Ατομική Διπλωματική Εργασία

ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

Σκεύη Μιχαήλ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Συστήματα πραγματικού χρόνου

Σκεύη Μιχαήλ

Επιβλέπων Καθηγητής
Γιώργος Παπαδόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2014

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κύριο Γιώργο Παπαδόπουλο που μου έδωσε την ευκαιρία να ασχοληθώ με αυτό το ιδιαίτερα ενδιαφέρον αντικείμενο μελέτης καθώς και για την πολύτιμη καθοδήγηση του καθ' όλη την πορεία της διπλωματικής μου εργασίας.

Τέλος, το πιο μεγάλο ευχαριστώ το οφείλω στην οικογένεια και τους φίλους μου που με στήριξαν όλα τα χρόνια της φοίτησης μου στο Πανεπιστήμιο Κύπρου. Ειδικά, χωρίς τη βοήθεια, την αγάπη και υποστήριξη των γονιών και παππούδων μου δεν θα ξεπερνούσα τα εμπόδια που συνάντησα στο δρόμο μου τη δύσκολη περίοδο των σπουδών μου.

Περίληψη

Η διπλωματική εργασία αυτή ασχολείται με τα συστήματα πραγματικού χρόνου (real- time systems). Στα συστήματα αυτά, όπως δηλώνει και το όνομα τους έχουμε την έννοια του χρόνου που τα διαφοροποιεί από τα υπόλοιπα συστήματα. Ο χρόνος έχει κρίσιμη σημασία στα εν λόγω συστήματα. Αυτό σημαίνει ότι ένα τέτοιο σύστημα πρέπει να ολοκληρώνει το έργο του και να παράγει τα αποτελέσματα του μέσα σε καθορισμένα χρονικά όρια.

Τα τελευταία χρόνια έχουν γίνει πολλές μελέτες στον τομέα αυτό και πλέον αναπτύσσονται όλο και περισσότερα συστήματα πραγματικού χρόνου. Στην παρούσα διπλωματική εργασία εξετάσαμε πως η UML (Unified Modeling Language – Γλώσσα Ενοποιημένης Μοντελοποίησης) μπορεί να συνεισφέρει στην μοντελοποίηση των ιδιαιτεροτήτων που έχουν τα συστήματα πραγματικού χρόνου. Επίσης, ασχοληθήκαμε με δύο συγκεκριμένα συστήματα, το σύστημα ελέγχου ανελκυστήρα και το σύστημα video player. Αυτά τα συστήματα τα μελετήσαμε και μοντελοποιήσαμε τη συμπεριφορά τους χρησιμοποιώντας, όπως αναφέραμε πιο πάνω, την δημοφιλέστερη γλώσσα μοντελοποίησης συστημάτων UML.

Τέλος, αναπτύξαμε κάποιο κώδικα με την γλώσσα προγραμματισμού Java (για το σύστημα ελέγχου ανελκυστήρα) για να εξετάσουμε πάλι κάποιες διαφορές που υπάρχουν στον κώδικα των συστημάτων πραγματικού χρόνου ο οποίος διαφέρει σε μεγάλο βαθμό από τα υπόλοιπα κοινά συστήματα.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γενικές Πληροφορίες

Αντικείμενο μελέτης της παρούσας διπλωματικής εργασίας είναι τα συστήματα πραγματικού χρόνου. Τα τελευταία χρόνια έχουν γίνει πολλές μελέτες στο πεδίο αυτό. Υπάρχει ένα ευρύ φάσμα εφαρμογών των real-time systems. Κύριο χαρακτηριστικό τους, όπως εξάλλου δηλώνει και το όνομα τους, είναι ο χρόνος. Αυτή η έννοια του χρόνου κάνει τα συστήματα πραγματικού χρόνου να διαφέρουν από τα υπόλοιπα υπολογιστικά συστήματα. Στα εν λόγω συστήματα, δεν μας φτάνει η παραγωγή ορθών και ικανοποιητικών αποτελεσμάτων. Μας ενδιαφέρει αυτά τα αποτελέσματα να εξάγονται μέσα σε προκαθορισμένα χρονικά περιθώρια. Στις μέρες μας, οι εφαρμογές των real- time συστημάτων καλύπτουν ένα πολύ μεγάλο πεδίο. Τα συστήματα πραγματικού χρόνου βρίσκουν εφαρμογή στα πολυμέσα, για παράδειγμα αναπαραγωγή εικόνας και ήχου. Επίσης βρίσκουν εφαρμογή στην τηλεφωνία, στην βιομηχανία, στις τραπεζικές συναλλαγές, στη ιατρική.

1.2 Σκοπός Διπλωματικής Εργασίας

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η μελέτη των συστημάτων πραγματικού χρόνου. Να εξετάσουμε τις ιδιότητες που έχουν αυτά τα συστήματα και πως

διαφέρουν από τα υπόλοιπα κοινά συστήματα. Ακόμη σκοπός μας είναι να μελετήσουμε πόσο καλή είναι η γλώσσα μοντελοποίησης UML και τι μηχανισμοί υπάρχουν και έχουν γίνει για τα συστήματα πραγματικού χρόνου σε αυτήν. Για να το εξετάσουμε αυτό στην πράξη ασχοληθήκαμε με δύο συστήματα πραγματικού χρόνου. Τέλος να εξετάσουμε τα χαρακτηριστικά και τις ιδιαιτερότητες που έχει ο κώδικας των εν λόγω συστημάτων χρησιμοποιώντας μια γλώσσα προγραμματισμού.

1.3 Δομή Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από έξι κεφάλαια. Τελειώνοντας με το κεφάλαιο 1, στο οποίο παρουσιάστηκε η γενικά ιδέα και ο σκοπός της διπλωματικής εργασίας, ακολουθεί μια σύντομη περιγραφή για το τι θα περιέχουν τα επόμενα κεφάλαια.

Στο κεφάλαιο 2, γίνεται αναφορά στα συστήματα πραγματικού χρόνου. Συγκεκριμένα εξηγούμε τι είναι τα συστήματα πραγματικού χρόνου, αναλύουμε τις κατηγορίες στις οποίες χωρίζονται τα συστήματα αυτά, αναφέρουμε τα χαρακτηριστικά τους. Ακόμη, εξηγούμε τους χρονικούς περιορισμούς και γιατί είναι απαραίτητα η χρονική ανάλυση σε ένα τέτοιο σύστημα. Στο τελευταίο υποκεφάλαιο του κεφαλαίου 1, εξηγούμε τη διαδικασία του σχεδιασμού των συστημάτων πραγματικού χρόνου.

Στο κεφάλαιο 3, αναφερόμαστε στην UML(Unified Modeling Language). Κάνουμε μια σύντομη εισαγωγή στην UML, εξηγούμε πως αυτή χρησιμοποιείται στην ανάπτυξη συστημάτων πραγματικού χρόνου. Επίσης, σχολιάζουμε την ταξινόμηση των αντικειμένων στα συστήματα πραγματικού χρόνου και τέλος αναλύονται τα είδη των διαγραμμάτων της UML.

Στο κεφάλαιο 4, περιγράφεται η πρώτη μελέτη περίπτωσης: το σύστημα ελέγχου ανελκυστήρα. Αρχικά, κάνουμε μια σύντομη επεξήγηση και αναφερόμαστε στο γιατί αυτό το σύστημα ανήκει στην κατηγορία των συστημάτων πραγματικού χρόνου. Επιπρόσθετα εξηγούμε γιατί αυτό το σύστημα είναι αυστηρό σύστημα πραγματικού χρόνου. Ακολούθως αναφερόμαστε στην μεθοδολογία που χρησιμοποιήσαμε. Αναλύουμε τις προδιαγραφές του συστήματος. Ακόμη παρουσιάζουμε το μοντέλο περιπτώσεων χρήσης, το στατικό και το δυναμικό μοντέλο του συστήματος μέσα από ένα σύνολο διαγραμμάτων UML. Τέλος καταλήγουμε σε κάποια συμπεράσματα.

Στο κεφάλαιο 5, περιγράφεται η δεύτερη μελέτη περίπτωσης: video player. Υπάρχει μια σύντομη περιγραφή που ακολουθείται από την μεθοδολογία που χρησιμοποιήσαμε. Αναφερόμαστε στο γιατί αυτό το σύστημα ανήκει στην κατηγορία των συστημάτων πραγματικού χρόνου. Επιπρόσθετα εξηγούμε γιατί αυτό το σύστημα είναι χαλαρό σύστημα πραγματικού χρόνου. Τέλος, μοντελοποιούμε τη συμπεριφορά του συστήματος αναπαραγωγής βίντεο σχεδιάζοντας διαγράμματα UML.

Στο κεφάλαιο 6, τελευταίο κεφάλαιο της διπλωματικής εργασίας, υπάρχουν τα συμπεράσματα μας από όλη την μελέτη καθώς και αναφορά σε κάποια μελλοντική εργασία που μπορεί να γίνει σε σχέση με την παρούσα διπλωματική εργασία.

Κεφάλαιο 2

Συστήματα Πραγματικού Χρόνου

2.1 Τι είναι τα συστήματα πραγματικού χρόνου

Ένα σύστημα πραγματικού χρόνου είναι το σύστημα του οποίου η σωστή λειτουργία εξαρτάται από τα αποτελέσματα που παράγονται αλλά κατά κύριο λόγο εξαρτάται από τον χρόνο που παράγονται. Με άλλα λόγια, τα εν λόγω συστήματα δεν έχουν μόνο την ευθύνη να παράγουν σωστά αποτελέσματα αλλά ταυτόχρονα έχουν και την ευθύνη να ικανοποιούν χρονικούς περιορισμούς.

Οι εργασίες που περιλαμβάνονται σε ένα τέτοιο σύστημα πρέπει να εκτελούνται μέσα στο προβλεπόμενο διάστημα. Σε αντίθεση περίπτωση, θα υπάρξουν αρνητικές συνέπειες. Οι συνέπειες μπορεί να είναι από ασήμαντες μέχρι πολύ κρίσιμες για την λειτουργία του συστήματος. Αυτό εξαρτάται από το πόσο κρίσιμη είναι η κάθε εργασία καθώς και από το είδος του συστήματος.

Όταν λέμε σε «πραγματικό χρόνο» δεν εννοούμε πολύ γρήγορα αλλά εννοούμε ότι οι εκτελέσεις των εργασιών πρέπει να ολοκληρωθούν πριν λήξουν οι προθεσμίες, αλλιώς έχουν αποτύχει.

Συμπληρωματικά, κάποιο σύστημα πραγματικού χρόνου μπορεί να αποτύχει ή να επιτύχει. Αυτό εξαρτάται από το αν το σύστημα ανάλογα με τις εισόδους που δέχεται παράγει τις ανάλογες εξόδους και αυτό εννοείται ότι συμβαίνει σε αυστηρά χρονικά περιθώρια. Ένα σύστημα πραγματικού χρόνου θεωρείται αποτυχημένο όταν είτε δίνει λανθασμένα αποτελέσματα είτε τα αποτελέσματα του είναι ορθά αλλά δεν παράγονται την σωστή χρονική στιγμή. Αντιθέτως, ένα σύστημα πραγματικού χρόνου θεωρείται επιτυχημένο όταν δίνει ορθά και ακριβή αποτελέσματα μέσα στα χρονικά διαστήματα που του επιτρέπονται.

2.2 Κατηγορίες συστημάτων πραγματικού χρόνου

Τα συστήματα πραγματικού χρόνου έχουν χωριστεί σε τρεις βασικές κατηγορίες. Η πρώτη κατηγορία είναι αυτή των hard real-time systems (αυστηρά- σκληρά συστήματα πραγματικού χρόνου), η δεύτερη κατηγορία είναι αυτή των soft real-time systems (ανεκτικά- χαλαρά συστήματα πραγματικού χρόνου) και η τρίτη κατηγορία είναι αυτή των firm real- time systems (σταθερά συστήματα πραγματικού χρόνου).

Τα συστήματα αυτά μπορούν να χωριστούν σε αυτές τις τρεις κατηγορίες ανάλογα με το πόσο αυστηρά τηρούν τους χρονικούς περιορισμούς καθώς και με τις συνέπειες που έχουν να αντιμετωπίσουν όταν δεν τηρούν τις χρονικές τους απαιτήσεις.
Ας εξετάσουμε ξεχωριστά την κάθε κατηγορία. Έχουμε:

<u>Hard real-time systems (αυστηρά- σκληρά συστήματα πραγματικού χρόνου)</u>
Ένα σύστημα ανήκει στην κατηγορία των hard real-time systems όταν κάποια εργασία δεν τηρήσει την χρονική προθεσμία εκτέλεσης. Σ’ αυτή τη περίπτωση το σύστημα λειτουργεί λανθασμένα αποτυγχάνει και οι συνέπειες μπορεί να είναι ολέθριες όχι μόνο για τους χρήστες του συστήματος αλλά και για το ίδιο το σύστημα.

Σε τέτοια συστήματα υπάρχουν πάρα πολύ αυστηρά χρονικά περιθώρια και για κανένα λόγο δεν πρέπει να ξεπεραστούν. Πιο συγκεκριμένα, οι απαιτήσεις του χρόνου απόκρισης των αυστηρών συστημάτων πραγματικού χρόνου είναι της τάξεως των χιλιοστών του δευτερολέπτου. Από αυτό φαίνεται ξεκάθαρα ότι αν δεν πληρούνται αυτές οι αυστηρές απαιτήσεις του χρόνου απόκρισης, τότε θα υπάρξουν καταστροφικές συνέπειες (για παράδειγμα, απώλεια δεδομένων, πρόκληση υλικής καταστροφής, γενική αποτυχία και δυσλειτουργία του συστήματος).

Ένα χαρακτηριστικό παράδειγμα hard real- time system είναι ο έλεγχος των μηχανισμών σε ένα αεροπλάνο. Εάν το σύστημα δεν ανταποκρίνεται εγκαίρως σε νέα συμβάντα, μέσα στις συγκεκριμένες χρονικές προθεσμίες, τότε θα έχουμε ως αποτέλεσμα ένα ασταθές αεροπλάνο με τις επακόλουθες συνέπειες. Οι συνέπειες μπορεί να είναι ολέθριες και υπάρχει πιθανότητα να φτάσουμε μέχρι σε σημείο απώλειας ανθρώπινης ζωής.

Μερικά άλλα παραδείγματα τέτοιων συστημάτων είναι τα εξής: έλεγχος των συστημάτων της παραγωγής κάποιου εργοστασίου, έλεγχος φρένων αυτοκινήτων, έλεγχος φωτεινών σηματοδοτών μιας πόλης. Επίσης παράδειγμα αυστηρού συστήματος πραγματικού χρόνου αποτελεί το σύστημα ελέγχου ανελκυστήρα σε ένα κτήριο. Το συγκεκριμένο παράδειγμα θα το δούμε εκτενέστερα σε επόμενο κεφάλαιο της παρούσας διπλωματικής εργασίας.

Soft real-time systems (ανεκτικά- χαλαρά συστήματα πραγματικού χρόνου)

Ένα σύστημα ανήκει στην κατηγορία των soft real-time systems όταν συνεχίζει να λειτουργεί (με υποβαθμισμένη απόδοση κάποιες φορές), ακόμα και όταν το αποτέλεσμα δεν παράγεται σύμφωνα με τις χρονικές απαιτήσεις. Αν κάποια εργασία δεν τηρήσει κάποια χρονική προθεσμία δεν θα υπάρξουν καταστροφικές συνέπειες, απλά θα μειωθεί η ποιότητα του αποτελέσματος που θα παραχθεί. Κάθε εργασία στο σύστημα «δικαιούται» να χάσει (μπορεί να χάσει) κάποιες χρονικές προθεσμίες πριν το σύστημα αποτύχει εντελώς. Όμως, χωρίς αμφιβολία θα ήταν χρήσιμο να τηρούνται οι χρονικοί περιορισμοί για λόγους απόδοσης του συστήματος.

Γενικά, στα χαλαρά συστήματα πραγματικού χρόνου, σε περίπτωση αποτυχίας το αποτέλεσμα δεν είναι καταστρεπτικό για το σύστημα όπως συμβαίνει με τα αυστηρά συστήματα πραγματικού χρόνου.

Ένα χαρακτηριστικό παράδειγμα soft real- time system είναι το σύστημα τηλεδιάσκεψης (μια πολυμεσική εφαρμογή ροής video). Το σύστημα αυτό πρέπει να αποδίδει νέο πλαίσιο κάθε συγκεκριμένο χρονικό διάστημα (για παράδειγμα κάθε τριακοστό του δευτερολέπτου). Επίσης ο ήχος και η εικόνα πρέπει να συγχρονίζονται πάλι μέσα σε συγκεκριμένη χρονική προθεσμία (για παράδειγμα σε 90 millisecond). Ωστόσο, μερικές απώλειες στη ροή των πλαισίων ή στο συγχρονισμό ήχου και εικόνας είναι αποδεκτές, πάντα μέσα σε λογικά πλαίσια. Αν όμως χάνονται πολλές προθεσμίες, τότε το σύστημα γίνεται λιγότερο χρήσιμο.

Κάποια άλλα παραδείγματα τέτοιων συστημάτων είναι τα εξής: παιχνίδια και γενικά διαδραστικές εφαρμογές. Επίσης, παράδειγμα soft real-time αποτελεί το σύστημα για συγχρονισμό υποτίτλων σε ένα βίντεο.

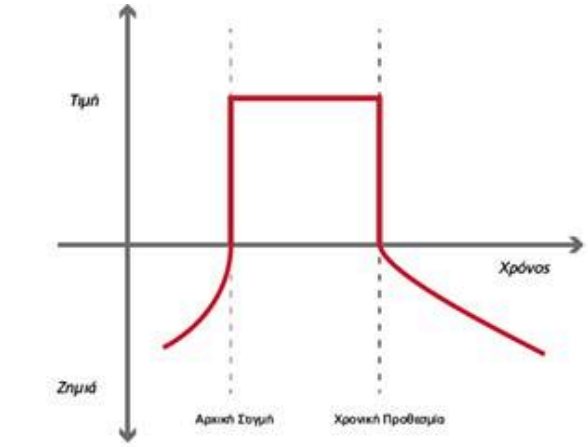
Firm real- time systems (σταθερά συστήματα πραγματικού χρόνου)

Ένα σύστημα ανήκει στην κατηγορία των firm real- time systems όταν σε αυτό το σύστημα η χρονική προθεσμία δηλώνει το χρόνο αναμονής πριν από την έκδοση του αποτελέσματος. Όταν μια χρονική προθεσμία δεν ικανοποιείται, τότε δεν έχουμε πολύ άσχημες συνέπειες. Όμως, όταν κάποιο αποτέλεσμα λαμβάνεται μετά την λήξη της χρονικής προθεσμίας, τότε αυτό το αποτέλεσμα δεν χρησιμοποιείται και δεν λαμβάνεται καθόλου υπόψη από το σύστημα.

