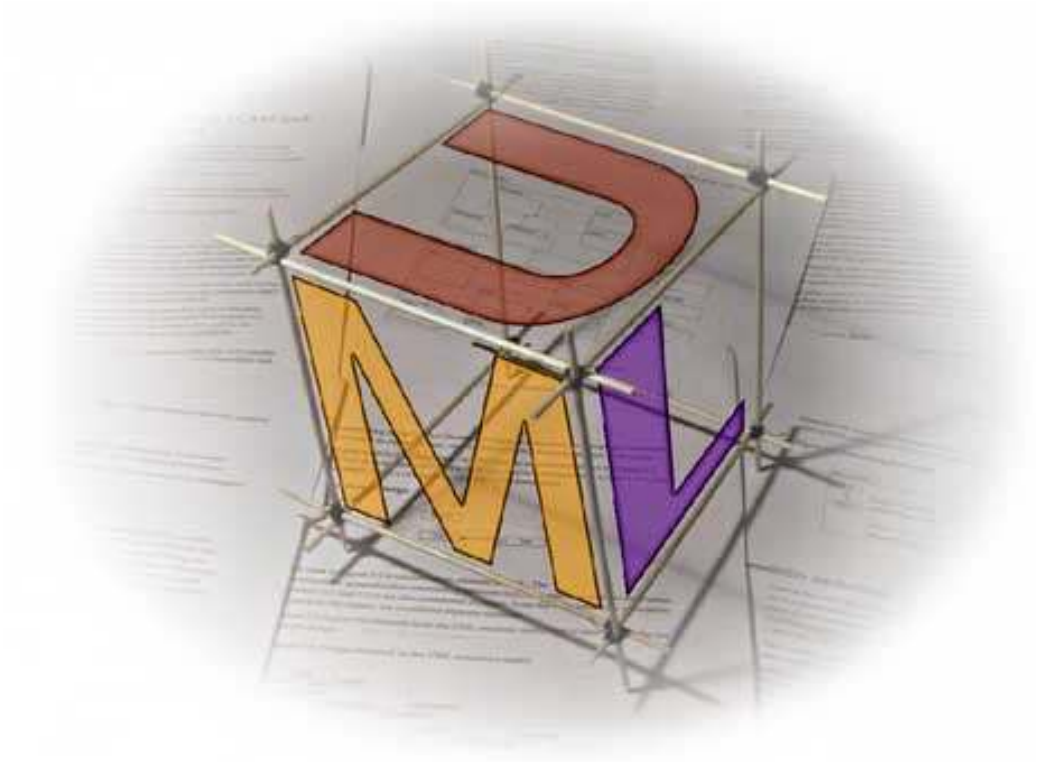
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή Εργασία

«ΣΧΕΔΙΑΣΗ ΣΥΣΤΗΜΑΤΟΣ ΔΙΑΧΕΙΡΙΣΗΣ ΤΩΝ ΕΣΩΤΕΡΙΚΩΝ ΔΙΑΔΙΚΑΣΙΩΝ ΕΝΟΣ ΣΩΜΑΤΕΙΟΥ ΜΕ ΧΡΗΣΗ ΤΗΣ ΕΝΟΠΟΙΗΜΕΝΗΣ ΓΛΩΣΣΑΣ ΜΟΝΤΕΛΟΠΟΙΗΣΗΣ UML»



Του φοιτητή
Λάιου Αναστάσιου Χαράλαμπου
Αρ. Μητρώου: 03/2358

Επιβλέπων καθηγητής
Κωνσταντινίδου Κυριακή

Θεσσαλονίκη 2010

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 5: ΔΙΑΓΡΑΜΜΑΤΑ ΚΑΤΑΣΤΑΣΕΩΝ

ΚΕΦΑΛΑΙΟ 6: ΔΙΑΓΡΑΜΜΑΤΑ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ

ΚΕΦΑΛΑΙΟ 7: ΦΥΣΙΚΑ ΔΙΑΓΡΑΜΜΑΤΑ

ΚΕΦΑΛΑΙΟ 8: ΜΕΘΟΔΟΛΟΓΙΑ CASE

ΠΡΟΛΟΓΟΣ

Η Μηχανική Λογισμικού είναι ένας σχετικά καινούριος κλάδος στην επιστήμη των ηλεκτρονικών υπολογιστών που καλείται να αντιμετωπίσει πολύπλοκα προβλήματα που αντιμετωπίζουν εκατομμύρια επιχειρήσεις σήμερα. Το λογισμικό γενικά, όσο σύνθετο και να είναι, προορίζεται για να συμβάλει στην αυτοματοποίηση επίπονων και επιρρεπών σε σφάλματα ανθρώπινων εργασιών. Όταν λέμε λογισμικό δεν εννοούμε μόνο τον εκτελέσιμο κώδικα, αλλά όλο το σύνολο των ενδιάμεσων προϊόντων όπως τις προδιαγραφές, τα σχέδια, τον πηγαίο κώδικα κ.ά. Όλα αυτά αποτελούν παράγωγα προϊόντα του κύκλου ζωής του λογισμικού ο οποίος περιλαμβάνει όλες τις φάσεις από τη σύλληψη της ιδέας μέχρι και την απόσυρση μιας εφαρμογής λογισμικού από την χρήση. Παρά τη σημαντική πρόοδο που έχει επιτευχθεί στον τομέα του υλικού των υπολογιστών, η κατασκευή του λογισμικού παρουσιάζει ορισμένα χρόνια σημαντικά προβλήματα που σχετίζονται με την ποιότητα, το κόστος και την γενική επάρκεια του τρόπου με τον οποίο αυτό γίνεται. Η Τεχνολογία Λογισμικού είναι η περιοχή εκείνη της επιστήμης της μηχανικής που ασχολείται με την εύρεση και θεμελίωση μεθόδων για την περιγραφή, την κατασκευή και την συντήρηση Λογισμικού καλής ποιότητας, με τη μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος. Η Τεχνολογία Λογισμικού δεν είναι μια θεωρητική επιστήμη, αλλά στοχεύει στην υποστήριξη των κατασκευαστών να παράγουν αποδοτικά προϊόντα λογισμικού.

Έτσι με τα χρόνια συντάχθηκε η UML η οποία κατάφερε με οπτικές αναπαραστάσεις να επιτρέψει τη μοντελοποίηση συστημάτων με βάση τις αρχές των αντικειμενοστραφών μοντέλων. Μπόρεσε να απλοποιήσει την πολυπλοκότητα στην όλη διαδικασία σχεδίασης λογισμικού. Η χρήση μοντέλων γίνεται επειδή έτσι απλουστεύεται η πραγματικότητα. Με αυτά μπορούμε να αναπαραστήσουμε τα σημαντικά στοιχεία της οντότητας που μοντελοποιείται, απλοποιώντας ή παραλείποντας τα υπόλοιπα, τις άσχετες πλευρές και τις δευτερεύουσας σημασίας λεπτομέρειες. Το μέσο επάνω στο οποίο αναπτύσσεται το μοντέλο πρέπει να είναι εύχρηστο και βολικό για επεξεργασία. Έτσι και η UML βασισμένη στα μοντέλα μας βοηθάει να κατανοήσουμε καλύτερα τα συστήματα. Τα μοντέλα μας προσφέρουν ακρίβεια, συνέπεια, εύκολη επικοινωνία, είναι ευμετάβλητα και κατανοητά. Ότι θα μπορούσε δηλαδή κάποιος να ζητήσει για να σχεδιάσει ένα σύστημα και να τον βοηθήσει στην υλοποίηση ενός προγράμματος για να ικανοποιήσει τις ανάγκες των πελατών.

Στα επόμενα κεφάλαια θα εξετάσουμε εις βάθος την UML θα αναπτύξουμε τους στόχους της αλλά και θα δούμε πώς, με την βοήθεια των διαγραμμάτων μοντελοποιούνται οι διαδικασίες ενός υποθετικού σωματείου. Τέλος θα αναφερθούμε στην μεθοδολογία CASE, τις περιπτώσεις που αυτή εφαρμόζεται αλλά και στους κινδύνους που μπορεί να προκύψουν κατά την χρήση της και τις διαδικασίες ελέγχου που θα χρησιμοποιηθούν για να αντιμετωπιστούν αυτοί οι κίνδυνοι.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ ΣΤΗ UML

1.1 Εισαγωγή

Η UML διαδικασία είναι μια διαδικασία που έχει σαν σκοπό την ανάπτυξη συστημάτων λογισμικού χρησιμοποιώντας αντικείμενα. Αρχικά θα εξετάσουμε την UML διαδικασία σε βαθμό που να μπορέσει να την κατανοήσει κάποιος που δεν είχε προηγούμενη επαφή με αυτήν ή κάποια παρόμοια διαδικασία.

1.2 Πεδίο

Η πτυχιακή προορίζεται να είναι ένας συνοπτικός οδηγός στις διαδικασίες και στα διαγράμματα της UML. Με την εστίαση στις βασικές έννοιες πάντα.

1.3 Τι είναι η UML;

Η ενοποιημένη γλώσσα σχεδίασης ή αλλιώς UML είναι μια τυποποιημένη γλώσσα για τον καθορισμό, την απεικόνιση, την κατασκευή και την τεκμηρίωση των αντικειμένων των συστημάτων λογισμικού, καθώς επίσης και για την μοντελοποίηση επιχειρήσεων αλλά και άλλων συστημάτων που δεν ασχολούνται με λογισμικό υπολογιστών. Η UML περιλαμβάνει μια συλλογή από τις καλύτερες πρακτικές εφαρμοσμένης μηχανικής, οι οποίες έχουν αποδειχθεί επιτυχείς στην μοντελοποίηση μεγάλων και σύνθετων συστημάτων. Η UML είναι ένα πολύ σημαντικό κομμάτι στην ανάπτυξη λογισμικού προσανατολισμένο στα αντικείμενα καθώς και της ίδιας διαδικασίας ανάπτυξης αυτού του λογισμικού. Η UML χρησιμοποιεί κυρίως γραφικές αναπαραστάσεις για να διατυπώσει τα «μοντέλα» των προγραμμάτων λογισμικού. Η χρησιμοποίηση της UML βοηθά τις ομάδες των project να επικοινωνούν μεταξύ τους, βοηθάει στην εύρεση πιθανών μοντέλων και τέλος επικυρώνει το αρχιτεκτονικό μοντέλο του λογισμικού. Συνοψίζοντας θα μπορούσαμε να πούμε:

Η Ενοποιημένη Γλώσσα Μοντελοποίησης είναι γραφική γλώσσα γενικού σκοπού η οποία χρησιμοποιείται για:

1. Τον *προσδιορισμό* (specifying), με ακρίβεια και πληρότητα
2. Την *οπτικοποίηση* (visualizing), με γραφικά σύμβολα
3. Την *ανάπτυξη/κατασκευή* (constructing),με μοντέλα που μετασχηματίζονται σε κώδικα με εργαλεία CASE και
4. Την *τεκμηρίωση* (documenting), για όλα τα στάδια κύκλου ζωής λογισμικού των κατασκευασμάτων (artifacts) των συστημάτων λογισμικού

1.4 Στόχοι της UML

Οι αρχικοί στόχοι για την δημιουργία της UML ήταν:

1. Να εφοδιαστούν οι χρήστες με μια γλώσσα μοντελοποίησης, που θα βασιζόταν σε γραφικές αναπαραστάσεις και διαγράμματα, ώστε να μπορούν να αναπτύξουν και να ανταλλάξουν μοντέλα συστημάτων με σημασία.
2. Να παρέχει μηχανισμούς γενίκευσης και ειδίκευσης ώστε να μπορεί να γίνεται επέκταση του κύριου σχεδίου και της αρχικής ιδέας.
3. Να είναι ανεξάρτητη από τις γλώσσες προγραμματισμού αλλά και τις διαδικασίες ανάπτυξης τους.
4. Να παρέχει μια σταθερή βάση για τη γλώσσα σχεδίασης.
5. Να ενθαρρύνει την ανάπτυξη εργαλείων που είναι προσανατολισμένα στα αντικείμενα.
6. Να υποστηρίξει τις υψηλότερου επιπέδου έννοιες ανάπτυξης (Collaborations, frameworks, patterns, components).
7. Να ενσωματώσει τις καλύτερες πρακτικές που ήταν μέχρι τότε γνωστές.

1.5 Γιατί να χρησιμοποιούμε την UML;

Δεδομένου ότι η στρατηγική αξία του λογισμικού αυξάνεται για πολλές επιχειρήσεις, η βιομηχανία ψάχνει για τεχνικές οι οποίες θα αυτοματοποιήσουν την παραγωγή λογισμικού και θα βελτιώσουν την ποιότητα, θα μειώσουν όμως παράλληλα το κόστος αλλά και τον συνολικό χρόνο που χρειάζεται για να είναι έτοιμο προς χρήση. Αυτές οι τεχνικές παρέχουν τεχνολογία βασισμένη στα συστατικά (components), τον γραφικό προγραμματισμό καθώς και στα σχέδια και πλαίσια (Patterns-Frameworks). Οι επιχειρήσεις επιδιώκουν επίσης τεχνικές για να μπορέσουν να ρυθμίσουν την πολυπλοκότητα των συστημάτων τους καθώς αυξάνονται σε μέγεθος. Ειδικότερα, αναγνωρίζουν την ανάγκη να λυθούν τα επαναλαμβανόμενα αρχιτεκτονικά προβλήματα, όπως για παράδειγμα, η ασφάλεια, η εξισορρόπηση φόρτου εργασίας και η ανοχή ελαττωμάτων. Επιπλέον, η ανάπτυξη του Παγκόσμιου Ιστού (WWW), μπορεί να κατέστησε μερικά πράγματα απλούστερα, αλλά από την άλλη έχει επιδεινώσει αυτά τα αρχιτεκτονικά προβλήματα. Η ενοποιημένη γλώσσα σχεδίασης (UML) είχε ως σκοπό να ανταποκριθεί σε αυτές τις ανάγκες.

1.6 Ιστορία της UML

Οι αντικειμενοστραφείς γλώσσες σχεδιασμού άρχισαν να εμφανίζονται μεταξύ των μέσων του 1970 και της δεκαετίας του '80, όπου διάφοροι επιστήμονες που ασχολούνταν γενικά με τις μεθοδολογίες πειραματίστηκαν με τις διαφορετικές προσεγγίσεις στην αντικειμενοστραφή ανάλυση και το σχέδιο. Ο αριθμός των αναγνωρισμένων γλωσσών σχεδίασης από λιγότερο από 10, που ήταν, αυξήθηκε σε περισσότερο από 50, κατά τη διάρκεια της περιόδου

μεταξύ 1989-1994. Πολλοί χρήστες των αντικειμενοστραφών μεθόδων είχαν πρόβλημα στο να καλυφθούν από μια μόνο γλώσσα σχεδίασης, ξεκινώντας έτσι τον «πόλεμο των μεθόδων». Μέχρι τα μέσα της δεκαετίας του '90, νέες εκδοχές των υπαρχόντων μεθόδων άρχισαν να εμφανίζονται, οι οποίες ενσωμάτωναν τεχνικές διαφόρων μεθοδολογιών, με αποτέλεσμα να προκύψουν και μερικές σαφώς προεξέχουσες μέθοδοι.

Η ανάπτυξη της UML άρχισε στα τέλη του 1994 όταν οι Grady Booch, Jim Rumbaugh και Ivar Jacobson της Rational Software Corporation, μιας μεγάλης εταιρίας λογισμικού, άρχισαν την εργασία τους για την ενοποίηση των μεθόδων του Booch με τις μεθόδους OMT (object management technique - τεχνική διαχείρισης αντικειμένου). Το φθινόπωρο του 1995, ο Ivar Jacobson μαζί με την επιχείρησή του έγιναν μέλη σε αυτήν την συγχώνευση των 2 μεθόδων που αναφέραμε παραπάνω με αποτέλεσμα να προκύψει η μέθοδος OOSE (object oriented software engineering - αντικειμενοστραφής τεχνολογία λογισμικού).

Οι αρχικοί συντάκτες των μεθόδων Booch, OMT και OOSE, Grady Booch, Jim Rumbaugh, και ο Ivar Jacobson ήταν πεπεισμένοι ότι πρέπει να δημιουργήσουν μια ενοποιημένη γλώσσα σχεδίασης για τρεις λόγους. Κατ' αρχάς, αυτές οι μέθοδοι εξελίσσονταν ήδη η μια από την άλλη ανεξάρτητα. Έτσι αποφάσισαν να συνεχίσουν την εξέλιξή τους μαζί παρά χώρια, και έτσι θα μίκρυναν την ανώφελες διαφορές που υπήρχαν μεταξύ των μεθόδων και θα συνέχεαν περαιτέρω τους απλούς χρήστες. Δεύτερον, με την ενοποίηση της σημασιολογίας και της σημειογραφίας, θα μπορούσαν να φέρουν κάποια σταθερότητα στην αντικειμενοστραφή αγορά, επιτρέποντας στα project των εταιριών να καταλήγουν σε μία ώριμη γλώσσα σχεδίασης και ταυτόχρονα να επιτρέψει τους προγραμματιστές εργαλείων να εστιάσουν στην παράδοση πιο χρήσιμων χαρακτηριστικών του έργου. Τρίτον, ήλπιζαν ότι η συνεργασία τους θα απέφερε βελτιώσεις και στις τρεις προηγούμενες μεθόδους, βοηθώντας τους έτσι να μαθαίνουν από τα λάθη τους και να αντιμετωπίζουν προβλήματα που καμία από τις προηγούμενες μεθόδους τους πριν δεν κατάφερε να αντιμετωπίσει μόνη της.

Οι προσπάθειες του Booch, Rumbaugh και Jacobson οδήγησαν στην κυκλοφορία των UML 0,9 και 0,91 συγγραμμάτων τον Ιούνιο και τον Οκτώβριο του 1996 αντίστοιχα, Κατά τη διάρκεια του 1996, οι συντάκτες της UML ζήτησαν και έλαβαν σχόλια αλλά και επιπλέον πληροφορίες (feedback) από τη κοινότητα των χρηστών. Λαμβάνοντας υπόψη αυτά τα σχόλια, τα ενσωμάτωσαν και διόρθωσαν τα υπάρχοντα συγγράμματα. Ήταν όμως σαφές ότι χρειαζόταν να δοθεί ακόμα επιπρόσθετη προσοχή.

Καθώς η εταιρία Rational συναρμολογούσε κατά κάποιο τρόπο την UML, καταβάλλονταν προσπάθειες για την επίτευξη του ευρύτερου στόχου, μιας δηλαδή τυποποιημένης γλώσσας σχεδίασης που θα βοηθούσε τις βιομηχανίες. Στις αρχές του 1995, ο Ivar Jacobson (Πρόεδρος του τμήματος τεχνολογίας της Objectory) και ο Richard Soley (Πρόεδρος του τμήματος τεχνολογίας της OMG – Object Management Group) αποφάσισαν να προσπαθήσουν πιο σκληρά για να επιτευχθεί η τυποποίηση αυτή των μεθόδων στην αγορά. Τον Ιούνιο του 1995, σε μια συνεδρίαση της OMG όπου

παρευρίσκονταν όλοι οι γνωστοί αναλυτές (ή οι αντιπρόσωποι τους) οδήγησε στην πρώτη παγκόσμια συμφωνία ώστε να βρεθούν πρότυπα μεθοδολογίας, υπό την αιγίδα της OMG διαδικασίας (OMG process).

Κατά τη διάρκεια του 1996, έγινε ξεκάθαρο ότι η χρήση της UML ήταν μια καλή στρατηγική επιλογή για τις επιχειρήσεις. Ένα αίτημα για εισήγηση που εκδόθηκε από την OMG, έπαιξε καταλυτικό ρόλο σε αυτές τις οργανώσεις, που χρησιμοποιούσαν την UML, για να τις κάνει να ενώσουν τις δυνάμεις ώστε να πάρουν μια κοινή απόφαση και να απαντήσουν στο αίτημα της OMG. Η Rational καθιέρωσε μια κοινοπραξία συνεργατών της UML από διάφορες οργανώσεις πρόθυμες να συνεισφέρουν τους πόρους που χρειάζονταν για να συντάξουν και να καθορίσουν την UML 1.0. Αυτοί που συνέβαλλαν πιο πολύ στον καθορισμό της UML 1.0 ήταν οι ακόλουθοι: Digital Equipment Corp., HP, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI, και Unisys . Αυτή η συνεργασία είχε ως αποτέλεσμα την UML 1.0, μια γλώσσα σχεδίασης που ήταν πάρα πολύ καλά καθορισμένη, πλήρης από εκφράσεις, ισχυρή και κυρίως εφαρμόσιμη. Αυτό υποβλήθηκε στην OMG τον Ιανουάριο του 1997 ως αρχική απάντηση στο αίτημα της.

Τον Ιανουάριο του 1997 οι ακόλουθες εταιρείες, IBM, ObjecTime, Platinum Technology, Ptech, Taskon, Reich Technologies και Softeam υπέβαλαν επιπλέον μεμονωμένες απαντήσεις στο αίτημα της OMG. Αυτές οι επιχειρήσεις έγιναν έπειτα μέλη των συνεργατών της UML. Όλοι μαζί οι συνεργάτες παρήγαγαν την αναθεωρημένη UML 1.1. Εκεί που εστίαζε η νέα UML 1.1 ήταν στο να βελτιωθεί η σαφήνεια της σημασιολογίας της UML 1.0 και να ενσωματωθούν οι νέες ιδέες από τους νέους συνεργάτες. Τέλος, η αναθεωρημένη γλώσσα υποβλήθηκε στην OMG για την εκτίμησή της και υιοθετήθηκε το φθινόπωρο του 1997.

1.7 Τύποι διαγραμμάτων UML

Κάθε διάγραμμα της UML σχεδιάζεται για να επιτρέπει τους σχεδιαστές και τους πελάτες να δουν ένα σύστημα λογισμικού από μια διαφορετική προοπτική. Τα διαγράμματα UML που χρησιμοποιούνται συνήθως στα διάφορα εργαλεία σχεδίασης, περιλαμβάνουν:

Διαγράμματα Περιπτώσεων Χρήσης - Use Case Diagrams

Μας δείχνουν την σχέση μεταξύ των Δραστών-Actors (άτομα δηλαδή που αλληλεπιδρούν με το σύστημα) και των περιπτώσεων χρήσης. Περιγράφονται οι απαιτήσεις των χρηστών (πελατών) από το σύστημα και αναπαρίστανται στα διαγράμματα χωρίς να περιγράφεται ο τρόπος υλοποίησης αυτής της λειτουργικότητας.

Διαγράμματα Κλάσεων - Class Diagrams

Αναπαριστούν τη στατική δομή του συστήματος σε επίπεδο κλάσεων (class structure) καθώς και τα περιεχόμενά τους (contents). Επίσης, αναπαριστούν τις σχέσεις (relationships) μεταξύ των κλάσεων με τη χρήση σχέσεων όπως

είναι η συσχέτιση (association), η σχέση εξειδίκευσης (specialization), η σχέση γενίκευσης (generalization) και άλλες που θα αναλύσουμε στα επόμενα κεφάλαια λεπτομερώς.

Διαγράμματα Αλληλεπίδρασης - Interaction Diagrams

- *Διαγράμματα Ακολουθίας – Sequence Diagrams:* Επιδεικνύει τη χρονική ακολουθία που ακολουθούν τα αντικείμενα που συμμετέχουν στην αλληλεπίδραση. Αποτελείται από την κάθετη διάσταση τον χρόνο και την οριζόντια διάσταση που είναι τα διάφορα αντικείμενα.
- *Διαγράμματα Επικοινωνίας – Communication Diagrams:* Μας παρουσιάζουν τις αλληλεπιδράσεις που σχηματίζονται γύρω από τα αντικείμενα και στις μεταξύ τους συνδέσεις.

Διαγράμματα Καταστάσεων - State Diagrams

Παρουσιάζουν τις ακολουθίες των καταστάσεων που βιώνει ένα αντικείμενο κατά την διάρκεια ζωής του στο σύστημα σε απάντηση στα ερεθίσματα που δέχεται, μαζί με τις αντιδράσεις και τις ενέργειές του.

Διαγράμματα Δραστηριοτήτων - Activity Diagrams

Παρουσιάζουν ένα ειδικό διάγραμμα κατάστασης όπου οι περισσότερες από τις καταστάσεις είναι καταστάσεις δράσης και οι περισσότερες από τις μεταβάσεις προκαλούνται από την ολοκλήρωση ενεργειών στις αρχικές καταστάσεις. Αυτό το διάγραμμα εστιάζει κυρίως στις ροές που οδηγούνται από την εσωτερική επεξεργασία του συστήματος.

Φυσικά Διαγράμματα ή Διαγράμματα Δομής - Physical Diagrams

- *Διαγράμματα Συστατικών – Component Diagrams:* Παρουσιάζουν τα υψηλότερα επίπεδα του πηγαίου κώδικα, την δομή δηλαδή των πακέτων μέσα σε αυτόν. Επίσης, συμπεριλαμβάνονται και οι εξαρτήσεις μεταξύ των συστατικών, συμπεριλαμβανομένων των τμημάτων πηγαίου κώδικα, δυαδικού κώδικα και εκτελέσιμων συστατικών.
- *Διαγράμματα Ανάπτυξης – Deployment Diagrams:* Παρουσιάζουν την διαμόρφωση των στοιχείων κατά τον χρόνο εκτέλεσης και των τμημάτων λογισμικού, των διαδικασιών και των αντικειμένων που ζουν σε αυτά. Τα στιγμιότυπα των τμημάτων λογισμικού αντιπροσωπεύουν τις υλοποιήσεις κατά τον χρόνο εκτέλεσης των μονάδων κώδικα.

Τα αντικείμενα και οι κλάσεις μοντελοποιούνται με την χρήση διαγραμμάτων αντικειμένων και κλάσεων. Οι αλληλεπιδράσεις μεταξύ των αντικειμένων από την άλλη, αναπαρίστανται από τα διαγράμματα επικοινωνίας και ακολουθίας. Τα διαγράμματα επικοινωνίας αναπαριστούν τη φυσική σύνδεση των αντικειμένων σε ένα σύστημα ενώ τα διαγράμματα ακολουθίας τις αλληλεπιδράσεις μεταξύ αυτών των αντικειμένων στο χρόνο.

1.7.1 Ταξινόμηση διαγραμμάτων UML - Στατικά, Δυναμικά και Εφαρμογής

Ένα σύστημα λογισμικού μπορούμε να πούμε ότι έχει δύο ευδιάκριτα χαρακτηριστικά: ένα δομικό "στατικό" μέρος και ένα που παρουσιάζει την συμπεριφορά του, το "δυναμικό" μέρος δηλαδή. Εκτός από αυτά τα δύο χαρακτηριστικά, ένα πρόσθετο χαρακτηριστικό που τα συστήματα λογισμικού κατέχουν συσχετίζεται με την εφαρμογή. Προτού να ταξινομήσουμε τα διαγράμματα UML σε κάθε ένα από αυτά τα τρία χαρακτηριστικά, ας ρίξουμε μια γρήγορη ματιά σε αυτά τα χαρακτηριστικά που είναι:

- *Στατικά - Static:* Το στατικό χαρακτηριστικό ενός συστήματος είναι ουσιαστικά η δομή του συστήματος. Το στατικό μέρος δηλαδή καθορίζει από ποια τμήματα αποτελείται το σύστημα.
- *Δυναμικά - Dynamic:* Είναι τα χαρακτηριστικά που έχουν να κάνουν με την συμπεριφορά που έχει ένα σύστημα. Παραδείγματος χάριν, οι τρόποι που ένα σύστημα συμπεριφέρεται ως απάντηση σε ορισμένα γεγονότα ή ενέργειες είναι τα δυναμικά χαρακτηριστικά ενός συστήματος.
- *Εφαρμογής – Implementation:* Το χαρακτηριστικό που σχετίζεται με την εφαρμογή ενός συστήματος είναι το τρίτο χαρακτηριστικό γνώρισμα που περιγράφει τα διαφορετικά στοιχεία που απαιτούνται για την ανάπτυξη ενός συστήματος.

