της υπό εξέταση μεθόδου χωρίς να μας ενδιαφέρει η σειρά με την οποία πραγματοποιήθηκαν.

β) επαλήθευση της ύπαρξης των σύνθετων τμημάτων

Σε αυτό το σκέλος ελέγχουμε αν το προβλεπόμενο από το διάγραμμα κώδικα σύνθετο τμήμα υπάρχει στο σώμα της αντίστοιχης μεθόδου του πραγματικού κώδικα και αν η συνθήκη ελέγχου του σύνθετου τμήματος του διαγράμματος κώδικα είναι συνεπής με τη συνθήκη ελέγχου του πραγματικού κώδικα.

γ) επαλήθευση της σειράς των κλήσεων των μεθόδων

Εφόσον έχουμε επαληθεύσει ότι κλήσεις των μεθόδων όντως υπάρχουν μέσα στο σώμα της υπό εξέταση μεθόδου ελέγχουμε αν η σειρά με την οποία καλούνται στο διάγραμμα κώδικα αντιστοιχεί στη σειρά με την οποία καλούνται στην εφαρμογή. Σε περίπτωση σύνθετου τμήματος εξετάζουμε την θέση του σύνθετου τμήματος καθώς και την σειρά των κλήσεων των μεθόδων μέσα στο σύνθετο τμήμα.

Κριτήριο 5ο: Επαλήθευση της κληρονομικότητας

Ελέγχουμε αν η ιεραρχία των κλάσεων είναι αυτή που έχει οριστεί στο διάγραμμα κώδικα και αν υπάρχουν οι προβλεπόμενες διαπροσωπείες.

Εφαρμογή των κριτηρίων

Κριτήριο 1ο:

Εφαρμόζοντας το 1ο κριτήριο διαπιστώνουμε ότι κάθε κλάση του διαγράμματος κώδικα υπάρχει και στην πραγματική εφαρμογή. Δεν έχουν βρεθεί όμως όλες οι κλάσεις του συστήματος. Οι κλάσεις που δεν εντοπίστηκαν είναι οι κλάσεις που δεν δηλώθηκαν σε κάποια γραμμή ετοιμότητας. Μερικές από αυτές τις κλάσεις εμφανίζονται στα ακολουθιακά διαγράμματα σαν τύποι δεδομένων όπως για παράδειγμα η κλάση Money. Αυτοί οι τύποι δεδομένων δεν μπορούν να θεωρηθούν κλάσεις του συστήματος καθώς με αυτόν τον τρόπο θα έχουμε την δημιουργία κλάσεων int, String κλπ. Καταχωρούνται πάντως σαν τύποι δεδομένων που διαχειρίζεται το σύστημα.

Πίνακας 4.1: Πίνακας των κλάσεων της εφαρμογής και του διαγράμματος κώδικα.

Κλάσεις της εφαρμογής	**Κλάσεις του διαγράμματος κώδικα**
ATM	ATM
Session	Session
CardReader	CardReader
CashDispenser	CashDispenser
CustomerConsole	CustomerConsole
EnvelopAcceptor	EnvelopAcceptor
Log	Log
NetworkToBank	NetworkToBank
OperatorPaner	OperatorPanel

Deposit	Deposit
Inquiry	Inquiry
Transaction	Transaction
Transfer	Transfer
Withdrawal	Withdrawal
AccountInfomation	-
Balances	«Datatype» Balances
Card	«Datatype» Card
Message	Message
Money	«Datatype» Money
Receipt	Receipt
Status	«Datatype» Status

Κριτήριο 2ο:

Για κάθε κλάση του διαγράμματος κώδικα ελέγχεται εάν οι δηλωμένες σε αυτήν την κλάση μέθοδοι έχουν δηλωθεί και στην αντίστοιχη κλάση του πραγματικού κώδικα. Η συνέπεια με τον κώδικα ελέγχεται ως προς το όνομα της μεθόδου, τις παραμέτρους που δέχεται κάθε μέθοδος καθώς επίσης και τον τύπο επιστροφής της κάθε μεθόδου. Το αντίστροφο δεν το ελέγχουμε διότι θεωρούμε εν γένει ότι στα ακολουθιακά διαγράμματα δεν έχουν αναφερθεί όλες οι μέθοδοι των κλάσεων.

Πίνακας 4.2:Πίνακας επαλήθευσης ύπαρξης των μεθόδων του διαγράμματος κώδικα.

Κλάση διαγράμματος κώδικα	Μέθοδος διαγράμματος κώδικα	Επαλήθευση
ATM	switchOn()	ΝΑΙ
ATM	switchOff()	ΝΑΙ
ATM	performStartup()	ΝΑΙ
ATM	performShutdown()	ΝΑΙ
ATM	cardInserted()	ΝΑΙ
Session	performSession()	ΝΑΙ
Session	setPin(pin:int)	ΝΑΙ
CardReader	readCard():Card	ΝΑΙ
CardReader	ejectCard()	ΝΑΙ
CardReader	retainCard()	ΝΑΙ
CashDispenser	setInitialCash (initialCash:Money)	ΝΑΙ
CashDispenser	checkCashOnHand (amount:Money)	ΝΑΙ
CashDispenser	dispenseCash (amount:Money)	ΝΑΙ
CustomerConsole	readMenuChoice (s1:String,s2:String):int	ΝΑΙ
CustomerConsole	readMenuChoice (s1:String, s2:String):Money	ΟΧΙ
CustomerConsole	readAmout (s1:String):Money	ΝΑΙ
CustomerConsole	readPin(s1:String):int	ΝΑΙ
CustomerConsole	getMenuChoice (s1:String, s2String):Money	ΟΧΙ
EnvelopAcceptor	acceptEnvelop()	ΝΑΙ
Log	logSend (message:Message)	ΝΑΙ
Log	logResponse (status:Status)	ΝΑΙ