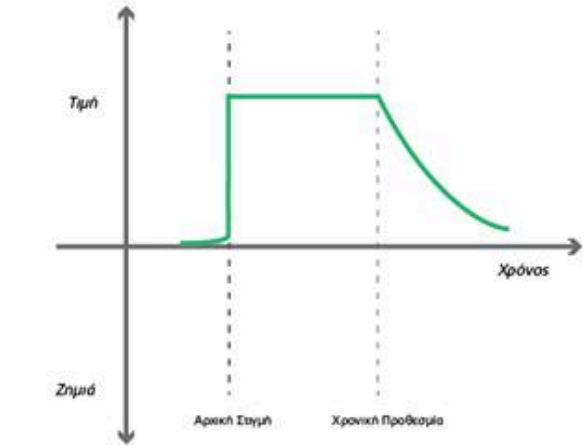
Βασικά, ένα firm real- time system είναι ένα soft real- time system με την διαφορά ότι στο πρώτο δεν θα έχουμε κανένα όφελος από καθυστερημένη παράδοση. Όπως αναφέραμε, το αποτέλεσμα που λαμβάνεται εκτός του παραθύρου της χρονικής προθεσμίας είναι άχρηστο, αλλά το σύστημα μπορεί να αντέξει κάποιες αστοχίες (η πιθανότητα για να συμβούν πρέπει να είναι μικρή) χωρίς να έχουμε καταστρεπτικές συνέπειες.

Κάποια παραδείγματα τέτοιων συστημάτων είναι τα εξής: σύστημα κράτησης εισιτηρίων, συστήματα πρόβλεψης, τραπεζικό σύστημα και πολλά άλλα.

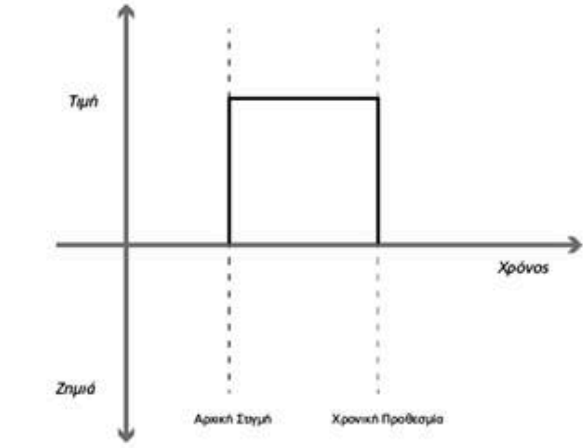
Ας συγκρίνουμε αυτές τις τρεις κατηγορίες με τη βοήθεια ενός σχήματος:



Χρονική προθεσμία εργασίας σε αυστηρό σύστημα πραγματικού χρόνου



Χρονική προθεσμία εργασίας σε χαλαρό σύστημα πραγματικού χρόνου



Χρονική προθεσμία εργασίας σε σταθερό σύστημα πραγματικού χρόνου

Σχήμα 2.1

Εξήγηση σχήματος:

- Αρχική στιγμή: Είναι η αρχική στιγμή που συμβαίνει κάποιο γεγονός στο σύστημα.
- Χρονική προθεσμία: Είναι η χρονική προθεσμία μέχρι την οποία πρέπει να ολοκληρωθεί κάποιο γεγονός.
- Τιμή: Είναι το αποτέλεσμα (δηλαδή η συμβολή, η προσφορά) της εκτελούμενης εργασίας στον σκοπό του συστήματος. Αυτή η τιμή μπορεί να είναι θετική, δηλαδή να έχει θετική προσφορά στο σύστημα, ή να είναι αρνητική, δηλαδή να έχει αρνητική προσφορά και να είναι καταστρεπτική για το σύστημα.

Στο hard real- time system, όπως φαίνεται στο σχήμα, αν ξεπεραστεί η χρονική προθεσμία τότε θα υπάρξει μεγάλη ζημιά για το σύστημα με μεγάλη πιθανότητα τη θανάσιμη καταστροφή του συστήματος.

Στο soft real- time system, όπως απεικονίζει το σχήμα, ακόμα και μετά το πέρας της χρονικής προθεσμίας, δεν μηδενίζεται η συμβολή της εργασίας στο σύστημα. Η συμβολή της εργασίας στο σύστημα είναι πάντα θετική και ελαττώνεται με το πέρασμα του χρόνου. Άρα καταλήγουμε στο ότι εάν υπάρξει αποτυχία στο σύστημα τότε απλώς χαλάει η ποιότητα των αποτελεσμάτων που θα παραχθούν.

Στο firm real- time system, όπως απεικονίζει το σχήμα, η συμβολή της εργασίας στο σύστημα μετά από την λήξη της προθεσμίας είναι μηδενική. Άρα δεν έχουμε ούτε αρνητική αλλά ούτε και θετική προσφορά στο σύστημα. Αυτό συνεπάγεται ότι το παραγόμενο αποτέλεσμα της συγκεκριμένης εργασίας δεν θα προσφέρει τίποτα στο σύστημα, εφόσον έχει χαθεί η προθεσμία.

2.3 Χαρακτηριστικά συστημάτων πραγματικού χρόνου

Το κύριο χαρακτηριστικό των real-time systems είναι οι απαιτήσεις τους σε συγκεκριμένες χρονικές προθεσμίες, τα λεγόμενα deadlines. Η αποτελεσματικότητα και η απόδοση των εν λόγω συστημάτων εξαρτάται κατά κύριο λόγο από τον χρόνο στον οποίο παράγουν τα αποτελέσματα τους και όχι μόνο από την ορθότητα των αποτελεσμάτων αυτών. Αυτά τα συστήματα λοιπόν, στοχεύουν στην ικανοποίηση

των λειτουργικών και οπωσδήποτε των χρονικών απαιτήσεων των εργασιών που εκτελούνται στο σύστημα.

Αδιαμφισβήτητα, στα συστήματα πραγματικού χρόνου, μας ενδιαφέρει σε μεγάλο βαθμό η προβλεψιμότητα. Η πλατφόρμα που θα αναπτυχθεί το σύστημα (hardware, software, operating systems) πρέπει να χαρακτηρίζεται από προβλεψιμότητα. Αυτό σημαίνει ότι πρέπει να υπολογίζεται ο χρόνος που χρειάζεται να ολοκληρωθεί η κάθε εργασία στην χειρότερη περίπτωση, και η πλατφόρμα να εγγυάται ότι κάθε φορά που θα εκτελείται η κάθε εργασία, θα ολοκληρώνεται συντομότερα από τον χρόνο που υπολογίστηκε ότι θα ολοκληρωθεί η εργασία (ή στον ίδιο χρόνο που υπολογίστηκε ότι θα ολοκληρωθεί η εργασία).

Επιπρόσθετα, στα συστήματα πραγματικού χρόνου, πρέπει να έχουμε εγγυημένους χρόνους απόκρισης. Δηλαδή πρέπει να προβλέπεται με σιγουριά ο χρόνος απόκρισης της χειρότερης περίπτωσης. Για να υποστηριχτεί η ανταπόκριση του συστήματος στην χειρότερη περίπτωση χρησιμοποιείται σχεδόν πάντα περισσότερη υπολογιστική ισχύς.

Επίσης, τα συστήματα αυτά πρέπει να είναι αξιόπιστα και ασφαλές. Συνήθως ελέγχουν το περιβάλλον στο οποίο λειτουργούν. Έτσι αν υπάρξουν αστοχίες, θα ακολουθήσουν πολύ άσχημες συνέπειες όπως βλάβη στο περιβάλλον τους, οικονομική-υλική ζημιά και πολλές άλλες.

Επιπλέον, τις περισσότερες φορές η απόδοση αυτών των συστημάτων θυσιάζεται για να μπορέσει το σύστημα να ανταποκριθεί και να ικανοποιήσει τους χρονικούς του περιορισμούς.

Συνοψίζοντας, τα συστήματα πραγματικού χρόνου έχουν βασικό και απαραίτητο χαρακτηριστικό την προβλεψιμότητα ενώ η αποδοτικότητα έρχεται σε δεύτερη μοίρα, παρόλο που και αυτή είναι πολύ σημαντική και πρέπει οπωσδήποτε να χαρακτηρίζει όλα τα συστήματα.

2.4 Χρονικοί περιορισμοί και χρονική ανάλυση

Ένα real- time σύστημα μοντελοποιείται ως ένα σύνολο από εργασίες πραγματικού χρόνου. Κάθε εργασία του συνόλου αυτού αντιπροσωπεύει μια υπολογιστική δραστηριότητα η οποία πρέπει να εκτελεστεί τηρώντας κάποιους περιορισμούς. Για να χαρακτηρίσουμε μια τέτοια δραστηριότητα χρησιμοποιούμε τις ακόλουθες χρονικές παραμέτρους: χρόνος άφιξης, χρόνος έναρξης, χρόνος εκτέλεσης, χρόνος λήξης, χρόνος απόκρισης, απόλυτος χρονικός περιορισμός και σχετικός χρονικός περιορισμός.

Θα εξετάσουμε ξεχωριστά την κάθε χρονική παράμετρο με την βοήθεια του πιο κάτω πίνακα:

ΧΡΟΝΟΣ ΑΦΙΞΗΣ	Η στιγμή που η εργασία είναι έτοιμη προς εκτέλεση.
ΧΡΟΝΟΣ ΕΝΑΡΞΗΣ	Ο χρόνος που η εργασία ξεκινάει για πρώτη φορά την εκτέλεση της.
ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ	Ο χρόνος που χρειάζεται ο επεξεργαστής για να εκτελέσει την εργασία χωρίς καμία διακοπή.
ΧΡΟΝΟΣ ΛΗΞΗΣ	Η χρονική στιγμή που ολοκληρώνεται η εκτέλεση της εργασίας.
ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	Το χρονικό διάστημα μεταξύ χρόνου άφιξης και χρόνου λήξης.
ΑΠΟΛΥΤΟΣ ΧΡΟΝΙΚΟΣ ΠΕΡΙΟΡΙΜΟΣ	Η χρονική στιγμή πριν από την οποία θα πρέπει να ολοκληρωθεί η εργασία.
ΣΧΕΤΙΚΟΣ ΧΡΟΝΙΚΟΣ ΠΕΡΙΟΡΙΣΜΟΣ	Το χρονικό διάστημα κατά τη διάρκεια του οποίου θα πρέπει η εργασία να ολοκληρωθεί.

Πίνακας 2.1

Για να εξασφαλίσουμε ότι τηρούνται όλοι οι χρονικοί περιορισμοί που έχει το σύστημα, θα πρέπει να γίνουν πολλοί πειραματισμοί και μακροχρόνιες και εκτεταμένες προσομοιώσεις.

Όπως έχουμε αναφέρει αρκετές φορές, η ορθότητα ενός συστήματος πραγματικού χρόνου εξαρτάται από τα αποτελέσματα που παράγει το σύστημα αλλά εξαρτάται επίσης και από το χρόνο που χρειάζεται για να παραχθούν τα αποτελέσματα αυτά. Άρα μια πολύ σημαντική διαδικασία που πρέπει να γίνεται για τα συστήματα αυτά είναι η χρονική ανάλυση. Αυτή η διαδικασία είναι πολύ δύσκολη. Είναι δύσκολη επειδή τα μη περιοδικά ερεθίσματα είναι απρόβλεπτα. Επομένως για το λόγο αυτό, για κάθε χρονική στιγμή, πρέπει να γίνονται οι κατάλληλες υποθέσεις για το πόσο πιθανό είναι να εμφανιστούν αυτά τα μη περιοδικά ερεθίσματα και να προβλέπεται τι ανάγκες χρειάζονται για την εξυπηρέτηση τους. Αν οι υποθέσεις αυτές είναι λανθασμένες τότε δεν θα είναι ικανοποιητική η απόδοση του συστήματος.
Τώρα θα δούμε κάποιους παράγοντες που υπεισέρχονται στην χρονική ανάλυση ενός συστήματος πραγματικού χρόνου. Αυτοί οι παράγοντες είναι οι εξής: οι προθεσμίες, ο χρόνος εκτέλεσης και η συχνότητα.

Πιο συγκεκριμένα, οι προθεσμίες αντιπροσωπεύουν τις χρονικές στιγμές μέχρι τις οποίες είναι προκαθορισμένο, δηλαδή πρέπει να γίνει η επεξεργασία των ερεθισμάτων και ακολούθως να δοθεί απάντηση από το σύστημα. Σε ένα αυστηρό σύστημα πραγματικού χρόνου, αν το σύστημα δεν τηρήσει τις προθεσμίες τότε αυτό αποτελεί σφάλμα. Σε ένα χαλαρό σύστημα πραγματικού χρόνου, αν το σύστημα δεν τηρήσει τις προθεσμίες τότε μειώνεται η απόδοση του συστήματος. Ο επόμενος παράγοντας, η συχνότητα, είναι το πόσες φορές μέσα σε κάποιο συγκεκριμένο χρονικό διάστημα, θα πρέπει κάποια διεργασία να εκτελεστεί ώστε να τηρούνται εγγυημένα οι προθεσμίες. Ο τελευταίος παράγοντας είναι ο χρόνος εκτέλεσης. Ο χρόνος εκτέλεσης αποτελεί το χρονικό διάστημα που κάποια διεργασία χρειάζεται για να επεξεργαστεί κάποιο ερέθισμα και να παράγει κάποια απάντηση.

Ας πάρουμε για παράδειγμα το σύστημα ελέγχου των ανελκυστήρων μέσα σε ένα πολυώροφο κτήριο (το οποίο αποτελεί την μελέτη περίπτωσης που θα εξετάσουμε στην συνέχεια) και να δούμε ένα παράδειγμα ερεθίσματος και την αντίστοιχη χρονική του απαίτηση. Αυτό το παράδειγμα είναι όταν συμβεί διακοπή ρεύματος τότε σε χρονικό όριο 50 milliseconds θα πρέπει να γίνει μετάβαση σε εφεδρική τροφοδοσία.

2.5 Σχεδιασμός συστημάτων πραγματικού χρόνου

Σε αυτό το υποκεφάλαιο θα ασχοληθούμε με την διαδικασία που απαιτείται για τον σχεδιασμό συστημάτων πραγματικού χρόνου. Πρώτα από όλα, πρέπει να γίνει σωστή επιλογή για την πλατφόρμα εκτέλεσης του συστήματος. Με αυτό εννοούμε το υλικό που θα χρησιμοποιηθεί καθώς επίσης και το λειτουργικό σύστημα. Για να γίνει σωστή επιλογή για τα πιο πάνω πρέπει κάποιος να λάβει υπόψη τους χρονικούς περιορισμούς του συστήματος και τους περιορισμούς της διαθέσιμης ενέργειας. Επίσης σημαντικοί παράγοντες που επηρεάζουν τις επιλογές αυτές είναι η πείρα της ομάδας που θα αναπτύξει το σύστημα και η τιμή του συστήματος.

Επιπρόσθετα, πρέπει να προσδιοριστούν τα ερεθίσματα τα οποία πρέπει να επεξεργάζεται το σύστημα και οι απαιτούμενες αποκρίσεις των ερεθισμάτων αυτών και στη συνέχεια να γίνει ο προσδιορισμός των χρονικών περιορισμών για την επεξεργασία του ερεθίσματος και της απόκρισης. Στη συνέχεια προχωρούμε στο στάδιο της συγκέντρωσης της επεξεργασίας των ερεθισμάτων και των αποκρίσεων σε μια σειρά από ταυτόχρονες διεργασίες. Οι διεργασίες αυτές πρέπει να εκτελούνται με τέτοιο τρόπο ώστε να αντανακλούνται οι απαιτήσεις του συστήματος.

Ακολούθως, πρέπει να σχεδιαστούν κατάλληλοι αλγόριθμοι για να εκτελούνται οι απαραίτητοι υπολογισμοί για κάθε ερέθισμα και για κάθε απόκριση. Για είναι κάποιος σε θέση να γνωρίζει την ποσότητα της απαιτούμενης επεξεργασίας και χρόνου για την ολοκλήρωση αυτής της επεξεργασίας, πρέπει οι αλγόριθμοι που έχω αναφέρει λίγο πιο πάνω, να αναπτυχθούν στην αρχή της διαδικασίας σχεδιασμού ενός συστήματος πραγματικού χρόνου.

Η διαδικασία σχεδιασμού ενός συστήματος πραγματικού χρόνου περιλαμβάνει επίσης τον σχεδιασμό των πληροφοριών που θα ανταλλάζουν οι διεργασίες και τα γεγονότα που θα συντονίζουν την ανταλλαγή των πληροφοριών. Συγκεκριμένα, για αυτή την ανταλλαγή των πληροφοριών πρέπει να σχεδιαστούν κατάλληλες δομές δεδομένων, οι οποίες θα διαμοιράζονται από ένα αριθμό ταυτόχρονων εκτελούμενων διεργασιών.

Τέλος, ακόμα ένα στάδιο στον σχεδιασμό συστημάτων πραγματικού χρόνου είναι ο σχεδιασμός συστήματος χρονοπρογραμματισμού. Αυτό γίνεται για να διασφαλίζεται το γεγονός ότι όλες οι διεργασίες του συστήματος ξεκινούν έγκαιρα και κυρίως ότι

τηρούν και ικανοποιούν του χρονικούς περιορισμούς που έχει το σύστημα. Οι διεργασίες σε ένα real- time system πρέπει να είναι συντονισμένες, δηλαδή όταν κάποια διεργασία επεξεργάζεται ή τροποποιεί ένα κοινόχρηστο πόρο, τότε οι άλλες διεργασίες δεν πρέπει να μεταβάλουν το πόρο αυτό.

Όλη αυτή η διαδικασία σχεδιασμού συστήματος πραγματικού χρόνου είναι επαναληπτική. Όταν και αφού σχεδιαστεί το σύστημα τότε γίνονται οι κατάλληλες δοκιμές και οι προσομοιώσεις για να γίνει ο έλεγχος των χρονικών περιορισμών και των υποθέσεων που έχουν γίνει. Σε περίπτωση που δεν πληρούνται οι χρονικοί περιορισμοί τότε το σύστημα σχεδιάζεται ξανά λαμβάνοντας υπόψη τα νέα δεδομένα που προέκυψαν από τις δοκιμές.

Κεφάλαιο 3

UML: Unified Modeling Language

3.1 Εισαγωγή στη UML

Η UML (Unified Modeling Language), είναι τρίτης γενιάς γλώσσα τύπου object-oriented modeling που χρησιμοποιείται σήμερα από χιλιάδες προγραμματιστές και εταιρίες. Η γλώσσα αυτή χρησιμοποιείται για την απεικόνιση του σχεδιασμού και την τεκμηρίωση των συστατικών των συστημάτων.

Η UML είναι ικανή για τη μοντελοποίηση όλων των ειδών των συστημάτων, όπως πραγματικού χρόνου, client-server και άλλων ειδών λογισμικών. Βασικό χαρακτηριστικό της είναι ότι είναι ανεξάρτητη από οποιαδήποτε μεθοδολογία και διαδικασία ανάπτυξης. Δηλαδή έχει τη δυνατότητα να προσδιορίσει το σύστημα με ένα τρόπο ανεξάρτητο της υλοποίησης. Συγκεκριμένα, παρέχει ένα πλούσιο σύνολο συμβολισμών για την υλοποίηση κάθε είδους συστήματος.

3.2 UML στην ανάπτυξη συστημάτων πραγματικού χρόνου

Η UML έχει μια πολύ πλούσια γραφική σημειολογία η οποία όταν συνδυάζεται με τις δυνατότητες μοντελοποίησης που παρέχει η γλώσσα αυτή, επιτρέπει τον προσδιορισμό και

την οπτικοποίηση της δομής αλλά και της συμπεριφοράς οποιουδήποτε συστήματος σε πολλαπλά επίπεδα αφαίρεσης. Τα στοιχεία αυτά που την χαρακτηρίζουν, καθιστούν ικανή την UML να χρησιμοποιηθεί στην μοντελοποίηση και την τεκμηρίωσή συστημάτων πραγματικού χρόνου.