Τα διαγράμματα UML εμπίπτουν σε κάθε μια από αυτές τις τρεις κατηγορίες ως εξής:

- Στατικά
  - Διαγράμματα Περιπτώσεων Χρήσης
  - Διαγράμματα Κλάσεων
- Δυναμικά
  - Διαγράμματα Αντικειμένων
  - Διαγράμματα Κατάστασης
  - Διαγράμματα Δραστηριοτήτων
  - Διαγράμματα Ακολουθίας
  - Διαγράμματα Επικοινωνίας
- Εφαρμογής – Δομής
  - Διαγράμματα Συστατικών
  - Διαγράμματα Ανάπτυξης

Στο παράρτημα στο τέλος της πτυχιακής μπορούμε να εξετάσουμε καλύτερα ένα σχήμα με το πώς ταξινομούνται τα διαγράμματα της UML.

1.8 Κύκλος Ζωής Λογισμικού UML

Τον κύκλο ζωής του Λογισμικού της UML θα μπορούσαμε να τον χωρίσουμε σε 3 Βήματα:

1^{ο}: ΑΝΑΛΥΣΗ Περιλαμβάνει ένα πρώτο εννοιολογικό μοντέλο. Εδώ επίσης γίνεται και μια αρχική αποτίμηση των απαιτήσεων του συστήματος. Σε αυτό το

πρώτο στάδιο έχουμε τα διαγράμματα κλάσεων στα οποία περιγράφονται οι σχέσεις μεταξύ των κλάσεων που είναι απαραίτητες για την εκτέλεση των διαγραμμάτων περιπτώσεων χρήσης. Επίσης, λαμβάνονται υπόψη μόνο οι κλάσεις που σχετίζονται με την περιοχή προβλήματος που μοντελοποιείται και όχι τεχνικές κλάσεις, όπως για παράδειγμα οι κλάσεις για τα συστήματα διεπαφής (user interfaces).

2^{ο}: ΣΧΕΔΙΑΣΜΟΣ Εδώ ξεκινάει ο σχεδιασμός του συστήματος μας σαν πρώτο μέρος και ύστερα ο λεπτομερειακός σχεδιασμός. Στα διαγράμματα κλάσεων τώρα περιγράφονται οι τεχνικές κλάσεις, όπως είναι οι κλάσεις για τα συστήματα διεπαφής (user interfaces), κλάσεις για τη διαχείριση βάσεων δεδομένων, κλάσεις για την επικοινωνία με άλλα συστήματα κ.α. καθώς και οι σχέσεις μεταξύ τους. Επιπλέον γίνεται ενσωμάτωση των τεχνικών κλάσεων με τις κλάσεις της περιοχής προβλήματος που προέκυψαν από το προηγούμενο στάδιο ανάλυσης του συστήματος. Τέλος τα αποτελέσματα του σχεδιασμού του συστήματος καταλήγουν σε λεπτομερείς προδιαγραφές για την υλοποίηση του συστήματος.

3^{ο}: ΥΛΟΠΟΙΗΣΗ Οι κλάσεις των μοντέλων του συστήματος, που προέκυψαν από το προηγούμενο στάδιο σχεδιασμού, μετατρέπονται σε πραγματικό κώδικα με τη χρήση μιας γλώσσας προγραμματισμού βασισμένη στα αντικείμενα.

1.9 Άξονες Μοντελοποίησης στη UML και Τμήματά της

Τρείς είναι οι βασικοί άξονες που μας ενδιαφέρουν όταν θα αποφασίσουμε να μοντελοποιήσουμε το σύστημά μας με UML:

Η Λειτουργικότητα: Τι κάνει δηλαδή το σύστημά μας καθώς και ποιες λειτουργίες υποστηρίζονται από αυτό. Εδώ περιλαμβάνεται η ανάλυση των απαιτήσεων τίς οποίες εξάγουμε από τις περιγραφές του κειμένου που έχει δώσει ο πελάτης.

Η Δομή: Εδώ υπάγονται τα Στατικά Διαγράμματα και μας ενδιαφέρει το πώς είναι οργανωμένα τα συστατικά τμήματα του συστήματος μας.

Η Συμπεριφορά: Τέλος η συμπεριφορά η οποία ασχολείται με τον τρόπο που αλληλεπιδρά το σύστημα με τους εξωτερικούς παράγοντες αλλά και το πώς ανταποκρίνεται στις διάφορες καταστάσεις που περνάει.

Βάση αυτών των τριών αξόνων η UML έχει καταφέρει να ορίσει ένα κοινό λεξιλόγιο για την αντικειμενοστραφή προσέγγιση και να προσφέρει διαγραμματικές τεχνικές ικανές να μοντελοποιήσουν οποιοδήποτε σύστημα από την ανάλυση έως και την υλοποίησή του.

Η UML αποτυπώνει τόσο τη στατική δομή όσο και την δυναμική συμπεριφορά ενός συστήματος. Ένα αντικειμενοστραφές σύστημα μοντελοποιείται ως μία συλλογή αντικειμένων που αλληλεπιδρούν για την εκτέλεση μιας λειτουργίας η

οποία είναι τελικά αξιοποιήσιμη από το χρήστη του συστήματος. Η στατική δομή του συστήματος καθορίζει τα είδη των αντικειμένων που είναι σημαντικά για το σύστημα καθώς και τις συσχετίσεις μεταξύ τους. Η δυναμική συμπεριφορά τέλος προσδιορίζει την εξέλιξη των αντικειμένων σε σχέση με τον χρόνο αλλά επίσης και την επικοινωνία μεταξύ τους.

1.10 Συνοψίζοντας

Σε αυτό το κεφάλαιο είδαμε τι είναι η UML, ποιοι είναι οι στόχοι της αλλά και για ποιο λόγο θα την προτιμήσουμε όταν έρθει η ώρα να μοντελοποιηθεί το σύστημά μας. Επίσης, ασχοληθήκαμε με τους τύπους διαγραμμάτων της UML και τις υποκατηγορίες τους και τέλος είδαμε από τι αποτελείται ο κύκλος ζωής του λογισμικού.

Στο επόμενο κεφάλαιο, θα ασχοληθούμε με τα διαγράμματα περιπτώσεων χρήσης και θα τα αναλύσουμε μέσα από παραδείγματα από την μοντελοποίηση ενός υποθετικού σωματείου.

ΚΕΦΑΛΑΙΟ 2. ΔΙΑΓΡΑΜΜΑΤΑ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ

2.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο, ρίξαμε μια συνοπτική ματιά στα εννέα διαγράμματα της UML και στην χρήση του καθενός. Στα επόμενα κεφάλαια αυτής της εργασίας θα μελετήσουμε αναλυτικότερα αυτά τα εννέα διαγράμματα. Θα ξεκινήσουμε με τα διαγράμματα περιπτώσεων χρήσης. Θα μάθουμε τα βασικά αυτών των διαγραμμάτων και θα δοκιμάσουμε να σχεδιάσουμε ένα τέτοιο διάγραμμα. Επιπλέον, θα δούμε τι είναι μία προδιαγραφή περίπτωσης χρήσης και τέλος θα προσπαθήσουμε να τα εφαρμόσουμε αυτά στην μοντελοποίηση του υποθετικού σωματείου μας, στο δικό μας δηλαδή πληροφοριακό σύστημα.

2.2 Ορισμός

Πριν ξεκινήσουμε, ας κάνουμε μια επανάληψη στον προσδιορισμό του διαγράμματος περιπτώσεων χρήσης που περιγράψαμε συνοπτικά στο πρώτο κεφάλαιο:

Το διάγραμμα περιπτώσεων χρήσης χρησιμοποιείται για να προσδιορίσει τα αρχικά στοιχεία και διαδικασίες που διαμορφώνουν το σύστημα μας. Τα αρχικά στοιχεία αναφέρονται ως "δράστες" και οι διαδικασίες ως "περιπτώσεις χρήσης". Το διάγραμμα περιπτώσεων χρήσης παρουσιάζει ποιοι δράστες αλληλεπιδρούν με κάθε περίπτωση χρήσης.

Η παραπάνω δήλωση συνοψίζει λίγο πολύ από τί αποτελείται πρώτιστα ένα διάγραμμα περίπτωσης χρήσης δηλαδή από τους δράστες (actors) και τις περιπτώσεις χρήσης (use cases).

Ένα διάγραμμα περίπτωσης χρήσης συλλαμβάνει τις λειτουργικές πτυχές ενός συστήματος. Πιο συγκεκριμένα, συλλαμβάνει τις επιχειρησιακές διαδικασίες (business processes) που πραγματοποιούνται μέσα στο σύστημα. Δεδομένου ότι συζητάμε τη λειτουργία και τις διαδικασίες του συστήματος, ανακαλύπτουμε τα σημαντικά χαρακτηριστικά του συστήματος που διαμορφώνετε στο διάγραμμα περίπτωσης χρήσης. Λόγω της απλότητάς τους και κυρίως επειδή είναι απογυμνωμένα όλης της τεχνικής ορολογίας, τα διαγράμματα περίπτωσης χρήσης είναι ένα μεγάλο εργαλείο που βοηθάει στην επικοινωνία με τους πελάτες του συστήματος. Τα διαγράμματα περίπτωσης χρήσης έχουν μια άλλη σημαντική χρήση. Καθορίζουν τις απαιτήσεις του συστήματος που θα μοντελοποιηθεί και ως εκ τούτου χρησιμοποιούνται για την συγγραφή δοκιμαστικών ελέγχων (scripts δοκιμής) για το μοντελοποιημένο σύστημα, ώστε να μπορέσουμε να το δοκιμάσουμε κάτω από διαφορετικές συνθήκες.

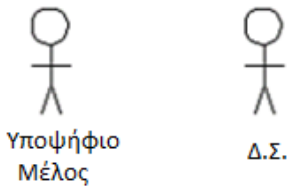
Οι περιπτώσεις χρήσης δημιουργούνται όταν πρέπει να συλληφθούν οι απαιτήσεις ενός συστήματος. Έτσι, στη δημιουργία τους πρέπει υποχρεωτικά να συμμετέχουν οι εμπειρογνώμονες και οι επιχειρησιακοί αναλυτές. Επειδή σε αυτό το σημείο δεν περιλαμβάνεται καμία διαδικασία ανάπτυξης, οι τεχνικοί σχεδιαστές δεν πρέπει να είναι ένα μέρος της διαδικασίας δημιουργίας των

περιπτώσεων χρήσης. Την εμπειρία τους την χρησιμοποιούμε στα επόμενα στάδια του κύκλου ζωής λογισμικού.

2.3 Στοιχεία Διαγράμματος Περίπτωσης Χρήσης

Ένα διάγραμμα περίπτωσης χρήσης είναι συνήθως απλής φύσης και απεικονίζει δύο τύπους στοιχείων: ένα που αντιπροσωπεύει τους επιχειρησιακούς ρόλους (business roles) και το άλλο που αντιπροσωπεύει τις επιχειρησιακές διαδικασίες (business processes). Ας εμβαθύνουμε τώρα λίγο στη χρήση των στοιχείων από τα οποία αποτελείται ένα διάγραμμα περίπτωσης χρήσης:

- *Δράστες - Actors:* Ένας δράστης απεικονίζει οποιαδήποτε οντότητα (ή οντότητες) που εκτελούν ορισμένους ρόλους σε ένα δεδομένο σύστημα. Αυτοί οι διαφορετικοί ρόλοι που αντιπροσωπεύει ο δράστης είναι οι πραγματικοί επιχειρησιακοί ρόλοι των χρηστών σε ένα δεδομένο σύστημα. Ένας δράστης σε ένα διάγραμμα περίπτωσης χρήσης αλληλεπιδρά με μια περίπτωση χρήσης. Παραδείγματος χάριν, στο σύστημά μας η Εκλογή Προεδρείου γίνεται με ψηφοφορία των μελών της Γενικής Συνέλευσης. Εδώ οι οντότητες των μελών της Γενικής Συνέλευσης αντιπροσωπεύουν έναν δράστη του συστήματος. Αλλά εξαρτάται από εμάς να εξεταστεί ο αντίκτυπος που έχουν οι δράστες στη λειτουργία που θέλουμε να διαμορφώσουμε. Εάν μια οντότητα δεν έχει επιπτώσεις σε ένα ορισμένο κομμάτι της λειτουργίας που σχεδιάζουμε, δεν έχει τότε κανένα νόημα να αντιπροσωπευτεί ως δράστης. Έναν δράστη του συστήματος τον απεικονίζουμε ως ένα μικρό ανθρωπάκι σε ένα διάγραμμα περίπτωσης χρήσης και τοποθετείται έξω από τα όρια του συστήματός μας, όπως φαίνεται στο σχήμα 2.1.



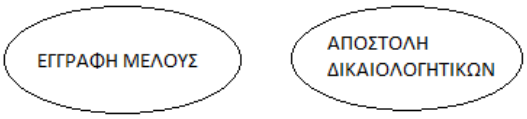
Σχήμα 2.1 Παράδειγμα δύο δραστών σε ένα διάγραμμα περίπτωσης χρήσης

Για να προσδιορίσουμε έναν δράστη, αναζητούμε στο κείμενο του προβλήματός μας τους επιχειρησιακούς όρους που απεικονίζουν ρόλους στο σύστημα. Παραδείγματος χάριν, «Τα υποψήφια μέλη του συλλόγου στέλνουν τις αιτήσεις τους στην Γενική Συνέλευση για έγκριση.» Τα υποψήφια μέλη και η Γενική Συνέλευση είναι οι επιχειρησιακοί ρόλοι που ψάχνουμε και μπορούν να προσδιοριστούν εύκολα ως δράστες στο σύστημά μας.

Υπάρχουν δύο κατηγορίες δραστών: οι Πρωτεύοντες (Primary) και οι Δευτερεύοντες (Secondary) Δράστες. Οι πρώτοι είναι αυτοί οι οποίοι

χρησιμοποιούν το σύστημα είναι δηλαδή οι τελικοί χρήστες (end users). Οι δευτερεύοντες δράστες από την άλλη παρέχουν πληροφορίες ή ενημερώνονται από το σύστημα για διάφορες λειτουργίες αλλά δεν αναλαμβάνουν πρωτοβουλία ενεργειών.

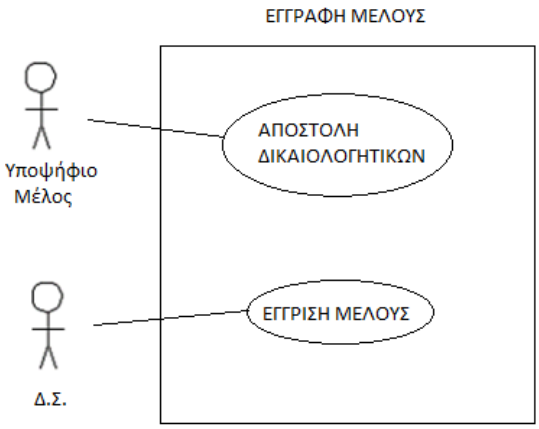
- *Περίπτωση Χρήσης - Use case:* Μια περίπτωση χρήσης σε ένα τέτοιο διάγραμμα είναι μια οπτική αντιπροσώπευση μιας ευδιάκριτης επιχειρησιακής λειτουργίας σε ένα σύστημα. Ο όρος κλειδί εδώ είναι η «ευδιάκριτη επιχειρησιακή λειτουργία». Για να επιλέξουμε μια επιχειρησιακή διαδικασία ως πιθανή υποψήφια για την σχεδίασή της ως περίπτωση χρήσης, πρέπει να εξασφαλίσουμε ότι η επιχειρησιακή διαδικασία είναι ιδιαίτερης φύσης δηλαδή εκτελεί κάποια σημαντική λειτουργία. Σαν πρώτο βήμα στον προσδιορισμό των περιπτώσεων χρήσης, πρέπει να απαριθμήσουμε αυτές τις επιχειρησιακές λειτουργίες μέσα από το κείμενο του προβλήματός μας (στην πραγματικότητα το κείμενο ανάλυσης απαιτήσεων (requirements analysis)). Κάθε μια από αυτές τις επιχειρησιακές λειτουργίες μπορεί να θεωρηθεί ως πιθανή περίπτωση χρήσης. Καθώς μια επιχειρησιακή λειτουργία γίνεται σαφέστερη, οι περιπτώσεις χρήσης που κρύβει γίνονται ευκολότερα εμφανείς. Μια περίπτωση χρήσης παρουσιάζεται ως μία έλλειψη σε ένα διάγραμμα περίπτωσης χρήσης (σχήμα 2.2).



Σχήμα 2.2 Περιπτώσεις χρήσης σε ένα διάγραμμα περίπτωσης χρήσης

Το σχήμα 2.2 παρουσιάζει 2 περιπτώσεις χρήσης: "εγγραφή μέλους" και "αποστολή δικαιολογητικών" σε ένα πληροφορικό σύστημα σαν το δικό μας. Σε αυτό το παράδειγμα βλέπουμε πως μια επιχειρησιακή διαδικασία όπως η εγγραφή νέου μέλους μπορεί στη συνέχεια να περιέχει υπό-διαδικασίες όπως την αποστολή δικαιολογητικών ή ακόμα και την έγκριση από το Δ.Σ.. Η ανακάλυψη τέτοιων υπονοούμενων περιπτώσεων χρήσης είναι δυνατή μόνο με μια λεπτομερή κατανόηση όλων των επιχειρησιακών διαδικασιών του συστήματος μέσω πάντα των συζητήσεων με τους τελικούς χρήστες του συστήματος και της σχετικής μας γνώσης πάνω στις διαδικασίες του συστήματος.

- *Όρια του Συστήματος:* Το σύστημα μπορεί να είναι ένα λογισμικό, μια επιχείρηση ή ένα μηχάνημα. Ένα σύστημα δεν μπορεί να έχει άπειρη λειτουργικότητα. Έτσι και οι περιπτώσεις χρήσης του συστήματος πρέπει να καθορίζουν τα όριά τους. Το όριο συστήματος σε ένα διάγραμμα περίπτωσης χρήσης καθορίζει αυτά τα όρια. Αναπαρίσταται με ένα παραλληλόγραμμο στο οποίο περικλείονται όλες οι περιπτώσεις χρήσης του συστήματος μας.



Σχήμα 2.3 Ένα διάγραμμα περίπτωσης χρήσης που απεικονίζει το όριο συστήματος στη εγγραφή μέλους.

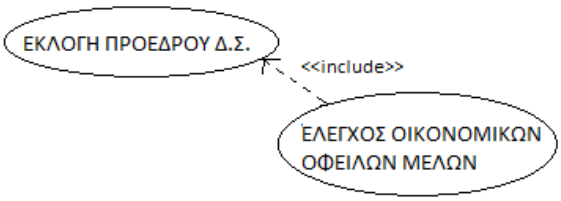
Το σχήμα 2.3 παρουσιάζει το όριο συστήματος στο σωματείο που μοντελοποιούμε για την εγγραφή νέου μέλους. Οι περιπτώσεις χρήσης αυτού του συστήματος εσωκλείονται σε ένα ορθογώνιο. Να σημειώσουμε ότι οι δράστες του συστήματός μας είναι έξω από το όριο συστήματος.

Το όριο του συστήματος είναι ενδεχομένως ολόκληρο το σύστημα όπως καθορίζεται στην ανάλυση απαιτήσεων. Αλλά αυτή δεν είναι η μόνη περίπτωση. Για τα μεγάλα και πιο σύνθετα συστήματα, κάθε μια από τις ενότητές του μπορεί να είναι όριο συστήματος. Παραδείγματος χάριν, σε ένα σύστημα προγραμματισμού επιχειρηματικών πόρων για μια οργάνωση, κάθε ένα από τα τμήματα του συστήματος όπως η διαχείριση προσωπικού, οι μισθολογικές καταστάσεις, οι λειτουργίες του λογιστηρίου και ούτω καθεξής, μπορούν να διαμορφώσουν το όριο του συστήματος για τις περιπτώσεις χρήσης συγκεκριμένα για κάθε μια από αυτές τις επιχειρησιακές λειτουργίες. Ολόκληρο το σύστημα μπορεί να περιέχει όλες αυτά τα τμήματα ξεχωριστά απεικονίζοντας έτσι το γενικό όριο του συστήματος. Η απόφαση σχετικά με τις απαιτήσεις που πρόκειται να υλοποιηθούν στο σύστημα είναι αρμοδιότητα και δικαίωμα των πελατών.

2.4 Τύποι Σχέσεων στις περιπτώσεις χρήσης

Οι περιπτώσεις χρήσης μπορούν να περιέχουν διαφορετικά είδη σχέσεων. Μια σχέση μεταξύ δύο περιπτώσεων χρήσης παρουσιάζει βασικά μια εξάρτηση μεταξύ των δύο περιπτώσεων χρήσης. Ο καθορισμός μιας τέτοιας σχέσης μεταξύ δύο περιπτώσεων χρήσης είναι απόφαση του σχεδιαστή του διαγράμματος περίπτωσης χρήσης. Η επαναχρησιμοποίηση μιας υπάρχουσας περίπτωσης χρήσης, στην οποία γίνεται χρήση των διαφορετικών τύπων σχέσεων, μειώνει τη γενική προσπάθεια που απαιτείται στον καθορισμό των περιπτώσεων χρήσης σε ένα σύστημα. Παρόμοιες σχέσεις θα συναντήσουμε και στα άλλα διαγράμματα UML. Οι σχέσεις που θα συναντήσουμε στις περίπτωσης χρήσης είναι οι ακόλουθες:

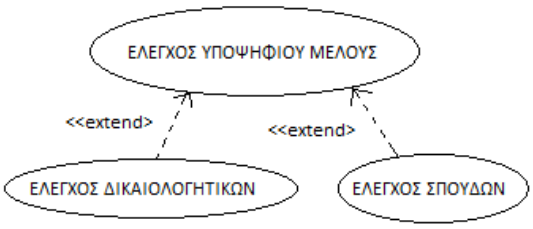
- *<u>Ένταξη - Include:</u>* Όταν μια περίπτωση χρήσης απεικονίζεται χρησιμοποιώντας τη λειτουργία μιας άλλης περίπτωσης χρήσης σε ένα διάγραμμα, αυτή η σχέση μεταξύ των περιπτώσεων χρήσης ονομάζεται σχέση *ένταξης*. Κυριολεκτικά μιλώντας, σε μια σχέση ένταξης, μια περίπτωση χρήσης περιέχει τη λειτουργία που περιγράφεται σε μια άλλη περίπτωση χρήσης ως μέρος της ροής της επιχειρησιακής της διαδικασίας. Η σχέση ένταξης απεικονίζεται με ένα κατευθυνόμενο βέλος που έχει έναν διακεκομμένο άξονα. Η άκρη του βέλους δείχνει την περίπτωση χρήσης γονέα ενώ στην βάση του βέλους τοποθετούμε την περίπτωση χρήσης παιδιού.



Σχήμα 2.4 Ένα παράδειγμα μιας σχέσης ένταξης.

Παραδείγματος χάριν, στο σχήμα 2.4, βλέπουμε ότι η λειτουργία της περίπτωσης χρήσης "έλεγχος οικονομικών οφειλών" περιλαμβάνεται μέσα στην περίπτωση χρήσης "εκλογή προέδρου Δ.Σ.". Ως εκ τούτου, όποτε η περίπτωση χρήσης "εκλογή προέδρου Δ.Σ." εκτελείται, τα επιχειρησιακά βήματα που καθορίζονται στην περίπτωση χρήσης "έλεγχος οικονομικών οφειλών" εκτελούνται επίσης.

- *<u>Επέκταση - Extend:</u>* Στην σχέση Επέκτασης μεταξύ δύο περιπτώσεων χρήσης, η περίπτωση χρήσης «παιδί» προσθέτει στην υπάρχουσα λειτουργία και τα χαρακτηριστικά της περίπτωσης χρήσης «γονέα». Αυτή η σχέση απεικονίζεται με ένα κατευθυνόμενο βέλος με διακεκομμένο άξονα, παρόμοιο με της σχέσης ένταξης. Και σε αυτή την περίπτωση η άκρη του βέλους δείχνει την περίπτωση χρήσης γονέα και η περίπτωση χρήσης παιδί συνδέεται στη βάση αυτού του βέλους.

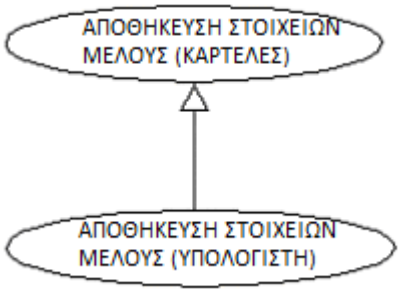


Σχήμα 2.5 Ένα παράδειγμα δύο σχέσεων επέκτασης

Στο σχήμα 2.5 δείχνει ένα παράδειγμα σχέσης επέκτασης μεταξύ των διαδικασιών «έλεγχος υποψήφιου μέλους» (γονέας) και «έλεγχος δικαιολογητικών», «έλεγχος σπουδών» (παιδιά). Οι δύο περιπτώσεις χρήσης «έλεγχος δικαιολογητικών» και «έλεγχος σπουδών» ενισχύουν τη λειτουργία της περίπτωσης χρήσης «έλεγχος δικαιολογητικών».

Ουσιαστικά, οι δύο περιπτώσεις χρήσης «έλεγχος δικαιολογητικών» και «έλεγχος σπουδών» είναι εξειδικευμένες εκδόσεις της γενικής περίπτωσης χρήσης «έλεγχος υποψηφίου». Εδώ έγκειται και η διαφορά με την σχέση ένταξης, ότι σε αυτή την περίπτωση τα παιδιά είναι εξειδικευμένες εκδόσεις της περίπτωσης χρήσης γονέα ενώ στην ένταξη τα παιδιά μπορούν να είναι τελείως διαφορετικά και να καθορίζουν με επιπλέον λειτουργίες την περίπτωση χρήσης γονέα.

- *Γενίκευση - Generalizations:* Μια σχέση *γενίκευσης* είναι επίσης μια σχέση γονέα-παιδιού μεταξύ των περιπτώσεων χρήσης. Η περίπτωση χρήσης παιδί στη σχέση γενίκευσης έχει ελλοχεύουσα σημασία σαν επιχειρησιακή διαδικασία, είναι μια επιπρόσθετη λειτουργία θα μπορούσαμε να πούμε της περίπτωσης χρήσης γονέα. Χρησιμοποιείται όταν υπάρχει μια περίπτωση χρήσης που είναι όμοια με κάποια άλλη, αλλά κάνει κάτι περισσότερο δηλαδή προσθέτει μια εναλλακτική συμπεριφορά στη βασική περίπτωση χρήσης. Σε ένα διάγραμμα περίπτωσης χρήσης, η γενίκευση παρουσιάζεται ως ένα κατευθυνόμενο βέλος με την άκρη του βέλους να είναι τρίγωνο (σχήμα 2.6). Εδώ η περίπτωση χρήσης παιδί συνδέεται στη βάση του βέλους ενώ στην άκρη του συνδέεται η περίπτωση χρήσης γονέα.