NetworkToBank	openConnection()	ΝΑΙ
NetworkToBank	closeConnection()	ΝΑΙ
NetworkToBank	send(message:Message, balances:Balances) :Status	ΟΧΙ διαφορετικό όνομα
OperatorPanel	getInitialCash():Money	ΝΑΙ
ReceiptPrinter	printReceipt (receipt:Receipt)	ΝΑΙ
Deposit	getSpecificsFromCustomer() :Message	ΝΑΙ
Deposit	completeTransaction():Receipt	ΝΑΙ
Inquiry	getSpecificsFromCustomer() :Message	ΝΑΙ
Inquiry	completeTransaction():Receipt	ΝΑΙ
Transaction	performTransaction():boolean	ΝΑΙ
Transaction	getSpecificsFromCustomer() :Message	ΝΑΙ
Transaction	completeTransaction():Receipt	ΝΑΙ
Transaction	performInvalidPinExtension(): Status	ΝΑΙ
Transfer	getSpecificsFromCustomer() :Message	ΝΑΙ
Transfer	completeTransaction():Receipt	ΝΑΙ
Withdrawal	getSpecificsFromCustomer() :Message	ΝΑΙ
Withdrawal	completeTransaction():Receipt	ΝΑΙ
Message	setPin(pin:int)	ΝΑΙ

Εξετάζοντας τον πίνακα 4.2 παρατηρούμε ότι οι περισσότερες μέθοδοι που εντοπίστηκαν είναι συνεπείς με τον κώδικα ως προς το όνομα, το πλήθος και τον τύπο των παραμέτρων και τον τύπο επιστροφής. Υπάρχουν όμως και μέθοδοι που είτε δεν υπάρχουν καθόλου στον κώδικα του συστήματος, όπως

για παράδειγμα η μέθοδος getMenuChoice() της κλάσης CustomerConsole, είτε υπάρχουν αλλά διαφέρουν ως προς κάποιο χαρακτηριστικό, όπως για παράδειγμα η μέθοδος send() της κλάσης NetworkToBank. Στον κώδικα υπάρχει η μέθοδος sendMessage() η οποία δέχεται τις ίδιες παραμέτρους και επιστρέφει τον ίδιο τύπο δεδομένων με την send(). Επίσης στην μέθοδο CustomerConsole έχει δηλωθεί δύο φορές η μέθοδος readMenuChoice(). Τη πρώτη φορά η μέθοδος δηλώνεται ότι ο τύπος επιστροφής είναι int ενώ τη δεύτερη φορά η μέθοδος δηλώνεται ότι ο τύπος επιστροφής είναι Money.

Οι ασυνέπειες μεταξύ του διαγράμματος κώδικα και του πραγματικού κώδικα του συστήματος οφείλονται σε σχεδιαστικές ασυνέπειες μεταξύ των ακολουθιακών διαγραμμάτων και του πραγματικού κώδικα

Κριτήριο 3ο:

Για κάθε κλάση του διαγράμματος κώδικα ελέγχεται εάν τα δηλωμένα σε αυτήν την κλάση πεδία έχουν δηλωθεί και στην αντίστοιχη κλάση του πραγματικού κώδικα. Το αντίστροφο δεν το ελέγχουμε διότι θεωρούμε εν γένει ότι στα ακολουθιακά διαγράμματα δεν έχουν αναφερθεί όλα τα πεδία των κλάσεων.
Στο πίνακα 4.3 παρατίθενται όλα τα πεδία των κλάσεων του διαγράμματος κώδικα καθώς και η επαλήθευση αν το πεδίο υπάρχει και στον πραγματικό κώδικα.

Πίνακας 4.3: Πίνακας επαλήθευσης ύπαρξης των πεδίων των κλάσεων του διαγράμματος κώδικα.

Κλάση διαγράμματος κώδικα	**Προβλεπόμενο πεδίο**	**Επαλήθευση πεδίου**
ATM	initialCash:Money	ΟΧΙ
ATM	session:Session	ΟΧΙ
Session	card:Card	ΟΧΙ
Session	pin:int	ΝΑΙ
Session	doAgain:boolean	ΟΧΙ
CashDispenser	initialCash:Money	ΝΑΙ-Άλλο όνομα
Deposit	to:int	ΝΑΙ

Deposit	amount:Money	NAI
Deposit	message:Message	NAI- protected πεδίο υπερκλάσης
Deposit	receipt:Receipt	OXI
Deposit	status:Status	OXI
Inquiry	from:int	NAI
Inquiry	message:Message	NAI-protected πεδίο υπερκλάσης
Inquiry	receipt:Receipt	OXI
Transaction	atm:ATM	NAI
Transaction	session:Session	NAI
Transaction	card:Card	NAI
Transaction	pin:int	NAI
Transaction	status:Status	OXI
Transaction	message:Message	NAI
Transaction	receipt:Receipt	OXI
Transfer	from	NAI
Transfer	to	NAI
Transfer	amount	NAI
Transfer	message	NAI-protetected πεδίο υπερκλάσης
Transfer	receipt	OXI
Withdrawal	from	NAI
Withdrawal	amount	NAI
Withdrawal	validAmount	OXI
Withdrawal	message	NAI
Withdrawal	receipt	OXI
Message	pin	NAI

Παρατηρούμε ότι υπάρχει σημαντική απόκλιση στην εύρεση των πεδίων των

κλάσεων του συστήματος. Η απόκλιση αυτή οφείλεται στο γεγονός ότι σύμφωνα με τις προδιαγραφές των ακολουθιακών διαγραμμάτων όπως αυτές περιγράφονται στο [8], δεν διακρίνεται στα ακολουθιακά διαγράμματα αν μια μεταβλητή αποτελεί τοπική μεταβλητή μιας μεθόδου ή πεδίο μιας κλάσης. Η πληροφορία αυτή μπορεί να εξαχθεί μόνο από τα διαγράμματα κλάσης του συστήματος. Οποιαδήποτε άλλη προσέγγιση εντοπισμού των πεδίων των κλάσεων του συστήματος και διαχωρισμού τους από τις τοπικές μεταβλητές των μεθόδων έχει ευριστικό χαρακτήρα και δεν μπορεί να εξασφαλιστεί εκ των προτέρων η ορθότητα του αποτελέσματος. Μόνος τρόπος είναι ο προσδιορισμός των πεδίων του συστήματος μέσα στο ακολουθιακό διάγραμμα. Θα πρέπει επίσης να προσθέσουμε ότι στόχος μας δεν είναι να εντοπίσουμε μέσω των ακολουθιακών διαγραμμάτων όλα τα πεδία των κλάσεων του συστήματος, αλλά μόνο εκείνα τα πεδία τα οποία συμμετέχουν στην επικοινωνία και την ανταλλαγή δεδομένων μεταξύ των κλάσεων.