Χρησιμοποιώντας την UML μπορούμε να μοντελοποιήσουμε τα σημαντικότερα χαρακτηριστικά των συστημάτων πραγματικού χρόνου. Σε αυτά τα χαρακτηριστικά περιλαμβάνονται ο χρόνος, η απόδοση και οι φυσικοί πόροι του συστήματος.

Η UML όπως αναφέραμε, είναι μια object-oriented modeling γλώσσα επομένως θα ήταν ενδιαφέρον να εξετάσουμε κατά πόσο η UML προσαρμόζεται στα συστήματα πραγματικού χρόνου. Έχουν προταθεί για αυτό το σκοπό πολλά UML profiles προκειμένου η UML να χρησιμοποιήσει τα χαρακτηριστικά και να προβάλει τις δυνατότητες της σε συστήματα με απαιτήσεις χρόνου.

3.2.1 Real- time χαρακτηριστικά στην UML

Η UML έχει την δυνατότητα να αναδείξει τα ιδιαίτερα χαρακτηριστικά των συστημάτων πραγματικού χρόνου όπως οι περιορισμοί χρόνου και η αλληλεπίδραση τους με το περιβάλλον τους. Για παράδειγμα, υποστηρίζει την μοντελοποίηση του συγχρονισμού (concurrency) παρέχοντας κάποιες έννοιες στις οποίες συμπεριλαμβάνονται τα ενεργά αντικείμενα (active objects), παράλληλες λειτουργίες και παράλληλες καταστάσεις. Η UML παρέχει δύο τύπους δεδομένων για να εκφράζει τους χρονικούς περιορισμούς: Time, TimeExpression. Αυτά τα χρησιμοποιούμε σε διαγράμματα ακολουθίας και καταστάσεων. Επιπλέον η UML εισάγει το timing diagram για να δείξει τις αλλαγές των καταστάσεων με την πάροδο του χρόνου. Ωστόσο η UML προσαρμόζεται καλύτερα για τα συστήματα πραγματικού χρόνου με τους ενσωματωμένους μηχανισμούς επεκτασιμότητας της, τα UML profiles όπως έχουμε αναφέρει πιο πάνω.

3.2.2 UML profile for Schedulability, Performance and Time (UML/SPT)

Το UML profile για real- time μοντελοποίηση λέγεται UML profile for Schedulability, Performance and Time (UML/SPT) ή αλλιώς Real-Time UML Profile. Αυτό αύξησε το ενδιαφέρον για την χρήση αντικειμενοστραφούς τεχνολογίας και UML, ιδίως την μοντελοποίηση και την ανάπτυξη συστημάτων πραγματικού χρόνου. Το UML/SPT αποτελεί

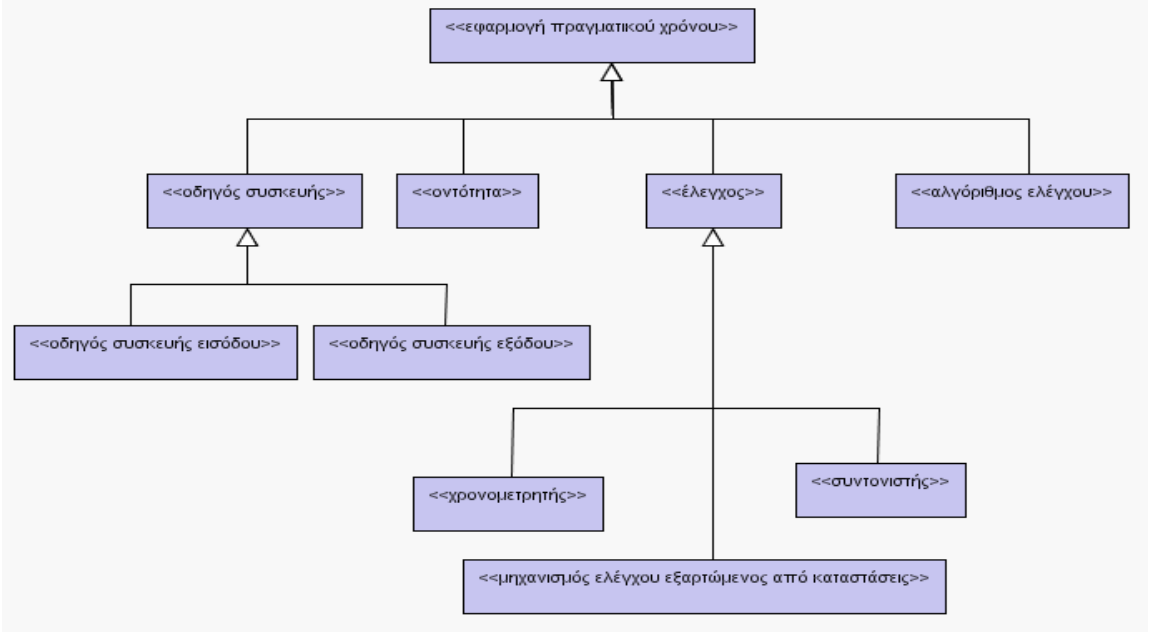
ένα πλαίσιο (framework) για την μοντελοποίηση συστημάτων πραγματικού χρόνου. Πιο ειδικά, μοντελοποίηση του χρόνου, του συγχρονισμού, των πόρων και της ποιότητας παρεχόμενων υπηρεσιών (QoS) και υποστηρίζει ποσοτική ανάλυση των UML models. Επιπλέον, το UML/SPT υποστηρίζει ανάλυση απόδοσης.

3.3 Ταξινόμηση αντικειμένων στα συστήματα πραγματικού χρόνου

Για τον προσδιορισμό των αντικειμένων ενός συστήματος, γενικά, ο αναλυτής συστημάτων καλείται να θεωρήσει τα αντικείμενα που υπάρχουν στον πραγματικό κόσμο όπου και λειτουργεί το σύστημα (real-world objects) και στη συνέχεια να προδιαγράψει αντίστοιχα αντικείμενα λογισμικού (software objects) που αναπαριστούν αυτόν τον πραγματικό κόσμο. Μια καλή τεχνική που εξυπηρετεί το σκοπό αυτό είναι να ταξινομηθούν οι κλάσεις αντικειμένων σε κατηγορίες.

Στην περίπτωση της ανάλυσης ενός συστήματος πραγματικού χρόνου εμφανίζονται γενικές κατηγορίες αντικειμένων, που μπορούν να ορίσουν αντιστοίχως κατάλληλα στερεότυπα για τις κλάσεις αντικειμένων που περιλαμβάνουν.

Στο παρακάτω σχήμα παρουσιάζεται ένα διάγραμμα κλάσεων που παρουσιάζει αυτή τη γενική κατηγοριοποίηση των κλάσεων αντικειμένων στις εφαρμογές πραγματικού χρόνου.



Σχήμα 3.1

3.4 Είδη UML διαγραμμάτων

Ο λόγος που η UML έχει γίνει τόσο ευρέως γνώστη και μπορεί να χρησιμοποιηθεί για συστήματα πραγματικού χρόνου είναι επειδή ο προγραμματιστής μπορεί να σχεδιάσει όλα τα διαγράμματα που θα τον βοηθήσουν να υλοποιήσει το τελικό πηγαίο κώδικα – πρόγραμμα αν ακολουθήσει του βασικούς άξονες. Οι πέντε βασικοί άξονες της UML είναι :

1. Στοιχεία του μοντέλου (model elements)
2. Συσχετίσεις (relationships)
3. Μηχανισμοί (mechanisms)
4. Διαγράμματα (diagrams)
5. Αρχιτεκτονικές όψεις (architectural views)

Χρησιμοποιώντας αυτούς τους άξονες μπορούμε να αναλύσουμε και μετά να υλοποιήσουμε το real time system μας γρήγορα και αποδοτικά.

Αλλά πρώτα πρέπει να δούμε τι είδη διαγραμμάτων έχουν δημιουργηθεί στην UML έτσι ώστε να είναι εφικτό να δημιουργηθεί ένα real time system.

Στην UML υπάρχουν τα εξής είδη διαγραμμάτων:

Δομικά Διαγράμματα (Structural Diagrams): Περιγράφουν την εσωτερική λογική δομή ενός συστήματος, δηλαδή τα συστατικά του και τις σχέσεις μεταξύ τους.

- Διάγραμμα Κλάσεων (Class Diagram)
- Διάγραμμα Αντικειμένων (Object Diagram)
- Διάγραμμα Συνιστωσών (Component Diagram)
- Παραταξιακό Διάγραμμα (Deployment Diagram)

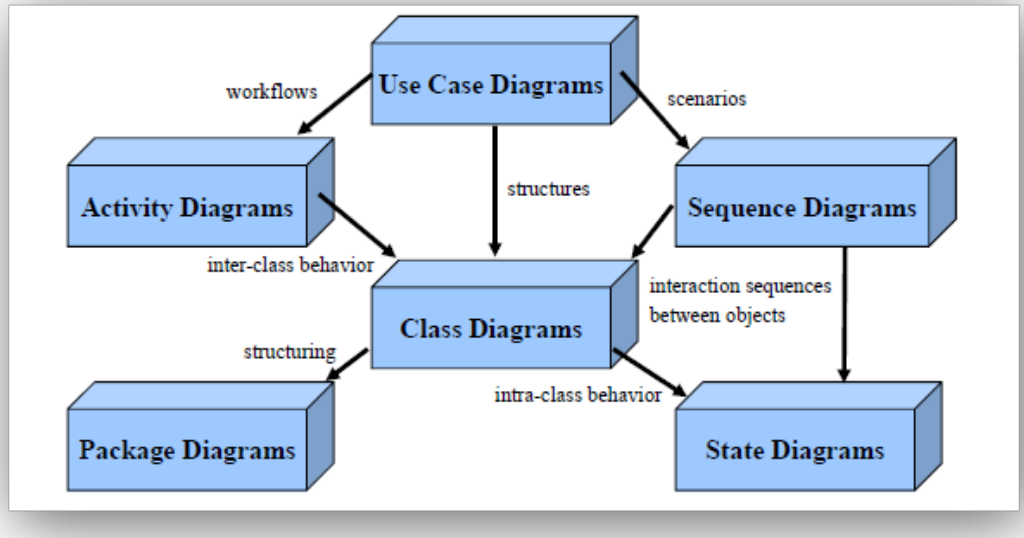
Διαγράμματα Συμπεριφοράς (Behavior Diagrams): Περιγράφουν τη δυναμική συμπεριφορά ενός συστήματος, δηλαδή την απόκρισή του σε γεγονότα του περιβάλλοντός του.

- Διάγραμμα Περιπτώσεων Χρήσης (Use Case Diagram)
- Διάγραμμα Αλληλουχίας (Sequence Diagram)
- Διάγραμμα Δραστηριοτήτων (Activity Diagram)
- Διάγραμμα Συνεργασίας (Collaboration Diagram)
- Διάγραμμα Καταστάσεων (Statechart Diagram)

Διαγράμματα Διαχείρισης Μοντέλου (Model Management Diagrams): Περιγράφουν τη φυσική δομή ενός συστήματος, δηλαδή τις μονάδες λογισμικού που το αποτελούν, σε όρους περιβάλλοντος υλοποίησης.

- Διάγραμμα Πακέτων (Package Diagram)
- Διάγραμμα Υποσυστημάτων (Subsystem Diagram)
- Διάγραμμα Μοντέλων (Model Diagram)

Τώρα θα δούμε πως μπορούν να συνδυαστούν αυτά τα διαγράμματα έτσι ώστε να οδηγηθούμε στο σωστό αποτέλεσμα. Οι σχέσεις των διαγραμμάτων αυτών στην UML είναι όπως φαίνετε πιο κάτω:



Σχήμα 3.2

Ας δούμε όμως λίγο πιο αναλυτικά τι είναι αυτά τα διαγράμματα και πως μπορούμε να τα χρησιμοποιήσουμε:

Τα Class Diagrams παρουσιάζουν κάποια στατικά στοιχεία μοντελοποίησης και τις σχέσεις μεταξύ τους.

Τα Object Diagrams δίνουν παράδειγμα εμφάνισης στιγμιότυπων (instances) των κλάσεων και τις μεταξύ τους σχέσεις σε μια συγκεκριμένη χρονική στιγμή.

Τα Component Diagrams δείχνουν τα συστατικά μέρη του κώδικα και την φυσική τους δομή. Παρουσιάζουν την οργάνωση και τις εξαρτήσεις των μελών ενός συνόλου συνιστωσών.

Τα Deployment Diagrams παρουσιάζουν την τοπολογία των υπολογιστικών κόμβων ενός συστήματος και τύπο των συνδέσεων, ενώ περιέχουν εκτελέσιμα αντικείμενα που φανερώνουν ποιες μονάδες λογισμικού εκτελούνται σε κάθε κόμβο.

Τα Use Case Diagrams περιγράφουν τη λειτουργικότητα του συστήματος όπως αυτή γίνεται αντιληπτή από εξωτερικές οντότητες (actors).

Τα Sequence Diagrams παρουσιάζουν μια αλληλεπίδραση αντικειμένων με έμφαση στην χρονική σειρά ανταλλαγής μηνυμάτων.

Τα Collaboration Diagrams παρουσιάζουν μια αλληλεπίδραση αντικειμένων με έμφαση στη δομική τους οργάνωση.

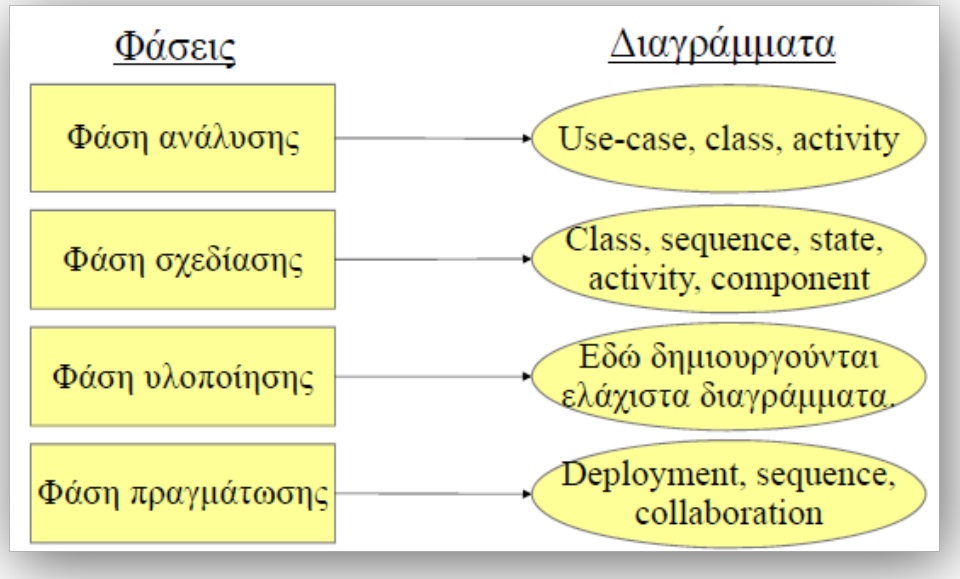
Τα Activity Diagrams παρουσιάζουν την ακολουθιακή ροή των δραστηριοτήτων και περιέχουν προσδιορισμούς των μηνυμάτων που στέλνονται.

Τα Statechart Diagrams παρουσιάζουν τις καταστάσεις του κύκλου ζωής των αντικειμένων.

Το κάθε είδος διαγράμματος έχει δική του προοπτική και άποψη του μοντέλου του συστήματος που βασίζεται. Μπορεί για τον κάθε τύπο να υπάρχουν πολλά διαγράμματα.

Τα sequence, collaboration, statechart και deployment diagrams χρησιμοποιούνται περισσότερο και έχουν τις περισσότερες δυνατότητες για πραγματικού χρόνου συστήματα.

Πως μπορούμε όμως να χρησιμοποιήσουμε αυτά τα διαγράμματα στις διάφορες φάσεις ανάπτυξης κάποιου συστήματος πραγματικού χρόνου ή και οποιουδήποτε συστήματος; Η απάντηση βρίσκετε στο σχήμα πιο κάτω.



Σχήμα 3.3

Περιληπτικά, η UML είναι μια γλώσσα αντικειμενοστραφής σχεδίασης (object-oriented modelling language) που είναι κατάλληλη για συστήματα πραγματικού χρόνου. Παρέχει

υποστήριξη για τις κλάσεις μοντέλων , των αντικειμένων και άλλων ειδών σχέσεων μεταξύ τους.

Οι χρήσεις διαγραμμάτων περιπτώσεων χρήσης (use cases) υποστηρίζονται άμεσα με διάφορα σενάρια για λεπτομερής περιγραφές της απαιτουμένης συμπεριφοράς του συστήματος. Τα διαγράμματα αλληλεπίδρασης απεικονίζουν γραφικά τα σενάρια του μοντέλου και μπορούν να περιέχουν πληροφορίες χρονισμού και συγχρονισμού μηνυμάτων. Η ενισχυμένη μηχανή σχεδίασης πεπερασμένων καταστάσεων (enhanced finite state machine modeling) υποστηρίζει μια σειρά από χαρακτηριστικά πραγματικού χρόνου όπως ταυτοχρονισμού-συγχρονισμού, διάδοση γεγονότων καθώς και τις ένθετες καταστάσεις.
Η UML είναι από μόνη της επεκτάσιμη για ορισμό πρόσθετων στερεότυπων. Οι προγραμματιστές-αναλυτές μπορούν να χρησιμοποιούν την UML είτε για απλά είτε για πολύπλοκα πραγματικού χρόνου συστήματα με συντομία και σαφήνεια.

Κεφάλαιο 4

Μελέτη Περίπτωσης 1: Σύστημα Ελέγχου Ανελκυστήρα

4.1 Γενική επεξήγηση

Όπως αναφέρθηκε στην εισαγωγή της διπλωματικής εργασίας, μελετήσαμε δύο συστήματα πραγματικού χρόνου, ένα hard και ένα soft real- time system.

Σαν περίπτωση μελέτης του αυστηρού συστήματος πραγματικού χρόνου επιλέξαμε να ασχοληθούμε με το σύστημα ελέγχου του ανελκυστήρα. Ο ανελκυστήρας είναι ένα μεταφορικό όχημα που κινείται μεταξύ των ορόφων κάποιου κτηρίου για να μεταφέρει ανθρώπους (ή/και αντικείμενα).

Σε αυτό το υποκεφάλαιο θα εξηγήσουμε γιατί οδηγηθήκαμε στην επιλογή του συγκεκριμένου συστήματος. Καταρχάς, αυτό είναι ένα σύστημα ελέγχου που έχει τα χαρακτηριστικά ενός συστήματος πραγματικού χρόνου που πραγματοποιεί τις εξής λειτουργίες: ελέγχει την κίνηση των ανελκυστήρων ώστε να μετακινούνται κατάλληλα στους ορόφους και ελέγχει αν οι ανελκυστήρες ανταποκρίνονται στις επιλογές των χρηστών που είτε βρίσκονται μέσα στο όχημα του ανελκυστήρα είτε βρίσκονται σε κάποιο όροφο.

Επιλέξαμε το συγκεκριμένο σύστημα επειδή μας παρέχει την δυνατότητα να παρουσιάσουμε τον τρόπο με τον οποίο η γλώσσα μοντελοποίησης που επιλέξαμε, χρησιμοποιείται για να καλύψει τις απαιτήσεις αυτού του συστήματος που παρουσιάζει χαρακτηριστικά ενός real-time system.