Σχήμα 2.6: Ένα παράδειγμα μιας σχέσης γενίκευσης.

Με μία πρώτη ματιά η γενίκευση και η επέκταση θα μπορούσε να πει κανείς ότι είναι πάνω κάτω παρόμοιες. Αλλά υπάρχει μια λεπτή διαφορά μεταξύ μιας σχέσης γενίκευσης και επέκτασης. Όταν δημιουργούμε μια σχέση γενίκευσης μεταξύ των περιπτώσεων χρήσης, αυτό υπονοεί ότι η περίπτωση χρήσης «γονέων» μπορεί να αντικατασταθεί από την περίπτωση χρήσης «παιδιών» χωρίς να αλλάξει η επιχειρησιακή ροή. Από την άλλη, χρησιμοποιώντας σχέση επέκτασης μεταξύ των περιπτώσεων χρήσης υπονοεί ότι η περίπτωση χρήσης «παιδιών» ενισχύει και εμπλουτίζει τη λειτουργία της περίπτωσης χρήσης «γονέων» σε μια εξειδικευμένη λειτουργία. Η περίπτωση χρήσης υψηλότερου επιπέδου επεκτείνει τη σχέση, δεν μπορεί να αντικατασταθεί από την περίπτωση χρήσης κατώτερου επιπέδου.

2.4.1 Κανόνες Χρήσης των Σχέσεων

Χρησιμοποιούμε τη σχέση <<include>> όταν επαναλαμβάνεται κάποιο τμήμα σε περισσότερες από μια διαφορετικές περιπτώσεις χρήσης και θέλουμε να αποφύγουμε την επανάληψη αυτή.

Χρησιμοποιούμε τη σχέση γενίκευσης όταν περιγράφουμε μια παραλλαγή της κανονικής συμπεριφοράς μιας λειτουργίας και θέλουμε να την απεικονίσουμε στο διάγραμμα.

Χρησιμοποιούμε τη σχέση <<extend>> όταν περιγράφουμε μια παραλλαγή της κανονικής συμπεριφοράς μιας λειτουργίας δηλώνοντας τα σημεία επέκτασης στη βασική περίπτωση χρήσης.

2.5 Προδιαγραφή Περίπτωσης Χρήσης (Use Case Specification)

Ένα διάγραμμα περίπτωσης χρήσης, όπως έχουμε δει, είναι μια οπτική απεικόνιση των διαφορετικών σεναρίων της αλληλεπίδρασης μεταξύ ενός δράστη και μιας περίπτωσης χρήσης. Η χρησιμότητα των διαγραμμάτων περίπτωσης χρήσης είναι περισσότερο ως ένα εργαλείο επικοινωνίας μεταξύ τις ομάδας που συλλέγει τις απαιτήσεις του συστήματος και της ομάδας χρηστών. Το επόμενο βήμα μετά από την οριστικοποίηση των διαγραμμάτων περίπτωσης χρήσης είναι να τεκμηριωθεί η επιχειρησιακή λειτουργία σε ευδιάκριτες και λεπτομερείς προδιαγραφές περίπτωσης χρήσης. Επειδή τις περιπτώσεις χρήσης τις χρησιμοποιούμε στις άλλες φάσεις του project όπως για παράδειγμα στον σχεδιασμό, στην ανάπτυξη και στην δοκιμή, πρέπει να εξασφαλίσουμε ότι η οπτική απεικόνιση των επιχειρησιακών απαιτήσεων είναι μεταφρασμένη σε σαφείς και καλά καθορισμένες προδιαγραφές. Οι καλά γραμμένες προδιαγραφές περίπτωσης χρήσης χρησιμοποιούνται για την ανάπτυξη των περιπτώσεων δοκιμής (μονάδας, συστήματος και δοκιμές οπισθοδρόμησης, ανάλογα με την περίσταση πάντα).

Οι προδιαγραφές περιπτώσεων χρήσης πρέπει να τεκμηριώνουν την επιχειρησιακή ροή. Οι πληροφορίες της προδιαγραφής περιλαμβάνουν ποιοι δράστες αλληλεπιδρούν, τα βήματα που εκτελεί η περίπτωση χρήσης, τους επιχειρησιακούς κανόνες και ούτω καθεξής. Συνοψίζοντας ένα έγγραφο προδιαγραφών περιπτώσεων χρήσης πρέπει να περιλαμβάνει τα ακόλουθα:

- *Σύντομη περιγραφή:* Περιγράφεται ο στόχος και η συμπεριφορά της περίπτωσης χρήσης που εκτελείται.
- *Δράστες:* Απαρίθμηση των δραστών που αλληλεπιδρούν και συμμετέχουν στην περίπτωση χρήσης.
- *Προϋποθέσεις:* Οι προϋποθέσεις που πρέπει να ικανοποιηθούν ώστε η περίπτωση χρήσης να αποδοθεί σωστά.
- *Μετά-Συνθήκες:* Καθορισμός των διαφορετικών φάσεων στις οποίες αναμένουμε να βρεθεί το σύστημα αφού εκτελεστεί η ομαλή λειτουργία που περιγράφουμε στην περίπτωση χρήσης.
- *Βασική ροή:* Απαρίθμηση των βασικών γεγονότων που θα συμβούν κατά την εκτέλεση της περίπτωσης χρήσης. Εδώ θα συμπεριληφθούν όλες οι πρωτογενές δραστηριότητες που θα εκτελέσει η περίπτωση χρήσης μας. Πρέπει να είμαστε αρκετά περιγραφικοί όταν καθορίζουμε τις ενέργειες που εκτελεί ο δράστης καθώς και στην απάντηση της περίπτωσης χρήσης σε αυτές τις ενέργειες. Αυτή η περιγραφή των ενεργειών και των ανταποκρίσεων θα είναι οι λειτουργικές μας απαιτήσεις. Αυτές θα αποτελέσουν τη βάση για τα σενάρια περιπτώσεων δοκιμής στο σύστημά μας.

- *Εναλλακτικές ροές:* Τα δευτερεύοντα γεγονότα που μπορούν να συμβούν στην περίπτωση χρήσης πρέπει να απαριθμηθούν χωριστά. Κάθε τέτοιο γεγονός πρέπει να ολοκληρώνεται ώστε να απαριθμείται ως εναλλακτική ροή. Μια περίπτωση χρήσης μπορεί να έχει τόσες εναλλακτικές ροές όσες απαιτούνται. Αλλά πρέπει να προσέχουμε γιατί εάν υπάρχουν πάρα πολλές εναλλακτικές ροές πρέπει να κάνουμε ανασκόπηση στο σχέδιο της περίπτωσης χρήσης για να το καταστήσουμε απλούστερο και αν είναι απαραίτητο να σπάσουμε την περίπτωση χρήσης σε μικρότερες, πιο διακριτές, μονάδες.
- *Ειδικές απαιτήσεις:* Οι βασικές και εναλλακτικές ροές πρέπει να απαριθμηθούν ως ειδικές απαιτήσεις της περίπτωσης χρήσης. Οι ειδικές απαιτήσεις θα χρησιμοποιηθούν ακόμα για το γράψιμο των περιπτώσεων δοκιμής. Σε αυτό το κομμάτι θα πρέπει επίσης να περιγραφούν τα σενάρια επιτυχίας και αποτυχίας.
- *Σχέσεις στις περίπτωσης χρήσης:* Στα σύνθετα συστήματα, συνιστάται να αναφέρουμε τις σχέσεις μεταξύ των περιπτώσεων χρήσης. Εάν μία περίπτωση χρήσης επεκτείνεται σε άλλες περιπτώσεις χρήσης ή περιλαμβάνει τη λειτουργία άλλων περιπτώσεων χρήσης, πρέπει να αναφερθεί εδώ.

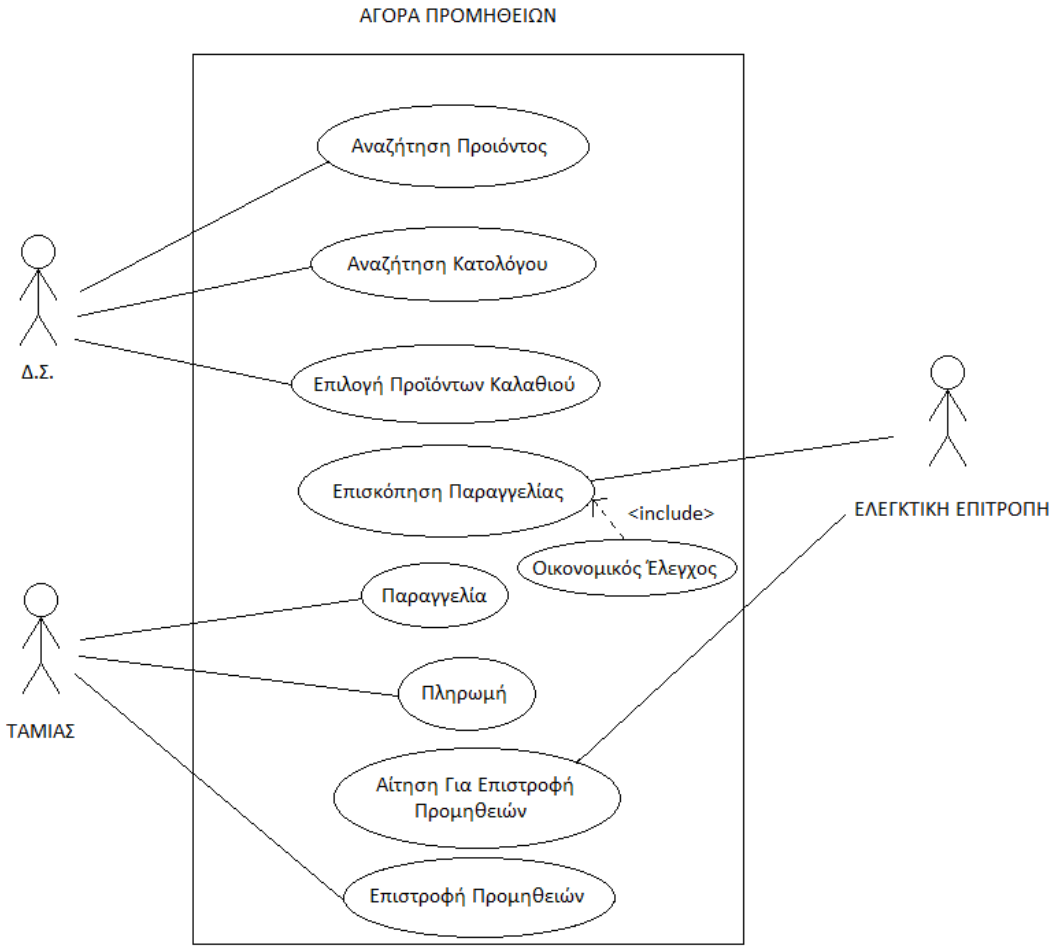
2.6 Τι πρέπει να έχουμε κατά νου

Οι περιπτώσεις χρήσης δεν πρέπει να χρησιμοποιηθούν για να συμπεριλαμβάνουν όλες τις λεπτομέρειες ενός συστήματος. Ο βαθμός λεπτομέρειας των περιπτώσεων χρήσης σε ένα διάγραμμα πρέπει να είναι τέτοιος, ώστε να κρατήσει το διάγραμμα τακτοποιημένο και ευανάγνωστο, παράλληλα όμως, να είναι πλήρες, χωρίς την απώλεια σημαντικών πτυχών της απαραίτητης λειτουργίας. Αυτό το πρόβλημα με το βαθμό λεπτομέρειας θα τον συναντήσουμε και στα υπόλοιπα διαγράμματα της UML.

Ένας σημαντικός κανόνας που συχνά ξεχνάμε να ακολουθήσουμε κατά τη διάρκεια της δημιουργίας μιας περίπτωσης χρήσης είναι ότι πρέπει να εστιάσουμε στην περιγραφή του συστήματός μας. Οι περιπτώσεις χρήσης προορίζονται να συλλάβουν το "τι" είναι το σύστημα, όχι το "πώς" το σύστημα θα σχεδιαστεί ή θα χτιστεί. Οι περιπτώσεις χρήσης δεν θα πρέπει να περιέχουν καθόλου χαρακτηριστικά σχεδίασης. Εάν καταλήξουμε να καθορίζουμε χαρακτηριστικά σχεδίασης του συστήματος σε μια περίπτωση χρήσης, πρέπει να ξεκινήσουμε πάλι από την αρχή.

2.7 Παράδειγμα Περίπτωσης Χρήσης

Το σωματείο που μοντελοποιούμε έχει ανάγκες για προμήθειες οι οποίες αποφασίζονται κάθε μήνα από το Διοικητικό Συμβούλιο. Ας δούμε το διάγραμμα περίπτωσης χρήσης και ύστερα να την περιγράψουμε με μια σύντομη περιγραφή.



Σχήμα 2.7 Περίπτωση Χρήσης Αγοράς Προμηθειών

Κάθε μήνα συγκαλείται το Διοικητικό Συμβούλιο όπου γίνεται καταγραφή των προϊόντων για τα οποία υπάρχει έλλειψη. Το επόμενο βήμα είναι να γίνει αναζήτηση στον κατάλογο των προμηθευτών για τα προϊόντα αυτά και αφού βρεθούν οι προμηθευτές, να γίνει η επιλογή των πιο οικονομικών και ποιοτικών υλικών. Ύστερα, η ελεγκτική επιτροπή θα εγκρίνει την παραγγελία αφού πρώτα γίνει η επισκόπησή της με τον έλεγχο των οικονομικών που διαθέτει το σωματείο και της ποσοτικής δαπάνης που θα χρειαστεί για τις προμήθειες. Μετά την έγκριση, ο ταμίας θα κάνει την παραγγελία, θα δώσει χρήματα από το ταμείο για την πληρωμή των προμηθευτών και θα παραλάβει τις προμήθειες. Θα ελέγξει βάση του τιμολογίου αν λείπει κάτι από την παραγγελία ή αν κάτι είναι ελαττωματικό και θα ενημερώσει την ελεγκτική επιτροπή. Η ελεγκτική επιτροπή με την σειρά της, αν βρεθεί κάτι ελαττωματικό, θα ζητήσει από τον προμηθευτή την επιστροφή των προμηθειών. Τέλος, ο ταμίας θα επιστρέψει αν χρειαστεί τις προμήθειες.

2.8 Συνοψίζοντας

Τα διαγράμματα περιπτώσεων χρήσης ήταν η αφετηρία του ταξιδιού μας στην εξερεύνηση κάθε ενός από τα διαγράμματα της UML. Η επιχειρησιακή λειτουργία μπορεί να παρασταθεί γρήγορα με τη χρησιμοποίηση των διαγραμμάτων περίπτωσης χρήσης. Μόλις ολοκληρωθεί η προκαταρκτική

εργασία με την απεικόνιση των περιπτώσεων χρήσης, το επόμενο βήμα, όπως είδαμε, είναι να γραφούν τα λεπτομερή σενάρια των περιπτώσεων χρήσης που θα χρησιμοποιηθούν ως οι βασικές λειτουργικές απαιτήσεις για το σύστημά μας.

Στο επόμενο κεφάλαιο θα μελετήσουμε τα διαγράμματα Κλάσεων και πως αυτά χρησιμοποιούνται στην μοντελοποίηση του συστήματός μας.

ΚΕΦΑΛΑΙΟ 3. ΔΙΑΓΡΑΜΜΑΤΑ ΚΛΑΣΕΩΝ

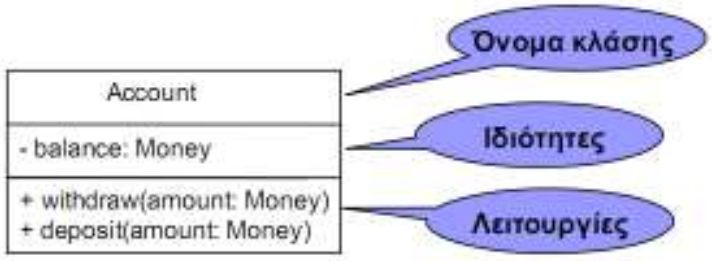
3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα μιλήσουμε για τα Διαγράμματα Κλάσεων. Τα διαγράμματα κλάσεων είναι η κύρια δομική μονάδα στον αντικειμενοστραφή σχεδιασμό και μοντελοποιούν την στατική δομή του συστήματος. Το πρώτο ερώτημα που μας έρχεται στο μυαλό είναι γιατί να χρειάζεται να μοντελοποιήσουμε την δομή; Τρείς είναι οι κύριοι λόγοι που χρειάζεται να το κάνουμε αυτό. Πρώτον για να μειώσουμε το σημασιολογικό χάσμα μεταξύ του πραγματικού κόσμου και του κόσμου του λογισμικού. Δεύτερον για να ορίσουμε ένα κοινό λεξιλόγιο μεταξύ των αναλυτών και των χρηστών και τέλος για να παραστήσουμε αντικείμενα, ιδέες και έννοιες που είναι σημαντικές για το πεδίο εφαρμογής.

3.2 Ορισμός

Όπως είπαμε στο προηγούμενο κεφάλαιο τα διαγράμματα περιπτώσεων χρήσης είναι διαγράμματα καταγραφής προδιαγραφών. Τα διαγράμματα κλάσεων όμως, καταγράφουν τις κλάσεις που ανήκουν σε ένα σύστημα. Τους τύπους δηλαδή των αντικειμένων και τις συσχετίσεις που υπάρχουν μεταξύ τους. Τα αντικείμενα αποτελούν στιγμιότυπα των κλάσεων και δημιουργούνται κατά τη διάρκεια εκτέλεσης του προγράμματος. Δηλαδή μια κλάση περιγράφει ένα σύνολο αντικειμένων με: Όμοιες Ιδιότητες, Κοινή Συμπεριφορά και Κοινές Συσχετίσεις με άλλα αντικείμενα.

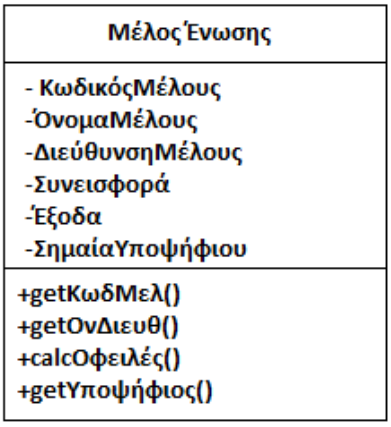
Ένα σύστημα τυπικά έχει πολλά διαγράμματα κλάσεων και μία κλάση μπορεί να συμμετέχει ταυτόχρονα σε πολλά διαγράμματα κλάσεων. Στο διάγραμμα κλάσεων οι κλάσεις αντιπροσωπεύονται με μικρά ορθογώνια παραλληλόγραμμα που αποτελούνται από τρία διαμερίσματα:



Σχήμα 3.1 Παράδειγμα μίας κλάσης με τα 3 της μέρη.

Όπως βλέπουμε και στο παραπάνω παράδειγμα μια κλάση αποτελείται από:

- Το ανώτερο μέρος που περιέχει το όνομα της κλάσης.
- Το ενδιάμεσο μέρος που περιέχει τις ιδιότητες της κλάσης και
- Το κατώτατο μέρος που μας πληροφορεί για τις μεθόδους ή τις διαδικασίες που η κλάση μπορεί να εμπεριέχει ή να αναλάβει.



Ας πάρουμε ένα άλλο παράδειγμα μιας κλάσης το «Μέλος Ένωσης» που βλέπουμε στο διπλανό σχήμα. Αυτή η κλάση αναπαριστά την οντότητα του Μέλους στο σύστημα που μοντελοποιούμε. Ενσωματώνει πληροφορίες για το «Μέλος» όπως το μοναδικό κωδικό του μέλους, το όνομά του αλλά και επιπλέον ιδιότητες όπως η ΣημαίαΥποψηφίου η οποία καθορίζει αν το μέλος είναι Υποψήφιο ή αν έχει επιλεγεί βάση των προϋποθέσεων οπότε είναι ενεργό μέλος. Η κλάση «Μέλος Ένωσης» έχει επίσης κάποια «Λειτουργικότητα», η οποία αναπαρίσταται με λειτουργίες, όπως είναι οι getΚωδΜελ(), getΟνΔιεύθ() ή και οι calcΟφειλές() και getΥποψήφιος() οι οποίες υπολογίζουν αν το μέλος έχει οφειλές και μας πληροφορούν αν το μέλος είναι υποψήφιο ή όχι αντίστοιχα. Αυτό που πρέπει να έχουμε κατά νου είναι ότι συνήθως ορίζουμε τις ιδιότητες ως ιδιωτικές (private) και τις λειτουργίες ως δημόσιες (public). Με τον όρο ιδιωτικές εννοούμε τα χαρακτηριστικά μιας κλάσης Α που δεν είναι προσπελάσιμα απευθείας από αντικείμενα άλλων κλάσεων, παρά μόνο μέσω της χρήσης δημόσιων μεθόδων της κλάσης Α. Από την άλλη όταν αναφερόμαστε σε δημόσιες λειτουργίες εννοούμε λειτουργίες που είναι άμεσα προσπελάσιμες από άλλες κλάσεις. Έτσι, οι μέθοδοι αποτελούν ουσιαστικά την διεπαφή ανάμεσα στην κλάση και το υπόλοιπο σύστημα.

Για να διαχωρίσουμε τα παραπάνω έχουμε τους προσδιοριστές πρόσβασης που μπαίνουν μπροστά από τις ιδιότητες και τις λειτουργίες-μεθόδους και μας ενημερώνουν για τον τύπο πρόσβασης που αυτές έχουν.

1. « - » Ιδιωτική Πρόσβαση
2. « + » Δημόσια Πρόσβαση
3. « # » Προστατευμένη Πρόσβαση, όπου η ιδιότητα ή λειτουργία είναι προσπελάσιμη από την κλάση και τις τυχόν υποκλάσεις της.
4. « ~ » Πρόσβαση σε επίπεδο Πακέτου, όπου η ιδιότητα ή λειτουργία είναι προσπελάσιμη από την κλάση στην οποία δηλώνεται και τις άλλες κλάσεις που βρίσκονται στο ίδιο πακέτο με αυτήν.

Στο εννοιολογικό σχέδιο ενός συστήματος μπορεί να προσδιοριστεί και να συγκεντρωθεί μεγάλος αριθμός κλάσεων σε ένα διάγραμμα κλάσεων και έτσι να γίνει και ο καθορισμός των στατικών σχέσεων μεταξύ αυτών των αντικειμένων. Με τον λεπτομερή σχεδιασμό, οι κλάσεις του εννοιολογικού σχεδίου συχνά χωρίζονται σε έναν αριθμό από υποκλάσεις.

Για να περιγράψουμε λίγο ακόμα την συμπεριφορά των συστημάτων, θα μπορούσαμε να πούμε ότι τα διαγράμματα κλάσεων μπορούν να συμπληρωθούν από τα διαγράμματα καταστάσεων τα οποία θα παρουσιάσουμε σε παρακάτω κεφάλαιο.

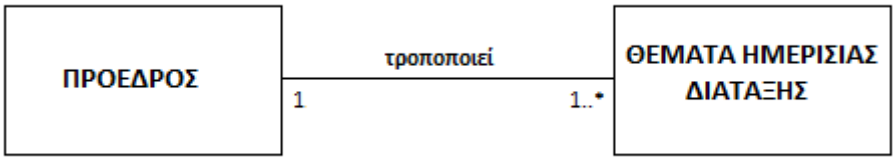
3.3 Σχέσεις μεταξύ Κλάσεων

Μια σχέση (ή συσχέτιση) είναι ένας γενικός όρος, που αφορά τους τύπους των λογικών συνδέσεων που συναντάμε στα διαγράμματα κλάσεων και αντικειμένων. Η UML μας παρέχει τα εξής στοιχεία τα οποία μπορούμε να έχουμε προαιρετικά σε μια συσχέτιση:

- Όνομα Συσχέτισης
- Όνομα Άκρων Συσχέτισης
- Πολλαπλότητα (Multiplicity)
- Πλοηγησιμότητα (Navigability)

Όνομα Συσχέτισης

Είναι το γνώρισμα που υποδηλώνει το νόημα της συσχέτισης. Το όνομα, όταν αναγράφεται, θα πρέπει να τοποθετείται στο μέσο του συνδέσμου έτσι ώστε να μη συγχέεται με τα ονόματα που πιθανώς θα υπάρχουν στα άκρα των συσχετίσεων.

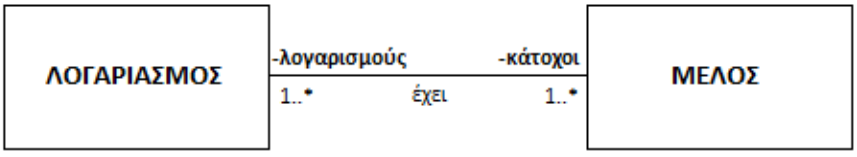


Σχήμα 3.2 Παράδειγμα ονόματος συσχέτισης.

Η λέξη «τροποποιεί» πρέπει να διαβάζεται από την κλάση Πρόεδρος προς την κλάση Θ.Η.Δ. δηλαδή ο Πρόεδρος (ένας) τροποποιεί τα Θέματα Ημερήσιας Διάταξης (πολλά).

Ονόματα Άκρων Συσχέτισης

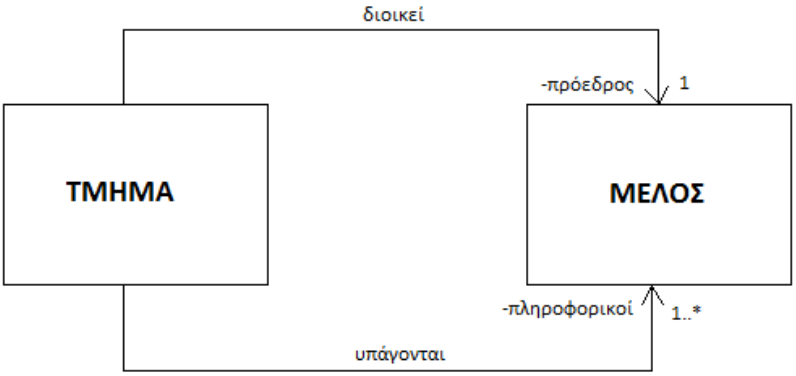
Σε κάθε άκρο συσχέτισης προσθέτουμε προαιρετικά το όνομα που υποδηλώνει το ρόλο της κλάσης στη συσχέτιση.



Σχήμα 3.3 Παράδειγμα ονομάτων άκρων μίας συσχέτισης

Στο άκρο της συσχέτισης που αφορά την κλάση «Μέλος» έχουμε ως όνομα «κάτοχοι» επειδή η συσχέτιση αφορά τα μέλη που είναι δικαιούχοι ενός λογαριασμού. Το όνομα του άκρου συσχέτισης προσδιορίζει το ρόλο μιας κλάσης στη συσχέτιση και κάποιες φορές μπορεί να το συναντήσουμε και ως

όνομα ρόλου (role name). Όταν η ίδια κλάση συσχετίζεται με μια άλλη κλάση με δύο διαφορετικές συσχετίσεις τότε το όνομα άκρου συσχέτισης είναι απαραίτητο και όχι προαιρετικό έτσι ώστε να υποδηλώνεται ποιος είναι ο ρόλος της πρώτης και ποιος της δεύτερης.



Σχήμα 3.4 Παράδειγμα διπλής συσχέτισης.