Κριτήριο 4ο:

α)
Για κάθε κλάση του διαγράμματος κώδικα για την οποία είναι διαθέσιμο το σώμα των μεθόδων της εξετάζουμε τις κλήσεις που γίνονται στο σώμα κάθε μεθόδου ανεξάρτητα από τη σειρά που κλήθηκαν. Τα αποτελέσματα της σύγκρισης συνοψίζονται στον πίνακα 4.4.

Στην περίπτωση της κλάσης ATM η ασυνέπεια με τον κώδικα οφείλεται στο γεγονός ότι η κλάση ATM υλοποιείται πολυνηματικά και οι μέθοδοι καλούνται με την χρήση μιας δομής switch και την κλήση της μεθόδου notify() μέσα στο σώμα της μεθόδου run(). Αντίθετα στο ακολουθιακό διάγραμμα η εκκίνηση και η παύση του συστήματος μοντελοποιούνται ως σειριακή κλήση μεθόδων. Κατά συνέπεια ενώ η σειρά των καλούμενων μεθόδων είναι αυτή που προβλέπεται από το διάγραμμα κώδικα, οι κλήσεις των μεθόδων performStartup() και performShutdown() δεν γίνονται μέσα στα σώματα των μεθόδων switchOn() και switchOff() αντίστοιχα αλλά μέσα στο σώμα της μεθόδου run().

Πίνακας 4.4: Πίνακας επαλήθευσης των κλήσεων των μεθόδων μέσα στο σώμα κάθε μεθόδου

Κλάση διαγράμματος κώδικα	Μέθοδος κλάσης	Επαλήθευση
ATM	switchOn()	ΟΧΙ
ATM	switchOff()	ΟΧΙ
ATM	performStartup()	ΝΑΙ
ATM	performShutdown()	ΝΑΙ
Session	performSession()	ΝΑΙ
NetworkToBank	send()	ΝΑΙ
Deposit	getSpecificsFromCustomer()	ΝΑΙ
Deposit	completeTransaction()	ΝΑΙ
Inquiry	getSpecificsFromCustomer()	ΝΑΙ
Inquiry	completeTransaction()	ΝΑΙ
Transaction	performInvalidPinExtension()	ΝΑΙ
Transfer	getSpecificsFromCustomer()	ΝΑΙ
Transfer	completeTransaction	ΝΑΙ
Withdrawal	getSpecificsFromCustomer()	ΝΑΙ
Withdrawal	completeTransaction()	ΝΑΙ

β)

Στο β σκέλος του κριτηρίου για κάθε μέθοδο του διαγράμματος κώδικα που εξετάστηκε στο α σκέλος και βρέθηκε ότι είναι συνεπής ως προς τον κώδικα της εφαρμογής και περιέχει σύνθετο τμήμα εξετάζουμε την ύπαρξη βρόγχων ανεξαρτήτως θέσης μέσα στο σώμα της μεθόδου και συγκρίνουμε τις συνθήκες ελέγχου. Τα αποτελέσματα της σύγκρισης παρουσιάζονται στον πίνακα 4.5

Πίνακας 4.5 Πίνακας επαλήθευσης των βρόγχων μέσα στο σώμα κάθε μεθόδου

Κλάση διαγράμμ. κώδικα	Μέθοδος κλάσης	Σύνθετο τμήμα	Επαλήθευση
Transaction	performInvalidPinExtension()	while(efforts<3)	ΝΑΙ
Transaction	performInvalidPinExtension()	if not (status ==invalidPin)	ΝΑΙ
Transaction	perfomInvalidPinExtension()	if(failures>3)	ΝΑΙ
Session	performSession()	while (customer wants to perform Transactions)	ΝΑΙ
Withdrawal	getSpecificsFromCustomer()	while (not validAmount)	ΝΑΙ

γ)

Στο τελευταίο σκέλος του 4^{ου} κριτηρίου, για κάθε μέθοδο που βρέθηκε συνεπής με τον κώδικα στο α΄ σκέλος, εξετάζουμε την σειρά με την οποία καλούνται οι μέθοδοι και την θέση των βρόγχων μέσα στο σώμα κάθε μεθόδου του διαγράμματος κώδικα και συγκρίνουμε τα δεδομένα με τον πραγματικό κώδικα της εφαρμογής.

Πίνακας 4.6: Πίνακας επαλήθευσης της σειράς των κλήσεων των μεθόδων μέσα στο σώμα κάθε μεθόδου.

Κλάση διαγράμματος κώδικα	Μέθοδος κλάσης	Επαλήθευση
ATM	performStartup()	ΝΑΙ
ATM	performShutdown()	ΝΑΙ
Session	performSession()	ΝΑΙ
NetworkToBank	send()	ΝΑΙ
Deposit	getSpecificsFromCustomer()	ΝΑΙ
Deposit	completeTransaction()	ΝΑΙ
Inquiry	getSpecificsFromCustomer()	ΝΑΙ
Inquiry	completeTransaction()	ΝΑΙ
Transaction	performInvalidPinExtension()	ΝΑΙ
Transfer	getSpecificsFromCustomer()	ΝΑΙ
Transfer	completeTransaction	ΝΑΙ
Withdrawal	getSpecificsFromCustomer()	ΝΑΙ
Withdrawal	completeTransaction()	ΝΑΙ

Κριτήριο 5ο:

Για κάθε διαπροσωπεία ελέγχουμε αν υπάρχει διαπροσωπεία στον κώδικα με το ίδιο σύνολο μεθόδων. Για κάθε υπερκλάση του διαγράμματος κώδικα ελέγχουμε αν η κλάση αυτή είναι υπερκλάση και στον πραγματικό κώδικα και σε περίπτωση που όντως είναι συγκρίνουμε τις υποκλάσεις.