Επιπρόσθετα, θα εξηγήσουμε γιατί αυτό το σύστημα κατατάσσεται στην κατηγορία των αυστηρών συστημάτων πραγματικού χρόνου. Το σύστημα ελέγχου ανελκυστήρων αποτελεί ένα πολύ εξελιγμένο σύστημα που έχει να φέρει εις πέρας διάφορες δραστηριότητες εντός αυστηρά καθορισμένων χρονικών περιορισμών. Για παράδειγμα, αν μια εργασία του συστήματος που είναι υπεύθυνη για την κατακόρυφη κίνηση του ανελκυστήρα, χάσει μια συγκεκριμένη χρονική προθεσμία τότε θα μπορούσε να ακολουθήσει η άσχημη συνέπεια να σταματήσει το όχημα του ανελκυστήρα μεταξύ δύο ορόφων.

Έτσι λοιπόν, στο παράδειγμα αυτό του hard real-time system, το λογισμικό πρέπει να αντιδρά σε κάποιο συμβάν προκειμένου να συμπεριφέρεται σωστά (λόγου χάρη να σταματήσει ο ανελκυστήρας στον επιθυμητό όροφο).

Ανακεφαλαιώνοντας, το σύστημα ελέγχου ανελκυστήρα ανήκει στην κατηγορία των αυστηρών συστημάτων πραγματικού χρόνου επειδή έχει και τα εξής χαρακτηριστικά:

- ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ →ΠΟΛΥ ΑΥΣΤΗΡΟΣ
- ΑΣΦΑΛΕΙΑ → ΣΥΧΝΑ ΚΡΙΣΙΜΗ
- ΜΕΓΕΘΟΣ ΑΡΧΕΙΩΝ ΔΕΔΟΜΕΝΩΝ → ΜΙΚΡΑ/ΜΕΣΑΙΑ
- ΑΚΑΙΡΕΟΤΗΤΑ ΔΕΔΟΜΕΝΩΝ → ΒΡΑΧΥΠΡΟΘΕΣΜΑ
- ΑΝΙΧΝΕΥΣΗ ΣΦΑΛΜΑΤΩΝ → ΑΥΤΟΝΟΜΗ

4.2 Μεθοδολογία που χρησιμοποιήθηκε

Για να μοντελοποιήσω την συμπεριφορά του συστήματος έπρεπε να κατασκευάσω μερικά διαγράμματα UML. Ως εργαλείο για την μοντελοποίηση χρησιμοποίησα το ArgoUML

(έκδοση:0.34, λειτουργικό σύστημα: windows all, κατασκευαστής: CollabNet, πηγή: argouml.tigris.org).

Παρόλο που υπάρχουν πολλά εργαλεία που είναι πιο χρήσιμα σε περιπτώσεις συστημάτων πραγματικού χρόνου (όπως το Rational Rhapsody, Enterprise Architect), επιλέξαμε αυτό γιατί είναι το κορυφαίο εργαλείο μοντελοποίησης και υποστηρίζει πολλά είδη διαγραμμάτων UML. Το κύριο πλεονέκτημα του ArgoUML είναι ότι διατίθεται ελεύθερα στο διαδίκτυο και μπορεί κάποιος εύκολα να το χρησιμοποιήσει ακόμα και αν δεν υπάρχει το κατάλληλο υπόβαθρο.

Αξίζει να αναφέρουμε ότι το ArgoUML προορίζεται για προγραμματιστές, σχεδιαστές εφαρμογών, αναλυτές συστημάτων που ασχολούνται με την ανάλυση, τον σχεδιασμό και την ανάπτυξη συστημάτων. Έτσι λοιπόν χρησιμοποιώντας το κατασκευάσαμε τα διαγράμματα του συστήματος ελέγχου του ανελκυστήρα.

Τέλος, υλοποιήσαμε κάποιο κώδικα σε γλώσσα προγραμματισμού Java χρησιμοποιώντας την πλατφόρμα Eclipse. Ο κώδικας στηρίζεται σε νήματα και συγχρονισμένες μεθόδους που είναι απαραίτητα για την υλοποίηση ενός συστήματος πραγματικού χρόνου.

4.3 Προδιαγραφές συστήματος

Το πρόβλημα που μελετούμε αφορά την κίνηση των ανελκυστήρων μεταξύ των ορόφων ενός κτηρίου. Υπάρχει ένα σύστημα ελέγχου το οποίο είναι υπεύθυνο να παίρνει τις αιτήσεις από τους χρήστες καθώς επίσης και να ελέγχει τους ανελκυστήρες.

Σε ένα πολυώροφο κτήριο υπάρχει ένας αριθμός από ανελκυστήρες. Μέσα σε κάθε ανελκυστήρα υπάρχουν κάποια κουμπιά και κάθε κουμπί αντιστοιχεί σε ένα όροφο. Ο χρήστης για να πάει στον προορισμό του (δηλαδή σε συγκεκριμένο όροφο), πιέζει το αντίστοιχο κουμπί. Ακόμα, υπάρχουν οι φωτιζόμενες ενδείξεις (φωτισμός - illumination) δίπλα από κάθε κουμπί του ανελκυστήρα. Κάθε φωτισμός δείχνει τον όροφο από τον οποίο διέρχεται ο ανελκυστήρας. Όταν ο ανελκυστήρας φτάσει στον αντίστοιχο όροφο, ο φωτισμός του αντίστοιχου κουμπιού σβήνει.

Η πόρτα κάθε ανελκυστήρα ελέγχεται από το σύστημα ώστε να κλείνει όταν το όχημα του ανελκυστήρα βρίσκεται σε κίνηση και να ανοίγει όταν το όχημα του ανελκυστήρα σταματάει σε κάποιο όροφο.

Σε κάθε όροφο υπάρχουν δυο κουμπιά. Το ένα κουμπί έχει ένδειξη προς τα πάνω και το άλλο ένδειξη προς τα κάτω. Αυτό ισχύει για κάθε όροφο εκτός από τον τελευταίο όροφο και το ισόγειο όπου υπάρχει ένα κουμπί. Ο χρήστης πατάει το αντίστοιχο κουμπί και καλεί ένα ανελκυστήρα για να πάει στην κατεύθυνση που επιθυμεί, δηλαδή πάνω ή κάτω. Επίσης σε κάθε όροφο υπάρχουν δυο φωτισμοί – illumination (φωτιζόμενες ενδείξεις) οι οποίοι δείχνουν την κατεύθυνση προορισμού που έχει επιλεγεί (προς τα κάτω ή προς τα πάνω). Όταν ο ανελκυστήρας φτάσει στον αντίστοιχο όροφο και κινηθεί προς την επιθυμητή κατεύθυνση, τότε ο φωτισμός σβήνει.

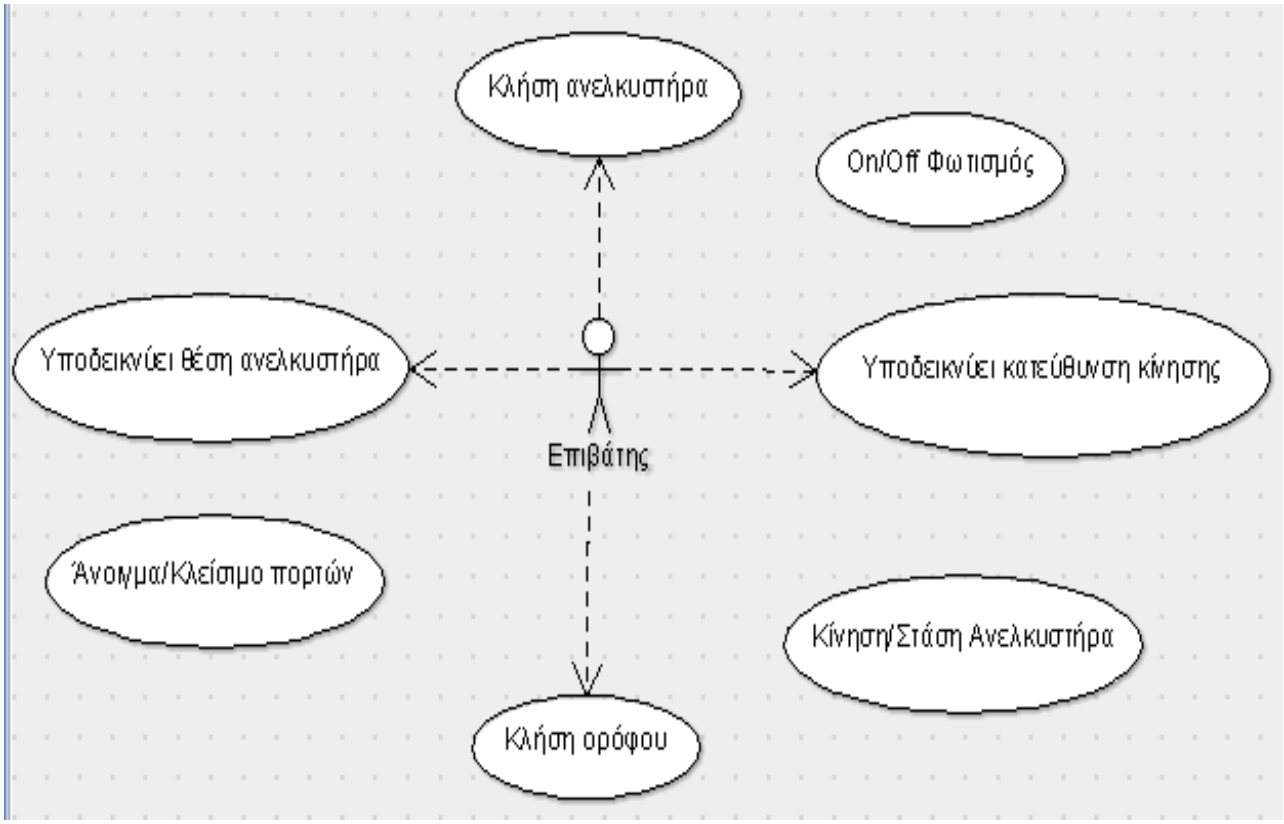
Τέλος, όταν δεν υπάρχουν καθόλου αιτήματα από κανένα υποψήφιο επιβάτη, τότε ο ανελκυστήρας παραμένει στο τρέχον όροφο (στον όροφο που είχε σταματήσει) και περιμένει για κάποιο νέο αίτημα, έχοντας φυσικά κλειστές τις πόρτες του.

4.4 Μοντέλο Περιπτώσεων χρήσης του συστήματος ανελκυστήρα

Στο υποκεφάλαιο αυτό, θα δούμε τις λειτουργικές απαιτήσεις του συστήματος δηλαδή τι κάνει το σύστημα. Στην UML ορίζουμε τις λειτουργικές απαιτήσεις με τον καθορισμό των actors και των use case. Συγκεκριμένα, οι actors είναι οι χρήστες του συστήματος και use case οι αντίστοιχες περιπτώσεις χρήσης. Οι περιπτώσεις χρήσης παρουσιάζουν τις βασικές λειτουργίες του συστήματος χωρίς να παίζει ρόλο η σειρά με την οποία εκτελείται η κάθε μια από αυτές. Οι χρήστες είναι οι άνθρωποι που αλληλεπιδρούν με το σύστημα ή ακόμα και αυτοί που επωφελούνται από τη χρήση του.

4.4.1 Διάγραμμα περιπτώσεων χρήσης

Οι απαιτήσεις χρήσης παρουσιάζονται γραφικά στην UML με το use case diagram (διάγραμμα περιπτώσεων χρήσης). Συνεπώς με το διάγραμμα αυτό θα παρουσιάσουμε τις γενικές ενέργειες που μπορεί να κάνει κάποιος χρήστης με το σύστημα. Το αντίστοιχο use case diagram για το σύστημα του ανελκυστήρα παρουσιάζεται στο σχήμα που ακολουθεί.



Σχήμα 4.1

Actors:

Ο μόνος χρήστης του συστήματος ανελκυστήρα είναι ο επιβάτης. Ο επιβάτης αλληλεπιδρά με το σύστημα με τους εξής τρόπους:

- Καλεί τον ανελκυστήρα και τον όροφο.
- Υποδεικνύει την κατεύθυνση και τη θέση του ανελκυστήρα.
- Παίρνει την απόφαση αν θα μπει/βγει από τον ανελκυστήρα.

Use cases:

- Κλήση ανελκυστήρα
  Στην διαδικασία κλήσης του ανελκυστήρα έχουμε το σενάριο στο οποίο ο ανελκυστήρας λαμβάνει κλήσεις από τους επιβάτες. Επίσης αν ανάβει ή σβήνει ο φωτισμός των κουμπιών του ανελκυστήρα.
- Κλήση ορόφου
  Στην διαδικασία κλήσης του ορόφου έχουμε το σενάριο στο οποίο ο ανελκυστήρας λαμβάνει κλήσεις ορόφου από τους επιβάτες . Επίσης αν ανάβει ή σβήνει ο φωτισμός των κουμπιών του ορόφου.
- Υποδεικνύει κατεύθυνση κίνησης
  Ο ανελκυστήρας πρέπει να είναι σε θέση να αφήνει τους επιβάτες να γνωρίζουν την τρέχουσα κατεύθυνση κίνησης του ανελκυστήρα ούτως ώστε ο επιβάτης να μπορεί να αποφασίσει αν θα εισέλθει στον ανελκυστήρα ή όχι.
- Υποδεικνύει θέση ανελκυστήρα
  Ο ανελκυστήρας πρέπει να είναι σε θέση να αφήνει τους επιβάτες να γνωρίζουν αν έχει φτάσει ο ανελκυστήρας στον προορισμό τους ούτως ώστε ο επιβάτης να μπορεί να αποφασίσει αν θα εξέλθει από τον ανελκυστήρα ή όχι.
- Κίνηση/στάση ανελκυστήρα
  Αυτή είναι η κύρια λειτουργία του ανελκυστήρα. Η λειτουργία αυτή ουσιαστικά είναι το πώς ο ανελκυστήρας θα πάρει την απόφαση να κινηθεί προς κάποια συγκεκριμένη κατεύθυνση ή αν θα σταματήσει.
- Άνοιγμα/κλείσιμο πορτών
  Ο ανελκυστήρας πρέπει να είναι σε θέση να ανοίξει ή να κλείσει τις πόρτες του για να βγουν ή/και να μπουν επιβάτες. Οι επιβάτες έχουν το δικαίωμα να επέμβουν στον αν θα κλείσουν οι πόρτες γιατί μπορεί εκείνη την στιγμή που κλείνουν οι πόρτες κάποιος να θέλει να εισέλθει στον ανελκυστήρα.
- On/off φωτισμός
  Τα κουμπιά του ανελκυστήρα και του ορόφου πρέπει να πληροφορούν τον επιβάτη ότι έχει προγραμματιστεί η κλήση που έκανε και τον ανελκυστήρα ότι έχει νέο αίτημα για να φωτιστούν τα κατάλληλα κουμπιά.

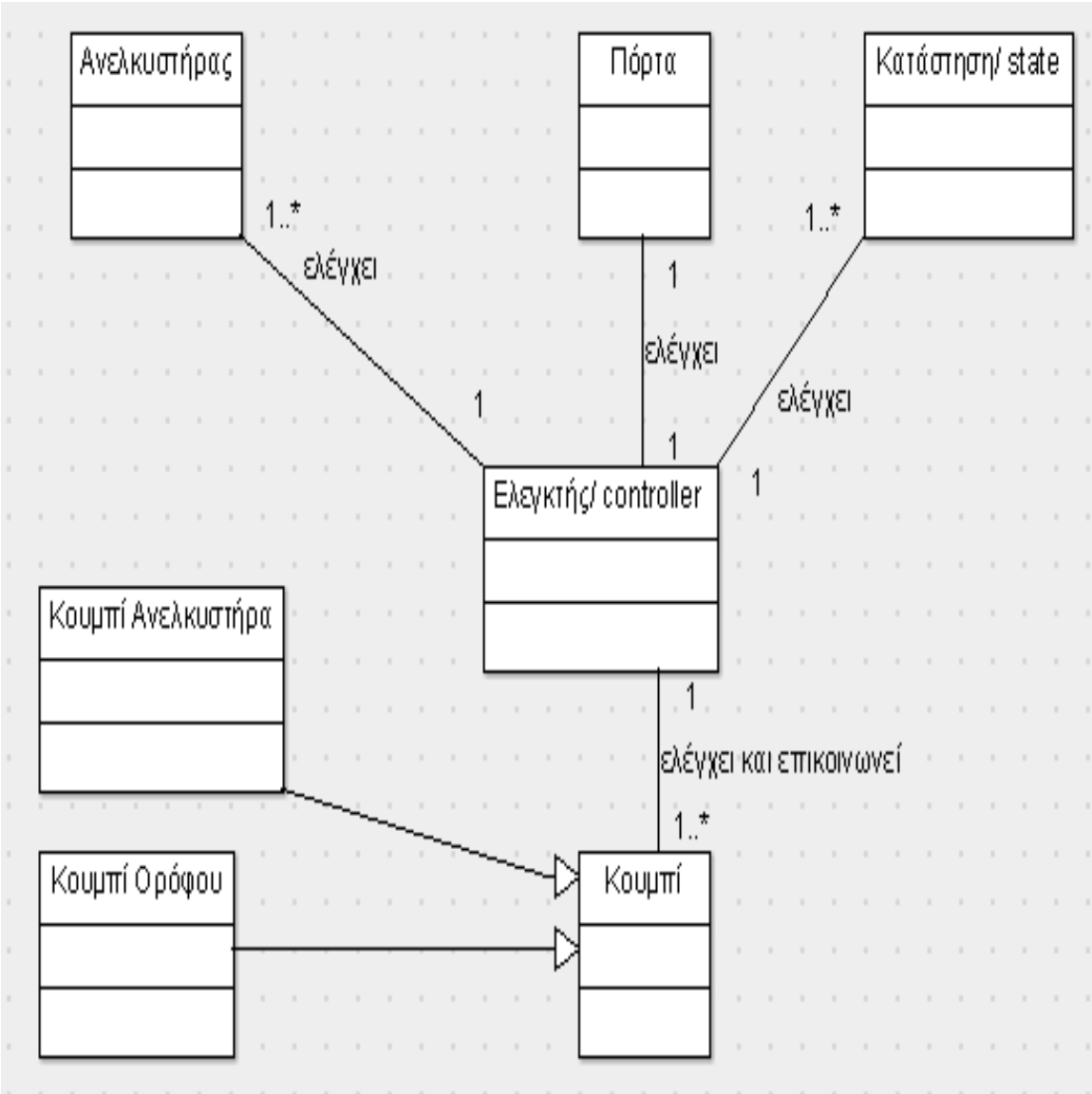
4.5 Στατικό μοντέλο του συστήματος ανελκυστήρα

Σ' αυτό το υποκεφάλαιο θα εξετάσουμε το στατικό μοντέλο του συστήματος. Με αυτό εννοούμε τα συστατικά του συστήματος δηλαδή την εικόνα της δομής του συστήματος και τις βασικές δομικές του ενότητες. Για την μοντελοποίηση της στατικής δομής του συστήματος, η UML χρησιμοποιεί τα εξής διαγράμματα: διαγράμματα κλάσεων, διαγράμματα αντικειμένων, παραταξιακά και ψηφιδικά διαγράμματα. Έτσι λοιπόν, η στατική δομή του συστήματος περιγράφεται με κλάσεις, συσχετίσεις, συστατικά και αντικείμενα.

4.5.1 Διάγραμμα Κλάσεων

Εδώ θα παρουσιάσουμε τα συστατικά στοιχεία του συστήματος με τη χρήση του διαγράμματος κλάσεων (class diagram).