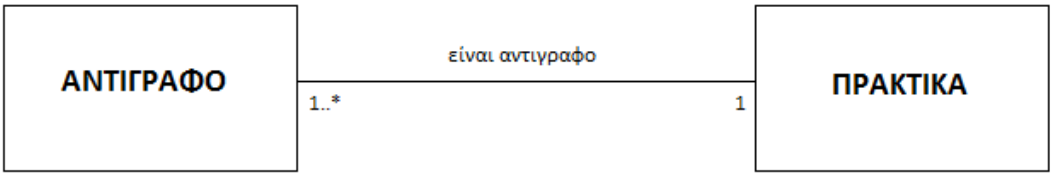
Βλέπουμε και στο σχήμα πως η κλάση Τμήμα συσχετίζεται με την κλάση Μέλος με δύο διαφορετικές συσχετίσεις. Έτσι, έχουμε τα απλά μέλη του σωματείου μας που είναι αντικείμενα της κλάσης Μέλος και υπάγονται σε ένα τμήμα αλλά επιπλέον έχουμε την συσχέτιση που αφορά τον πρόεδρο του τμήματος που είναι επίσης Μέλος και διοικεί το τμήμα.

Πολλαπλότητα

Η πολλαπλότητα αφορά το ένα άκρο μιας συσχέτισης και μας πληροφορεί για το πλήθος των αντικειμένων που μπορούν να μετέχουν σε μια συσχέτιση και εκφράζονται με κάτω και πάνω όρια. Έχουμε τις παρακάτω πολλαπλότητες:

- 1 ή 1..1 : ένα και μόνο ένα ή αλλιώς μονότιμη συσχέτιση.
- * ή 0..* : 0 ή περισσότερα δηλαδή κανένας περιορισμός.
- 1..* : ένα ή περισσότερα, πλειότιμη συσχέτιση.
- 0..1 : μηδέν ή ένα, δηλαδή προαιρετική συσχέτιση.
- 11 : κάποιος συγκεκριμένος αριθμός.
- 2..5 : κάποια συγκεκριμένη περιοχή τιμών.
- 3,6 : μη συνεχόμενες σειρές τιμών.

Όπως βλέπουμε και στο σχήμα 3.3 ένας λογαριασμός μπορεί να έχει έναν ή περισσότερους δικαιούχους (1..* στο άκρο Κάτοχοι) και ένα μέλος μπορεί να έχει έναν ή περισσότερους λογαριασμούς (1..* στο άκρο Λογαριασμούς).



Σχήμα 3.5 Παράδειγμα πολλαπλότητας 1 προς πολλά.

Όπως βλέπουμε στο σχήμα 3.5 κάθε αντικείμενο της κλάσης «Αντίγραφο» συσχετίζεται μέσω της συσχέτισης «είναι αντίγραφο» με ακριβώς «ένα» αντικείμενο της κλάσης «Πρακτικά», που αναπαριστά τα πρακτικά μιας συνέλευσης. Για αυτό βάλαμε 1 στο άκρο της συσχέτισης που αφορά την κλάση «Πρακτικά». Από την άλλη πλευρά, μπορεί να υπάρχουν πολλά αντίγραφα ενός Πρακτικού συνέλευσης στο σύστημά μας. Επομένως η πολλαπλότητα στο άκρο της συσχέτισης που αφορά την κλάση «Αντίγραφο» είναι 1..*.

Πλοηγησιμότητα

Η πλοηγησιμότητα συμβολίζεται με ένα βέλος στο πέρας της συσχέτισης και υποδηλώνει την πλοηγησιμότητα μόνο προς τη φορά του βέλους. Αφορά τη δυνατότητα που έχουμε από μια κλάση να ανακτήσουμε αντικείμενα μιας άλλης κλάσης μέσα από μια συσχέτιση.

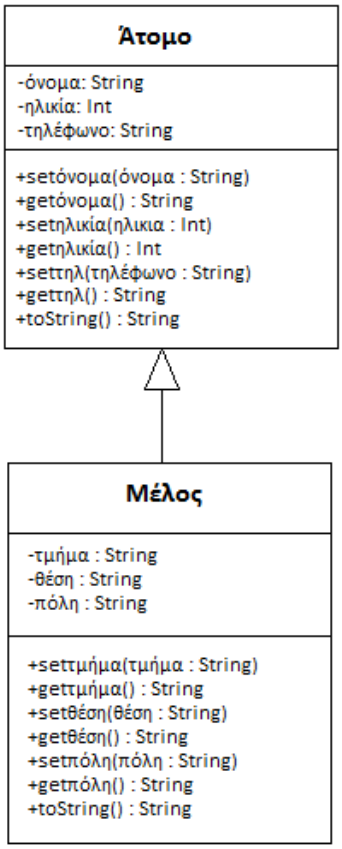
Στο σχήμα 3.3 δεν βλέπουμε κάποιο βέλος και θα υποθέταμε πως δεν υπάρχει πλοηγησιμότητα. Η αλήθεια είναι πως όταν δεν σημειώνουμε κάποιο βέλος σημαίνει ότι υπάρχει πλοηγησιμότητα και προς τις δύο κατευθύνσεις. Έχοντας ένα αντικείμενο «Λογαριασμός» μπορούμε να βρούμε τα μέλη που είναι δικαιούχοι του λογαριασμού και ταυτόχρονα έχοντας ένα αντικείμενο «Μέλος» μπορούμε να βρούμε τους λογαριασμούς του. Η πλοηγησιμότητα και προς τις δύο κατευθύνσεις έχει την έννοια της αμοιβαιότητας δηλαδή αν το αντικείμενο-λογαριασμός Α συσχετίζεται με το αντικείμενο-μέλος Β, τότε θα πρέπει και το μέλος Β να συσχετίζεται με το λογαριασμό Α.

Αντίθετα στο σχήμα 3.4 υπάρχει πλοηγησιμότητα μόνο από την κλάση «Τμήμα» προς την κλάση «Μέλος» που σημαίνει πως έχοντας ένα αντικείμενο Τμήμα μπορούμε να βρούμε ποιοι αποτελούν αυτό το τμήμα ή ποιος είναι ο πρόεδρος του αλλά έχοντας ένα αντικείμενο Μέλος δεν μπορούμε να βρούμε σε ποιο τμήμα ανήκει. Το γεγονός ότι η κλάση «Τμήμα» έχει την υποχρέωση να γνωρίζει τα Μέλη της αλλά ένα Μέλος δεν έχει την αντίστοιχη υποχρέωση να γνωρίζει σε ποιο Τμήμα ανήκει, ίσως μας φανεί λίγο περίεργο. Αυτό δείχνει ότι ο πραγματικός κόσμος και ο κόσμος του λογισμικού δεν είναι αναγκαστικά ταυτόσημοι. Στον πραγματικό κόσμο ένα μέλος γνωρίζει πάντα σε ποιο τμήμα ανήκει, όπως επίσης και ένας διευθυντής ξέρει πάντα ποιο τμήμα διευθύνει. Στον κόσμο του λογισμικού, μπορεί να μην κριθεί απαραίτητο για τις λειτουργίες που θέλουμε να παρέχει το λογισμικό μας η συσχέτιση να είναι δύο κατευθύνσεων.

3.3.1 Συσχέτιση Γενίκευσης

Η γενίκευση αποτελεί μια ειδική μορφή συσχέτισης, όπου μια γενική κλάση αποτελεί τη βάση για τη δήλωση μιας ή περισσότερων ειδικότερων, υπο κάποια έννοια, κλάσεων. Στην περίπτωση της γενίκευσης η γενική κλάση ονομάζεται υπερκλάση και οι ειδικές κλάσεις ονομάζονται υποκλάσεις. Η γενίκευση στις περισσότερες γλώσσες προγραμματισμού υλοποιείται με το μηχανισμό της κληρονομικότητας ή της επέκτασης. Η γενική κλάση ή υπερκλάση όπως είπαμε παρέχει λειτουργίες, ιδιότητες και συσχετίσεις οι οποίες είναι χρήσιμες σε όλες τις υποκλάσεις της. Έτσι οι υποκλάσεις

επεκτείνουν τη λειτουργικότητα της υπερκλάσης και παρέχουν επιπλέον λειτουργίες όπου είναι απαραίτητο ή εξειδικεύουν τη συμπεριφορά τους.



Η κλάση «Άτομο» επεκτείνεται για να δημιουργηθεί η υποκλάση «Μέλος» και η τελευταία κληρονομεί όλες τις ιδιότητες και τις λειτουργίες της από την υπερκλάση της. Στην κλάση «Μέλος» προστίθενται επίσης και κάποιες επιπλέον λειτουργίες για τις αντίστοιχες ιδιότητες και επιτρέπουν την ανάκτηση αλλά και την τροποποίηση του τμήματος στο οποίο δραστηριοποιείται το μέλος, την θέση στην οποία είναι υπεύθυνο αλλά και την πόλη όπου εντάσσεται.

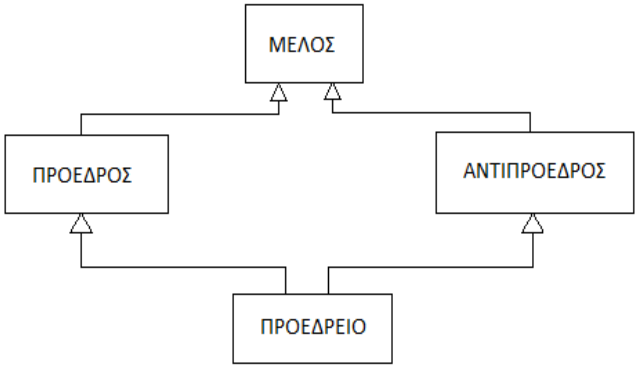
Όπως βλέπουμε και στο διπλανό σχήμα το σύμβολο με το οποίο απεικονίζουμε την γενίκευση είναι ένα βέλος το οποίο έχει ως βάση την ειδική κλάση (απογονική) και καταλήγει στην γενική κλάση (γονική).

Όταν έχουμε γενίκευση, μια λειτουργία της κλάσης «Μέλος» υπερβαίνει την αντίστοιχη λειτουργία της κλάσης «Άτομο» (override) για τα αντικείμενα που είναι μέλη. Όταν δηλαδή καλούμε μια λειτουργία σε αντικείμενα της κλάσης «Άτομο» τότε αν το αντικείμενο είναι όντως «Άτομο» θα κληθεί η λειτουργία της υπερκλάσης ενώ αν είναι «Μέλος» θα κληθεί η λειτουργία της υποκλάσης.

Η συσχέτιση της Γενίκευσης μπορεί να είναι:

I. Υπερκαλυπτόμενη
II. Πλήρης ή
III. Ελλιπής.

Στην υπερκαλυπτόμενη γενίκευση όπως βλέπουμε και στο σχήμα 3.6 η κλάση Προεδρείο μπορεί να κληρονομήσει όλα τα χαρακτηριστικά και τις λειτουργίες από τις δύο υποκλάσεις, Πρόεδρος και Αντιπρόεδρος, οι οποίες κληρονομούν με την σειρά τους από μία υπερκλάση «Μέλος».



Σχήμα 3.6 Υπερκαλυπτόμενη Γενίκευση

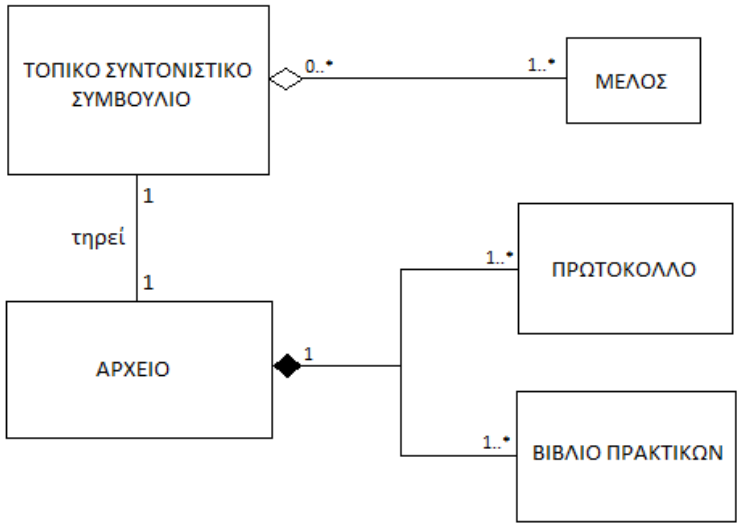
Σε μία πλήρη γενίκευση δεν επιτρέπεται να προστεθεί άλλη υποκλάση εκτός από αυτές που έχουν ήδη οριστεί και είναι οι επιτρεπτές. Το αντίθετο μίας πλήρους γενίκευσης είναι μία ελλιπής γενίκευση όπου επιπλέον υποκλάσεις επιτρέπεται να προστεθούν στο μέλλον. Στο μεγαλύτερο μέρος τους οι σχέσεις γενίκευσης είναι ελλιπείς.

3.3.2 Συσσώρευση (Aggregation) και Σύνθεση (Composition)

Η συσχέτιση συσσώρευσης είναι μια ειδική περίπτωση συσχέτισης και υποδηλώνει τη εξάρτηση μιας κλάσης από κάποια άλλη κλάση που αποτελεί μέρος της. Εκεί που διαφέρει από μια απλή συσχέτιση είναι ότι εδώ δεν επιτρέπεται η κυκλική συσχέτιση του «μέρους» από το «όλο», αλλά μόνο μία συσχέτιση από το όλο στο μέρος. Η συσσώρευση συμβολίζεται με μια συσχέτιση από το όλο προς το μέρος, στην οποία τοποθετείται ένας άσπρος ρόμβος στην πλευρά του όλου. Συνήθως θα μπορούμε να καταλάβουμε ότι έχουμε συσσώρευση όταν μια κλάση αποτελεί συλλογή από άλλες κλάσεις και υπονοούνται στην συσχέτιση τα ρήματα: Αποτελείται από, Περιέχει, Περιλαμβάνει και Έχει.

Η σύνθεση είναι μια ισχυρή μορφή συσχέτισης στην οποία το όλο περιέχει αποκλειστικά τα μέρη του, δεν μπορεί κάποιο άλλο όλο να περιέχει το ίδιο αντικείμενο. Δηλαδή ένα τμήμα ανήκει μόνο σε ένα όλο δημιουργώντας έτσι μια σχέση «ζωής και θανάτου» μεταξύ των δύο. Τα μέρη δημιουργούνται και καταστρέφονται ταυτόχρονα με το όλο. Την σύνθεση την συμβολίζουμε με μια συσχέτιση από το όλο προς το μέρος και τοποθετείται ένας μαύρος ρόμβος στην πλευρά του όλου. Μπορεί να γεννηθεί η απορία σε τι διαφέρει η σύνθεση από την συσσώρευση. Η συσσώρευση χρησιμοποιείται όταν οι επιμέρους κλάσεις έχουν διάρκεια ζωής ανεξάρτητη από την σύνθετη κλάση. Έτσι για παράδειγμα ένας πρόεδρος-μέλος δεν μπορεί να υπάρξει ανεξάρτητα από ένα σύλλογο, που σημαίνει ότι ορίζεται πρόεδρος-μέλος όσο υπάρχει ο σύλλογος, αν διαλυθεί ο σύλλογος ο πρόεδρος θα πάψει να υπάρχει. Η σύνθεση χρησιμοποιείται όταν οι επιμέρους κλάσεις έχουν διάρκεια ζωής που εξαρτάται και συμπίπτει με αυτή της σύνθετης κλάσης. Αυτό με λίγα λόγια σημαίνει ότι η διαγραφή του όλου θα έχει ως αποτέλεσμα την διαγραφή και των μερών του.

Ας εξετάσουμε ένα παράδειγμα συσσώρευσης και σύνθεσης πάνω στο σωματείο μας.



Σχήμα 3.7 Παράδειγμα Συσσώρευσης και Σύνθεσης.

Όπως βλέπουμε στο σχήμα 3.7 ένα Τοπικό Συντονιστικό Συμβούλιο ενός Εργασιακού Τομέα αποτελείται από ένα ή περισσότερα Μέλη. Η κλάση «Μέλος» συσχετίζεται με την κλάση «Τοπικό Συντονιστικό Συμβούλιο» με Συσσώρευση από την στιγμή που αν διαλυθεί το τοπικό Σ.Σ. τα μέλη θα υφίστανται. Σε αντίθετη περίπτωση αν διαλυθεί το Αρχείο, που τηρεί το τοπικό Σ.Σ. και το οποίο περιέχει διάφορα Πρωτόκολλα και Βιβλία Πρακτικών, θα διαγραφούν και τα αντικείμενα των κλάσεων Πρωτόκολλο και Βιβλίο Πρακτικών αφού δεν θα μπορούν να υπάρξουν χωρίς το Αρχείο. Έτσι βλέπουμε εδώ την διαφορά της Συσσώρευσης από την Σύνθεση όπου στην δεύτερη η διάρκεια ζωής των επιμέρους κλάσεων (μέρος) εξαρτάται και συμπίπτει με αυτή της σύνθετης κλάσης (όλο). Καταλαβαίνουμε ότι μια συσχέτιση πολύ πιθανόν να είναι σύνθεση αν έχει νόημα να πούμε ότι κάτι είναι μέρος κάποιου άλλου όπως για παράδειγμα ένα δωμάτιο είναι μέρος ενός κτιρίου ενώ μια διεύθυνση δεν είναι μέρος ενός ανθρώπου.

3.3.3 Αυτοπαθής Συσχέτιση (Reflexive Association)

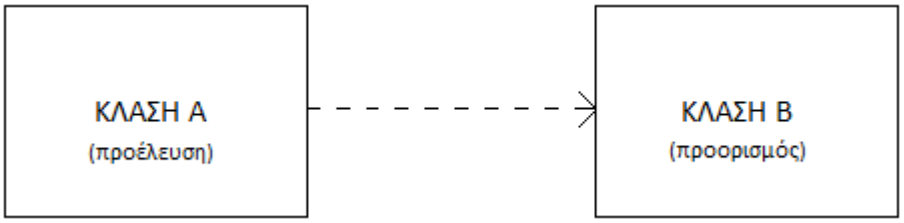
Μια κλάση μπορεί να συσχετίζεται και με τον εαυτό της και σε αυτή την περίπτωση λέμε πως έχουμε μια αυτοπαθή συσχέτιση. Η κλάση Μέλος μπορεί να συσχετίζεται με τον εαυτό της μέσω του ρόλου του Εκπροσώπου/Αντιπροσωπεύει όπως βλέπουμε στο παρακάτω σχήμα.

Σχήμα 3.8 Παράδειγμα Αυτοπαθής Συσχέτισης Μέλους

Όταν μια κλάση συσχετίζεται με τον εαυτό της τότε ένα στιγμιότυπό της, συσχετίζεται με ένα άλλο στιγμιότυπο της ίδιας κλάσης. Έτσι ένα στιγμιότυπο της κλάσης «Μέλος» μπορεί να είναι ο Εκπρόσωπος ενώ πολλά άλλα στιγμιότυπα τα Μέλη. Άρα, σε αυτή την αυτοπαθή συσχέτιση παρατηρώντας την πολλαπλότητα βλέπουμε ότι το στιγμιότυπο Εκπρόσωπος αντιπροσωπεύει ένα σύνολο από τα στιγμιότυπα Μέλη.

3.4 Εξαρτήσεις και Πακέτα

Οι εξαρτήσεις στην UML εμφανίζονται ώστε να δείχνουν τις μεταβολές που θα υποστεί ένα στοιχείο αν αλλάξει κάποιο άλλο στοιχείο από το οποίο το πρώτο εξαρτάται. Συμβολίζονται με διακεκομμένα βέλη και συνδέουν ένα διαγραμματικό στοιχείο από το οποίο ξεκινούν και ονομάζεται στοιχείο προέλευσης με ένα άλλο διαγραμματικό στοιχείο που ονομάζεται στοιχείο προορισμός. Έτσι, αν μεταβληθεί το στοιχείο προέλευσης θα πρέπει κατά πάσα πιθανότητα να αλλάξει και το στοιχείο προορισμός. Στόχος της ανάπτυξης είναι η ελαχιστοποίηση των εξαρτήσεων. Στο σχήμα 3.9 φαίνεται πως συμβολίζεται η εξάρτηση.

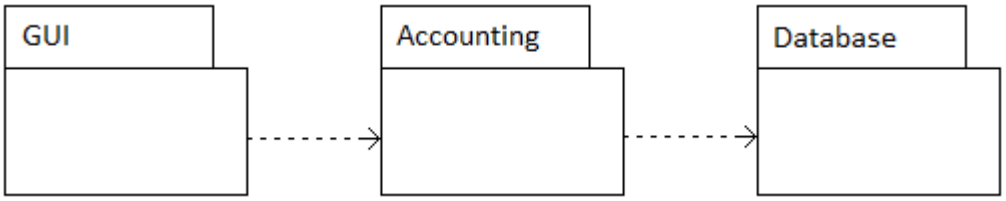


Σχήμα 3.9 Συμβολισμός Εξάρτησης

Τα μοντέλα ενός μεγάλου συστήματος είναι καλό να οργανώνονται σε πακέτα. Τα πακέτα προσφέρουν τη δυνατότητα ομαδοποίησης παρόμοιων, υπό κάποια έννοια μοντέλων μέσω της τοποθέτησης αυτών των μοντέλων στο ίδιο πακέτο. Επίσης παρέχουν την δυνατότητα ελέγχου εξαρτήσεων αφού μπορούμε να περιορίσουμε την πρόσβαση στα στοιχεία του πακέτου εκθέτοντας μόνο ένα υποσύνολό τους. Για παράδειγμα, τα στοιχεία με πρόσβαση πακέτου δεν είναι ορατά έξω από το πακέτο τους. Τέλος, τα πακέτα περιέχουν ένα χώρο ονομάτων, αφού είναι δυνατόν να έχουμε σε διαφορετικά πακέτα στοιχεία με το ίδιο όνομα χωρίς να προκαλείται σύγχυση. Ένα πακέτο συμβολίζεται με ένα φάκελο όπως φαίνεται στο σχήμα 3.10. Το όνομα του πακέτου αναγράφεται είτε μέσα στο πακέτο, είτε στο μικρό τετράγωνο στην πάνω αριστερή γωνία. Μπορούμε να εμφανίσουμε τα περιεχόμενα του πακέτου μέσα στο πακέτο ή να τα παραλείψουμε. Είναι χρησιμότερο να έχουμε ένα διάγραμμα στο οποίο να εμφανίζονται μόνο τα πακέτα και οι εξαρτήσεις τους, και σε ένα άλλο διάγραμμα τα περιεχόμενα των πακέτων. Τα πακέτα ενδέχεται να περιέχουν άλλα πακέτα. Οι εξαρτήσεις

μεταξύ των πακέτων είναι πολύ σημαντικές καθώς παίζουν καθοριστικό ρόλο για τον έλεγχο των αλλαγών σε ένα έργο.

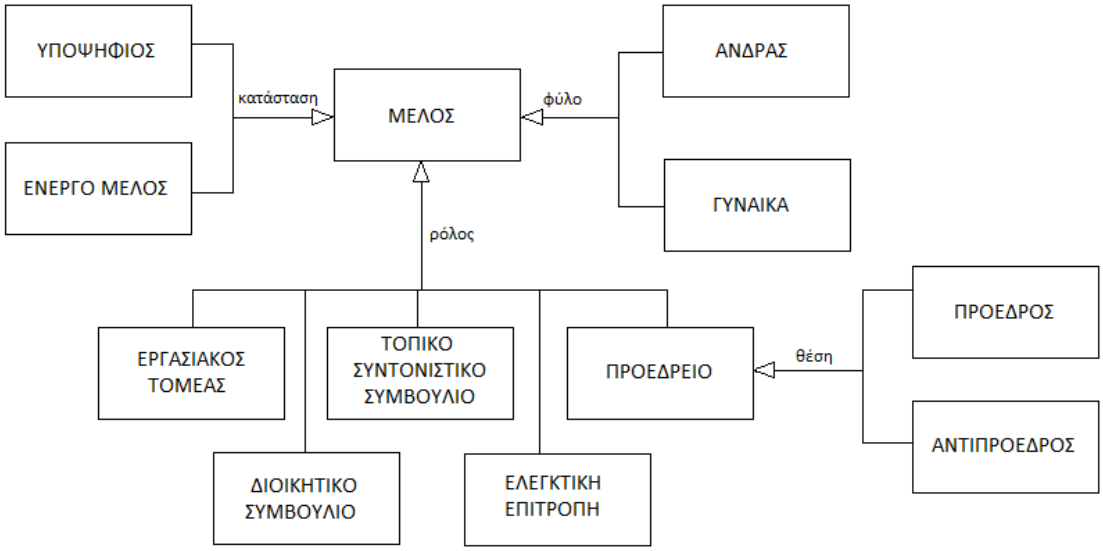
Ας εξετάσουμε το παράδειγμα που βλέπουμε στο σχήμα 3.10. Όπως βλέπουμε η γραφική διασύνδεση χρήστη (GUI) εξαρτάται, δηλαδή χρησιμοποιεί στοιχεία, από το πακέτο της λογιστικής (Accounting). Επίσης το πακέτο της λογιστικής εξαρτάται από στοιχεία του πακέτου της Βάσης Δεδομένων (Database). Έτσι αν αλλάξει κάτι στο πακέτο της λογιστικής δεν χρειάζεται να ανησυχήσουμε για αλλαγές στο πακέτο Βάσης Δεδομένων παρά μόνο για αλλαγές στο πακέτο του GUI, επειδή το πακέτο αυτό εξαρτάται από το πακέτο της λογιστικής. Στις γλώσσες προγραμματισμού ένα πακέτο της UML θα αντιστοιχεί σε κάποια αντίστοιχη έννοια πακέτου της γλώσσας προγραμματισμού, όπως για παράδειγμα σε ένα package της Java.



Σχήμα 3.10 Παράδειγμα Εξαρτήσεων σε πακέτα

3.5 Παράδειγμα Διαγράμματος Κλάσεων

Ήρθε η ώρα να δώσουμε ένα παράδειγμα για το πώς είναι οργανωμένο το σωματείο μας με ένα διάγραμμα κλάσεων. Όπως βλέπουμε στο σχήμα 3.11, στο διάγραμμα του σωματείου μας υπάρχουν πολλές κλάσεις που αφορούν την κλάση μέλος. Ανάλογα με τον τύπο της κάθε συσχέτισης έχουμε και μια ομάδα υποκλάσεων. Έτσι όσον αναφορά την *κατάσταση* του μέλους της ένωσης έχουμε τις υποκλάσεις υποψήφιος και ενεργό μέλος, ανάλογα με το *φύλο* τον άνδρα και τη γυναίκα, ανάλογα με τον *ρόλο* τον εργασιακό τομέα, το τοπικό συντονιστικό συμβούλιο, την ελεγκτική επιτροπή, το Δ.Σ. και το προεδρείο. Αλλά και η κλάση προεδρείο έχει δύο υποκλάσεις τον πρόεδρο και τον αντιπρόεδρο βάση της συσχέτισης *θέση*. Έτσι κατευθείαν κάποιος με την χρήση ενός διαγράμματος κλάσεων μπορεί να πάρει μια πρώτη εικόνα για το πώς είναι οργανωμένο το σύστημα που μοντελοποιείται.



Σχήμα 3.11 Διάγραμμα Κλάσεων τρόπου οργάνωσης Σωματείου.