Το διάγραμμα κώδικα περιέχει μια διαπροσωπεία Interface01 που υλοποιείται από την κλάση Transaction. Η κλάση αυτή είναι υπερκλάση των

κλάσεων Deposit,Transfer,Inquiry και Withdrawal. Συγκρίνοντας το διάγραμμα κώδικα με τον πραγματικό κώδικα παρατηρούμε ότι δεν υπάρχει κάποια διαπροσωπεία που να υλοποιείται από την κλάση Transaction. Η ύπαρξη της διαπροσωπείας δεν αποτελεί ουσιαστική ασυνέπεια με τον κώδικα. Η διαπροσωπεία αυτή στην ουσία είναι ενδεικτική της κοινής συμπεριφοράς των κλάσεων που υλοποιούν άμεσα ή έμμεσα την διαπροσωπεία. Η κλάση όμως Transaction είναι όντως υπερκλάση των υπολοίπων κλάσεων που αναφέρθηκαν πιο πάνω. Συνολικά η προσέγγιση της κληρονομικότητας των κλάσεων του συστήματος έγινε σε ικανοποιητικό βαθμό

4.4 Περίληψη

Σε αυτό το κεφάλαιο εφαρμόσαμε τους κανόνες μετασχηματισμού των ακολουθιακών διαγραμμάτων πάνω σε μία ολοκληρωμένη εφαρμογή η οποία έχει υλοποιηθεί σύμφωνα με τις αρχές του αντικειμενοστραφούς προγραμματισμού και της Μοντελοκεντρικής Αρχιτεκτονικής. Η σύγκριση του παραγόμενου διαγράμματος κώδικα από τον μετασχηματισμό των ακολουθιακών διαγραμμάτων με τον πραγματικό κώδικα μας έδωσε τη δυνατότητα να ορίσουμε σημαντικό μέρος της αρχιτεκτονικής της εφαρμογής,να κατασκευάσουμε έναν σκελετό του κώδικα ο οποίος να μοντελοποιεί τις σημαντικότερες λειτουργίες του συστήματος και να εντοπίσουμε ασυνέπειες μεταξύ του πραγματικού κώδικα και των ακολουθιακών διαγραμμάτων καθώς και τις σχεδιαστικές ελλείψεις των ακολουθιακών διαγραμμάτων χωρίς τις οποίες θα είχαμε ένα πληρέστερο διάγραμμα κώδικα και συνεπώς ακριβέστερη μοντελοποίηση της δομής και των λειτουργιών του συστήματος.

Κεφάλαιο 5

Επίλογος

Στο κεφάλαιο αυτό κάνουμε μια ανακεφαλαίωση της διπλωματικής εργασίας παρουσιάζοντας την συμβολή της στον ερευνητικό χώρο της τεχνολογίας λογισμικού. Αναφέρουμε τα συμπεράσματα και την κάλυψη των αρχικών στόχων που είχαμε θέσει στο πρώτο κεφάλαιο. Επίσης παρουσιάζουμε κάποιες μελλοντικές ερευνητικές κατευθύνσεις που προκύπτουν από την εργασία αυτή.

5.1 Αποτίμηση

Παρακάτω παρουσιάζουμε τα γενικά συμπεράσματα της διπλωματικής εργασίας, αντίστοιχα με τους στόχους τους οποίους θέσαμε στο πρώτο κεφάλαιο.

Ορίσαμε ένα μεταμοντέλο ακολουθιακών διαγραμμάτων αρκετά πιο λιτό και μινιμαλιστικό από αυτό που ορίζεται στην UML και περιγράφεται στο [8]. Το μεταμοντέλο αυτό αν και δεν έχει τις εκφραστικές ικανότητες του μεταμοντέλου που περιγράφεται στο [8] μοντελοποιεί τα περισσότερα και πιο ευρέως χρησιμοποιούμενα χαρακτηριστικά των διαγραμμάτων, δίνοντας έμφαση σε εκείνα τα χαρακτηριστικά τα οποία επεξεργαζόμαστε σε αυτήν την εργασία. Κάθε χαρακτηριστικό το οποίο δεν έχει συμπεριληφθεί, μπορεί να προστεθεί στο διάγραμμα επεκτείνοντας το, ενώ ταυτόχρονα να μην προκληθούν ασυμβατότητες με ήδη υπάρχοντα διαγράμματα. Το μεταμοντέλο ακολουθιακών διαγραμμάτων υπακούει στο πρότυπο Ecore του Eclipse το οποίο είναι παραπλήσιο του EMOF και συνεπώς είναι εύκολα μετατρέψιμο σε ένα μεταμοντέλο που να υπακούει στο EMOF. Ο λόγος για τον οποίο επιλέχθηκε το Ecore είναι η πληθώρα εργαλείων που διατίθενται για την επεξεργασία Ecore διαγραμμάτων. Μέσω των εργαλείων του Eclipse παρέχουμε ταυτόχρονα μια γραφική απεικόνιση του μεταμοντέλου μας αλλά και μια σειριοποιημένη απεικόνιση μέσω ενός αρχείου XML το οποίο υπακούει στο πρότυπο XMI. Ταυτόχρονα μέσω της δημιουργίας μιας διεπαφής προγραμματισμού εφαρμογών(API) μας δίνεται η δυνατότητα να

κατασκευάσουμε ακολουθιακά διαγράμματα τα οποία θα αποθηκεύονται επίσης σε σειριοποιημένη μορφή συμβατή με το πρότυπο XMI. Η σειριοποιημένη μορφή του μεταμοντέλου και των ακολουθιακών διαγραμμάτων καθώς και η διεπαφή προγραμματισμού εφαρμογών μας δίνει τη δυνατότητα να επεξεργαστούμε τα ακολουθιακά διαγράμματα σε οποιαδήποτε πλατφόρμα ανεξαρτήτως αν συνεργάζεται με την πλατφόρμα κατασκευής των διαγραμμάτων η όχι.