Στο διάγραμμα κλάσεων ενός συστήματος πραγματικού χρόνου, παρουσιάζονται τα φυσικά υποσυστήματα του συστήματος αλλά και το πώς συνδέονται μεταξύ τους. Στην περίπτωση του συστήματος που μελετούμε εμείς θα παρουσιάσουμε στο στατικό μοντέλο τις στατικές σχέσεις ανάμεσα στις συσκευές του συστήματος (βασικές κλάσεις συστήματος). Αυτές είναι οι συσκευές που ένας χρήστης του συστήματος μπορεί να τις βλέπει και, στα real – time systems χρησιμοποιούνται για αναπαράσταση χρηστών, χρονομετρητών, εξωτερικών συσκευών και άλλα. Το αντίστοιχο διάγραμμα κλάσεων για το σύστημα ελέγχου ανελκυστήρα παρουσιάζεται στο σχήμα που ακολουθεί:



Σχήμα 4.2

Εξήγηση διαγράμματος:
Για το σύστημα ελέγχου ανελκυστήρα έχουμε τις εξής κλάσεις:

Ανελκυστήρας: Ο ανελκυστήρας ελέγχεται για να σταματά στους ορόφους του κτηρίου όποτε είναι απαραίτητο από τις κλήσεις που δέχεται από τους επιβάτες. Επίσης ελέγχεται για να μετακινείται ανάλογα με τα request πάνω ή κάτω.

Πόρτα: Οι πόρτες ελέγχονται για να ανοίγουν και να κλείνουν ανάλογα με την κατάσταση την οποία βρίσκεται το σύστημα.

Κατάσταση/State: Η κατάσταση αναφέρεται στις πληροφορίες σχετικά με την κατεύθυνση που κινείται ο ανελκυστήρας και την τρέχουσα θέση που βρίσκεται.

Ελεγκτής/Controller: Ο controller είναι το κύριο αντικείμενο ελέγχου του συστήματος του ανελκυστήρα ο οποίος ελέγχει και επικοινωνεί με όλα τα υπόλοιπα αντικείμενα που απαρτίζουν το σύστημα.

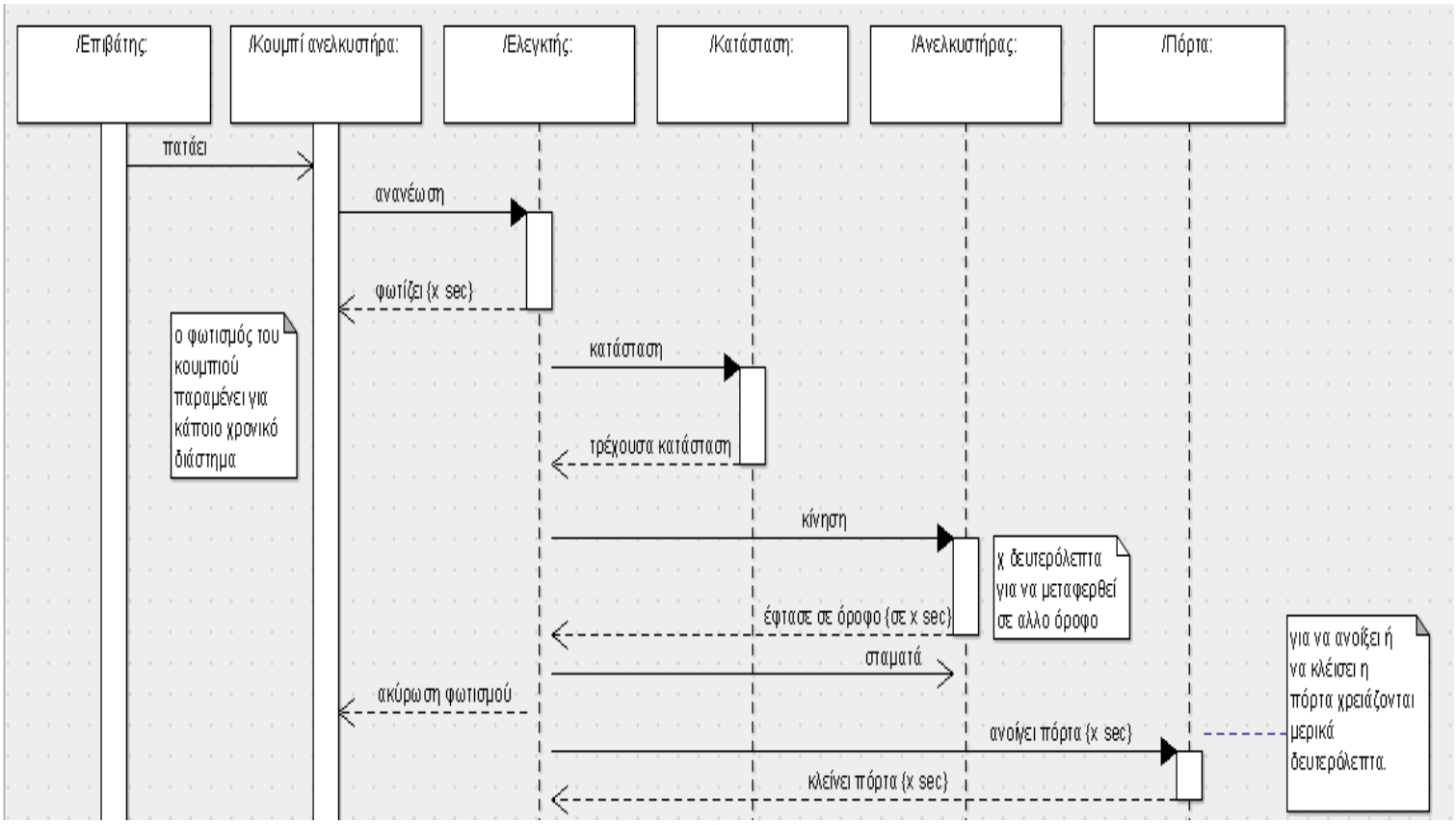
Κουμπί:
→Κουμπί ανελκυστήρα
→Κουμπί ορόφου
Το κουμπί ελέγχεται από τον controller. Το κουμπί ανελκυστήρα και το κουμπί ορόφου κληρονομούν από την κλάση κουμπί. Ο controller επικοινωνεί με το κουμπί για να πληροφορηθεί αν έχει πατηθεί κάποιο κουμπί έτσι ώστε να ανάψει ο φωτισμός του κατάλληλου κουμπιού.

4.6 Δυναμικό μοντέλο του συστήματος ανελκυστήρα

Το υποκεφάλαιο αυτό του δυναμικού μοντέλου του συστήματος αποτελεί το επόμενο βήμα της ανάλυσης μας. Εδώ θα καθορίσουμε τις αλληλεπιδράσεις μεταξύ των αντικειμένων του συστήματος καθώς επίσης και τις ακολουθίες των ενεργειών που γίνονται για διάφορες περιπτώσεις χρήσης. Γι’ αυτό το σκοπό θα χρησιμοποιήσουμε κάποια διαγράμματα της UML όπως τα διαγράμματα ακολουθίας και διαγράμματα συνεργασίας τα οποία ανήκουν στην κατηγορία των διαγραμμάτων αλληλεπίδρασης. Συγκεκριμένα τα διαγράμματα ακολουθίας κρίνονται κατάλληλα για συστήματα πραγματικού χρόνου επειδή δείχνουν ρητά την ακολουθία των μηνυμάτων.

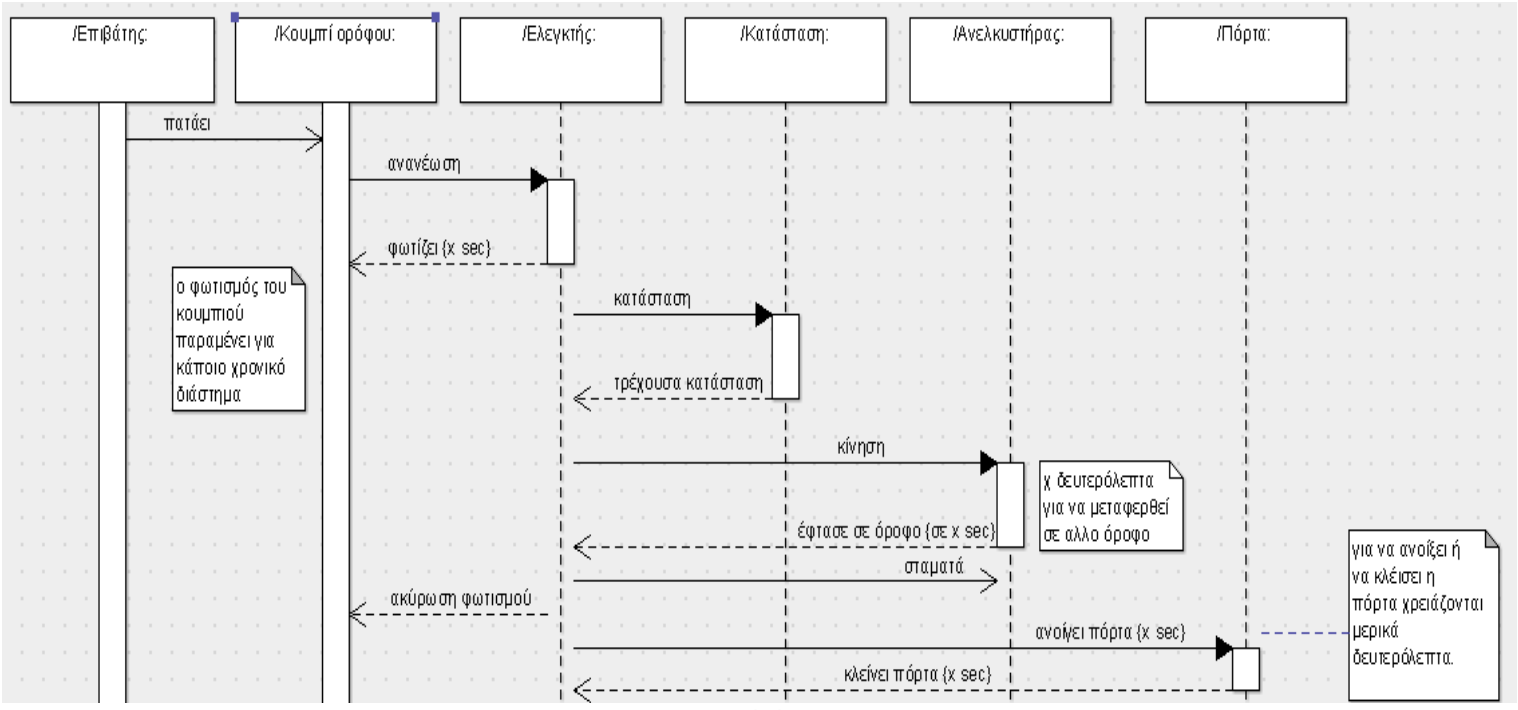
4.6.1 Διαγράμματα ακολουθίας (sequence diagrams)

→Το ακόλουθο διάγραμμα αναπαριστά την περίπτωση της εξυπηρέτησης κουμπιού του ανελκυστήρα, δηλαδή όταν κάποιος χρήστης πατήσει κάποιο κουμπί που βρίσκεται μέσα στον ανελκυστήρα. Χρησιμοποιούμε ετικέτες που προσδιορίζουν κάποιους περιορισμούς σχετικά με το χρόνο.



Σχήμα 4.3

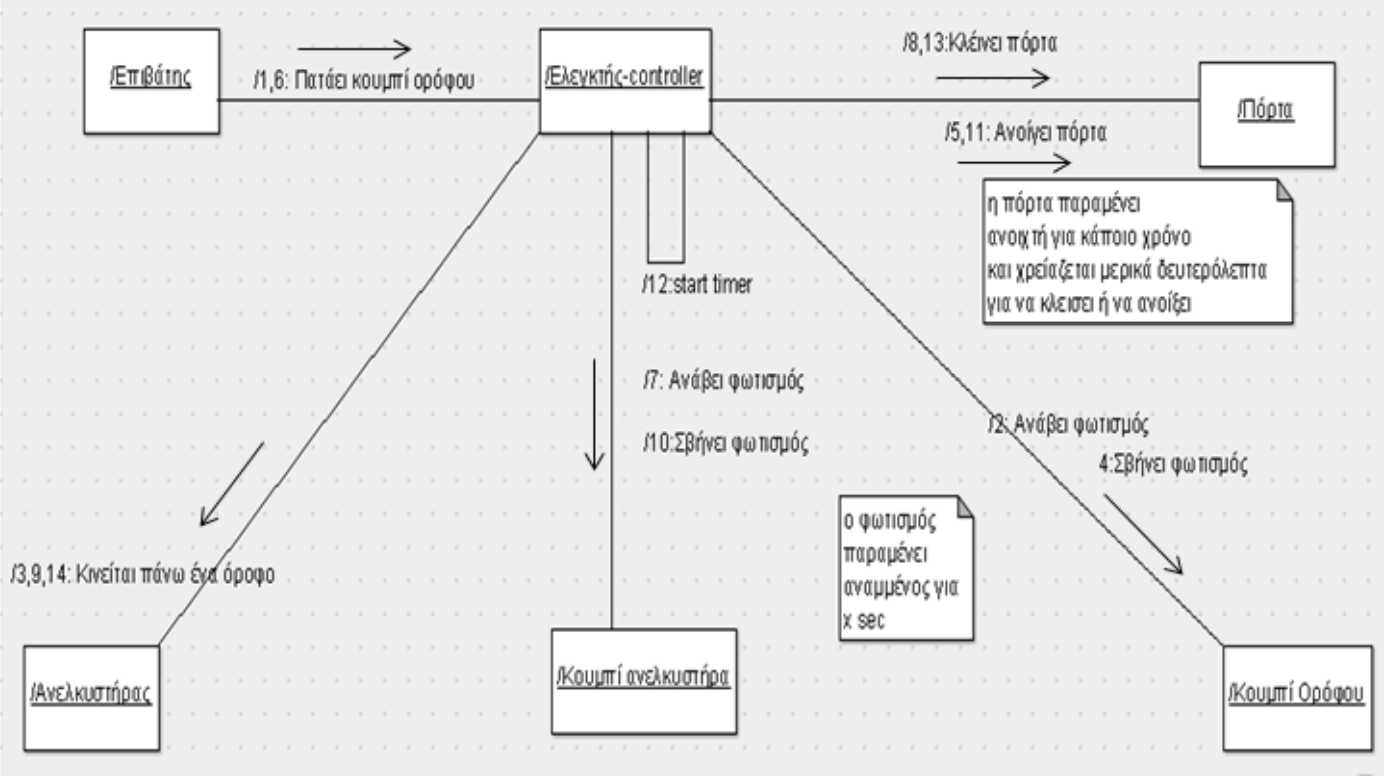
→ Το ακόλουθο διάγραμμα αναπαριστά την περίπτωση της εξυπηρέτησης κουμπιού του ορόφου, δηλαδή όταν κάποιος χρήστης πατήσει κάποιο κουμπί που βρίσκεται στον όροφο.



Σχήμα 4.4

4.6.2 Διάγραμμα συνεργασίας

Στο επόμενο σχήμα απεικονίζεται το διάγραμμα συνεργασίας (collaboration diagram) του συστήματος.



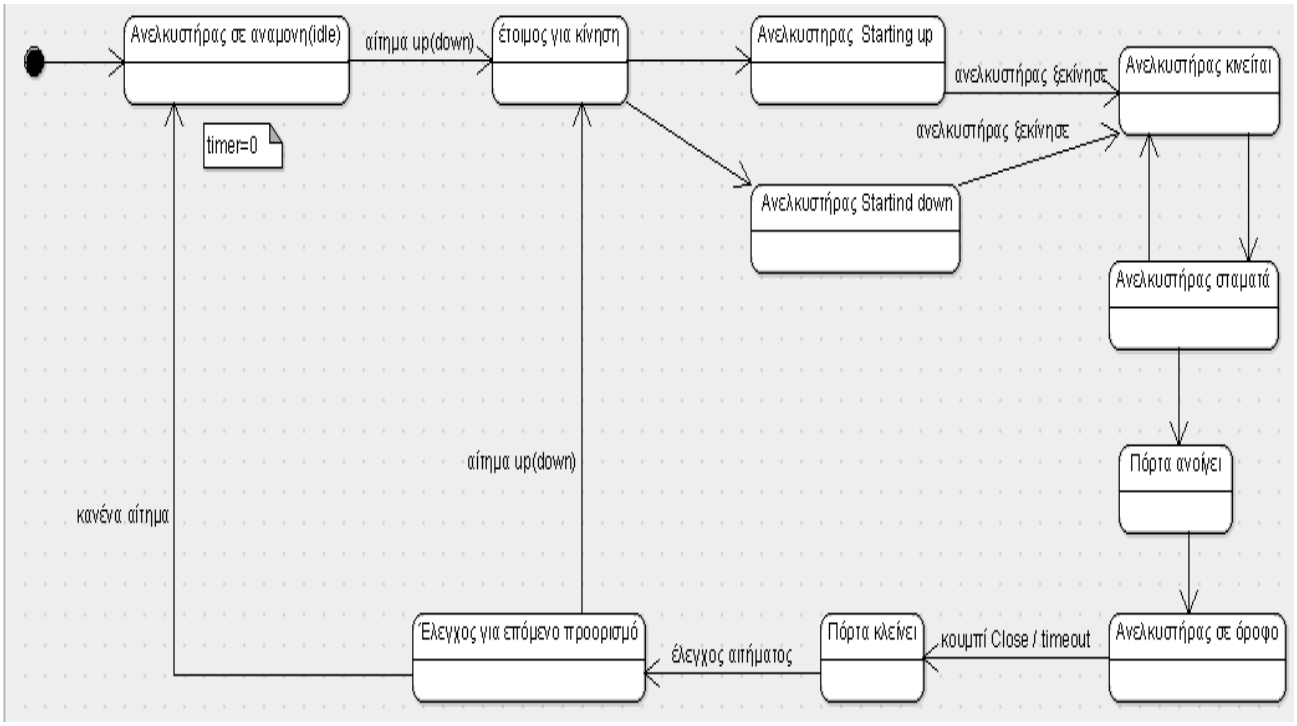
Σχήμα 4.5

Στο πιο πάνω σχήμα περιγράφεται το διάγραμμα συνεργασίας του συστήματος το οποίο δείχνει τα αντικείμενα και τους μεταξύ τους συνδέσμους. Επίσης δείχνει τα μηνύματα που στέλνονται ανάμεσα στα συνδεδεμένα αντικείμενα. Συγκεκριμένα πιο πάνω φαίνεται ότι ο επιβάτης στέλνει ένα μήνυμα (πάτημα του κουμπιού ορόφου) στον ελεγκτή καθώς επίσης φαίνονται και όλα τα υπόλοιπα μηνύματα ανάμεσα στα αντικείμενα που συνδέονται.