3.6 Συνοψίζοντας

Τα διαγράμματα κλάσεων που είδαμε σε αυτό το κεφάλαιο μας βοήθησαν να διακρίνουμε τις αλληλεπιδράσεις της εφαρμογής και των αντικείμενων που πρόκειται να προγραμματίσουμε για το υποθετικό μας σωματείο. Μέσα από παραδείγματα είδαμε τα μέρη μιας κλάσης, τις σχέσεις που μπορούν να αναπτυχθούν μεταξύ των κλάσεων και τέλος πως δημιουργούνται εξαρτήσεις μεταξύ των κλάσεων και γιατί είναι καλό να οργανώνουμε το σύστημά μας σε πακέτα. Τέλος, είδαμε ένα παράδειγμα πως μπορούμε να μοντελοποιήσουμε με ένα διάγραμμα κλάσεων την οργάνωση του σωματείου μας. Στο επόμενο κεφάλαιο θα εξετάσουμε τα διαγράμματα ακολουθίας καθώς και τα στοιχεία από τα οποία αποτελούνται μέσα από παραδείγματα.

ΚΕΦΑΛΑΙΟ 4. ΔΙΑΓΡΑΜΜΑΤΑ ΑΚΟΛΟΥΘΙΑΣ

4.1 Εισαγωγή

Ένα από τα ευρύτερα χρησιμοποιούμενα δυναμικά διαγράμματα της UML είναι το διάγραμμα ακολουθίας, με το οποίο θα ασχοληθούμε σε αυτό το κεφάλαιο. Μέχρι το τέλος αυτού του κεφαλαίου, θα γνωρίζουμε τι είναι ένα διάγραμμα ακολουθίας, από τι στοιχεία αποτελείται και στο τέλος θα παρουσιάσουμε κάποια διαγράμματα ακολουθίας από το σύστημά μας δηλαδή το υποθετικό σωματείο μας.

4.2 Ορισμός

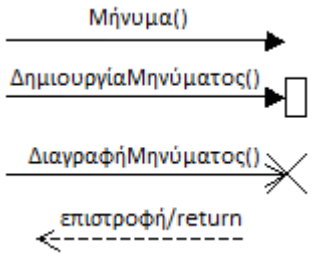
Ένα διάγραμμα ακολουθίας απεικονίζει την ακολουθία των ενεργειών που συμβαίνουν σε ένα σύστημα. Η κλήση μεθόδων για κάθε αντικείμενο του συστήματος αλλά και η σειρά με την οποία γίνεται αυτή συλλαμβάνονται σε ένα διάγραμμα ακολουθίας. Αυτό κάνει το διάγραμμα ακολουθίας ένα πολύ χρήσιμο εργαλείο το οποίο μπορεί εύκολα να απεικονίσει την δυναμική συμπεριφορά ενός συστήματος.

Ένα διάγραμμα ακολουθίας αναπαριστά τη χρονική σειρά εκτέλεσης των γεγονότων που λαμβάνουν χώρα κατά την διάρκεια μίας αλληλεπίδρασης με αποτέλεσμα να δίνει έμφαση στη χρονική σειρά που εκτελούνται οι αλληλεπιδράσεις μεταξύ των αντικειμένων. Αυτό που πρέπει να προσέξουμε είναι ότι τα διαγράμματα ακολουθίας δεν αναπαριστούν τις σχέσεις μεταξύ των αντικειμένων αλλά μόνο τον τρόπο αλληλεπίδρασής τους. Συνοψίζοντας μπορούμε να πούμε ότι ένα τέτοιο διάγραμμα απεικονίζει τη σειρά των ενεργειών ενός σεναρίου στον άξονα του χρόνου και εξηγεί ποιος δράστης εκκινεί μια ενέργεια και με ποιον δράστη συνεργάζεται αυτός. Παρουσιάζει μόνο την κύρια ροή με την επιτυχημένη κατάληξη και όχι εναλλακτικές πορείες.

4.3 Στοιχεία ενός Διαγράμματος Ακολουθίας

Ένα διάγραμμα ακολουθίας αποτελείται από δύο βασικά στοιχεία. Τα αντικείμενα (objects) και τα μηνύματα (messages).

Αντικείμενο: Το πρωτεύον στοιχείο που περιλαμβάνεται σε ένα διάγραμμα ακολουθίας είναι το αντικείμενο, το στιγμιότυπο δηλαδή μιας κλάσης. Ένα διάγραμμα ακολουθίας αποτελείται από την αλληλουχία αλληλεπιδράσεων μεταξύ των διαφορετικών αντικειμένων σε μια συγκεκριμένη χρονική περίοδο. Ένα αντικείμενο αντιπροσωπεύεται από ένα ορθογώνιο που περιέχει ένα χαρακτηριστικό. Αυτό το χαρακτηριστικό συντίθεται από το όνομα του αντικειμένου, τον χαρακτήρα «:»	Παράδειγμα: 

και τέλος το όνομα της κλάσης όπου ανήκει αυτό το αντικείμενο, όπως φαίνεται και στο παράδειγμα.	
Μήνυμα: Οι αλληλεπιδράσεις μεταξύ των διαφορετικών αντικειμένων σε ένα διάγραμμα ακολουθίας ονομάζονται μηνύματα. Ένα μήνυμα παρουσιάζεται ως ένα κατευθυνόμενο βέλος. Ανάλογα με τον τύπο μηνύματος, το βέλος έχει διαφορετικές μορφές. Με αυτά μπορούμε να αναπαραστήσουμε απλά μηνύματα, ειδικά μηνύματα για την δημιουργία και την καταστροφή αντικείμενων αλλά και μηνύματα αντιδράσεων για το πώς δηλαδή αντιδρούν τα αντικείμενα σε κάποια γεγονότα.	Παραδείγματα: 
Πλαίσιο Ενεργοποίησης – Life Line: Κάτω από κάθε αντικείμενο εκτείνεται μια διακεκομμένη γραμμή που αντιστοιχεί στη γραμμή ζωής του αντικειμένου. Όταν λαμβάνεται ένα μήνυμα ξεκινά ένα πλαίσιο ενεργοποίησης (διπλή γραμμή) το οποίο συνεχίζεται σε όλη την διάρκεια εκτέλεσης της λειτουργίας.	

4.3.1 Τύποι Μηνυμάτων

Κλήση-Call: Αυτό το είδος χρησιμοποιείται για να κληθεί μια λειτουργία. Δηλαδή ένα αντικείμενο εκτελεί μια λειτουργία ενός άλλου αντικειμένου. Απεικονίζεται με ένα βέλος από το καλών αντικείμενο προς το αντικείμενο λήψης. Πάνω από το βέλος αναγράφεται το όνομα της λειτουργίας που καλείται, με τις τυχόν παραμέτρους σε παρενθέσεις.

Επιστροφή-Return: Παρουσιάζεται ως μια διακεκομμένη ακμή που ξεκινά από το κληθέν αντικείμενο και καταλήγει σε αυτό που το κάλεσε. Αν υπάρχει τιμή επιστροφής αυτή αναγράφεται πάνω στην ακμή. Για να έχουμε τέτοιο είδος μηνύματος θα πρέπει να έχει προηγηθεί μήνυμα κλήσης και να έχουν ολοκληρωθεί και άλλες πιθανές υπο-κλήσεις που περιέχει η εκτέλεση της συγκεκριμένης λειτουργίας.

Αποστολή-Send: Στέλνει ένα σήμα-ειδοποίηση από ένα αντικείμενο σε ένα άλλο. Εδώ υπάρχει η περίπτωση ένα αντικείμενο να στείλει ένα μήνυμα στον εαυτό του. Απεικονίζεται με ένα βέλος με μισή αιχμή προς το αντικείμενο λήπτη του σήματος.

Δημιουργία-Create: Καλεί μια κλάση και δημιουργεί ένα αντικείμενό της. Απεικονίζεται με ένα βέλος και την λέξη «create» πάνω του, με κατεύθυνση από το αντικείμενο που ζητάει την δημιουργία προς το αντικείμενο που θα δημιουργηθεί.

Διαγραφή-Destroy: Καλεί ένα αντικείμενο να διαγραφεί. Η διαγραφή μπορεί να αφορά είτε άλλο αντικείμενο είτε τον εαυτό του. Σηματοδοτεί την καταστροφή ενός αντικειμένου ως αποτέλεσμα ενός μηνύματος από ένα άλλο αντικείμενο. Απεικονίζεται με ένα βέλος και την λέξη «destroy» πάνω του, με κατεύθυνση

από το αντικείμενο που ζητάει την καταστροφή προς το προς καταστροφή αντικείμενο.

Μια ειδική κατηγορία μηνύματος κλήσης είναι η *αυτόκληση*. Σε αυτήν την περίπτωση ένα αντικείμενο καλεί μία λειτουργία στον εαυτό του και απεικονίζεται σαν μία κλήση που ξεκινάει από το αντικείμενο και καταλήγει πάλι σε αυτό.

4.3.2 Είδος Μηνύματος

Τα μηνύματα μπορεί να είναι σύγχρονα ή ασύγχρονα.

ΣΥΓΧΡΟΝΟ: Ένα σύγχρονο μήνυμα δεσμεύει το αντικείμενο που πραγματοποιεί την σύγχρονη κλήση το οποίο πρέπει να περιμένει μέχρι αυτή να ολοκληρωθεί. Συμβολίζεται με ένα βέλος με ολόκληρη συμπαγή αιχμή.

ΑΣΥΓΧΡΟΝΟ: Ένα ασύγχρονο μήνυμα δεν εμποδίζει το αντικείμενο που πραγματοποιεί την κλήση να συνεχίσει την όποια λειτουργία του. Συνεπώς ένα τέτοιο μήνυμα δημιουργεί ένα νέο νήμα εκτέλεσης ή ακόμα μπορεί να δημιουργήσει και ένα νέο αντικείμενο. Ως αποτέλεσμα έχουμε την ασύγχρονη κλήση, η οποία συμβολίζεται με μία ακμή που καταλήγει σε μισό βέλος. Οι ασύγχρονες κλήσεις χρησιμοποιούνται για την μοντελοποίηση πολυνηματικών εφαρμογών.

4.4 Βρόχοι και Τελεστές Ελέγχου

Τα διαγράμματα ακολουθίας δεν είναι γενικά κατάλληλα στο να απεικονίζουν λεπτομέρειες, αλγόριθμους δηλαδή με βρόχους, loops και συμπεριφορές υπό συνθήκη. Από την άλλη όμως δείχνουν πολύ ξεκάθαρα τις κλήσεις που λαμβάνουν χώρα μεταξύ των συμμετεχόντων αλλά και ποιοι συμμετέχοντες εκτελούν ποια επεξεργασία. Παρόλα αυτά η UML προσφέρει για τη μοντελοποίηση βρόχων και συνθηκών τα πλαίσια αλληλεπίδρασης ή αλλιώς interaction frames. Θα εξετάσουμε τους πιο γνωστούς τελεστές ελέγχου:

Τελεστής alt: Γνωστός και ως υποθετικός τελεστής. Το σώμα ενός υποθετικού τελεστή χωρίζεται σε περιοχές με τη χρήση διακεκομμένων γραμμών. Κάθε περιοχή αναπαριστά ένα εναλλακτικό τμήμα μιας συνθήκης και περιέχει μια μόνο συνθήκη. Αν η συνθήκη μιας περιοχής ικανοποιείται, τότε αυτή η περιοχή εκτελείται. Αν από την άλλη δεν ικανοποιηθεί τότε δεν εκτελείται καμία περιοχή. Στον υποθετικό τελεστή υπάρχει η δυνατότητα μια περιοχή να έχει τη συνθήκη else όπου σε αυτήν την περίπτωση θα εκτελείται όταν καμία άλλη συνθήκη δεν είναι αληθής.

Τελεστής opt: Ή αλλιώς προαιρετικός τελεστής. Τα μηνύματα που περιέχονται σε έναν προαιρετικό τελεστή εκτελούνται μόνο αν ικανοποιείται η λογική Boolean συνθήκη που τοποθετείται μέσα στις αγκύλες [].

Τελεστής par: Ή αλλιώς τελεστής παράλληλης εκτέλεσης. Αυτός ο τελεστής παρουσιάζει μια παράλληλη συμπεριφορά ή εκτέλεση και χωρίζει σε περιοχές με τη χρήση διακεκομμένων γραμμών το τμήμα που μας ενδιαφέρει. Κάθε

περιοχή εκτελείται παράλληλα με τις υπόλοιπες περιοχές. Η εκτέλεση από την άλλη των μηνυμάτων σε κάθε περιοχή εκτελείται σειριακά. Θα πρέπει να προσέξουμε καθώς ο τελεστής παράλληλης εκτέλεσης δεν χρησιμοποιείται όταν οι διαφορετικές περιοχές αλληλεπιδρούν.

Τελεστής loop: Σε αυτή την περίπτωση έχουμε επαναληπτική εκτέλεση η οποία καθορίζεται από μια συνθήκη η οποία ορίζει πόσες φορές θα εκτελεστεί το σώμα του τελεστή αυτού. Όσο η συνθήκη είναι αληθής, εκτελείται επαναληπτικά το σώμα του τελεστή έως ότου να γίνει ψευδής και όπου η εκτέλεση θα συνεχιστεί έξω από το σώμα του τελεστή.

Τελεστής break: Ο τελεστής διακοπής αλλιώς, περιέχει μια συνθήκη, η οποία αν ικανοποιείται εκτελούνται οι ενέργειες που περιέχονται στο σώμα του τελεστή διακοπής και ύστερα διακόπτεται η διαδικασία. Αν η συνθήκη δεν ικανοποιείται συνεχίζεται η διαδικασία μετά τον τελεστή διακοπής.

Τελεστής neg: Παρουσιάζει την άρνηση. Το περιεχόμενο του τμήματος δείχνει μη έγκυρη αλληλεπίδραση.

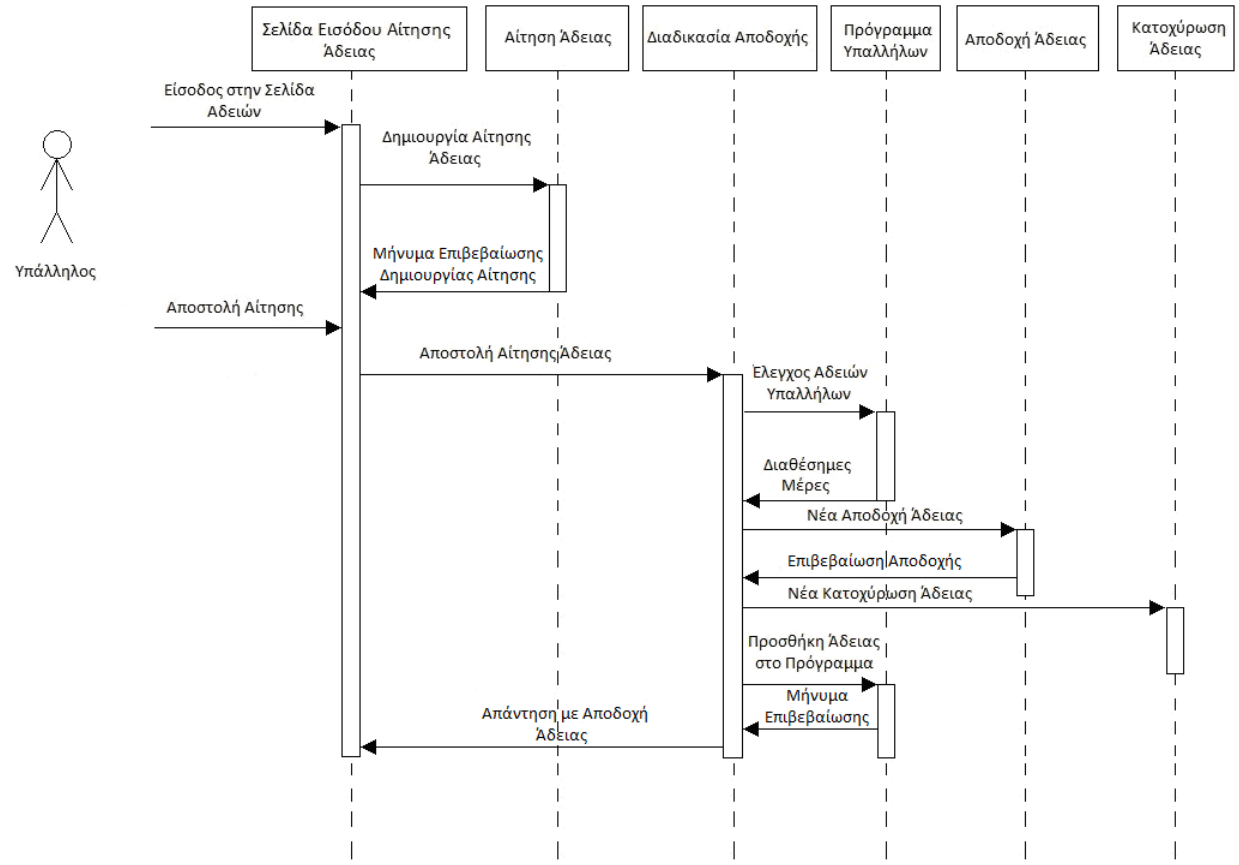
Τελεστής ref: Παρουσιάζει αναφορά. Το περιεχόμενο του τμήματος μέσα στον τελεστή αναφέρεται σε αλληλεπίδραση που ορίζεται σε άλλο διάγραμμα.

4.5 Παράδειγμα Διαγράμματος Ακολουθίας

Το σωματείο που μοντελοποιούμε απασχολεί Υπαλλήλους. Θα δούμε ένα παράδειγμα της ακολουθίας διαδικασιών που πρέπει να κάνει ένας υπάλληλος για να πάρει άδεια.

Όπως βλέπουμε στο σχήμα 4.1 ένας υπάλληλος για να πάρει άδεια πρέπει πρώτα να μπει στο σύστημα στην Σελίδα Εισόδου Αίτησης Άδειας. Ύστερα δημιουργεί μέσα σε αυτή την σελίδα μια αίτηση συμπληρωμένη με τις επιθυμητές μέρες άδειας και την στέλνει στο σύστημα. Το σύστημα τότε ξεκινάει την διαδικασία Αποδοχής της Άδειας του υπαλλήλου η οποία περιλαμβάνει κάποιες άλλες διαδικασίες. Το σύστημα ελέγχει τις άδειες που έχουν κατοχυρώσει οι άλλοι υπάλληλοι στο πρόγραμμα υπαλλήλων και επιστρέφει τις διαθέσιμες μέρες. Από την στιγμή που είναι διαθέσιμες κατοχυρώνει την άδεια και την προσθέτει στο πρόγραμμα αδειών. Τέλος στέλνει ένα μήνυμα επιβεβαίωσης στον Υπάλληλο ότι η άδεια του κατοχυρώθηκε.

Αυτή την διαδικασία που μόλις περιγράψαμε μπορούμε να την δούμε στο παρακάτω διάγραμμα ακολουθίας.



Σχήμα 4.1 Παράδειγμα Διαγράμματος Ακολουθίας για ανάληψη Άδειας.

4.6 Συνοψίζοντας

Σε αυτό το κεφάλαιο είδαμε τα διαγράμματα ακολουθίας και πώς αυτά απεικονίζουν την ακολουθία των ενεργειών που συμβαίνουν σε ένα σύστημα. Είδαμε επίσης τα στοιχεία από τα οποία απαρτίζεται ένα τέτοιο διάγραμμα, τους τύπους μηνυμάτων που ανταλλάσσονται από τα αντικείμενα και τους τελεστές που μπορούν να χρησιμοποιηθούν. Τέλος, είδαμε μια εφαρμογή αυτού του τύπου διαγραμμάτων στο σωματείο μας. Στο επόμενο κεφάλαιο θα ασχοληθούμε με τα διαγράμματα καταστάσεων και θα δούμε πως συμβάλουν στην μοντελοποίηση ενός συστήματος.

ΚΕΦΑΛΑΙΟ 5. ΔΙΑΓΡΑΜΜΑΤΑ ΚΑΤΑΣΤΑΣΕΩΝ

5.1 Εισαγωγή

Μέχρι τώρα, έχουμε δει τα διαγράμματα περιπτώσεων χρήσης, τα διαγράμματα κλάσεων και τα διαγράμματα ακολουθίας. Τα διαγράμματα καταστάσεων είναι ο τέταρτος τύπος διαγραμμάτων που θα εξετάσουμε και χρησιμοποιούνται προκειμένου να μοντελοποιήσουμε τη δυναμική συμπεριφορά των αντικειμένων μιας κλάσης και τον τρόπο που μεταβάλλεται η κατάστασή τους ως αποτέλεσμα εκτέλεσης κάποιων συμβάντων. Ένα διάγραμμα καταστάσεων απεικονίζει το σύστημα σαν μια μηχανή πεπερασμένων καταστάσεων και περιλαμβάνει:

1. Διάφορες καταστάσεις στις οποίες μπορεί να βρεθεί το αντικείμενο κατά τη λειτουργία του συστήματος.
2. Τις μεταβάσεις από τη μια κατάσταση στην άλλη ανταποκρινόμενο σε εσωτερικά ή εξωτερικά γεγονότα.
3. Δραστηριότητες που ενδεχομένως αντιδρούν σε διάφορα συμβάντα.

5.2 Ορισμός

Καθώς γράφουμε κώδικα, είναι απαραίτητο να γίνουν κατανοητές οι λεπτομέρειες των καταστάσεων/φάσεων στις οποίες μπορεί να βρεθεί ένα αντικείμενο μιας κλάσης καθώς και τις μεταβάσεις του σε διάφορες χρονικές στιγμές καθώς σημειώνονται διάφορα γεγονότα.

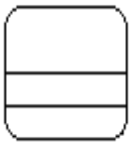
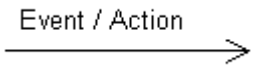
Τα διαγράμματα κατάστασης χρησιμοποιούνται για να βοηθήσουν τον υπεύθυνο ανάπτυξης ενός έργου να κατανοήσει καλύτερα οποιεσδήποτε σύνθετες και ασυνήθιστες λειτουργίες ή επιχειρησιακές ροές κάποιων εξειδικευμένων τμημάτων του συστήματος. Εν ολίγοις, τα διαγράμματα κατάστασης απεικονίζουν τη δυναμική συμπεριφορά ολόκληρου του συστήματος ή ενός υποσυστήματος ή ακόμα και μόνο ενός αντικειμένου σε ένα σύστημα. Αυτό γίνεται με τη βοήθεια των *στοιχείων συμπεριφοράς.* Ως στοιχεία συμπεριφοράς μπορούμε να θεωρήσουμε μια κατάσταση, μια μετάβαση ή μια δραστηριότητα. Αυτούς τους όρους θα τους αναλύσουμε αναλυτικά στην επόμενη παράγραφο.

Είναι σημαντικό να σημειωθεί ότι η δημιουργία διαγράμματος καταστάσεων για το σύστημά μας δεν είναι υποχρεωτική για όλα τα τμήματα του συστήματός μας αλλά το σχεδιάζουμε μόνο αν το κρίνουν οι ανάγκες μας.

5.3 Στοιχεία ενός Διαγράμματος Καταστάσεων

Ακριβώς όπως καθορίσαμε τις κλάσεις σε ένα διάγραμμα κλάσεων έτσι είναι απαραίτητο να καθοριστούν και τα στοιχεία ενός διαγράμματος καταστάσεων. Αρχικά θα εξετάσουμε ποια είναι τα στοιχεία ενός διαγράμματος καταστάσεων.

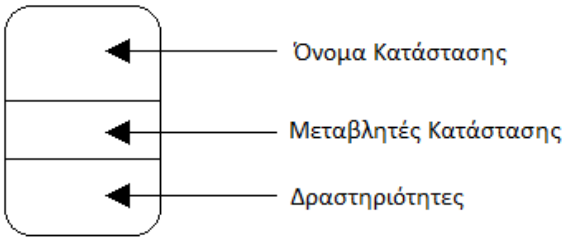
Ένα διάγραμμα καταστάσεων αποτελείται από τα ακόλουθα στοιχεία:

Στοιχεία και η περιγραφή τους	Σύμβολο
Αρχική Κατάσταση: Έτσι παρουσιάζεται η αφετηρία ή η πρώτη δραστηριότητα της ροής. Παρουσιάζεται ως ένας γεμάτος κύκλος. Αυτή η κατάσταση καλείται επίσης *"ψευδό κατάσταση"* επειδή δεν περιέχει καμία μεταβλητή που να την περιγράφει περαιτέρω αλλά ούτε και καμία δραστηριότητα.	
Κατάσταση: Αντιπροσωπεύει την κατάσταση του αντικειμένου σε μια συγκεκριμένη χρονική στιγμή. Σε ένα διάγραμμα καταστάσεων, θα συναντήσουμε πολλαπλά τέτοια σύμβολα, ένα για κάθε κατάσταση του αντικειμένου που συζητάμε. Την σχεδιάζουμε με ένα ορθογώνιο με στρογγυλευμένες γωνίες και διαμερίσματα (όπως μια κλάση απλά με στρογγυλευμένες γωνίες). Θα περιγράψουμε αυτό το σύμβολο λεπτομερώς λίγο αργότερα.	
Μετάβαση: Ένα βέλος που παρουσιάζει την μετάβαση ενός αντικειμένου από την μία κατάσταση στην άλλη. Το γεγονός/ενέργεια που πυροδοτεί και προκαλεί τη μετάβαση γράφεται πάνω από το βέλος. Οι μεταβάσεις που συμβαίνουν επειδή σε μία κατάσταση ολοκληρώθηκε μία δραστηριότητα καλούνται "*triggerless*" μεταβάσεις. Εάν ένα γεγονός πρέπει να συμβεί μετά από την ολοκλήρωση κάποιου άλλου γεγονότος ή δράσης, το γεγονός ή η δράση καλείται συνθήκη φρουρός. Η μετάβαση πραγματοποιείται αφότου πραγματοποιηθεί η συνθήκη φρουρός. Αυτή η συνθήκη απεικονίζεται με ένα τετράγωνο και με μεταβάσεις στις κορυφές του που υποδηλώνουν τις διαφορετικές περιπτώσεις είτε όταν εκπληρωθεί η συνθήκη είτε όταν δεν εκπληρωθεί (με άλλα λόγια, υπό μορφή έκφρασης Boolean (Α/Ψ)).	
Κατάσταση Αναστολής (History State): Μια ροή μπορεί να απαιτήσει από ένα αντικείμενο να μείνει σε κατάσταση αναμονής-αναστολής και όταν συμβεί ένα συγκεκριμένο περιστατικό να επιστρέψει στη κατάσταση που ήταν πριν μπει σε κατάσταση αναμονής - την τελευταία ενεργή του κατάσταση δηλαδή. Σε ένα διάγραμμα καταστάσεων αυτό απεικονίζεται με τη βοήθεια του γράμματος Η (από το History) που εσωκλείεται μέσα σε έναν κύκλο και «θυμάται» την τελευταία ενεργή κατάσταση ενός αντικειμένου και στην οποία μπορεί να επιστρέψει.	

Σήμα: Όταν ένα γεγονός προκαλεί την αποστολή ενός μηνύματος/εναύσματος σε μία κατάσταση, οι οποίες αργότερα προκαλούν την μετάβαση, τότε εκείνο το μήνυμα που στέλνεται καλείται σήμα. Το αναπαριστούμε με ένα ορθογώνιο που περιέχει το λεκτικό <<Σήμα>>/<<Signal>> επάνω από το γεγονός/δράση.	<<Signal>> Event / Action Event / Action
Τελική Κατάσταση: Το τέλος του διαγράμματος καταστάσεων παρουσιάζεται με έναν κύκλο που περιέχει μια βούλα μέσα, αποκαλούμενο επίσης ως τελική κατάσταση. Η τελική κατάσταση είναι ακόμα ένα παράδειγμα μιας «ψευδό-κατάστασης» επειδή δεν περιγράφει καμία μεταβλητή ή δράση.	