Παρόμοια ορίσαμε ένα μεταμοντέλο κώδικα το οποίο απεικονίζει την δομή αλλά και τις λειτουργίες ενός συστήματος ή μέρους αυτού. Κάθε διάγραμμα το οποίο αποτελεί στιγμιότυπο του συγκεκριμένου μεταμοντέλου απεικονίζει τις κλάσεις του συστήματος, τα πεδία και τις μεθόδους κάθε κλάσης, καθώς και την δομή του σώματος των μεθόδων με εντολές υπό μορφή κλήσης μεθόδων ή δημιουργίας αντικειμένων. Επιπρόσθετα απεικονίζονται και οι δομές ελέγχου παρέχοντας για αυτές πληροφορίες που αφορούν το είδος της δομής, την συνθήκη ελέγχου της δομής, τη θέση και τις κλήσεις μεθόδων που περιέχουν. Επίσης δίνονται πληροφορίες σχετικά με την κληρονομικότητα των κλάσεων. Με αυτόν τον τρόπο κατασκευάζεται ένας σκελετός του κώδικα του συστήματος ο οποίος σκιαγραφεί τη δομή και τις βασικές λειτουργίες του συστήματος. Όπως και με τα ακολουθιακά διαγράμματα, τόσο τα διαγράμματα κώδικα όσο και το μεταμοντέλο των διαγραμμάτων αποθηκεύονται σε σειριοποιημένη μορφή ενώ παρέχεται και μια διεπαφή προγραμματισμού εφαρμογών και έχουμε έτσι τη δυνατότητα να επεξεργαστούμε τα διαγράμματα αυτά σε οποιαδήποτε πλατφόρμα.

Για να μετασχηματίσουμε τα ακολουθιακά διαγράμματα σε διαγράμματα κώδικα και να συνδέσουμε την επικοινωνία των κλάσεων του συστήματος, με την δομή του συστήματος αλλά και με τον κώδικα του, χρησιμοποιήσαμε ένα σύνολο κανόνων μετασχηματισμού. Οι κανόνες αυτοί βασίζονται στα μοντέλα που προαναφέραμε και οι αλγόριθμοι τους περιγράφονται σε ψευδοκώδικα χρησιμοποιώντας απλές δομές οι οποίες υπάρχουν σε κάθε γλώσσα προγραμματισμού ή μετασχηματισμού μοντέλων. Στην περίπτωση της χρησιμοποίησης της Java για την υλοποίηση των αλγορίθμων χρησιμοποιείται η διαπροσωπεία προγραμματισμού εφαρμογών κάθε μεταμοντέλου η οποία κατασκευάζεται αυτόματα μαζί με το μεταμοντέλο. Η διαπροσωπεία επιτρέπει στον σχεδιαστή να επεξεργαστεί τα μοντέλα χωρίς να εμπλακεί με την εσωτερική δομή των αρχείων των διαγραμμάτων και τις λεπτομέρειες που έχουν σχέση με τα πρότυπα XML και XMI.

Τα μοντέλα και τους κανόνες μετασχηματισμού τους εφαρμόσαμε σε μια πραγματική εφαρμογή ανοιχτού κώδικα η οποία σχεδιάστηκε με τις αρχές της Μοντελοκεντρικής Αρχιτεκτονικής και του Αντικειμενοστραφούς Προγραμματισμού. Επεξεργαστήκαμε τα ακολουθιακά διαγράμματα της εφαρμογής και τα μετασχηματίσαμε σε ένα ενιαίο διάγραμμα κώδικα το οποίο απεικονίζει τη γενικότερη αρχιτεκτονική της εφαρμογής,τις μεθόδους κάθε κλάσης αλλά και έναν σκελετό του σώματος των μεθόδων, όπου αυτό είναι δυνατό, σκιαγραφώντας με αυτόν τον τρόπο την λειτουργία των μεθόδων.

Τέλος ορίσαμε ένα σύνολο κριτηρίων τα οποία προσδιορίζουν τη συνέπεια του διαγράμματος κώδικα με τον πραγματικό κώδικα της εφαρμογής με στόχο να εντοπίσουμε τις αστοχίες του μετασχηματισμού αλλά και τις ασυνέπειες των ακολουθιακών διαγραμμάτων με τον κώδικα του συστήματος. Τα κριτήρια αυτά εξετάζουν βαθμιαία τη συνέπεια του διαγράμματος κώδικα με τον κώδικα της εφαρμογής έτσι ώστε να προσδιοριστεί με όσο το δυνατό μεγαλύτερη ακρίβεια η ασυνέπεια μεταξύ του κώδικα και του διαγράμματος κώδικα και κατ' επέκταση των ακολουθιακών διαγραμμάτων. Εφαρμόσαμε τα κριτήρια αυτά και συγκρίναμε το διάγραμμα κώδικα που παρήχθη από την εφαρμογή των κανόνων μετασχηματισμού με τον πραγματικό κώδικα του συστήματος. Με την χρήση των κριτηρίων αυτών εντοπίσαμε τις ασυνέπειες των ακολουθιακών διαγραμμάτων ως προς τον κώδικα αλλά και τις αδυναμίες που παρουσιάζουν οι αλγόριθμοι μετασχηματισμού των ακολουθιακών διαγραμμάτων σε κώδικα.