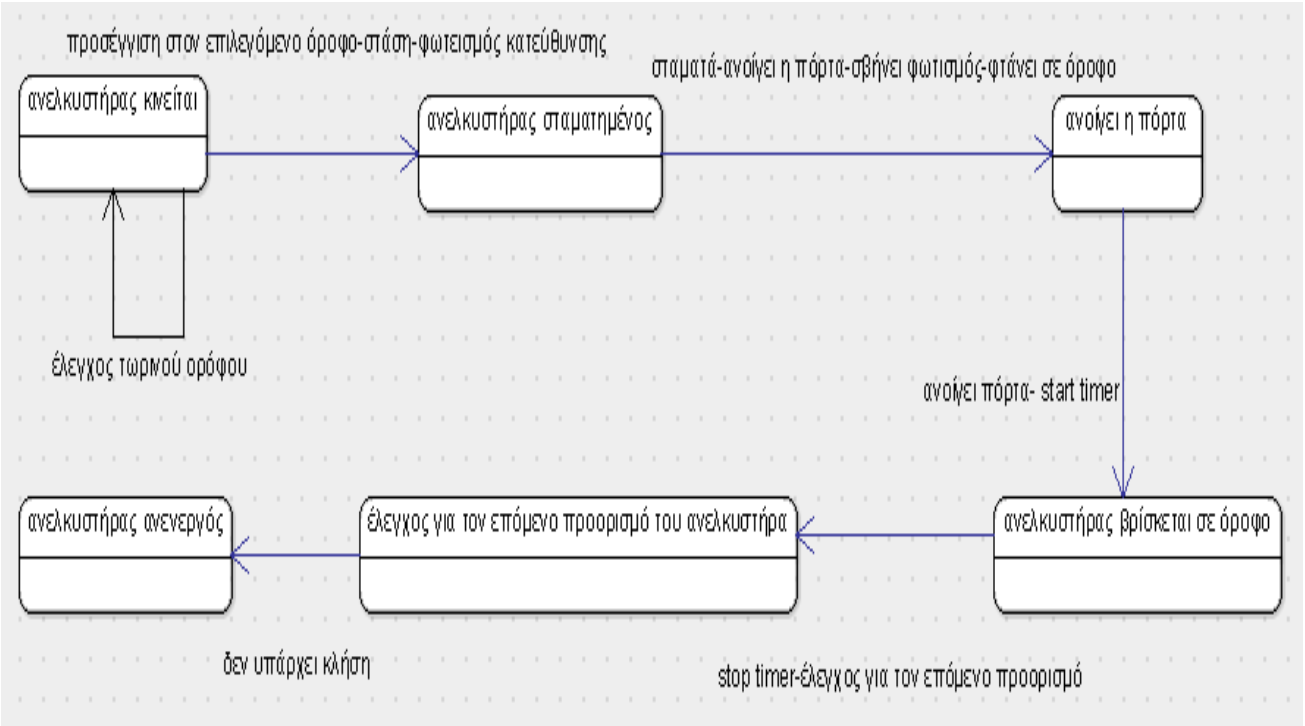
4.6.3 Διαγράμματα καταστάσεων

Σ' αυτό το υποκεφάλαιο θα παρουσιάσουμε ορισμένα διαγράμματα καταστάσεων (state chart diagrams) για διάφορες περιπτώσεις χρήσης του συστήματος.

➔ Διαγράμματα καταστάσεων για σύστημα ελέγχου του ανελκυστήρα

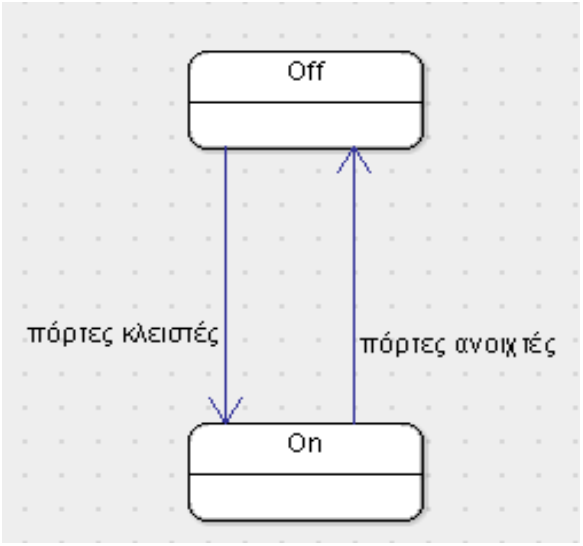


Σχήμα 4.6



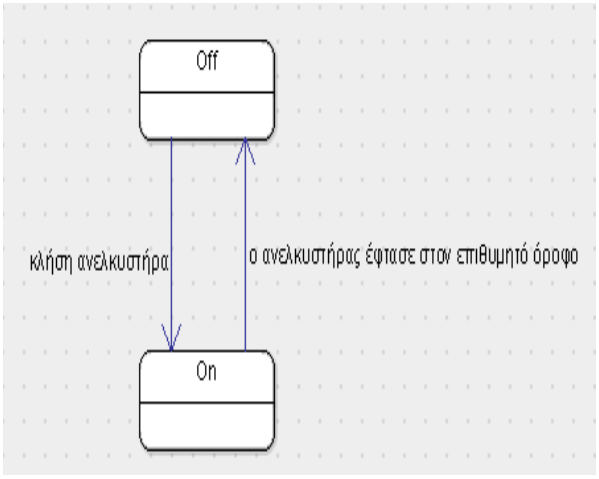
Σχήμα 4.7

➔ Διάγραμμα καταστάσεων για τη θέση του ανελκυστήρα



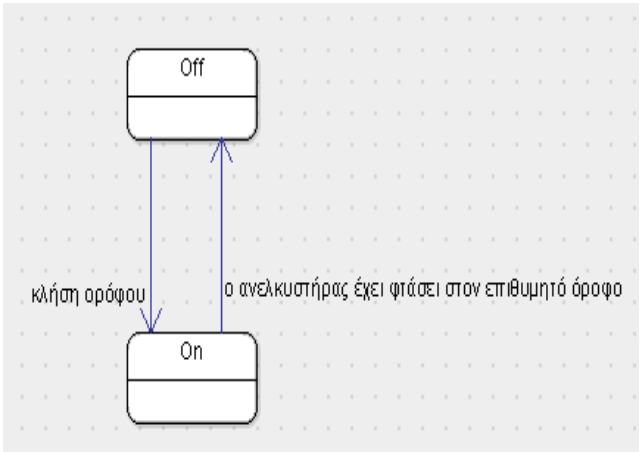
Σχήμα 4.8

→Διάγραμμα καταστάσεων για το κουμπί του ανελκυστήρα



Σχήμα 4.9

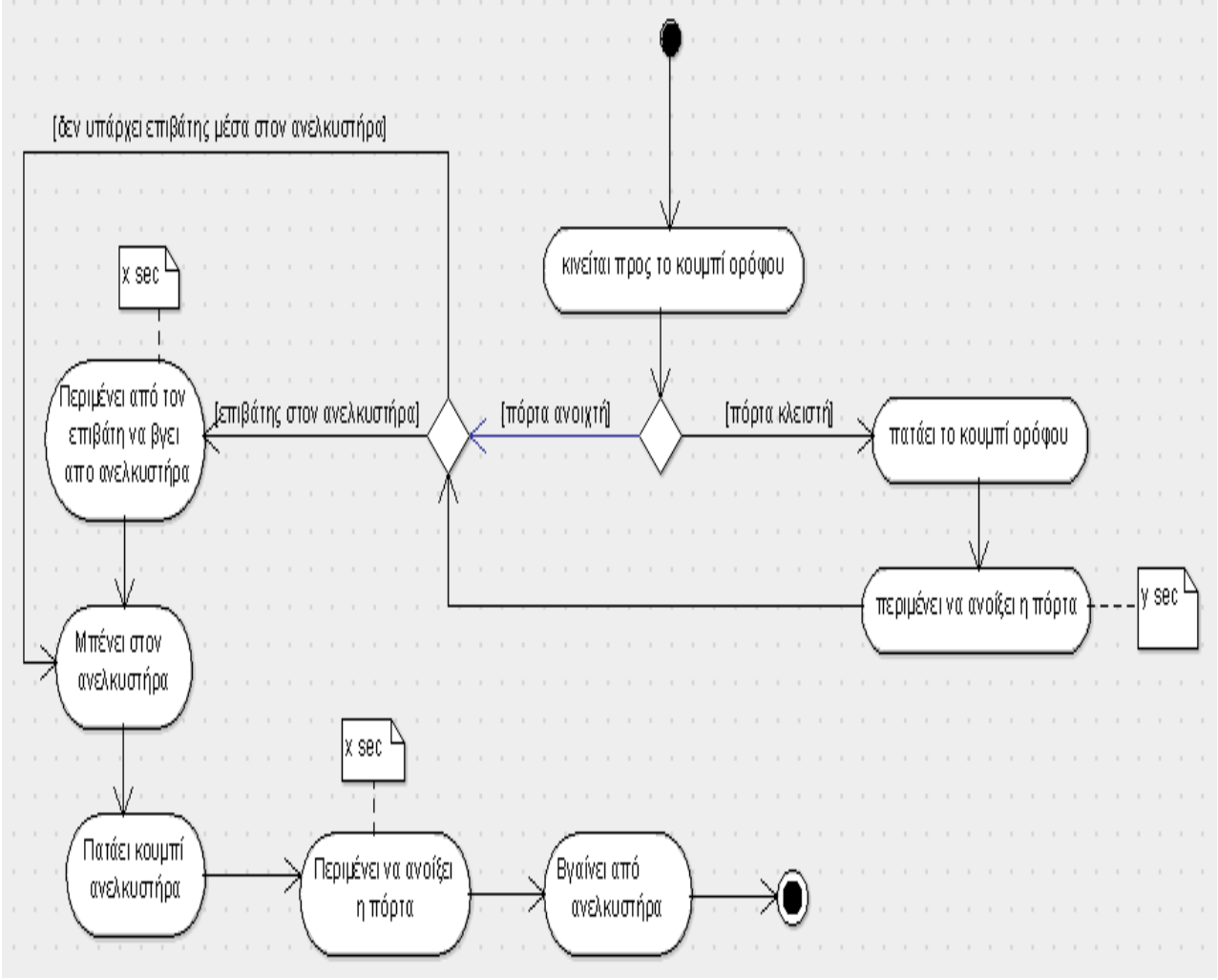
➔ Διάγραμμα καταστάσεων για το κουμπί του ορόφου



Σχήμα 4.10

4.6.4 Διάγραμμα δραστηριοτήτων

Εδώ σε αυτό το υποκεφάλαιο θα παρουσιάσουμε το διάγραμμα δραστηριοτήτων (activity diagram) που απεικονίζει τις ενέργειες του χρήστη όταν θέλει να χρησιμοποιήσει τον ανελκυστήρα.



Σχήμα 4.11

4.7 Συμπεράσματα

Στο κεφάλαιο αυτό είδαμε τον τρόπο με τον οποίο εφαρμόζεται η UML και οι μεθοδολογίες της αντικειμενοστραφούς τεχνολογίας στην ανάλυση εφαρμογών που έχουν απαιτήσεις πραγματικού χρόνου.

Η ανάλυση και η σχεδίαση τέτοιων συστημάτων δεν είναι εύκολο αλλά αντιθέτως είναι πολύπλοκο πρόβλημα. Αυτό οφείλεται στα εξής ιδιαίτερα χαρακτηριστικά των real time systems:

1. Οι αυξημένες λειτουργικές και χρονικές απαιτήσεις

2. Η ασύγχρονη φύση της εισόδου στο σύστημα
3. Η πολυπλοκότητα της στατικής δομής και της συμπεριφοράς.

Για να αντιμετωπίσουμε αυτό το πρόβλημα στην πράξη εφαρμόσαμε κανόνες αντικειμενοστραφούς ανάλυσης και σχεδίασης χρησιμοποιώντας την UML. Με αφορμή την περίπτωση μελέτης του συστήματος ελέγχου ανελκυστήρα, εξετάσαμε μια σειρά ζητημάτων που συνδέονται ειδικότερα με την ανάλυση και τη σχεδίαση συστημάτων πραγματικού χρόνου.

Κεφάλαιο 5

Μελέτη Περίπτωσης 2: Video Player

5.1 Σύντομη περιγραφή

Όπως αναφέρθηκε στην εισαγωγή της διπλωματικής εργασίας, μελετήσαμε δύο συστήματα πραγματικού χρόνου, ένα hard και ένα soft real- time system. Σαν περίπτωση μελέτης του χαλαρού συστήματος πραγματικού χρόνου επιλέξαμε να ασχοληθούμε με την εφαρμογή αναπαραγωγής βίντεο (video player).

5.2 Μεθοδολογία που χρησιμοποιήθηκε

Για να μοντελοποιήσω την συμπεριφορά του συστήματος έπρεπε να κατασκευάσω μερικά διαγράμματα UML. Ως εργαλείο για την μοντελοποίηση χρησιμοποίησα το ArgoUML (έκδοση:0.34, λειτουργικό σύστημα: windows all, κατασκευαστής: CollabNet, πηγή: argouml.tigris.org).

Παρόλο που υπάρχουν πολλά εργαλεία που είναι πιο χρήσιμα σε περιπτώσεις συστημάτων πραγματικού χρόνου (όπως το Rational Rhapsody, Enterprise Architect), επιλέξαμε αυτό γιατί είναι το κορυφαίο εργαλείο μοντελοποίησης και υποστηρίζει πολλά είδη διαγραμμάτων UML. Το κύριο πλεονέκτημα του ArgoUML είναι ότι διατίθεται ελεύθερα στο διαδίκτυο και μπορεί κάποιος εύκολα να το χρησιμοποιήσει ακόμα και αν δεν υπάρχει το κατάλληλο υπόβαθρο.

Αξίζει να αναφέρουμε ότι το ArgoUML προορίζεται για προγραμματιστές, σχεδιαστές εφαρμογών, αναλυτές συστημάτων που ασχολούνται με την ανάλυση, τον σχεδιασμό και την ανάπτυξη συστημάτων. Έτσι λοιπόν χρησιμοποιώντας το κατασκευάσαμε τα διαγράμματα για την εφαρμογή του video player.

Τέλος, μελετήσαμε κάποιο κώδικα σε Real Time Java και εξάγαμε κάποια συμπεράσματα. Ο κώδικας στηρίζεται σε νήματα και συγχρονισμένες μεθόδους που είναι απαραίτητα για την υλοποίηση ενός συστήματος πραγματικού χρόνου.

5.3 Εξήγηση επιλογής της μελέτης περίπτωσης

Σε αυτό το υποκεφάλαιο θα εξηγήσουμε γιατί οδηγηθήκαμε στην επιλογή του συγκεκριμένου συστήματος. Καταρχάς, αυτό είναι ένα σύστημα που έχει τα χαρακτηριστικά ενός συστήματος πραγματικού χρόνου. Επιλέξαμε το συγκεκριμένο σύστημα επειδή μας παρέχει την δυνατότητα να παρουσιάσουμε τον τρόπο με τον οποίο η γλώσσα μοντελοποίησης που επιλέξαμε, χρησιμοποιείται για να καλύψει τις απαιτήσεις αυτού του συστήματος που παρουσιάζει χαρακτηριστικά ενός real- time system.

Επιπρόσθετα, θα εξηγήσουμε γιατί αυτό το σύστημα κατατάσσεται στην κατηγορία των χαλαρών συστημάτων πραγματικού χρόνου. Από τον ορισμό ενός soft real time system: « Ένα ‘soft’ σύστημα πραγματικού χρόνου είναι το σύστημα του οποίου η λειτουργία υποβαθμίζεται αν τα αποτελέσματα δεν παράγονται σύμφωνα με τους συγκεκριμένους χρονικούς περιορισμούς». Σε ένα πρόγραμμα αναπαραγωγής βίντεο, σε περίπτωση καθυστέρησης θα ελευθερώσει-απορρίψει μερικά frames αλλά θα συνεχίσει να λειτούργει κανονικά και να αναπαραγάγει την ταινία. Οπότε η μη ικανοποίηση της προθεσμίας οδηγεί σε μειωμένη απόδοση αλλά η λειτουργιά συνεχίζει κανονικά και δεν οδηγείτε σε αποτυχία όπως στο hard real time system.

Αξίζει να αναφέρουμε πως με τον κώδικα σε Real time java δικαιολογείται πως το video player είναι παράδειγμα χαλαρού συστήματος πραγματικού χρόνου. Το πρόγραμμα πρέπει να διέπεται από τις αρχές των συστημάτων πραγματικού χρόνου δηλαδή προβλεψιμότητα, καθοριστικότητα, ταυτοχρονισμός, παραλληλισμός. Στην Real time java χρησιμοποιώντας τον Real Time Garbage Collector (RTGC) επιτυχαίνεται το πρόγραμμα να είναι πραγματικού χρόνου και να υπάρχουν όσο το δυνατόν λιγότερες απώλειες και καθυστερήσεις. Η Real-Time Java έχει ως στόχο να διατηρήσει όλα εκείνα τα χαρακτηριστικά της Java που εξασφαλίζουν αυξημένη παραγωγικότητα και παράλληλα να παρέχει μια πλατφόρμα κατάλληλη για την ανάπτυξη συστημάτων πραγματικού χρόνου.

Αυτό επιτυγχάνεται αρχικά με την χρήση πραγματικού χρόνου νημάτων (RealTimeThreads) και πολυνημάτωσης (multithreading). Στη περίπτωση που μελετούμε εμείς, γίνεται δημιουργία διαφορετικών νημάτων πραγματικού χρόνου, για κάθε βίντεο που ανοίγει ο χρήστης , για να χειριστεί το decoding του video όσο πιο γρήγορα και απροβλημάτιστα γίνεται. Στο κάθε νήμα που δημιουργείτε ορίζετε η προτεραιότητα στο μέγιστο (maximum priority). Ακόμη οι θέσεις μνήμης για τα frames που αποθηκεύονται είναι προκαθορισμένες έτσι ώστε να υπάρχει ελάχιστη χρησιμοποίηση μνήμης και να είναι όσο το δυνατόν πιο γρήγορο. Αν χρειαστεί θα χαθούν-ελευθερωθούν μερικά frames προκειμένου να μην υπάρχει καθυστέρηση στο πρόγραμμα μας.

Επίσης, τα frames αποθηκεύονται με scheduling για να συγχρονιστούν όταν το νήμα τελειώσει με το decoding του audio και video. Ακόμη, οι synchronized μεθόδοι χρησιμοποιούνται για να γίνει η αποκωδικοποίηση γρήγορα από το realtime thread και να υπάρξουν όσο γίνεται λιγότερες απώλειες των frame της ταινίας.

Αυτό που το διαφοροποιεί από αυστηρά συστήματα πραγματικού χρόνου: Στα hard δεν ορίζουμε critical real time threads ούτε συγκεκριμένη μνήμη που χρησιμοποιεί το κάθε thread. Έτσι εάν το σύστημα ξεπερνούσε την μνήμη αυτή θα «κατάστρεφε» τα threads μας και η ταινία θα τερματιζόταν.

Τέλος όπως αναφέραμε πιο πάνω το πρόγραμμα μας μπορεί να έχει απώλειες σε frames σε περίπτωση που γεμίσει η ουρά (queue) ή η μνήμη. Έτσι το πρόγραμμα μας σε περίπτωση καθυστέρησης θα ελευθερώσει-απορρίψει μερικά frames αλλά θα συνεχίσει να λειτούργει κανονικά και να αναπαραγάγει την ταινία. Οπότε η μη ικανοποίηση της προθεσμίας οδηγεί σε μειωμένη απόδοση αλλά η λειτουργιά συνεχίζει κανονικά και δεν οδηγείτε σε αποτυχία όπως στο hard real time system.