Σημείωση: Οι αλλαγές που συμβαίνουν στο σύστημα, όπως για παράδειγμα μία διεργασία στο παρασκήνιο ενώ εκτελείται η κύρια διαδικασία, καλούνται *"υπό-καταστάσεις"*. Ακόμα κι αν έχουν επιπτώσεις στο κύρια διαδικασία, μία υπό-κατάσταση δεν παρουσιάζεται ως τμήμα του κύριας κατάστασης. Ως εκ τούτου, απεικονίζεται σαν να περιέχεται μέσα στην ροή της κύριας κατάστασης.

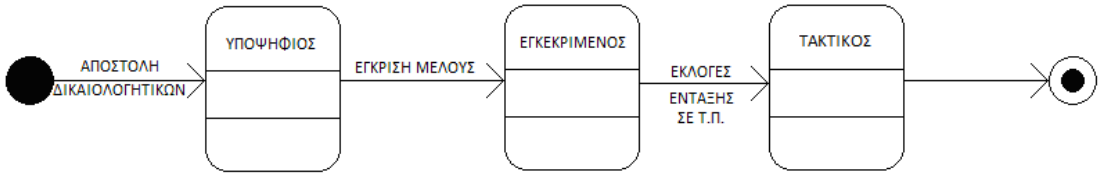
Όπως είδατε παραπάνω, μία κατάσταση αντιπροσωπεύεται από ένα ορθογώνιο με στρογγυλευμένες άκρες. Μέσα σε αυτό περιέχονται, το όνομά της, οι μεταβλητές της, και οι δραστηριότητές της, όπως φαίνεται στο σχήμα 5.1.



Σχήμα 5.1: Η δομή του στοιχείου Κατάσταση.

5.4 Σχεδιάζοντας ένα Διάγραμμα Καταστάσεων

Ας υποθέσουμε ότι ένα άτομο επιθυμεί να γίνει μέλος του υποθετικού σωματείου μας. Το σχήμα 5.2 θα ήταν ένα διάγραμμα καταστάσεων για ένα τέτοιο απλό σενάριο. Αντιπροσωπεύει την ομαλή ροή. Δεν παρουσιάζει τις υποκαταστάσεις για αυτό το σενάριο, οι οποίες θα έδειχναν τα εναλλακτικά σενάρια αν κάτι πήγαινε στραβά. Όπως βλέπουμε στο σχήμα 5.2 όταν ένα άτομο ενδιαφέρεται να γίνει μέλος του σωματείου στέλνει τα απαραίτητα δικαιολογητικά οπότε η κατάσταση του γίνεται «Υποψήφιος». Η γενική συνέλευση και το Διοικητικό Συμβούλιο εγκρίνουν την αίτηση του Υποψηφίου οπότε ο «Υποψήφιος» μεταβαίνει στην κατάσταση «Εγκεκριμένος». Τέλος το εγκεκριμένο μέλος θα πρέπει να ενταχθεί σε κάποιο Τοπικό Παράρτημα οπότε μετά το γεγονός αυτό το άτομο μπορεί πλέον να οριστεί ως «Τακτικό» Μέλος.



Σχήμα 5.2: Παράδειγμα μιας απλής ροής ενός διαγράμματος καταστάσεων.

Κατά τον σχεδιασμό ενός διαγράμματος καταστάσεων ο κύριος στόχος μας είναι να προσδιορίσουμε την αρχική και την τελική κατάσταση ενός τέτοιου διαγράμματος. Ύστερα θα πρέπει να αναρωτηθούμε από τι άλλες καταστάσεις ή στάδια της ζωής του ένα αντικείμενο περνάει. Συχνά μπορούμε να βρούμε τις καταστάσεις αυτές με την εξέταση των οριακών τιμών των χαρακτηριστικών του συστήματος μας. Παραδείγματος χάριν, ένα σεμινάριο γίνεται πλήρες αν ο αριθμός των μελών που μπορούν να το παρακολουθήσουν φτάσει στο μέγιστο. *Το Πλήρες* είναι έγκυρη κατάσταση επειδή ισχύουν διαφορετικοί κανόνες τώρα: όταν ένα μέλος προσπαθεί να εγγραφεί για το σεμινάριο, μπαίνει σε έναν κατάλογο αναμονής και άρα το σεμινάριο έχει τώρα δύο ακραίες καταστάσεις (λόγω 2 είδη μελών-αυτοί που πρόλαβαν και αυτοί που είναι στην λίστα αναμονής).

Αφού τελειώσουμε με τον προσδιορισμό όλων των δυνατών καταστάσεων μπορούμε να αρχίσουμε να προσδιορίζουμε τις μεταβάσεις. Για κάθε κατάσταση θα πρέπει να αναρωτηθούμε πώς το αντικείμενο, για το οποίο εντοπίσαμε τις καταστάσεις, μπορεί να βγει από αυτή (αν αυτό είναι βέβαια δυνατόν). Αυτή η διαδικασία θα μας δώσει μια μετάβαση. Επειδή όλες οι μεταβάσεις οδηγούν από μία κατάσταση σε μία άλλη, θα πρέπει να αναρωτηθούμε σε ποια νέα κατάσταση μας οδηγεί η μετάβαση αυτή. Πρέπει επίσης να εξετάσουμε τις μεθόδους που προσδιορίσαμε στο διάγραμμα κλάσεων αφού μερικές από αυτές αντιστοιχούν σε μια μετάβαση στο διάγραμμα καταστάσεων.

Ο προσδιορισμός πιθανών συνθηκών λάθους, καθώς σχεδιάζουμε ένα διάγραμμα καταστάσεων, είναι συχνό φαινόμενο επειδή πρέπει να ρωτάμε συνεχώς «Πρέπει να επιτραπεί αυτή η μετάβαση όταν είναι το αντικείμενο σε αυτή την κατάσταση;». Όταν η απάντηση είναι ναι, θα πρέπει να προσθέσουμε τη μετάβαση στο διάγραμμά μας. Όταν η απάντηση είναι όχι θα πρέπει να σημειώσουμε αυτό το πιθανό ζήτημα έτσι ώστε οι προγραμματιστές που θα αναπτύξουν τον κώδικα ελέγχου λάθους, να μην επιτρέψουν στη μετάβαση να συμβεί.

Παρόλο που θα ήταν καλό να μπορούσαμε να χρησιμοποιήσουμε την κληρονομικότητα στα διαγράμματα καταστάσεων δεν το κάνουμε. Ο ορισμός της κληρονομικότητας λέει ότι αν και η υποκλάση είναι παρόμοια με την υπερκλάση της, αυτή παραμένει διαφορετική. Η συμπεριφορά της υποκλάσης είναι συνήθως διαφορετική από αυτή των υπερκλάσεων. Αυτό σημαίνει ότι πρέπει να επανεξετάσουμε το διάγραμμα καταστάσεων όταν έχουμε κληρονομικότητα. Το καλό είναι ότι πολλές από τις καταστάσεις και τις μεταβάσεις είναι επαναχρησιμοποιήσιμες.

5.5 Τι πρέπει να προσέχουμε

Τα διαγράμματα καταστάσεων τα σχεδιάζουμε όταν η επιχειρησιακή λογική για μια ιδιαίτερη ροή είναι πολύ σύνθετη ή χρειάζεται κατανοηθεί περισσότερο από τον υπεύθυνο της ανάπτυξης έτσι ώστε να μπορέσει να κωδικοποιηθεί τέλεια.

Θα πρέπει να ταξινομούμε τις καταστάσεις και τις μεταβάσεις έτσι ώστε να εξασφαλίσουμε ότι οι γραμμές που τέμνονται μεταξύ τους στην διαδρομή ανάμεσα στα δύο άκρα να είναι ελάχιστες. Οι γραμμές που τέμνονται μπορεί να προκαλέσουν σύγχυση στη κατανόηση της απεικόνισης του διαγράμματος.

5.6 Συνοψίζοντας

Σε αυτό το κεφάλαιο ασχοληθήκαμε με τα διαγράμματα καταστάσεων. Είδαμε τα στοιχεία που απαρτίζουν ένα τέτοιο διάγραμμα καθώς και το τι θα πρέπει να προσέχουμε όταν δημιουργούμε ένα από αυτά. Μέσα από ένα παράδειγμα, είδαμε τις καταστάσεις από τις οποίες περνάει κατά τη διάρκεια ζωής του ένα αντικείμενο. Τέλος, παρατηρήσαμε ότι θα πρέπει να προσέχουμε να μην μπαίνουμε σε λεπτομέρειες ώστε να μην γίνονται σύνθετα τα διαγράμματά μας. Στο κεφάλαιο που ακολουθεί θα δούμε αναλυτικά τα διαγράμματα δραστηριοτήτων και τι μας βοηθούν να μοντελοποιήσουμε.

ΚΕΦΑΛΑΙΟ 6. ΔΙΑΓΡΑΜΜΑΤΑ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ

6.1 Εισαγωγή

Στο προηγούμενο κεφάλαιο, ασχοληθήκαμε με τα διαγράμματα καταστάσεων, τα στοιχεία από τα οποία αποτελείται ένα τέτοιο διάγραμμα καθώς και την σημασία τους και τέλος, πώς μπορούμε να χτίσουμε ένα διάγραμμα καταστάσεων για ένα συγκεκριμένο αντικείμενο στην περίπτωση του σωματείου μας.

Σε αυτό το κεφάλαιο, θα μιλήσουμε για το επόμενο δυναμικό διάγραμμα —το διάγραμμα δραστηριοτήτων. Μέχρι το τέλος αυτού του κεφαλαίου θα γνωρίζουμε τι είναι ένα διάγραμμα δραστηριοτήτων, από ποια στοιχεία αποτελείται και θα σχεδιάσουμε ένα τέτοιο διάγραμμα για το σύστημά μας.

6.2 Ορισμός

Ο ευκολότερος τρόπος για να κατανοήσουμε ένα διάγραμμα δραστηριοτήτων είναι να σκεφτούμε ένα διάγραμμα ροής του κώδικα ενός προγράμματος (flow chart). Τα διαγράμματα ροής χρησιμοποιούνται για να απεικονίσουν τη ροή της επιχειρησιακής λογικής αλλά και τα γεγονότα που οδηγούν στην λήψη αποφάσεων όσον αναφορά τις ενέργειες που θα πρέπει να πραγματοποιηθούν στον κώδικα.

Τα διαγράμματα δραστηριοτήτων αντιπροσωπεύουν τις επιχειρησιακές και λειτουργικές ροές ενός συστήματος. Το διάγραμμα δραστηριοτήτων είναι ένα δυναμικό διάγραμμα που παρουσιάζει τις δραστηριότητες και τα γεγονότα που προκαλούν ένα αντικείμενο να μπει σε μία συγκεκριμένη κατάσταση.

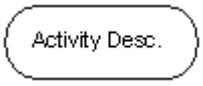
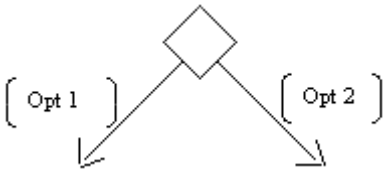
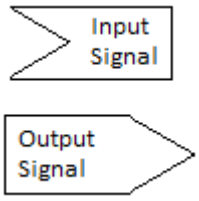
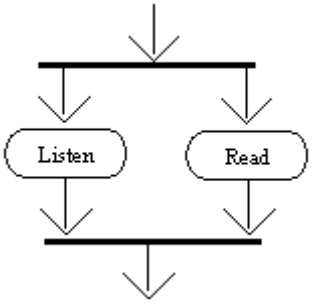
Άρα, σε τι διαφέρει ένα διάγραμμα δραστηριοτήτων από ένα διάγραμμα καταστάσεων; Τα διαγράμματα καταστάσεων παρουσιάζουν τις διαφορετικές καταστάσεις τις οποίες ένα αντικείμενο μπορεί να βιώσει κατά τη διάρκεια του κύκλου της ζωής του μέσα στο σύστημα αλλά και τις μεταβάσεις στις καταστάσεις αυτές. Αυτές οι μεταβάσεις απεικονίζουν τις δραστηριότητες που προκαλούν αυτές τις μεταβάσεις, και παρουσιάζονται με βέλη.

Ένα διάγραμμα δραστηριοτήτων ασχολείται περισσότερο με αυτές καθαυτές τις μεταβάσεις και τις δραστηριότητες που προκαλούν τις αλλαγές στις καταστάσεις του αντικειμένου.

6.3 Στοιχεία ενός Διαγράμματος Δραστηριοτήτων

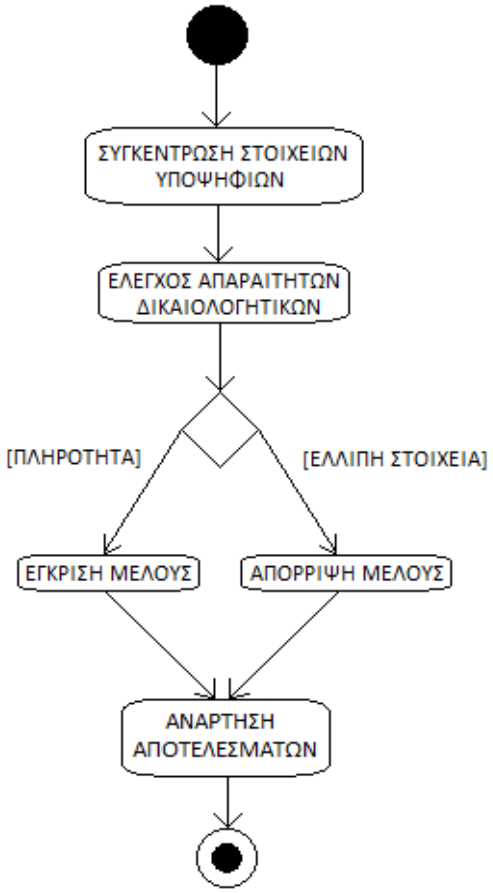
Ας δούμε τώρα τις δομικές μονάδες από τις οποίες μπορεί να απαρτίζεται ένα διάγραμμα δραστηριοτήτων.

Ένα διάγραμμα δραστηριοτήτων αποτελείται από τα ακόλουθα στοιχεία συμπεριφοράς:

Στοιχεία και η περιγραφή τους	Σύμβολο
Αρχική Δραστηριότητα: Παρουσιάζει την αφετηρία ή την πρώτη δραστηριότητα στην ροή που θα αναπαραστήσουμε. Παρουσιάζεται ως ένας στερεός κύκλος ο οποίος είναι παρόμοιος με αυτόν που χρησιμοποιείται για την αρχική κατάσταση.	
Δραστηριότητα: Σχεδιάζεται ως ένα ορθογώνιο με στρογγυλευμένες (σχεδόν ωοειδείς) γωνίες.	
Αποφάσεις: Όπως και στα διαγράμματα ροής έτσι και εδώ η ανάγκη λήψης μιας απόφασης απεικονίζεται με ένα τετράγωνο, με τις επιλογές να παρουσιάζονται σε κάθε ένα από τα βέλη που ξεκινούν από τις πλευρές του τετραγώνου. Μέσα σε αγκύλες σημειώνουμε μια μικρή περιγραφή των προϋποθέσεων για να ληφθεί αυτή η απόφαση.	
Σήμα: Όταν μια διαδικασία στέλνει ή δέχεται ένα μήνυμα τότε αυτή η δραστηριότητα καλείται σήμα. Υπάρχουν δύο είδη σήματος: Το Σήμα Εισόδου (Δραστηριότητα Εισερχόμενου Μηνύματος) το οποίο παρουσιάζεται με ένα κοίλο πολύγωνο και το Σήμα Εξόδου (Δραστηριότητα Αποστολής Μηνύματος) που παρουσιάζεται από ένα κυρτό πολύγωνο.	
Παράλληλες Δραστηριότητες: Μερικές δραστηριότητες συμβαίνουν ταυτόχρονα ή παράλληλα. Τέτοιες δραστηριότητες καλούνται παράλληλες δραστηριότητες. Παραδείγματος χάριν, ακούοντας έναν ομιλητή και διαβάζοντας μια παρουσίαση στον πίνακα είναι μια παράλληλη δραστηριότητα. Μπορούμε να τις απεικονίσουμε με μια οριζόντια παχιά γραμμή από την οποία ξεκινούν οι ταυτόχρονες δραστηριότητες η μια δίπλα στην άλλη. Μία δεύτερη οριζόντια παχιά γραμμή στην οποία καταλήγουν οι ταυτόχρονες δραστηριότητας ορίζει το τέλος της παράλληλης δραστηριότητας.	
Τελική δραστηριότητα: Το τέλος του διαγράμματος δραστηριοτήτων παρουσιάζεται με έναν κύκλο που περιέχει μια βούλα μέσα, αποκαλούμενο επίσης ως τελική δραστηριότητα.	

6.4 Δημιουργία ενός Διαγράμματος Δραστηριοτήτων

Ας πάρουμε για παράδειγμα την πολύ απλή διαδικασία για την εγγραφή ενός μέλους στο σωματείο αφού βέβαια έχει ολοκληρωθεί η αποστολή των δικαιολογητικών του.



Σχήμα 6.1 Παράδειγμα ενός Διαγράμματος Δραστηριοτήτων για την εγγραφή μέλους στο σωματείο

Όπως μπορούμε να δούμε και στο σχήμα 6.1, η πρώτη δραστηριότητα που περιέχεται στο διάγραμμα, είναι να συγκεντρώσει το Διοικητικό Συμβούλιο τα στοιχεία και δικαιολογητικά όλων των υποψηφίων που ενδιαφέρονται να γίνουν μέλη στο σωματείο. Έπειτα θα πρέπει να γίνει εκτενής έλεγχος όλων των δικαιολογητικών που έχουν αποσταλεί για να ληφθεί μια απόφαση για την αποδοχή ή μη του υποψηφίου. Εάν διαπιστωθεί πληρότητα στα δικαιολογητικά που έχει αποστείλει ο υποψήφιος, τότε γίνεται έγκριση της αίτησής του. Διαφορετικά αυτή θα απορριφθεί. Η τελική δραστηριότητα αφορά την ανάρτηση των αποτελεσμάτων είτε σε γραπτή, είτε σε ηλεκτρονική μορφή. Μετά από αυτήν το διάγραμμα δραστηριότητας ολοκληρώνεται.

6.5 Swim Lanes / Λωρίδες «Κολύμβησης»-Δράσης

Τα διαγράμματα δραστηριοτήτων μας παρέχουν άλλη μία δυνατότητα, να διευκρινίσουμε ποιος δράστης/actor εκτελεί κάποια δραστηριότητα. Ας

εξετάσουμε το διάγραμμα δραστηριότητας του σχήματος 6.1. Θα μπορούσαμε να είχαμε διασπάσει τις δραστηριότητές μας περαιτέρω, για παράδειγμα θα μπορούσαμε να χωρίσουμε τη δραστηριότητα κατά την οποία γίνεται ο έλεγχος των δικαιολογητικών σε:

1. Έλεγχος σπουδών υποψηφίου.
2. Έλεγχος για καταδίκες, για αδικήματα και κακουργήματα στο μητρώο του υποψηφίου.
3. Έλεγχος συνταξιοδότησης υποψηφίου.
4. Έλεγχος για τυχόν δικαστικές απαγορεύσεις που έχουν εκδοθεί για τον υποψήφιο και ούτω καθεξής...

Αντίστοιχα η έγκριση του μέλους θα περιελάμβανε το να στείλει το Διοικητικό Συμβούλιο την έγκριση του στην Γενική Συνέλευση. Επίσης, στην ανάρτηση αποτελεσμάτων θα πρέπει να ενημερωθεί το νέο μέλος για το ποσό που πρέπει να καταβάλει, να του αποσταλεί η κάρτα μέλους του και να ενημερωθεί για τα δικαιώματα και τις υποχρεώσεις που έχει πλέον ως μέλος του σωματείου. Εδώ μπορούμε να δούμε ότι εμπλέκονται δύο επιπλέον δράστες, ο ένας είναι η οντότητα «Γενική Συνέλευση» και ο άλλος είναι το «Νέο Μέλος». Εάν επιθυμούμε να διακρίνουμε σε ένα διάγραμμα δραστηριοτήτων τις επιπλέον δραστηριότητες που πραγματοποιούνται από τους μεμονωμένους δράστες, θα πρέπει πρώτα να σχεδιάσουμε τις Λωρίδες Κολύμβησης ή αλλιώς Δράσης δηλαδή κάθετες στήλες οι οποίες χωρίζονται από διακεκομμένες κάθετες γραμμές. Σε κάθε στήλη δίνουμε ένα όνομα ανάλογα με τον δράστη που εμπλέκεται κάθε φορά και τοποθετούμε κάθε δραστηριότητα κάτω από κάθε δράστη που εκτελεί αυτές τις δραστηριότητες. Τέλος δείχνουμε πως συνδέονται μεταξύ τους αυτές οι δραστηριότητες.

6.6 Συνοψίζοντας

Μέχρι τώρα έχουμε αναλύσει τα διαγράμματα περιπτώσεων χρήσης, κλάσεων, ακολουθίας, καταστάσεων και τώρα των δραστηριοτήτων. Σε αυτό το κεφάλαιο ορίσαμε τι είναι ένα διάγραμμα δραστηριοτήτων, είδαμε από τι στοιχεία αποτελείται και μέσα από παραδείγματα είδαμε πως μπορούμε να σχεδιάσουμε ένα. Στο επόμενο κεφάλαιο, κλείνει ο κύκλος παρουσίασης των διαγραμμάτων με τα φυσικά διαγράμματα ή αλλιώς διαγράμματα υλοποίησης που όπως υποδηλώνουν και με το όνομά τους, ασχολούνται με την φυσική αρχιτεκτονική που θα έχει το σύστημα που μας ενδιαφέρει να μοντελοποιήσουμε, δηλαδή τις υλικές υποδομές που θα υποστηρίζονται.

ΚΕΦΑΛΑΙΟ 7. ΦΥΣΙΚΑ ΔΙΑΓΡΑΜΜΑΤΑ Ή ΔΙΑΓΡΑΜΜΑΤΑ ΥΛΟΠΟΙΗΣΗΣ

7.1 Εισαγωγή

Τα φυσικά διαγράμματα ή αλλιώς διαγράμματα υλοποίησης περιλαμβάνουν τα εξής δύο διαγράμματα:

1. Διαγράμματα Συστατικών (Component)

2. Διαγράμματα Ανάπτυξης (Deployment)

Υπάγονται στην κατηγορία των διαγραμμάτων δομής, και ως εκ τούτου στα στατικά διαγράμματα, που μας περιγράφουν το πώς είναι οργανωμένα τα συστατικά τμήματα του συστήματός μας.

Μέχρι τώρα ασχοληθήκαμε κυρίως με την λογική αρχιτεκτονική η οποία μελετά τη λειτουργικότητα του συστήματος και πώς αυτή αναπτύσσεται. Περιέχει την λογική της εφαρμογής και όχι τη φυσική διανομή αυτής της λογικής σε διαφορετικές διεργασίες, προγράμματα ή υπολογιστές.

Τα διαγράμματα της UML που χρησιμοποιούνται για τον προσδιορισμό της λογικής αρχιτεκτονικής είναι τα:

1. Διαγράμματα Περιπτώσεων Χρήσης
2. Διαγράμματα Κλάσεων
3. Διαγράμματα Κατάστασης
4. Διαγράμματα Δραστηριοτήτων
5. Διαγράμματα Ακολουθίας και

Κατά την σχεδίαση του πληροφοριακού μας συστήματος κάποια στιγμή προκύπτουν κάποιες βασικές ερωτήσεις. Όπως γνωρίζουμε οι υπολογιστικές πλατφόρμες αποτελούνται από το υλικό, το λογισμικό και την δικτύωση η οποία ενώνει τα διάφορα υποσυστήματα μεταξύ τους σε ένα ενιαίο σύστημα. Έτσι γεννιούνται ερωτήσεις για το πώς θα επιλέξουμε αυτό το υλικό και λογισμικό αλλά επίσης, και τι αρχιτεκτονικές και τι υλικό δικτύωσης θα χρησιμοποιήσουμε. Με λίγα λόγια μας ενδιαφέρει το πώς θα εκφράσουμε τη φυσική αρχιτεκτονική του συστήματος με μία στάνταρ διαγραμματική μορφή.

7.2 Φυσική Αρχιτεκτονική

Η φυσική αρχιτεκτονική ενός συστήματος περιγράφει τα τμήματα που συγκροτούν ένα σύστημα στα πλαίσια τόσο της λογικής όσο και της φυσικής αρχιτεκτονικής. Κατά κάποιο τρόπο θα μπορούσαμε να πούμε ότι λειτουργεί ως χάρτης του συστήματος με τον οποίο ο αναλυτής μπορεί εύκολα να προσδιορίσει το που βρίσκεται ή που υλοποιείται μια συγκεκριμένη λειτουργία του συστήματος. Η φυσική αρχιτεκτονική δεν είναι αμετάβλητη. Έχουμε την δυνατότητα να κάνουμε αλλαγές ή ακόμα και να επεκτείνουμε μια συγκεκριμένη θέση του συστήματος χωρίς βέβαια να επηρεάζεται αρνητικά το

υπόλοιπο σύστημα. Επίσης, θα πρέπει να προσέχουμε το πώς αναπτύσσουμε τα διάφορα τμήματα έτσι ώστε να είναι δυνατή η επαναχρησιμοποίηση τους και από άλλα συστήματα. Ένα απλό και καλά ορισμένο interface, θα βοηθήσει κάποιον που αναπτύσσει ένα συγκεκριμένο κομμάτι του συστήματος χωρίς να χρειάζεται να κατανοήσει πλήρως όλες τις λεπτομέρειες του συστήματος.

Συνοψίζοντας θα μπορούσαμε να πούμε ότι η φυσική αρχιτεκτονική ασχολείται με τη λεπτομερή περιγραφή ενός συστήματος από άποψη υλικού και λογισμικού. Περιγράφει:

1. Τη δομή του υλικού (hardware), περιλαμβάνοντας διάφορους κόμβους και πως αυτοί συνδέονται μεταξύ τους,
2. Τη φυσική δομή και τις εξαρτήσεις του κώδικα που υλοποιεί τις έννοιες που ορίστηκαν στην λογική αρχιτεκτονική και τέλος
3. Την κατανομή του run-time λογισμικού σε διεργασίες, προγράμματα και άλλα προϊόντα λογισμικού.