5.2 Μελλοντική έρευνα

Σε αυτήν την ενότητα παρουσιάζουμε μελλοντικές επεκτάσεις της εργασίας αυτής και γενικότερα μελλοντικά ερευνητικά θέματα όπως αυτά προέκυψαν από την επεξεργασία και την συγγραφή της εργασίας.

5.2.1 Μελλοντικές επεκτάσεις

Στη συνέχεια θα παρουσιάσουμε κάποιες από τις επεκτάσεις και βελτιώσεις που επιδέχεται η εργασία αυτή. Μια βασική επέκταση η οποία θα βελτιώσει

τη λειτουργικότητα της εφαρμογής που αναπτύχθηκε σε αυτήν την εργασία, είναι η δημιουργία ενός ολοκληρωμένου γραφικού περιβάλλοντος το οποίο θα δίνει στον χρήστη τη δυνατότητα να σχεδιάζει απευθείας τα ακολουθιακά διαγράμματα με ένα σχεδιαστικό εργαλείο και να γίνεται αυτόματα η δημιουργία του xmi αρχείου που θα περιγράφει το κάθε ακολουθιακό διάγραμμα σύμφωνα με το μοντέλο που περιγράψαμε στο 3ο κεφάλαιο. Επιπρόσθετα θα μπορούσε να δημιουργηθεί και μια γραφική απεικόνιση του διαγράμματος κώδικα μέσω της οποίας ο χρήστης θα μπορεί να έχει εποπτική εικόνα του συστήματος και επίσης να έχει τη δυνατότητα να παρεμβαίνει άμεσα στο διάγραμμα κώδικα συμπληρώνοντας ή διορθώνοντας τον κώδικα του συστήματος. Αναγκαία είναι επίσης η δημιουργία μεταγλωττιστών του διαγράμματος κώδικα σε αντικειμενοστραφείς γλώσσες προγραμματισμού (Java, C++, Ada κλπ) έτσι ώστε να αυτοματοποιηθεί πλήρως η παραγωγή του κώδικα.

Σημαντική είναι και η αντίστροφη διαδικασία, η αυτόματη παραγωγή διαγραμμάτων κώδικα από τον πραγματικό κώδικα, έτσι ώστε να είναι δυνατή η αυτοματοποιημένη εφαρμογή των κριτηρίων σύγκρισης μέσω των οποίων επισημαίνονται οι ασυνέπειες των ακολουθιακών διαγραμμάτων και του πραγματικού κώδικα του συστήματος. Με την αυτοματοποιημένη επισήμανση των ασυνεπειών των ακολουθιακών διαγραμμάτων με τον κώδικα γίνεται πιο εύκολη, η υλοποίηση ενός επόμενου στόχου που είναι η αυτόματη διόρθωση του κώδικα στα σημεία ασυνέπειας. Με την επίτευξη αυτού του στόχου πλησιάζουμε στην υλοποίηση του απώτερου σκοπού μας που είναι ο συγχρονισμός ακολουθιακών διαγραμμάτων και κώδικα.

Χρήσιμη επίσης είναι η επέκταση του μοντέλου του ακολουθιακού διαγράμματος ώστε να περιλαμβάνει και τα υπόλοιπα στοιχεία των ακολουθιακών διαγραμμάτων όπως αυτά προδιαγράφονται από τον OMG στο [8]. Ταυτόχρονα πρέπει να επεκταθεί και ο μετασχηματισμός των ακολουθιακών διαγραμμάτων σε διαγράμματα κώδικα με την προσθήκη αλγορίθμων επεξεργασίας των νέων στοιχείων του μοντέλου του ακολουθιακού διαγράμματος. Μια τέτοια σημαντική προσθήκη είναι η επεξεργασία των σύνθετων τμημάτων par με τα οποία μοντελοποιούνται διεργασίες που εξελίσσονται μέσω πολυνηματισμού.

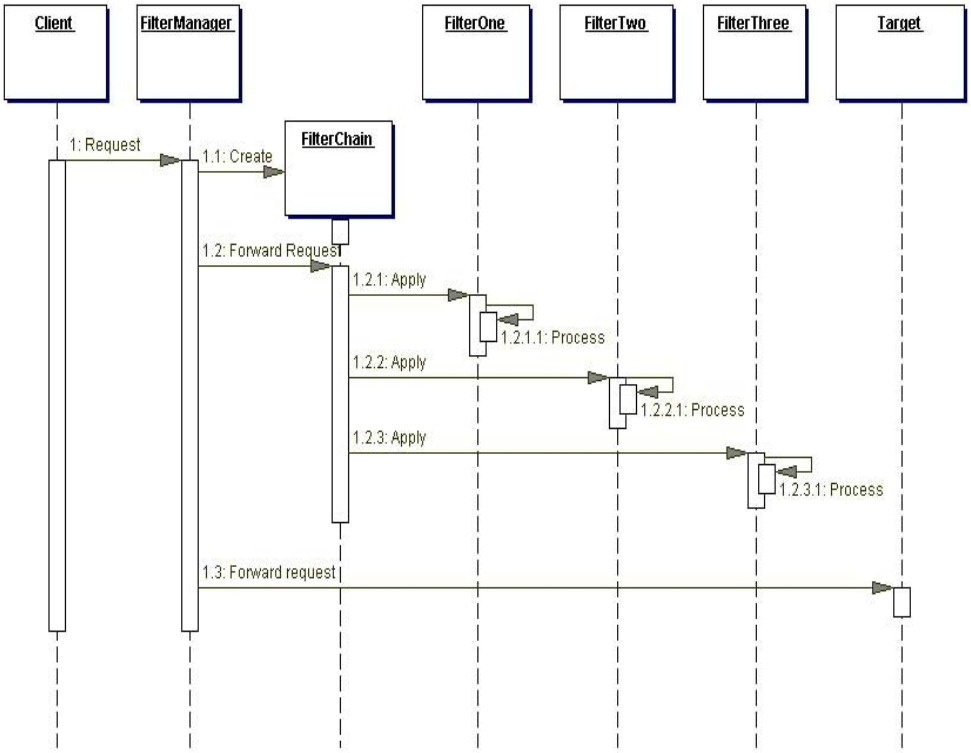
Συνοπτικά η εφαρμογή που αναπτύχθηκε στην εργασία αυτή μπορεί να επεκταθεί με τέτοιον τρόπο ώστε να αποτελέσει ένα πλήρες εργαλείο επεξεργασίας ακολουθιακών διαγραμμάτων και σύγκρισης τους με τον