Ξεφεύγοντας από την αναφορά της Real Time Java και μιλώντας πιο γενικά, το σύστημα αναπαραγωγής βίντεο ανήκει στην κατηγορία των χαλαρών συστημάτων πραγματικού χρόνου επειδή έχει και τα εξής χαρακτηριστικά:

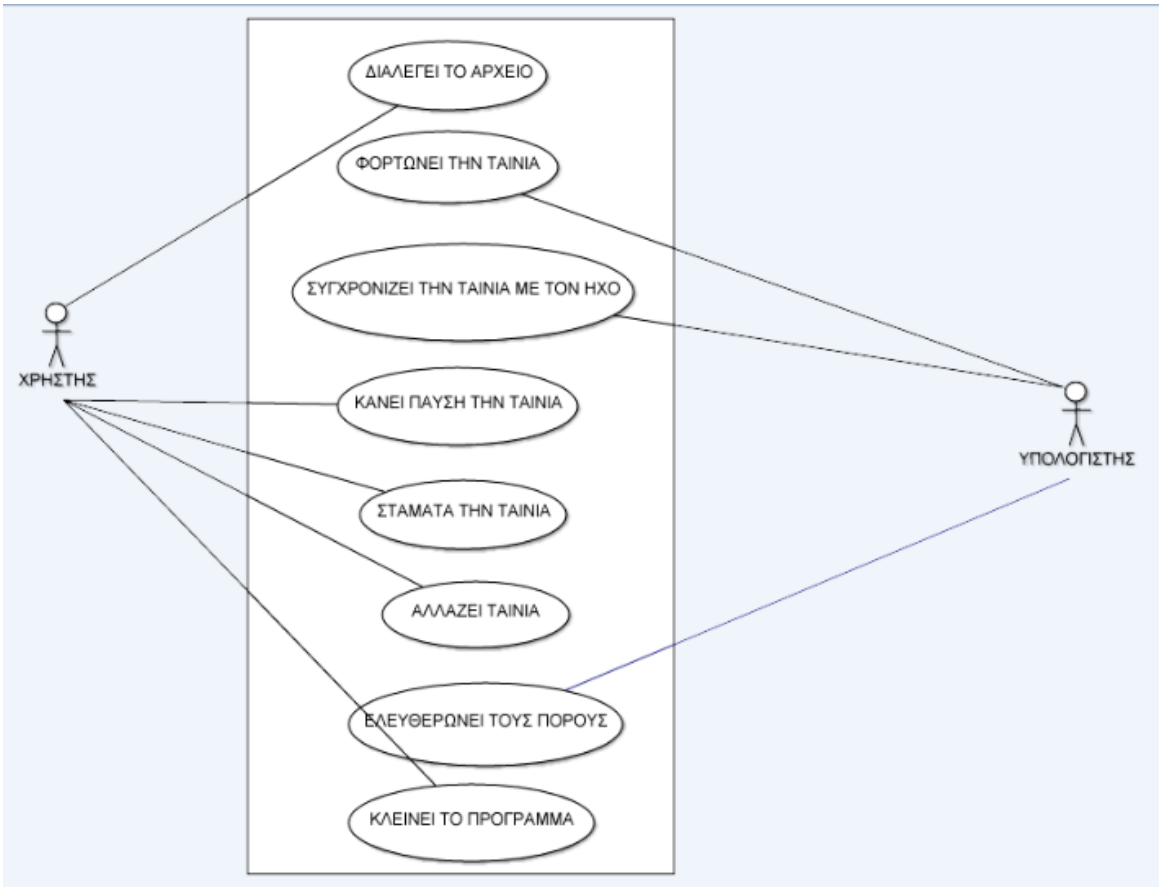
- ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ →ΠΟΛΥ ΧΑΛΑΡΟΣ
- ΑΣΦΑΛΕΙΑ → ΟΧΙ ΚΡΙΣΙΜΗ
- ΜΕΓΕΘΟΣ ΑΡΧΕΙΩΝ ΔΕΔΟΜΕΝΩΝ → ΜΕΓΑΛΑ
- ΑΚΑΙΡΕΟΤΗΤΑ ΔΕΔΟΜΕΝΩΝ → ΜΑΚΡΟΠΡΟΘΕΣΜΑ
- ΑΝΙΧΝΕΥΣΗ ΣΦΑΛΜΑΤΩΝ → ΜΕ ΒΟΗΘΕΙΑ ΤΟΥ ΧΡΗΣΤΗ

- ΑΝΟΧΗ → ΑΠΟΚΡΥΨΗ ΣΦΑΛΜΑΤΩΝ
- ΑΞΙΟΠΙΣΤΙΑ→ΟΧΙ ΚΑΤΑΣΤΡΟΦΙΚΕΣ ΕΠΙΠΤΩΣΕΙΣ
- ΠΕΡΙΟΔΙΚΟΤΗΤΑ→ ΙΣΟΧΡΟΝΗ ΛΕΙΤΟΥΡΓΙΑ
- ΠΡΟΣΑΡΜΟΓΗ→ΜΕΙΩΣΗ ΠΟΙΟΤΗΤΑΣ

5.4 UML διαγράμματα

Στο υποκεφάλαιο αυτό θα δούμε πως τελικά μοντελοποιήσαμε την συμπεριφορά αυτού του χαλαρού συστήματος πραγματικού χρόνου, χρησιμοποιώντας την UML.

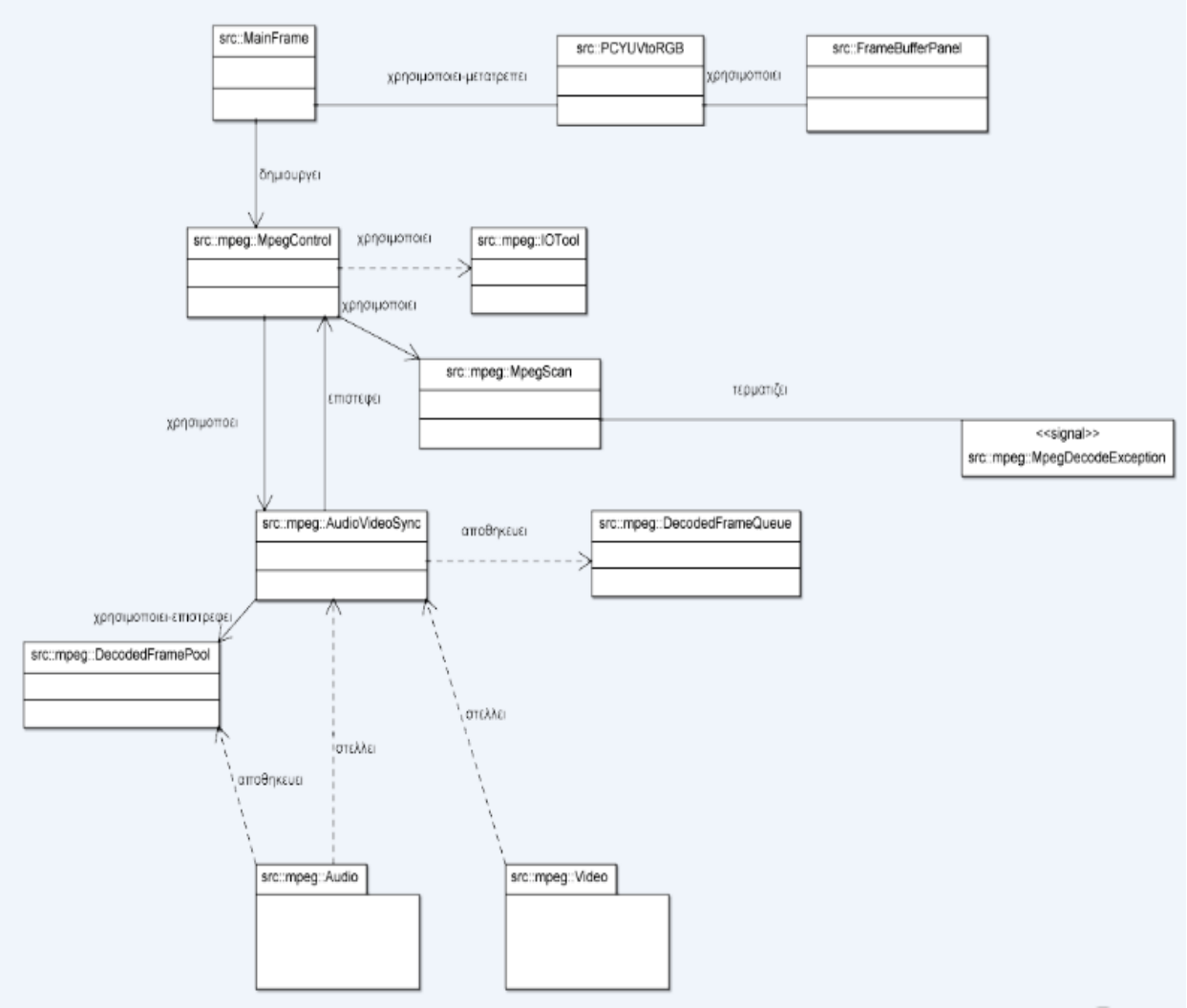
Use case diagram:



Σχήμα 5.1

Οι δύο actors της εφαρμογής είναι ο χρήστης (αυτός που χρησιμοποιεί την εφαρμογή) και ο υπολογιστής. Ο χρήστης μπορεί να διαλέξει το αρχείο που θέλει να παίξει, να κάνει παύση την ταινία, να σταματήσει την ταινία, να αλλάξει ταινία και να κλείσει το πρόγραμμα. Ο υπολογιστής φορτώνει την ταινία, συγχρονίζει την ταινία με τον ήχο και ελευθερώνει τους πόρους που δεσμεύτηκαν.

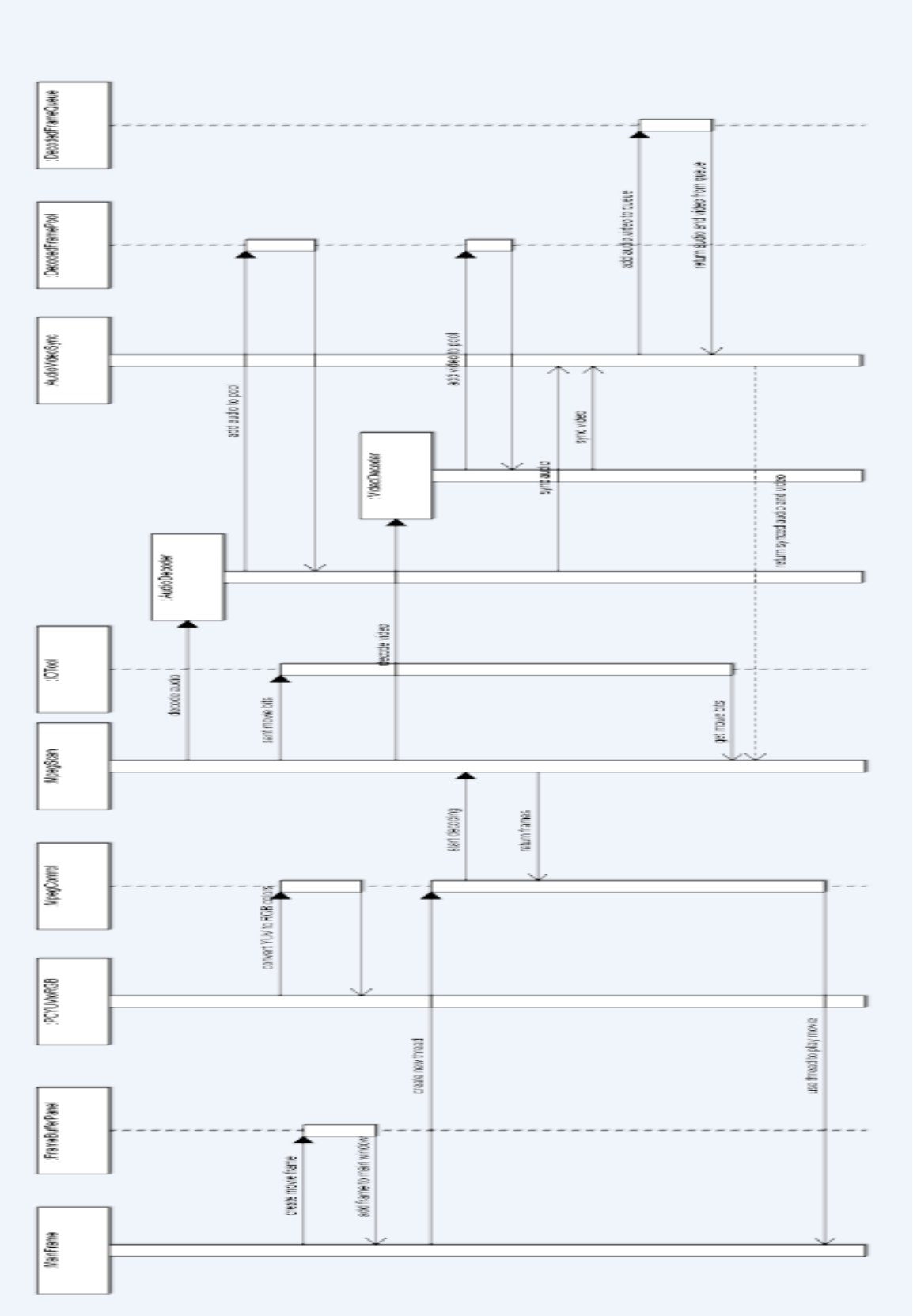
Class diagram:



Σχήμα 5.2

Εδώ παρουσιάζονται τα συστατικά στοιχεία του συστήματος με τη χρήση του διαγράμματος κλάσεων (class diagram). Οι κύριες κλάσεις του συστήματος είναι οι εξής: MainFrame, MpegControl, IOToll, MpegScan, AudioVideoSync, DecodedFrameQueue, DecodedFramePool, FrameBufferPanel.

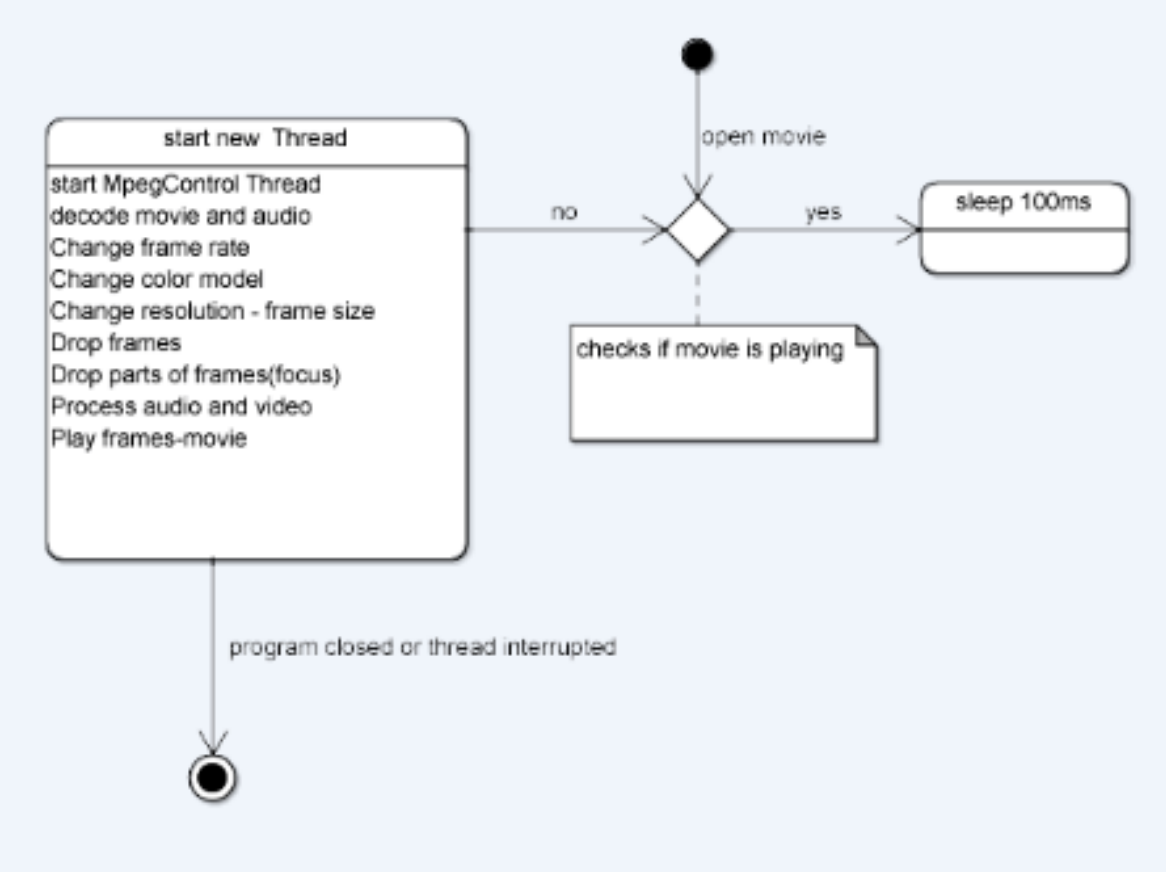
Sequence diagram:



Σχήμα 5.3

Στο πιο πάνω διάγραμμα απεικονίζεται το διάγραμμα ακολουθίας του συστήματος όπου περιγράφεται πως τα αντικείμενα αλληλεπιδρούν μεταξύ τους. Δείχνει πως στέλνονται και λαμβάνονται τα μηνύματα ανάμεσα στα αντικείμενα. Ο κάθετος άξονας δείχνει το χρόνο και ο οριζόντιος είναι το σύνολο των αντικειμένων του συστήματος. Έτσι βλέπουμε την ακολουθία των μηνυμάτων με την πάροδο του χρόνου.