Με λίγα λόγια, περιγράφει τα συστατικά του υλικού και του λογισμικού ενώ παράλληλα αντιστοιχίζει την λογική αρχιτεκτονική στη φυσική αρχιτεκτονική όπου, για παράδειγμα, οι κλάσεις κ.τ.λ. αντιστοιχούνται σε συστατικά, διεργασίες (processes) και υπολογιστές. Έτσι μπορούμε από ένα τμήμα της λογικής αρχιτεκτονικής να μεταβούμε στο αντίστοιχο τμήμα της φυσικής αρχιτεκτονικής και αντίστροφα. Η φυσική αρχιτεκτονική απαντά σε ερωτήματα όπως:

⇒ Σε ποια προγράμματα ή διεργασίες (processes) βρίσκονται οι κλάσεις και τα αντικείμενα;
⇒ Σε ποιους υπολογιστές εκτελούνται τα προγράμματα και οι διαδικασίες;
⇒ Ποιοι υπολογιστές και τι άλλα περιφερειακά βρίσκονται στο σύστημα και πώς συνδέονται μεταξύ τους;
⇒ Ποιες είναι οι εξαρτήσεις ανάμεσα σε διαφορετικά αρχεία κώδικα; Αν αλλάξει ένα αρχείο, ποια άλλα τμήματα πρέπει να επαναμεταγλωττιστούν (recompile);

Η Φυσική Αρχιτεκτονική μοντελοποιείται με τα Φυσικά Διαγράμματα τα οποία όπως είπαμε είναι τα:

- *Διαγράμματα Συστατικών (Component Diagrams):* Αναπαριστά σε υψηλό επίπεδο τη δομή του κώδικα. *Συστατικό ή συνιστώσα* είναι μια φυσική μονάδα υλοποίησης κώδικα (source code και binary code). Επιπλέον αναπαριστά τις αλληλοσυνδέσεις μεταξύ συστατικών, συμπεριλαμβανομένων συστατικών πηγαίου κώδικα, συστατικών δυαδικού κώδικα και συστατικών εκτελέσιμου κώδικα.
- *Διαγράμματα Ανάπτυξης (Deployment Diagrams):* Αναπαριστά τη φυσική/runtime αρχιτεκτονική του υλικού (hardware) και λογισμικού σε ένα σύστημα. Αναπαριστώνται οι μηχανές και οι υπολογιστές-κόμβοι (nodes) που χρησιμοποιούνται από το σύστημα καθώς και οι συνδέσεις τους. *Κόμβοι* είναι τα φυσικά αντικείμενα που στη γενική περίπτωση έχουν τουλάχιστον μνήμη και δυνατότητα επεξεργασίας.

7.2.1 Υλικό

Οι έννοιες του υλικού που χρησιμοποιούνται στη φυσική αρχιτεκτονική είναι οι εξής:

1) *Επεξεργαστές:* Σε αυτήν την κατηγορία περιλαμβάνονται οι υπολογιστές που εκτελούν το πρόγραμμα και οι οποίοι μπορεί να είναι από μικροεπεξεργαστές μέχρι υπερυπολογιστές.
2) *Περιφερειακές Συσκευές:* Υποστηρίζουν το σύστημα, για παράδειγμα εκτυπωτές, δρομολογητές, οθόνες κ.α. και είναι συνδεδεμένες σε έναν τουλάχιστον επεξεργαστή από τον οποίο κιόλας ελέγχονται.
3) *Συνδέσεις:* Οι επεξεργαστές συνδέονται με άλλους επεξεργαστές και με περιφερειακές συσκευές. Μια σύνδεση ανάμεσα σε δύο κόμβους περιγράφεται από δύο στοιχεία, πρώτον από το φυσικό μέσο για παράδειγμα μία οπτική ίνα και δεύτερον από το πρωτόκολλο λογισμικού που χρησιμοποιείται (π.χ. TCP/IP).

7.3 Βασικές Έννοιες

7.3.1 Κόμβος – Node

Γενικά

Ένας κόμβος είναι ένα φυσικό αντικείμενο που γενικά αναπαριστά έναν πόρο επεξεργασίας, ο οποίος έχει τουλάχιστον μνήμη και συχνά και δυνατότητα επεξεργασίας. Οι κόμβοι περιλαμβάνουν τις συσκευές υπολογισμού αλλά και ανθρώπινους πόρους ή πόρους μηχανικής επεξεργασίας. Οι κόμβοι μπορούν να αναπαρασταθούν ως τύποι και ως στιγμιότυπα. Τα υπολογιστικά στιγμιότυπα του χρόνου εκτέλεσης, τόσο στιγμιότυπα συνιστωσών, όσο και στιγμιότυπα αντικειμένων, μπορούν να βρίσκονται σε στιγμιότυπα κόμβων.

Συμβολισμός

Ένας κόμβος παρουσιάζεται ως ένα σχήμα που μοιάζει με την τρισδιάστατη όψη ενός κύβου. Ο τύπος ενός κόμβου έχει όνομα της μορφής:

τύπος-κόμβου

Ένα στιγμιότυπο κόμβου έχει όνομα και όνομα τύπου. Το όνομα ενός κόμβου και ο τύπος του μπορεί να είναι υπογραμμισμένα ή όχι και να βρίσκονται μέσα ή κάτω από τον κόμβο. Το όνομα έχει την εξής μορφή:

όνομα ‘:’ *τύπος-κόμβου*

Το όνομα είναι το όνομα ενός οποιουδήποτε κόμβου (εφόσον υπάρχει). Ο τύπος-κόμβου λέει τι είδους είναι ο κόμβος. Οποιοδήποτε ή και τα δύο στοιχεία είναι προαιρετικά. Αν παραληφθεί ο τύπος του κόμβου, μπορεί να παραληφθεί η άνω-κάτω τελεία.

Τα διακεκομμένα βέλη με τη λέξη κλειδί «support» παρουσιάζουν τη δυνατότητα ενός τύπου κόμβου να υποστηρίξει έναν τύπο συνιστώσας.

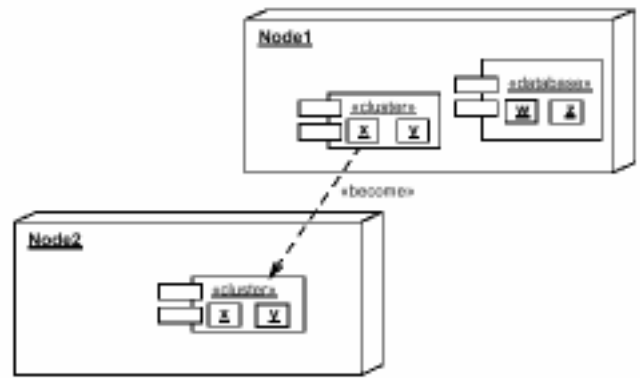
Εναλλακτικά, αυτό μπορεί να φανεί με ένθεση των συμβόλων της συνιστώσας στο σύμβολο του κόμβου.

Στιγμιότυπα συνιστωσών και αντικείμενα μπορούν να περιέχονται μέσα σε σύμβολα στιγμιότυπων κόμβων. Αυτό δηλώνει ότι τα στοιχεία βρίσκονται στα στιγμιότυπα των κόμβων.

Οι κόμβοι μπορεί να είναι συνδεδεμένοι με συσχετίσεις με άλλους κόμβους. Μία συσχέτιση ανάμεσα σε κόμβους δηλώνει μία διαδρομή επικοινωνίας ανάμεσα στους κόμβους. Η συσχέτιση μπορεί να έχει ένα στερεότυπο (stereotype) που να δηλώνει το είδος της διαδρομής επικοινωνίας (για παράδειγμα, το είδος του καναλιού ή του δικτύου).

Παράδειγμα

Το παρόν παράδειγμα δείχνει δύο κόμβους που περιέχουν μία συνιστώσα (cluster) η οποία μεταναστεύει από έναν κόμβο σε έναν άλλο και μία συνιστώσα (βάση δεδομένων) που παραμένει στη θέση της.



Σχήμα 7.1 Παράδειγμα επικοινωνίας δύο κόμβων.

7.3.2 Συνιστώσα – Component

Γενικά

Ένας τύπος συνιστώσας αναπαριστά ένα τμήμα της υλοποίησης του συστήματος το οποίο έχει τη δυνατότητα να κατανεμηθεί, συμπεριλαμβανομένου του κώδικα λογισμικού (πηγαίου ή εκτελέσιμου), αλλά και εταιρικών κειμένων κλπ. σε ένα φυσικό σύστημα. Οι συνιστώσες μπορούν να χρησιμοποιηθούν για να παρουσιάσουν εξαρτήσεις, όπως εξαρτήσεις μεταγλώττισης ή χρόνου εκτέλεσης ή πληροφοριακές εξαρτήσεις. Ένα στιγμιότυπο συνιστώσας αναπαριστά μία μονάδα υλοποίησης σε χρόνο εκτέλεσης και μπορεί να χρησιμοποιηθεί για να δείξει τις μονάδες υλοποίησης που έχουν ταυτότητα κατά το χρόνο εκτέλεσης, συμπεριλαμβανομένης της θέσης τους στους κόμβους.

Συμβολισμός

Μία συνιστώσα αναπαριστάται με ένα ορθογώνιο με δύο μικρά ορθογώνια να προεξέχουν στη μία του πλευρά. Ένας τύπος συνιστώσας έχει όνομα της μορφής:

τύπος-συνιστώσας

Ένα στιγμιότυπο συνιστώσας έχει όνομα και τύπο. Το όνομα μίας συνιστώσας και ο τύπος της μπορούν να εμφανιστούν υπογραμμισμένα είτε μέσα στο σύμβολο της συνιστώσας είτε πάνω ή κάτω από αυτό, με τη σύνταξη:

όνομα-συνιστώσας ':' *τύπος-συνιστώσας*

Κάθε ένα ή και τα δύο στοιχεία είναι προαιρετικά. Αν παραληφθεί ο τύπος-συνιστώσας, το ίδιο θα πρέπει να συμβεί και με την άνω-κάτω τελεία.

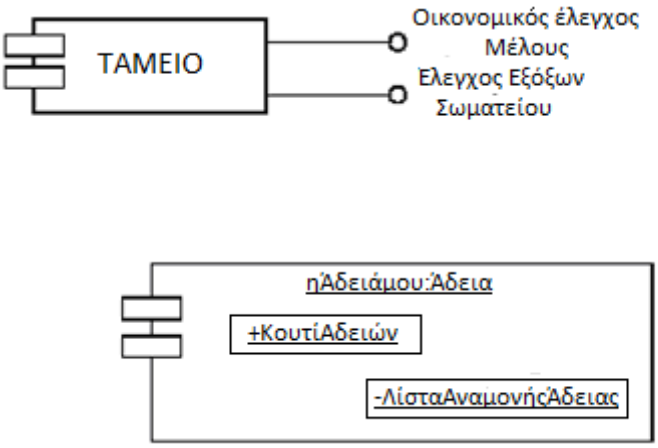
Μπορεί να χρησιμοποιηθεί μία ιδιότητα για να δηλώσει τη φάση του κύκλου ζωής του κώδικα που περιγράφει η συνιστώσα (πηγαίος, δυαδικός ή εκτελέσιμος). Οι συνιστώσες (συμπεριλαμβανομένων των προγραμμάτων, των DLLs, των εικόνων που συνδέονται κατά το χρόνο εκτέλεσης κλπ) μπορούν να βρίσκονται σε κόμβους.

Τα αντικείμενα που βρίσκονται σε ένα στιγμιότυπο συνιστώσας (δηλαδή που υλοποιούνται από αυτό) παρουσιάζονται ένθετα μέσα στο σύμβολο του στιγμιοτύπου της συνιστώσας. Κατ' αναλογία, οι κλάσεις που υλοποιούνται από μία συνιστώσα μπορούν να παρουσιαστούν ως ένθετες μέσα σε αυτή, αυτό δηλώνει υλοποίηση και όχι ιδιοκτησία.

Τα στοιχεία που βρίσκονται μέσα σε (δηλαδή υλοποιούνται από) μία συνιστώσα παρουσιάζονται ένθετα μέσα στο σύμβολο της συνιστώσας. Είναι δυνατό επίσης, να παρουσιάζεται η ορατότητα του στοιχείου σε άλλες συνιστώσες, χρησιμοποιώντας τον ίδιο συμβολισμό με αυτόν που είδαμε νωρίτερα στο κεφάλαιο 3.2 για την ορατότητα των χαρακτηριστικών και μεθόδων μιας κλάσης (τοποθετώντας δηλαδή το σύμβολο της ορατότητας πριν το όνομα του στοιχείου). Η έννοια της ορατότητας εξαρτάται από τη φύση της συνιστώσας. Για μία συνιστώσα πηγαίας γλώσσας (όπως ο κώδικας ενός προγράμματος), θα πρέπει να ελέγχεται η προσπελασιμότητα των δομών της πηγαίας γλώσσας. Για μία συνιστώσα κώδικα χρόνου εκτέλεσης, θα πρέπει να ελέγχεται η δυνατότητα του κώδικα που βρίσκεται σε άλλες συνιστώσες να καλεί ή να προσπελαύνει με κάποιο τρόπο τον κώδικα στην παρούσα συνιστώσα.

Παράδειγμα

Το παράδειγμα δείχνει μία συνιστώσα με διασυνδέσεις που δείχνουν την χρήση αυτής της συνιστώσας του Ταμείου για τον Οικονομικό έλεγχο ενός μέλους αλλά και για τον Έλεγχο εξόδων του Σωματείου και επίσης μία συνιστώσα που περιέχει αντικείμενα κατά το χρόνο εκτέλεσης όταν κάποιος κάνει αίτηση για άδεια.



Σχήμα 7.2 Παραδείγματα δύο συνιστωσών.

7.4 ΔΙΑΓΡΑΜΜΑΤΑ ΣΥΣΤΑΤΙΚΩΝ

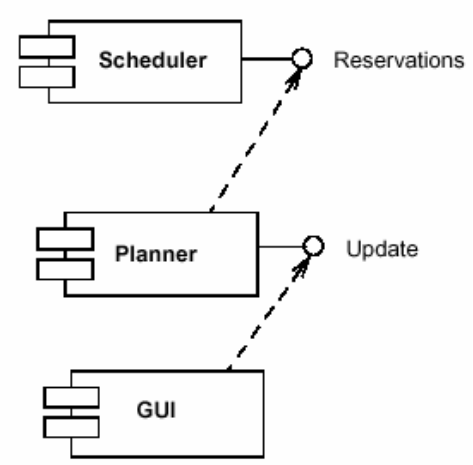
7.4.1 Ορισμός

Ένα διάγραμμα συστατικών παρουσιάζει τις εξαρτήσεις ανάμεσα στις συνιστώσες λογισμικού, συμπεριλαμβανομένων των συνιστωσών πηγαίου κώδικα, συνιστωσών δυαδικού κώδικα και εκτελέσιμων συνιστωσών. Για μία εταιρία, οι συνιστώσες «λογισμικού» είναι με την ευρεία έννοια οι εταιρικές διαδικασίες και τα κείμενα. Ένα τμήμα λογισμικού μπορεί να αναπαρασταθεί με ένα στερεότυπο-πρότυπο (stereotype) συνιστώσας. Ορισμένες συνιστώσες υπάρχουν κατά το χρόνο μεταγλώττισης, ορισμένες κατά το χρόνο σύνδεσης, ορισμένες κατά την εκτέλεση, ενώ άλλες μπορούν να υπάρχουν σε περισσότερους από έναν χρόνους. Μία συνιστώσα μόνο μεταγλώττισης είναι μία συνιστώσα που έχει νόημα μόνο κατά το χρόνο της μεταγλώττισης. Μία συνιστώσα εκτέλεσης στην περίπτωση αυτή θα μπορούσε να είναι ένα εκτελέσιμο πρόγραμμα.

Τα διαγράμματα συστατικών αποτελούνται από τρεις κύριες μονάδες, τα συστατικά, τις εξαρτήσεις και το σύστημα διεπαφής ή αλλιώς interface. Τα συστατικά αναπαριστούν το φυσικό πακετάρισμα ενός συνόλου από κώδικα. Οι εξαρτήσεις είναι οι σχέσεις μεταξύ των συστατικών και δείχνουν πώς οι αλλαγές που υφίσταται ένα συστατικό μπορούν να επηρεάσουν τα υπόλοιπα συστατικά. Οι εξαρτήσεις αναπαρίστανται με μια διακεκομμένη γραμμή μεταξύ δύο ή περισσότερων συστατικών. Τέλος το σύστημα διεπαφής χρησιμοποιείται από τα συστατικά για να μπορέσουν να επικοινωνήσουν μεταξύ τους. Η κυριότερη διαφορά των διαγραμμάτων συστατικών με τα υπόλοιπα διαγράμματα UML είναι ότι τα διαγράμματα αυτά αναπαριστούν την όψη υλοποίησης (deployment view) του συστήματος. Για την κατασκευή ενός συστήματος βασισμένο στα συστατικά, το σύστημα αποσυντίθεται σε συστατικά και ύστερα καθορίζονται οι διεπαφές του συστήματος. Στην συνέχεια αναπτύσσονται συστατικά τα οποία υλοποιούν τις διεπαφές καθώς και άλλα συστατικά που συνεργάζονται μεταξύ τους μέσω των διεπαφών.

7.4.2 Βασικά Στοιχεία και Συμβολισμός τους

Τα συστατικά ή αλλιώς συνιστώσες αναπαριστούν ένα modular, δηλαδή ένα αντικαταστήσιμο τμήμα του συστήματος μας. Όπως είπαμε είναι μια φυσική μονάδα υλοποίησης κώδικα που είναι επαναχρησιμοποιήσιμα και μπορούν να αντικατασταθούν χωρίς να επηρεαστεί το υπόλοιπο σύστημα. Θα πρέπει να παρέχουν μια διεπαφή για να επιτρέπουν και σε άλλα συστατικά να αλληλεπιδρούν με αυτά και να μπορούν έτσι να χρησιμοποιούν τις υπηρεσίες τους. Τέλος αποτελούνται ή υλοποιούνται από artifacts τα οποία είναι εκτελέσιμα αρχεία, βιβλιοθήκες εφαρμογών ή ακόμα και συστατικά αρχείων που αναπαριστούν τον πηγαίο κώδικα μιας εφαρμογής. Τα artifacts είναι φυσικά συστατικά του συστήματος που υπάρχουν σε επίπεδο υλοποίησης. Τα διαγράμματα συστατικών δείχνουν τύπους συστατικών και όχι τα διάφορα στιγμιότυπά τους τα οποία φαίνονται στα διαγράμματα ανάπτυξης.



Σχήμα 7.3 Παράδειγμα διαγράμματος συστατικών.

7.5 ΔΙΑΓΡΑΜΜΑΤΑ ΑΝΑΠΤΥΞΗΣ

7.5.1 Ορισμός

Τα διαγράμματα ανάπτυξης παρουσιάζουν τη σύνθεση των επεξεργαστικών στοιχείων κατά το χρόνο εκτέλεσης και τις συνιστώσες λογισμικού, τις διεργασίες και τα αντικείμενα που ζουν πάνω από αυτά. Τα στιγμιότυπα των συνιστωσών λογισμικού αναπαριστούν εμφανίσεις των μονάδων του κώδικα κατά το χρόνο εκτέλεσης. Οι συνιστώσες που δεν υφίστανται ως οντότητες του χρόνου εκτέλεσης (γιατί έφυγαν λόγω της μεταγλώττισης) δεν εμφανίζονται σε αυτά τα διαγράμματα και θα πρέπει να εμφανίζονται στα διαγράμματα συνιστωσών.

Για μοντελοποίηση εταιριών, τα επεξεργαστικά στοιχεία του χρόνου εκτέλεσης περιλαμβάνουν εργάτες και οργανωτικές μονάδες, ενώ οι συνιστώσες λογισμικού περιλαμβάνουν διαδικασίες και κείμενα που χρησιμοποιούνται από τους εργάτες και τις οργανωτικές μονάδες.

Σε ένα διάγραμμα ανάπτυξης παρουσιάζονται οι υπολογιστές και οι συσκευές (κόμβοι) σε ένα σύστημα καθώς και ο τύπος των συνδέσεων που χρησιμοποιούνται ώστε να επικοινωνούν τα παραπάνω μεταξύ τους. Τα εκτελέσιμα αντικείμενα φανερώνουν ποιες μονάδες λογισμικού εκτελούνται σε κάθε κόμβο. Μπορούν να χρησιμοποιηθούν για να δείξουν ποια συστατικά τρέχουν σε ποιους κόμβους. Ένας κόμβος ή αλλιώς node όπως είπαμε παραπάνω, είναι ένα φυσικό αντικείμενο που στην γενική περίπτωση έχει τουλάχιστον μνήμη και δυνατότητα επεξεργασίας.

Συνοψίζοντας μπορούμε να πούμε ότι ένα διάγραμμα ανάπτυξης αναπαριστά τα παρακάτω:

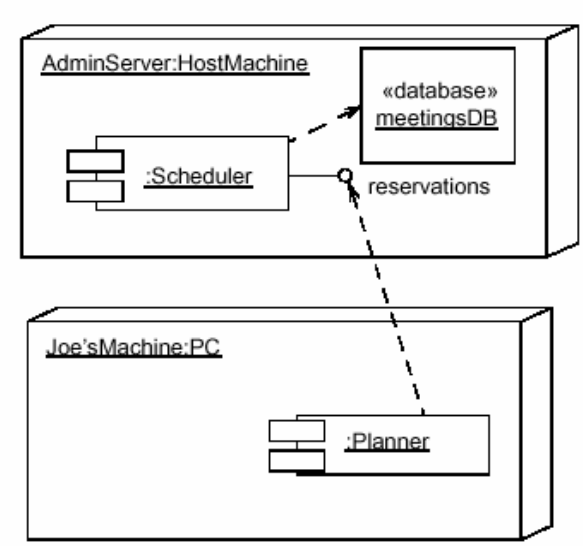
1. Την φυσική υλοποίηση του λογισμικού.
2. Ποια τμήματα του λογισμικού (software) τρέχουν σε ποια τμήματα του υλικού (hardware).
3. Πως μπορούν να επικοινωνούν οι κόμβοι.

7.5.2 Βασικά Στοιχεία και Συμβολισμός τους

Ένα διάγραμμα ανάπτυξης είναι ένας γράφος κόμβων οι οποίοι είναι συνδεδεμένοι με συσχετίσεις επικοινωνίας. Οι κόμβοι μπορούν να περιέχουν στιγμιότυπα συνιστωσών. Αυτό δηλώνει ότι οι συνιστώσες ζουν ή εκτελούνται στον κόμβο. Οι συνιστώσες μπορούν να περιέχουν αντικείμενα, αυτό δηλώνει ότι το αντικείμενο βρίσκεται μέσα στη συνιστώσα. Οι συνιστώσες είναι συνδεδεμένες με άλλες συνιστώσες με εξαρτήσεις διακεκομμένων βελών (πιθανά μέσω διασυνδέσεων). Αυτό δηλώνει ότι μία συνιστώσα χρησιμοποιεί τις υπηρεσίες άλλων συνιστωσών. Εφόσον χρειάζεται, μπορεί να χρησιμοποιηθεί ένα στιγμιότυπο για να δηλώσει την ακριβή εξάρτηση.

Το διάγραμμα ανάπτυξης μπορεί επίσης να χρησιμοποιηθεί για να δείξει ποιες συνιστώσες μπορούν να βρίσκονται σε ποιους κόμβους, χρησιμοποιώντας διακεκομμένα βέλη με το στερεότυπο «support» από το σύμβολο της συνιστώσας στο σύμβολο του κόμβου ή γραφικά εμφανίζοντας το σύμβολο της συνιστώσας ένθετο στο σύμβολο του κόμβου.

Η μετανάστευση των στιγμιότυπων των συνιστωσών από έναν κόμβο σε έναν άλλον ή των αντικειμένων από ένα στιγμιότυπο συνιστώσας σε ένα άλλο, μπορεί να φανεί με το στερεότυπο «become» στη σχέση εξάρτησης. Στην περίπτωση αυτή το στιγμιότυπο της συνιστώσας ή το αντικείμενο βρίσκεται στο στιγμιότυπο του κόμβου ή στο στιγμιότυπο της συνιστώσας μόνο για ένα μέρος του όλου χρόνου. Να σημειώσετε ότι οι διαδικασίες είναι ένα ειδικό είδος των αντικειμένων.



Σχήμα 7.4 Παράδειγμα διαγράμματος ανάπτυξης.

7.6 Συνοψίζοντας

Σε αυτό το κεφάλαιο ασχοληθήκαμε με τα φυσικά διαγράμματα τα οποία μοντελοποιούν την φυσική αρχιτεκτονική των συστημάτων. Όπως είδαμε τα διαγράμματα συστατικών και ανάπτυξης ανήκουν σε αυτή την κατηγορία διαγραμμάτων. Στα πρώτα παρουσιάσαμε τα τρία δομικά στοιχεία τους που είναι οι συνιστώσες, οι εξαρτήσεις και η διεπαφή και πως χρησιμοποιώντας τα μπορούν να παρουσιάσουν το πλήθος των εξαρτήσεων ανάμεσα στις συνιστώσες λογισμικού. Στην συνέχεια στα διαγράμματα ανάπτυξης είδαμε τον σκοπό που τα χρησιμοποιούμε δηλαδή για να αναπαραστήσουμε την φυσική υλοποίηση του λογισμικού, να ξεχωρίσουμε ποια τμήματα λογισμικού εκτελούνται σε ποια τμήματα υλικού και τέλος πως μπορούν να επικοινωνούν οι κόμβοι μεταξύ τους. Στο τελευταίο κεφάλαιο θα ασχοληθούμε με τη μεθοδολογία CASE, θα δούμε τι ακριβώς είναι και τι εργαλεία χρησιμοποιεί ώστε να υλοποιηθεί.

ΚΕΦΑΛΑΙΟ 8. ΜΕΘΟΔΟΛΟΓΙΑ CASE

8.1 Εισαγωγή

Ο όρος Μηχανική Λογισμικού με την Βοήθεια Υπολογιστή ή αλλιώς CASE (Computer-Aided Software Engineering), στον τομέα της Μηχανικής Λογισμικού που αναφερθήκαμε στο πρώτο κεφάλαιο, είναι η επιστημονική εφαρμογή ενός συνόλου εργαλείων και μεθόδων σε ένα σύστημα λογισμικού με σκοπό την παραγωγή υψηλής ποιότητας, συντηρήσιμων και χωρίς ελαττώματα προϊόντων λογισμικού. Αναφέρεται επίσης, στις μεθόδους που χρησιμοποιούνται για την ανάπτυξη πληροφοριακών συστημάτων μαζί με τα αυτοματοποιημένα εργαλεία που μπορούν να χρησιμοποιηθούν στην διαδικασία ανάπτυξης λογισμικού.

8.2 Επισκόπηση

Ο όρος CASE μπορεί να αναφέρεται στο λογισμικό που χρησιμοποιείται για την αυτοματοποιημένη ανάπτυξη λογισμικού συστημάτων, για παράδειγμα πηγαίου κώδικα. Οι λειτουργίες CASE αποτελούνται από την ανάλυση, τον σχεδιασμό και τον προγραμματισμό. Τα εργαλεία CASE αυτοματοποιούν τις μεθόδους για τον σχεδιασμό, την τεκμηρίωση και παραγωγή δομημένου πηγαίου κώδικα υπολογιστών στην επιθυμητή γλώσσα προγραμματισμού.

Οι δύο βασικές ιδέες που κρύβονται πίσω από την CASE είναι:

- Η υποβοήθηση μου μπορούν να προσφέρουν οι υπολογιστές στην ανάπτυξη λογισμικού αλλά και στις διαδικασίες συντήρησης λογισμικού και
- Μια προσέγγιση εφαρμοσμένης μηχανικής στην ανάπτυξη λογισμικού και την συντήρησή του.

Μερικά χαρακτηριστικά εργαλεία CASE είναι:

- Εργαλεία διαμόρφωσης (configuration).
- Εργαλεία σχεδίασης μοντέλου δεδομένων (data modeling).
- Εργαλεία μετασχηματισμού μοντέλων (model transformation).
- Εργαλεία μετασχηματισμού προγράμματος (program transformation).
- Εργαλεία επανα-παραγοντοποιήσης (refactoring tools)
- Εργαλεία παραγωγής πηγαίου κώδικα (source code generation) και
- Η Ενοποιημένη γλώσσα Σχεδίασης (UML).