κώδικα του συστήματος. Με τη χρήση του εργαλείου θα γίνεται αυτόματα η εξέταση της συνέπειας των ακολουθιακών διαγραμμάτων με τον πραγματικό κώδικα του συστήματος και θα εντοπίζονται τα σημεία ασυνέπειας. Επίσης, το εργαλείο αυτό μπορεί να ενσωματωθεί σε μια μεγαλύτερη εφαρμογή συγχρονισμού UML διαγραμμάτων και κώδικα ενός συστήματος.

5.2.2 Μελλοντικές κατευθύνσεις

Καθ' όλη την διάρκεια της διπλωματικής εργασίας ήρθαμε αντιμέτωποι με ερωτήματα και διλήμματα τα οποία χρήζουν περαιτέρω έρευνας ώστε να αντιμετωπιστούν αποτελεσματικά.

Ένα από τα πιο σημαντικά προβλήματα είναι η αναγνώριση μοτίβων επικοινωνίας μεταξύ των κλάσεων και γενικότερα των μονάδων του συστήματος. Μέσω της αναγνώρισης τέτοιων μοτίβων μπορούν να εξαχθούν συμπεράσματα όχι μόνο για τη δομή του κώδικα αλλά και για τον ρόλο που επιτελούν τα διάφορα λειτουργικά μέρη του συστήματος. Ο προσδιορισμός των ρόλων του συστήματος μπορεί να συμβάλλει στην συντήρηση ενός λογισμικού συστήματος, βοηθώντας τον υπεύθυνο συντήρησης να κατανοήσει σε βάθος την δομή του συστήματος υπό συντήρηση και τον τρόπο που το σύστημα εκτελεί τις διάφορες λειτουργίες. Παράδειγμα ενός τέτοιου μοτίβου είναι το μοτίβο του σχήματος 5.1 που απεικονίζει την σχεδίαση ενός φίλτρου μεταξύ ενός πελάτη που κάνει χρήση μιας υπηρεσίας και του εξυπηρετητή που υλοποιεί την παρεχόμενη υπηρεσία. Η αναγνώριση μοτίβων σαν αυτό αλλά και πολλών άλλων όπως αυτά που περιγράφονται στο [29] μπορεί να βοηθήσει τον συντηρητή ενός συστήματος να αποκτήσει μια πληρέστερη εικόνα για το σύστημα που συντηρεί.



Σχήμα 5.1: Μοτίβο υλοποίησης φίλτρου.

Η αναγνώριση μοτίβων σε ακολουθιακά διαγράμματα σε συνδυασμό με την παραγωγή κώδικα από ακολουθιακά διαγράμματα μπορεί να εφαρμοστεί για την αναπαραγωγή της δομής ενός συστήματος μέσω ακολουθιακών διαγραμμάτων που εξάγονται από την δυναμική εκτέλεση ενός συστήματος. Όπως αναφέρεται και στα [30] και [31] υπάρχει η δυνατότητα παραγωγής ακολουθιακών διαγραμμάτων από την εκτέλεση ενός συστήματος με τη χρήση εργαλείων όπως το TPTP [32],[33],[34],[35]. Με την επεξεργασία αυτών των διαγραμμάτων και επιπρόσθετα την αναγνώριση μοτίβων σε αυτά τα ακολουθιακά διαγράμματα μπορεί να παραχθεί ένας σκελετός της δομής και του κώδικα του συστήματος ο οποίος να χρησιμοποιηθεί είτε για τον έλεγχο της συνέπειας των UML διαγραμμάτων που προδιαγράφουν την δομή και τις λειτουργίες του συστήματος, είτε για την παραγωγή ενός αρχικού μοντέλου του κώδικα του συστήματος σε περίπτωση που αυτός δεν είναι διαθέσιμος.

Βιβλιογραφία

[1] Object Management Group, Inc, *MDA Guide, version 1.0.1*

[2] Object Management Group, Inc., *UML Infrastructure Specification, version 2.1.2*, November 2007

[3] L.C. Briand, Y. Labiche and Y. Miao. *Towards the Reverse Engineering of UML Sequence Diagrams*. in *Proceedings of the IEEE Working Conference in Reverse Enginnering*. pp.150-169,2004

[4] P. Selonen, K. Koskimies, and M. Sakkinen, *How to make apples from oranges in UML*, hicss, vol. 03, p. 3054, 2001.

[5] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, Timothy J. Grose, *Eclipse Modeling Framework: A Developer's Guide*. Publisher:Addison-Wesley Professional. ISBN: 0-13-142542-0

[6] Pilone D., Pitman N., *UML 2 in a Nutshell*, publisher: O'Reilly, June 2005 ISBN:0-596-00795-7.

[7] Donald Bell *UML's Sequence Diagram* http://www.ibm.com/developerworks/rational/library/3101.html

[8] Object Management Group, Inc., *UML Superstructure Specification, version 2.1.2*, November 2007.

[9] Object Management Group, Inc.,*Meta Object Facility (MOF) Specification version 1.4.1*, May 2005

[10] Object Management Group, Inc., *Meta Object Facility (MOF) Core Specification version 2.0,* January 2006.

[11] Atlas Group, LINA & INRIA, *Atlas Transformation Language, ATL User Manual, version 0.7*, February2006

[12] IBM Corporation, *IBM Model Transformation Framework User's Guide, version 1.0.2*, 2004

[13] Engels G., Hücking G., Sauer S. , Wagner A.: *UML Collaboration Diagrams and Their Transformation to Java*. In France R, Rumbe B. (eds) : *Proc. UML '99- Beyond the Standard*, Fort Collins,CO, USA, LNCS 1723, Springer, 1999 pp 473-488

[14] Mathupayas Thongmak , Pornsiri Muenchaisri, *Design of Rules for Transforming UML Sequence Diagrams into Java code*, in *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, p.485, December 04-06, 2002 .

[15] R. Van Der Straeten , *Inconsistency Detection between UML Models Using RACER and nRQL* in *Proceedings of the KI-04 Workshop on Applications of Description Logics*, Ulm, September 20-21, 2004. http://ceur-ws.org.

[16] Litvak B., Tyszberowicz S., Yehudai A. *Behavioral Consistency Validation of UML diagrams, in Proceedings of the First International Conference on Software Engineering and Formal Methods (SEFM'03),* ISBN:0-7695-1949-0/03 $17.00 © 2003 IEEE

[17] Matula M. *,NetBeans Metadata Repository.*, March 2003, URL:http://mdr.netbeans.org/MDR-whitepaper.pdf

[18] *What is AndroMDA*?, URL:http://galaxy.andromda.org/docs/whatisit.html

[19] Bjork R., *ATM Simulation*, URL:www.math-cs.gordon.edu/local/courses/cs211/ATMExample

[20] Pons C., Giandini R., Baum G., *Dependency Relations between models in the unified Process* in *Proceedings of the Tenth International Workshop on Software Specification and Design (IWSSD'00)* 0-7695-0884-7/00 $10.00 © 2000 IEEE

[21] Jan Olaf Blech, Sabine Glesner, Johannes Leitner, *Formal Verification of Java Code Generation from UML Models*, Fujaba Days, september 2005.

[22] Sangal, N.; Farrell, E.; Lieberherr, K.; Lorenz, D., *Interaction schemata: compiling interactions to code* ,in *Technology of Object-Oriented Languages and Systems*, 1999. TOOLS 30. Proceedings , vol., no., pp.268-277, Aug 1999 URL:http://ieeexplore.ieee.org/stamp/stamp.jsp?

arnumber=787555&isnumber=17059

[23] Jackson, D.; Waingold, A., *Lightweight extraction of object models from bytecode*, Software Engineering, IEEE Transactions on , vol.27, no.2, pp.156-169, Feb 2001
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=908960&isnumber=19623

[24] Xu F., Wood A., , *Reverse engineering UML class and sequence diagrams from Java code with IBM Rational Software Architect* IBM Corporation June 2008,
URL:
http://www.ibm.com/developerworks/rational/library/08/0610_xu-wood/index.html

[25] J. Niere, J. P. Wadsack, and A. Z¨undorf. R*ecovering UML diagrams from Java code using patterns*. i*n Engineering*, pages 89–97. Springer-Verlag, Feb. 2001

[26] Ivkovic, I.; Kontogiannis, K., T*racing evolution changes of software artifacts through model synchronization,* Software Maintenance, 2004. in *Proceedings. 20th IEEE International Conference,* Sept. 2004
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1357809&isnumber=29793

[27] Hößler J., Soden M.,Eichler H.,*Coevolution of Models, Metamodels and Transformations*, URL:http://www.ikv.co.jp/

[28] Demathieu S., Griffin C,. Sendal S., *Model Transformation with the IBM Model Transformation Framework,*
URL:http://www.ibm.com/developerworks/rational/library/05/503_sebas/index.html

[29] Alur D., John Crupi J. and Dan Malks D. *Core J2EE Patterns: Best Practices and Design Strategies,* Publisher: Prentice Hall / Sun Microsystems Press ISBN:0130648841; 1st edition (June 26, 2001)

[30] K.Taniguchi, T. Ishio, T. Kamiya, S. Kusumo and K. Inoue. *Extracting Sequence Diagram from Execution Trace of Java Program,* in *Proceedings of the 8th International Workshop on Principles of Software Evolution*(IWPS'05).1550-4077/05 IEEE.

[31] P. Tonella and A. Potrich. *Reverse Engineering of the Interaction Diagrams from C++ Code*.in *Proceedings of the International Conference on Software Maintenance*(ICSM'03). 1063-6773/03 IEEE.

[32] TPTP Platform Project. http://www.eclipse.org/tptp/platform/documents/index.php

[33] TPTP Monitoring Project http://www.eclipse.org/tptp/performance/documents/index.php

[34] TPTP Testing Project http://www.eclipse.org/tptp/test/index.php

[35] TPTP Trace and Profiling Project http://www.eclipse.org/tptp/performance/index.php

[36] A. Tsiolakis. *Consistency Analysis of UML Class and Sequence Diagrams based on Attributed Typed Graphs and their Transformation.* Technical Report 2000/3, Technical University of Berlin, March 2000. ISSN 1436-9915.

[37] Rasch H. and Wehrheim H., *Checking Consistency in UML Diagrams: Classes and State Machines* ,in E. Najm, U. Nestmann, and P. Stevens (Eds.): *FMOODS* 2003, LNCS 2884, pp. 229–243, 2003.

[38] Yeung W. L., *Checking Consistency between UML Class and State Models Based on CSP and B* in Journal of Universal Computer Science, vol. 10, no. 11 (2004), 1540-1558 submitted: 10/5/04, accepted: 20/8/04, appeared: 28/11/04

[39] Derrick J and Akehurst H. D., *A framework for UML Consistency in Blekinge,* Institute of Technology, Research Report 2002:06. *UML 2002, Model Engineering, Concepts and Tools*. Workshop on Consistency Problems in UML-based Software Development. Workshop Materials, publisher Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002

[40] W. Liu, S. M. Easterbrook, and J. Mylopoulos, R*ule-based detection of inconsistency in uml models*. In *Wkshp on Consistency Problems in UML-Based Software Development,* 5th Int. Conf. on the Unified Modeling Language, 2002.