Πολλά εργαλεία CASE δεν έχουν ως έξοδο μόνο κώδικα αλλά επίσης, παράγουν και άλλα χαρακτηριστικά διάφορων τεχνικών ανάλυσης και σχεδιασμού συστημάτων όπως:

- Διαγράμματα ροής Δεδομένων.
- Διαγράμματα Σχέσεων Οντοτήτων.
- Λογικά Διαγράμματα.
- Προδιαγραφές Προγράμματος.

- Εγχειρίδιο για τους χρήστες.

8.3 Ιστορία

Το πρότζεκτ ISDOS του πανεπιστημίου του Μίσιγκαν άρχισε με πολύ ενδιαφέρον στην όλη ιδέα της χρησιμοποίησης υπολογιστικών συστημάτων για την υποστήριξη των αναλυτών συστημάτων, στην δύσκολη διαδικασία της ανάλυσης των απαιτήσεων και στην ανάπτυξη συστημάτων γενικά. Διάφορες δημοσιεύσεις από τον Daniel Teichroew ήταν το έναυσμα για να γεννηθεί μια γενιά ανθρώπων οι οποίοι ενθουσιάστηκαν με την ιδέα της αυτοματοποιημένης ανάπτυξης συστημάτων. Το εργαλείο του Teichroew PSL/PSA ήταν ένα εργαλείο CASE αν και προηγήθηκε χρονικώς του όρου. Οι ιδέες του στη δύναμη των μοντέλων ήταν ενθαρρυντικές.

Ένα άλλο σημαντικό βήμα προέκυψε ως λογική επέκταση στον κατάλογο DBMS (Database Management System - πρόγραμμα διαχείρισης βάσεων δεδομένων). Με την αύξηση του πλήθους των μεταδεδομένων που αποθηκεύονταν, οι ιδιότητες μιας εφαρμογής θα μπορούσαν να κρατηθούν μέσα σε ένα λεξικό και να χρησιμοποιηθούν στο χρόνο εκτέλεσης. Αυτό το "ενεργό λεξικό" έγινε ο πρόδρομος στην πιο σύγχρονη "εκτέλεση προσανατολισμένη προς το μοντέλο" (MDE). Εντούτοις, το λεξικό δεν παρείχε μια γραφική απεικόνιση από κανένα από τα μεταδεδομένα.

Ο όρος CASE αρχικά χρησιμοποιήθηκε από την εταιρία Nastec, μιας επιχείρησης λογισμικού στο Μίσιγκαν το 1982, μαζί με τον ενσωματωμένο με γραφικά συντάκτη κειμένου GraphiText, ο οποίος ήταν επίσης το πρώτο σύστημα βασισμένο σε υπολογιστή που χρησιμοποίησε τους συνδέσμους υπέρ-κειμένου για να παραπέμψει τις ομάδες λέξεων σε έγγραφα - ένας πρώτος πρόδρομος στις σημερινές ιστοσελίδες. Το DesignAid, ο διάδοχος του GraphiText, ήταν το πρώτο βασισμένο σε υπολογιστή εργαλείο που αξιολογούσε τα διαγράμματα σχεδίασης λογισμικού και συστημάτων και έχτιζε ένα λεξικό δεδομένων.

Τα εργαλεία CASE ήταν στην αιχμή τους στις αρχές της δεκαετίας του '90. Τότε η IBM είχε προτείνει την AD/Cycle, η οποίος ήταν μια συμμαχία των προμηθευτών λογισμικού που εκμεταλλεύονταν την αποθήκη λογισμικού της IBM και χρησιμοποιούσαν την IBM DB2 στον κεντρικό υπολογιστή τους και το OS/2.

8.4 Κατηγορίες εργαλείων CASE

Ο Alfonso Fuggetta ταξινόμησε τα εργαλεία CASE σε 3 κατηγορίες:

1. *Τα εργαλεία (tools)* που υποστηρίζουν μόνο συγκεκριμένες διεργασίες στην ανάπτυξη λογισμικού.
2. *Τους μικρούς χώρους εργασίας (workbenches)* που υποστηρίζουν μόνο μία ή λίγες διεργασίες.
3. *Τα περιβάλλοντα εργασίας (environments)* που υποστηρίζουν το μεγαλύτερο μέρος την όλης διαδικασίας ανάπτυξης λογισμικού.

Τα δύο τελευταία αποτελούνται γενικά από συλλογές εργαλείων. Τα εργαλεία από την άλλη μπορεί να είναι είτε standalone εφαρμογές είτε συστατικά των χώρων και περιβαλλόντων εργασίας.

8.4.1 Εργαλεία

Τα εργαλεία CASE είναι μια κατηγορία λογισμικού που αυτοματοποιεί πολλές από τις δραστηριότητες που εμπλέκονται στα διάφορα στάδια του κύκλου ζωής του λογισμικού. Παραδείγματος χάριν, κατά τον ορισμό των λειτουργικών απαιτήσεων μιας προτεινόμενης εφαρμογής, τα εργαλεία διαμόρφωσης πρωτοτύπου μπορούν να χρησιμοποιηθούν για να αναπτύξουν γραφικά μοντέλα του πως θα φαίνεται η εφαρμογή στο τέλος, για να βοηθήσουν τους τελικούς χρήστες να σχηματίσουν μια εικόνα για το πώς θα φαίνεται η εφαρμογή μετά την ανάπτυξή της. Στη συνέχεια, οι σχεδιαστές συστημάτων μπορούν να χρησιμοποιήσουν τα αυτοματοποιημένα εργαλεία σχεδιασμού για να μετασχηματίσουν τις πρότυπες λειτουργικές απαιτήσεις στα λεπτομερή έγγραφα σχεδιασμού. Οι προγραμματιστές μπορούν έπειτα να χρησιμοποιήσουν τις αυτοματοποιημένες γεννήτριες κώδικα για να μετατρέψουν τα έγγραφα σχεδιασμού σε κώδικα. Τα αυτοματοποιημένα εργαλεία μπορούν να χρησιμοποιηθούν είτε συλλογικά είτε μεμονωμένα.

Τα υπάρχοντα εργαλεία CASE που υπάγονται σε αυτή την κατηγορία μπορούν να κατηγοριοποιηθούν βάση τεσσάρων διαφορετικών παραγόντων:

- Υποστήριξης Κύκλου Ζωής
- Ενσωμάτωσης (integration)
- Κατασκευής (construction)
- Γνωστικής Βάσης CASE (knowledge base)

8.4.2 Χώροι Εργασίας

Οι χώροι εργασίας ενσωματώνουν διάφορα εργαλεία CASE σε μια εφαρμογή για να υποστηριχθούν οι συγκεκριμένες δραστηριότητες λογισμικού-διαδικασίας. Ως εκ τούτου επιτυγχάνουν:

- μια ομοιογενή και συνεπή διεπαφή (interface).
- εύκολη κλήση των εργαλείων και των αλυσίδων εργαλείων (ένταξη ελέγχου).
- πρόσβαση σε ένα ομοιογενές σύνολο στοιχείων διοικούμενο με έναν συγκεντρωτικό κοινό τρόπο (ένταξη δεδομένων).

Οι χώροι εργασίας CASE μπορούν να ταξινομηθούν περαιτέρω στις ακόλουθες 8 κατηγορίες:

1. Επιχειρησιακού προγραμματισμού και σχεδιασμού
2. Ανάλυσης και σχεδίασης
3. Ανάπτυξης διεπαφής χρήστη
4. Προγραμματισμού
5. Επαλήθευσης και επικύρωσης
6. Συντήρησης και Αντίστροφης Μηχανικής

7. Διαχείρισης διαμόρφωσης (configuration management)
8. Διαχείρισης του σχεδίου (project management)

8.4.3 Περιβάλλοντα Εργασίας

Ένα περιβάλλον εργασίας είναι μια συλλογή από εργαλεία και χώρους εργασίας CASE που υποστηρίζει τη ανάπτυξη λογισμικού. Τα περιβάλλοντα CASE είναι ομαδοποιημένα βάση της ενοποίησης ως εξής:

1. Εργαλειοθήκες (toolkits)
2. Επικεντρωμένα στην Γλώσσα Προγραμματισμού (language centered)
3. Ενοποιημένα (integrated)
4. Τέταρτης Γενιάς (forth generation)
5. Επικεντρωμένα στην Διεργασία (process centered)

Εργαλειοθήκες

Οι εργαλειοθήκες είναι συλλογές προϊόντων που μπορούν εύκολα να επεκτείνονται με την άθροιση διαφορετικών εργαλείων και χώρων εργασίας. Συνήθως η υποστήριξη που παρέχεται από μία εργαλειοθήκη περιορίζεται στον προγραμματισμό, τη διαχείριση διαμόρφωσης και τη διαχείριση του έργου. Επιπλέον, η χαλαρή συνοχή των εργαλειοθηκών απαιτεί από το χρήστη να ενεργοποιήσει τα εργαλεία της με κλήση ή άλλους απλούς μηχανισμούς ελέγχου.

Επικεντρωμένα στην Γλώσσα Προγραμματισμού

Το ίδιο το περιβάλλον συντάσσεται στη ίδια γλώσσα προγραμματισμού με αυτήν για την οποία αναπτύχθηκε, κατά συνέπεια επιτρέπει στους χρήστες να το επαναχρησιμοποιήσουν, να το προσαρμόσουν και να το επεκτείνουν βάση των απαιτήσεων τους. Η ενσωμάτωση του κώδικα στις διάφορες γλώσσες είναι ένα σημαντικό ζήτημα για τα περιβάλλοντα που είναι επικεντρωμένα στην Γλώσσα προγραμματισμού. Η απουσία της διεργασίας και των δεδομένων είναι ένα σημαντικό πρόβλημα . Από την άλλη τα δυνατά σημεία αυτού του είδους περιβαλλόντων είναι το καλό επίπεδο παρουσίασης και οι διαδικασίες ελέγχου. Η Interlisp και η Smalltalk, είναι δύο χαρακτηριστικά παραδείγματα αυτών των περιβαλλόντων.

Ενοποιημένα

Αυτό το περιβάλλον επιτυγχάνει την ενσωμάτωση παρουσίασης με το να παρέχει ομοιόμορφες και συνεπείς διεπαφές στα εργαλεία και τους χώρους εργασίας. Η ενσωμάτωση δεδομένων επιτυγχάνεται μέσω της έννοιας των αποθηκών (repository). Οι αποθήκες είναι εξειδικευμένες βάσεις δεδομένων που διαχειρίζονται όλες τις πληροφορίες που παράγονται και που χρησιμοποιούνται, στο περιβάλλον. Παραδείγματα ενοποιημένων περιβαλλόντων είναι το AD/Cycle της IBM και το Cohesion της DEC.

Τέταρτης Γενιάς

Τα περιβάλλοντα τέταρτης γενιάς ήταν τα πρώτα ενοποιημένα περιβάλλοντα. Είναι σύνολα εργαλείων και χώρων εργασίας που υποστηρίζουν την ανάπτυξη μιας συγκεκριμένης κατηγορίας προγραμμάτων: ηλεκτρονική επεξεργασία δεδομένων και εφαρμογές προσανατολισμένες στις επιχειρησιακές διαδικασίες. Γενικά, περιλαμβάνουν εργαλεία προγραμματισμού, απλά διοικητικά εργαλεία διαμόρφωσης, εγκαταστάσεις που διαχειρίζονται έγγραφα και μερικές φορές, μια γεννήτρια πηγαίου κώδικα για να παραγάγουν κώδικα για γλώσσες χαμηλότερου επιπέδου. Παραδείγματα που εντάσσονται σε αυτήν την κατηγορία είναι η Informix 4GL της IBM και η Focus της Information Builders.

Επικεντρωμένα στη Διεργασία

Τα περιβάλλοντα σε αυτήν την κατηγορία εστιάζουν στην ολοκλήρωση της διεργασίας χρησιμοποιώντας ως αφετηρία άλλους παράγοντες (π.χ. τις εργαλειοθήκες) και όχι αυτή κάθε αυτή την διεργασία. Ένα περιβάλλον επικεντρωμένο στην διεργασία λειτουργεί με την μετάφραση ενός μοντέλου διεργασίας που δημιουργείται από τα πιο εξειδικευμένα εργαλεία. Αυτά αποτελούνται συνήθως από εργαλεία που χειρίζονται δύο λειτουργίες:

- Εκτέλεση Μοντέλου Διεργασίας
- Παραγωγή Μοντέλου Διεργασίας

8.5 Εφαρμογές των Εργαλείων CASE

Όλες οι πτυχές του κύκλου ζωής της ανάπτυξης λογισμικού μπορούν να υποστηριχθούν από εργαλεία λογισμικού και κατ' επέκταση από εργαλεία όλων των φασμάτων. Από λογισμικό διαχείρισης έργων μέχρι εργαλεία για την επιχειρησιακή και λειτουργική ανάλυση, για το σχεδιασμό συστημάτων, για την αποθήκευση κώδικα, μεταγλωττιστές, λογισμικό ελέγχων και πολλά άλλα.

Εντούτοις, τα εργαλεία που ασχολούνται με την ανάλυση και τη σχεδίαση, και μπορούν να χρησιμοποιηθούν για να δημιουργήσουν το μέρος (ή το όλο) ενός προϊόντος λογισμικού, θεωρούνται πολύ συχνά ως εργαλεία CASE. Τα εργαλεία CASE που εφαρμόστηκαν, παραδείγματος χάριν, σε ένα προϊόν λογισμικού βάσεων δεδομένων, φυσιολογικά θα περιλαμβάνουν:

- Διαδικασία Μοντελοποίησης / Ροή Δεδομένων.
- Ανάπτυξη μοντέλων δεδομένων με την μορφή διαγραμμάτων οντοτήτων και συσχετίσεων.
- Ανάπτυξη διεργασιών και λειτουργικών προδιαγραφών.
- Παραγωγή SQL από βάση δεδομένων και αποθηκευμένες διεργασίες.

8.6 Κίνδυνοι και διαδικασίες Ελέγχου

Η χρήση εργαλείων CASE υποκρύπτει κάποιους κινδύνους για αυτό το λόγο έχουν αναπτυχθεί σχετικές διαδικασίες ελέγχου. Οι σημαντικότεροι κίνδυνοι και οι αντίστοιχες διαδικασίες ελέγχου είναι:

- *Ανεπαρκής τυποποίηση*: Το να συνδυάζονται εργαλεία CASE από διαφορετικούς προμηθευτές (π.χ. το εργαλείο σχεδίασης από την εταιρία Α και το εργαλείο προγραμματισμού από την εταιρία Β) μπορεί να είναι δύσκολο εάν τα προϊόντα δεν χρησιμοποιούν τυποποιημένες δομές κώδικα και κατηγοριοποίηση δεδομένων. Αν και οι τύποι αρχείων μπορούν να μετατραπούν αυτό δεν συμφέρει οικονομικά. Οι διαδικασίες έλεγχου περιλαμβάνουν τη χρήση εργαλείων από τον ίδιο προμηθευτή ή τη χρήση εργαλείων βασισμένων σε τυποποιημένα πρωτόκολλα τα οποία εμμένουν στην συμβατότητα. Επιπλέον, εάν οι επιχειρήσεις χρησιμοποιούν εργαλεία μόνο για ένα τμήμα της διαδικασίας ανάπτυξης, πρέπει να λάβουν υπόψη τους να διαλέξουν έναν προμηθευτή που έχει πλήρη σειρά προϊόντων και εργαλείων, ώστε έτσι να εξασφαλιστεί μελλοντική συμβατότητα εάν προσθέτουν περισσότερα εργαλεία.

- *Μη ρεαλιστικές απαιτήσεις και προσδοκίες*: Οι εταιρίες εφαρμόζουν συχνά τις τεχνολογίες CASE για να μειώσουν τα έξοδα ανάπτυξης του λογισμικού. Στην πραγματικότητα η εφαρμογή των στρατηγικών CASE περιλαμβάνει συνήθως υψηλές αρχικές δαπάνες. Θα πρέπει οι εταιρίες να είναι πρόθυμες να δεχτούν ένα μακροπρόθεσμο σχέδιο απόδοσης. Οι έλεγχοι εδώ περιλαμβάνουν την εκμετάλλευση των ανώτερων στελεχών των επιχειρήσεων για να καθορίσουν το σκοπό και τις στρατηγικές που θα ακολουθήσουν για την ενσωμάτωση των τεχνολογιών CASE.

- *Βιαστική εφαρμογή*: Η ενσωμάτωση τεχνολογιών CASE μπορεί να προκαλέσει σημαντική αλλαγή στον κύκλο ανάπτυξης λογισμικού σε σχέση με τα παραδοσιακά περιβάλλοντα ανάπτυξης. Συνήθως, οι εταιρίες δεν θα πρέπει να χρησιμοποιούν εργαλεία CASE για πρώτη φορά σε κρίσιμα έργα ή σε έργα με μικρές χρονικές προθεσμίες, λόγω της εκτενούς εκπαίδευσης που χρειάζεται κάποιος για να τα χρησιμοποιήσει. Αντίθετα, οι επιχειρήσεις θα πρέπει να αναλογιστούν την χρήση εργαλείων CASE σε μικρότερα και λιγότερο σύνθετα έργα και έτσι να γίνει βαθμιαία η κατάρτισή τους σε αυτά

- *Αδυναμία ελέγχου αποθηκών*: Η αποτυχία να ελεγχθεί επαρκώς η πρόσβαση στις αποθήκες CASE μπορεί να οδηγήσει στην παραβίαση της ασφάλειάς τους ή ακόμα και στην φθορά των εγγράφων εργασίας, των σχεδίων συστήματος ή ακόμα και των προτύπων κώδικα που αποθηκεύονται σε αυτή. Οι έλεγχοι περιλαμβάνουν την προστασία αυτών των αποθηκών με κατάλληλους ελέγχους πρόσβασης αλλά και τακτική λήψη αντιγράφων ασφαλείας.

8.7 Συνοψίζοντας

Σε αυτό το κεφάλαιο μιλήσαμε για τη μεθοδολογία CASE και τα εργαλεία που χρησιμοποιούνται για να την υλοποιήσουμε. Είδαμε τις κατηγορίες εργαλείων CASE όπως τους χώρους εργασίας καθώς και τα μεγαλύτερα περιβάλλοντα εργασίας. Μιλήσαμε για το που μπορούν να βρουν εφαρμογή αυτά τα εργαλεία και για τους κινδύνους που μπορεί να προκύψουν όταν θα έρθει η ώρα να χρησιμοποιήσουμε τα εργαλεία της CASE. Τέλος, είδαμε και τις διαδικασίες ελέγχου που πρέπει να ακολουθήσουμε για να αντιμετωπίσουμε τους κινδύνους αυτούς. Οι εταιρίες και οι επιχειρήσεις ωφελούνται από την χρήση των εργαλείων CASE γιατί πρώτον αυξάνουν την παραγωγικότητα με την αυτοματοποίηση διαδικασιών ρουτίνας, δεύτερον βελτιώνουν την όλη ποιότητα του παραγόμενου λογισμικού με την προϋπόθεση ότι γίνεται σωστή χρήση των τεχνικών και έτσι περιορίζονται σημαντικά τα λάθη και τέλος έχουν ως αποτέλεσμα την μείωση ανάγκης συντήρησης του συστήματος δίνοντάς μας έτσι χρόνο για ανάπτυξη νέων συστημάτων. Δεν πρέπει να ξεχνάμε ότι η επιτυχία εφαρμογής ενός εργαλείου CASE στην ανάπτυξη ενός προϊόντος λογισμικού εξαρτάται από την σωστή χρήση της μεθοδολογίας.

ΕΠΙΛΟΓΟΣ

Από όλα τα προηγούμενα κεφάλαιο μπορούμε να βγάλουμε το συμπέρασμα ότι η UML αποτελεί απαραίτητο εφόδιο για κάποιον που απλά θέλει να σχεδιάσει την μικρή επιχείρησή του μέχρι και για τον επαγγελματία σχεδιαστή σύνθετων πληροφοριακών συστημάτων. Η γλώσσα της UML δεν περιορίζεται στο χώρο της πληροφορικής αφού είναι σχεδιασμένη έτσι ώστε να είναι ανεξάρτητη από διαδικασίες ανάπτυξης και μπορεί να χρησιμοποιηθεί για την μοντελοποίηση επιχειρηματικών διαδικασιών και συνεπώς η χρηστικότητά της επεκτείνεται και στον χώρο διοίκησης επιχειρήσεων.

Η UML μπορεί να αποδειχτεί ένα εξαιρετικό εργαλείο ανταλλαγής ιδεών για τους μηχανικούς πληροφορικής. Η δύναμη της UML δεν είναι η δυνατότητα σχεδίασης απλών τετραγώνων και βελών αλλά ο σχεδιασμός συγκεκριμένων μορφών σχημάτων και τύπων βελών. Εάν δύο μηχανικοί ξέρουν να χρησιμοποιούν για παράδειγμα τα διαγράμματα κλάσεων της UML και ο ένας από τους δύο ξεκινήσει να σχεδιάζει μια ιεραρχία κλάσεων, τότε ο άλλος ξέρει αμέσως ποια κλάση κληρονομεί από την άλλη. Αυτές οι συμβάσεις των μοντέλων και των συμβόλων που χρησιμοποιούνται επιτρέπουν στους μηχανικούς που έχουν τη γνώση αυτών των συμβάσεων να επικοινωνούν γρήγορα και αποτελεσματικά. Το μέλλον της UML φαίνεται να είναι λαμπρό εφόσον τα εργαλεία που αναπτύσσονται για αυτήν χρειάζεται να είναι σταθερά, γρήγορα, προσιτά οικονομικά και φιλικά στον χρήστη ώστε να μπορούν εύκολα να υιοθετηθούν.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Arlow,J. Neustadt, I. (2006). UML 2 And The Unified Process: Practical Object-Oriented Analysis And Design. 2nd Edition. Addison-Wesley

- Booch, G. Jacobson, I. Rumbaugh, J. (1999). The Unified Software Development Process. Addison-Wesley

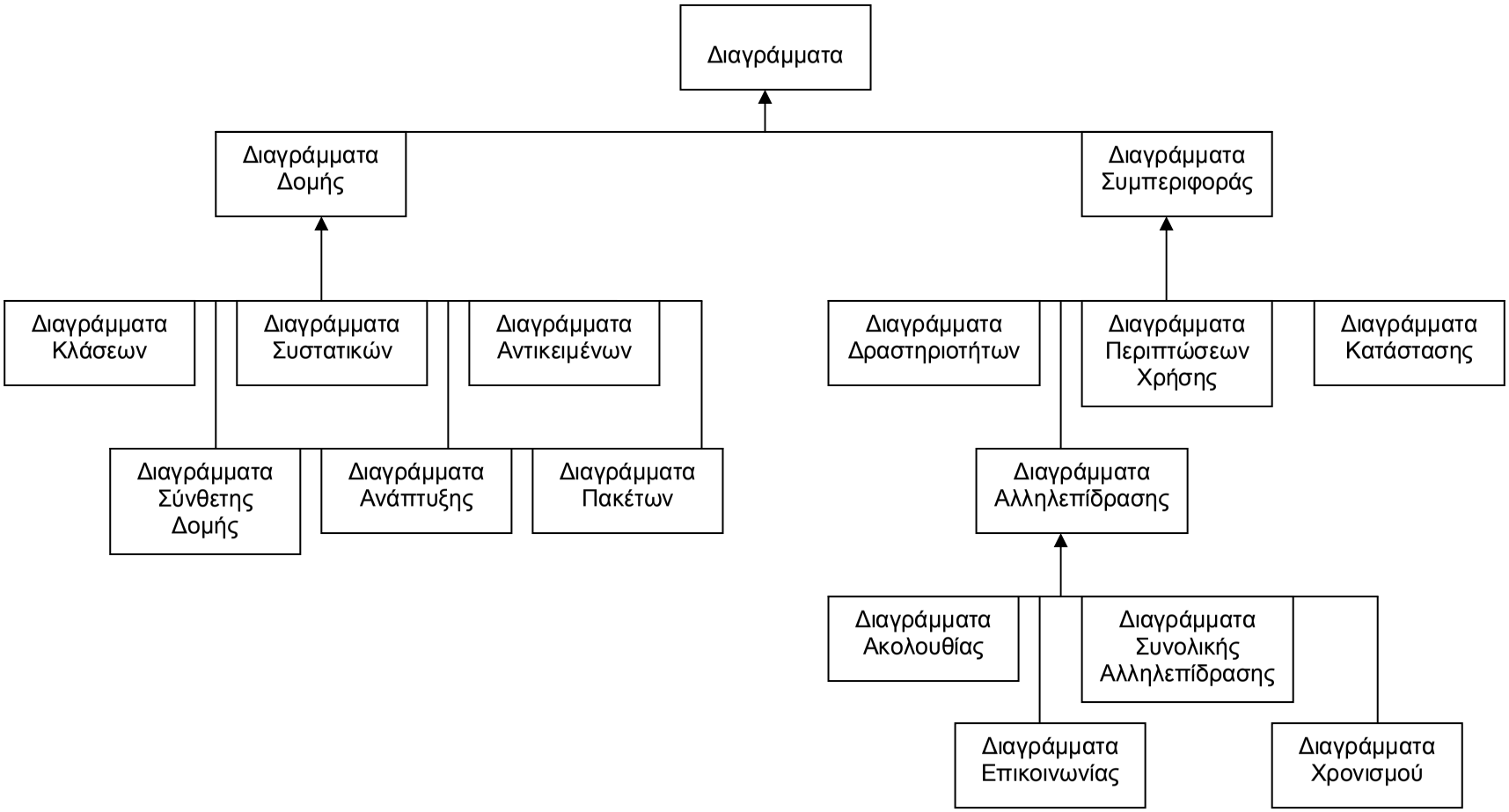
- Fowler, M. (2003). UML Distilled A Brief Guide To the Standard Object
- Modeling Language. 3rd Edition. Addison-Wesley

- Kental, S. (2001). The Unified Process Explained. Addison-Wesley

- Larman, C. (2002). Applying UML And Patterns, An Introduction To Object-
- Oriented Analysis And Design And The Unified Process. 2nd edition. NJ:
- Prentice Hall PTR

- Internet: <http://www.omg.org>
- Internet: <http://en.wikipedia.org/wiki/Sequence_diagram>
- Internet: <http://www.agilemodeling.com/artifacts/classDiagram.htm>
- Internet: <http://www.developer.com/design/article.php/1553851/UML-Overview.htm>

ΠΑΡΑΡΤΗΜΑ


Διαγράμματα
Διαγράμματα Δομής
Διαγράμματα Συμπεριφοράς
Διαγράμματα Κλάσεων
Διαγράμματα Συστατικών
Διαγράμματα Αντικειμένων
Διαγράμματα Σύνθετης Δομής
Διαγράμματα Ανάπτυξης
Διαγράμματα Πακέτων
Διαγράμματα Δραστηριοτήτων
Διαγράμματα Περιπτώσεων Χρήσης
Διαγράμματα Κατάστασης
Διαγράμματα Αλληλεπίδρασης
Διαγράμματα Ακολουθίας
Διαγράμματα Συνολικής Αλληλεπίδρασης
Διαγράμματα Επικοινωνίας
Διαγράμματα Χρονισμού