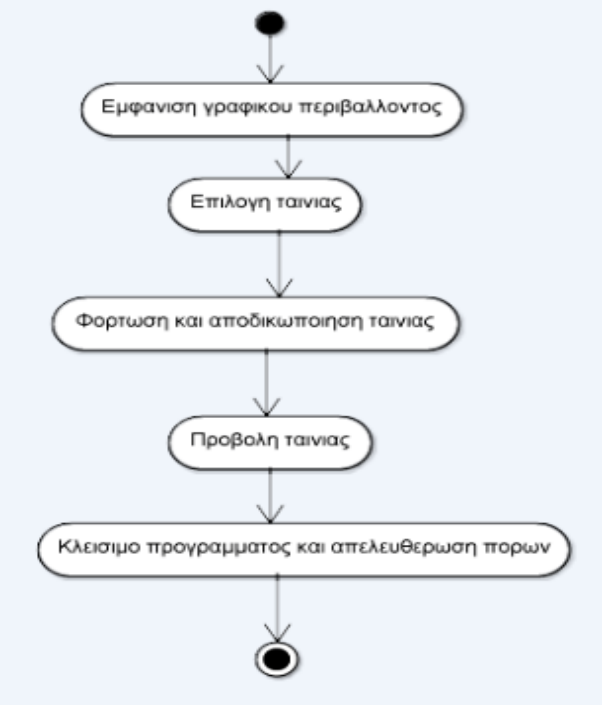
Statechart diagram:



Σχήμα 5.4

Στο πιο πάνω διάγραμμα απεικονίζεται ένα διάγραμμα καταστάσεων όπου περιγράφεται η συμπεριφορά κάποιας κλάσης. Εστιάζει στο πώς τα αντικείμενα αλλάζουν καταστάσεις στο χρόνο ανάλογα με το τί συμβαίνει: γεγονότα, συμπεριφορά και δραστηριότητες μέσα στις καταστάσεις, και μεταβάσεις. Ένα γεγονός μπορεί να είναι η επαλήθευση μιας συνθήκης, η λήψη ενός μηνύματος ή απλά η πάροδος του χρόνου. Εδώ βλέπουμε πως όταν ανοίγει μια ταινία δημιουργείται και ξεκινά ένα καινούριο νήμα και γίνονται συγκεκριμένα όλα αυτά που περιγράφονται στο σχήμα

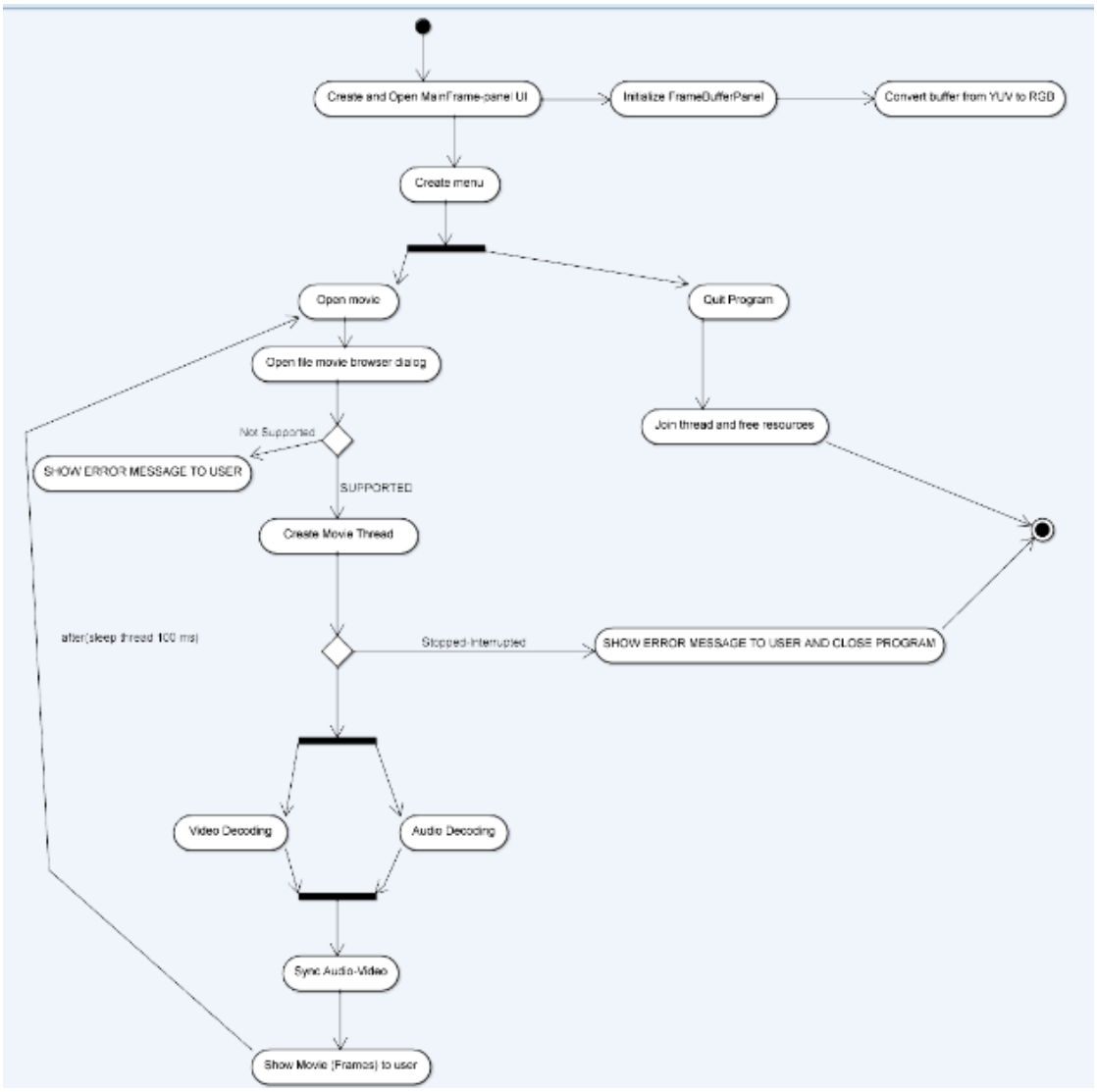
Activity diagrams:



Σχήμα 5.5

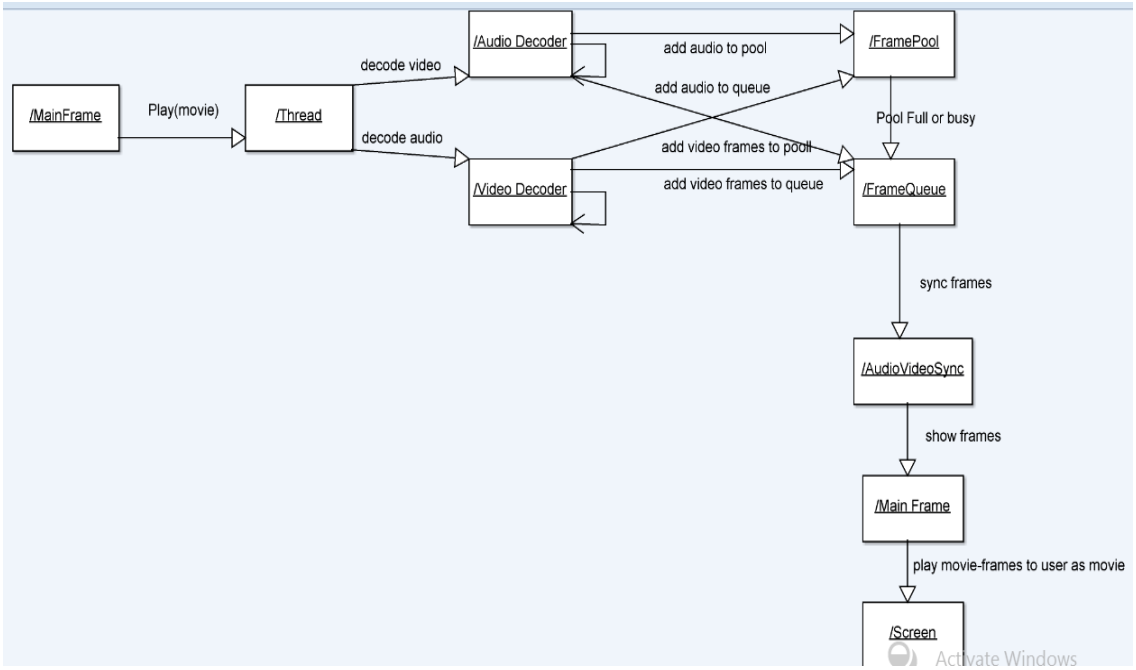
Στο σχήμα 5.5 και 5.6 παρουσιάζονται διαγράμματα δραστηριοτήτων. Στα εν λόγω διαγράμματα καταγράφονται οι ενέργειες που εφαρμόζονται κατά την εκτέλεση μιας λειτουργίας. Αποτελούν δηλαδή στιγμιότυπο της υλοποίησης κάποιας λειτουργίας. Πιο συγκεκριμένα στο σχήμα 5.5 απεικονίζεται το γενικό διάγραμμα δραστηριοτήτων του συστήματος. Απεικονίζεται η δραστηριότητα κατά την οποία όταν ο χρήστης ανοίξει την εφαρμογή αναπαραγωγής βίντεο τότε εμφανίζεται το γραφικό περιβάλλον της εφαρμογής. Ο χρήστης επιλέγει την ταινία που θέλει να δει. Αμέσως η ταινία φορτώνει και αποκωδικοποιείται. Η ταινία προβάλλεται και τέλος η εφαρμογή κλείνει και όλοι οι πόροι που χρησιμοποιήθηκαν απελευθερώνονται.

Το σχήμα 5.6 (που απεικονίζεται αμέσως μετά), αποτελεί το πιο λεπτομερές διάγραμμα δραστηριοτήτων του συστήματος. Δηλαδή περιγράφεται πιο αναλυτικά πως γίνονται οι διάφορες δουλειές που πρέπει να εκτελεστούν,

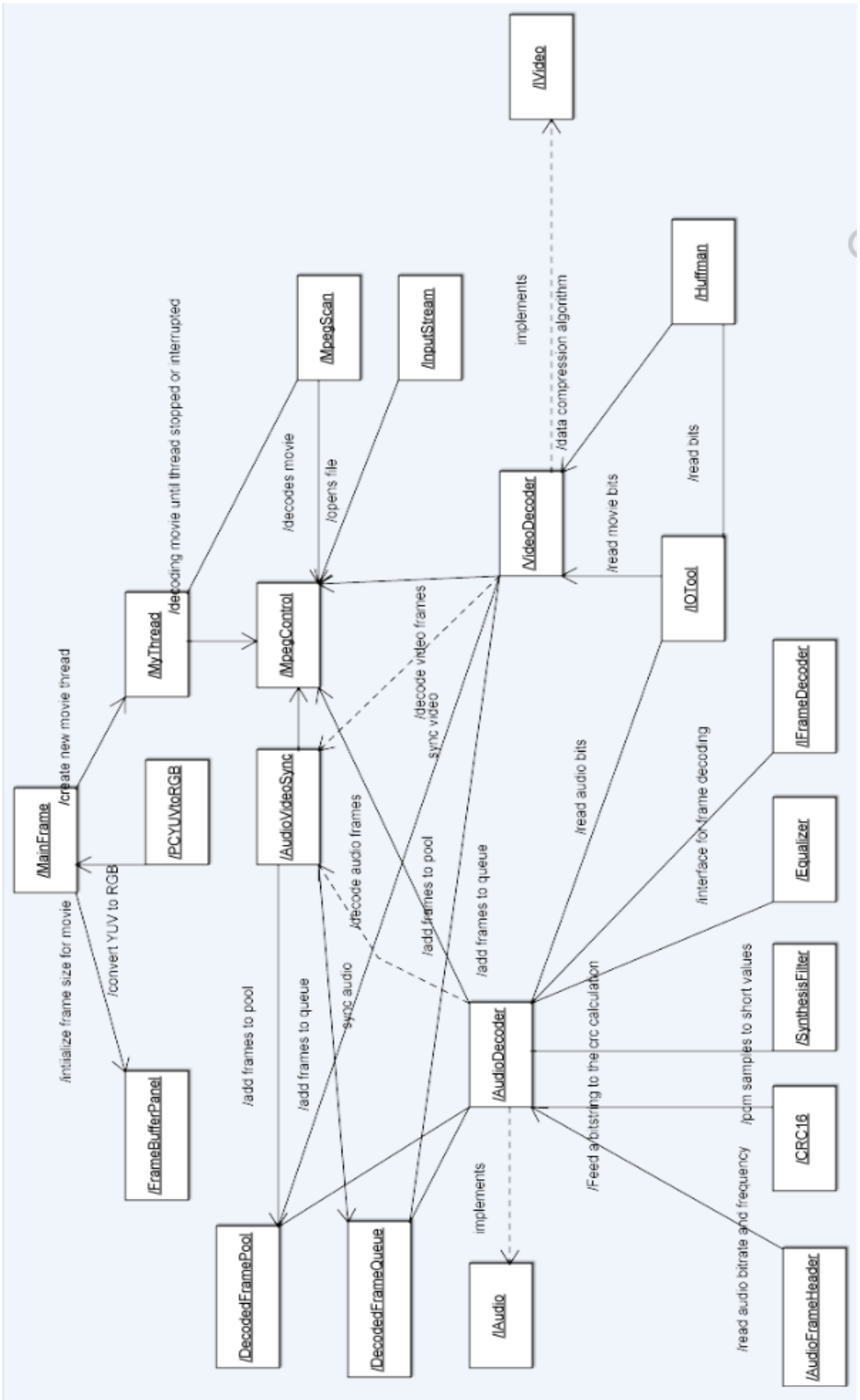


Σχήμα 5.6

Collaboration diagrams:



Σχήμα 5.7


/IVideo
/Huffman
/MpegScan
/InputStream
implements
/data compression algorithm
/decodes movie
/opens file
/read bits
/VideoDecoder
/read movie bits
/IOTool
/MyThread
/decoding movie until thread stopped or interrupted
/MpegControl
/decode video frames
sync video
/create new movie thread
/MainFrame
/PCYUVtoRGB
/AudioVideoSync
/FrameDecoder
/Equalizer
/SynthesisFilter
/CRC16
/AudioFrameHeader
/read audio bits
/interface for frame decoding
/decode audio frames
/add frames to pool
/add frames to queue
sync audio
/AudioDecoder
/Feed a bitstring to the crc calculation
/pcm samples to short values
/read audio bitrate and frequency
/Initialize frame size for movie
/convert YUV to RGB
/FrameBufferPanel
/DecodedFramePool
/DecodedFrameQueue
implements
/IAudio

Σχήμα 5.8

Στο σχήμα 5.7 και στο σχήμα 5.8 φαίνονται τα διαγράμματα συνεργασίας (collaboration diagrams). Σε αυτά τα διαγράμματα περιγράφεται πως αλληλεπιδρούν τα αντικείμενα στο χώρο. Αυτό σημαίνει πως εκτός από τη δυναμική αλληλεπίδραση των αντικειμένων, τα διαγράμματα συνεργασία, δείχνουν επίσης και πως συνδέονται μεταξύ τους τα αντικείμενα.

Κεφάλαιο 6

Συμπεράσματα

6.1 Συμπεράσματα

Στα πλαίσια της παρούσας διπλωματικής εργασίας μελετήσαμε τα συστήματα πραγματικού χρόνου. Ένα σύστημα με απαιτήσεις πραγματικού χρόνου δεν έχει μόνο την ευθύνη να παράγει σωστά αποτελέσματα, αλλά έχει και την ευθύνη να ικανοποιεί συγκεκριμένους χρονικούς περιορισμούς.

Υπάρχουν διάφορες μεθοδολογίες και τρόποι ανάλυσης και σχεδιασμού συστημάτων πραγματικού χρόνου. Η δημοφιλέστερη γλώσσα μοντελοποίησης, ανάλυσης και σχεδιασμού συστημάτων είναι η UML, την οποία χρησιμοποιήσαμε σε αυτή την εργασία για να αναλύσουμε δύο περιπτώσεις συστημάτων πραγματικού χρόνου (σύστημα ελέγχου ανελκυστήρα, video player). Χρησιμοποιήσαμε τους κανόνες σχεδιασμού συστημάτων πραγματικού χρόνου και σχεδιάσαμε τα απαραίτητα διαγράμματα για κάθε σύστημα ξεχωριστά. Κάθε διάγραμμα UML είναι απλά μια γραφική αναπαράσταση μερικών από τις πτυχές του συστήματος και πρέπει πολλά διαγράμματα να συνδυαστούν μαζί για να περιγράψουν πλήρως το σύστημα.

Η πολυπλοκότητα της συμπεριφοράς των συστημάτων πραγματικού χρόνου κάνει την ανάλυση και τη σχεδίαση τους ένα περίπλοκο πρόβλημα. Το πρόβλημα αυτό το αντιμετωπίσαμε στην πράξη εφαρμόζοντας κανόνες αντικειμενοστραφούς ανάλυσης και σχεδίασης με χρήση της UML. Δηλαδή, μέσα από τις περιπτώσεις μελέτης προσπαθήσαμε να

περιγράψουμε κάποια βήματα για να έχουμε μια αποδοτική λύση. Με αφορμή λοιπόν, τις περιπτώσεις μελέτης εξετάσαμε ζητήματα που συνδέονται ειδικότερα με την ανάλυση και τη σχεδίαση real- time systems. Για παράδειγμα, για την περιγραφή των χρονικών περιορισμών, η UML μας παρέχει τα διαγράμματα ακολουθίας και τα διαγράμματα συνεργασίας, τα οποία είναι σε θέση να προσδιορίσουν τη λειτουργία του συστήματος σε πραγματικό χρόνο μέσω της σήμανσης χρονικών περιορισμών (στην άκρη από το όνομα του μηνύματος και του αντικειμένου).

Επιπρόσθετα υλοποιήσαμε κάποιο κώδικα σε γλώσσα προγραμματισμού Java, για το σύστημα ελέγχου ανελκυστήρα. Ο κώδικας προσπαθήσαμε να εμπεριέχει στοιχεία συστημάτων πραγματικού χρόνου και αυτό επιτεύχθηκε με την χρήση νημάτων, συγχρονισμένων μεθόδων, προσθήκη timers (χρονικοί περιορισμοί). Επίσης, για την εφαρμογή αναπαραγωγής βίντεο μελετήσαμε κώδικα σε Real Time Java (RTJ) και τότε καταλήξαμε σε κάποια συμπεράσματα.

Η Java δεν μπορεί να εγγυηθεί προβλέψιμη εκτέλεση των εφαρμογών και επομένως δεν μπορεί να χρησιμοποιηθεί σε μεγάλο εύρος συστημάτων πραγματικού χρόνου. Αντιθέτως, η Real Time Java παρέχει κάποιους μηχανισμούς που βοηθούν στον προγραμματισμό της εκτέλεσης των εργασιών του συστήματος ούτως ώστε όλες οι εργασίες να εκτελούνται μέσα στα προδιαγεγραμμένα χρονικά περιθώρια. Επίσης η RTJ εγγυάται προβλέψιμη εκτέλεση. Αυτό σημαίνει ότι οι χρονικές απαιτήσεις ικανοποιούνται πάντα. Επομένως η RTJ μπορεί να χρησιμοποιηθεί για την ανάπτυξη αυστηρών και χαλαρών συστημάτων πραγματικού χρόνου.

6.2 Μελλοντική εργασία

Τα συστήματα που μελετήσαμε έχουν μεγάλη πολυπλοκότητα όσο πολλαπλασιάζονται οι απαιτήσεις. Έτσι η περαιτέρω ανάλυση των συστημάτων που μας απασχόλησαν θα ήταν χρήσιμη. Παρόλα αυτά, η παρούσα διπλωματική εργασία θα μπορούσε να αποτελέσει μια βάση για μια τέτοια ανάλυση.

Βιβλιογραφία

[1] Booch, G.: Object-Oriented Analysis and Design with Applications, Benjamin/Cummings, 1994.

[2] Rumbaugh, J., et al.: Object-Oriented Modeling and Design, Prentice Hall, 1991.

[3] Jacobson, I., et al.: Object-Oriented Software Engineering, Addison-Wesley, 1992.

[4] Booch, G. and Rumbaugh, J.: "Unified Method for Object-Oriented Development," Documentation Set Version 0.8, October 1995.

[5] Booch, G. and Jacobson, I. and Rumbaugh, J.: "The Unified Modeling Language for Object-Oriented Development," Documentation Set Version 0.91 Addendum UML Update, September 1996.

[6] OMG Unified Language Specification, version 1.3, June 1999

[7] Χατζηγεωργίου Α., Αντικειμενοστρεφής Σχεδίαση, Κλειδάριθμος 2005

[8] Θραμπουλίδης Κ., Ανάλυση και Σχεδιασμός Συστημάτων Λογισμικού, Εκδόσεις Πανεπιστημίου Πατρών 2007

[9] A.Gherbi and F.Khendek, UML Profiles for Real-Time Systems and their Applications, 2006

[10] B.Horne and J.Urban, Formal Specification For Real-Time Object Oriented Systems With UML Design, 2012

[11] J.Kratz, Unified Modeling Language for Real-Time Systems Development

[12] T.Method and R.With, Designing Real-Time and Embedded Systems with the COMET/UML method, 2001

[13] B.Douglass, Real - Time UML

[14] L.Vanzandt, Developing Real-time Systems using Real-time UML

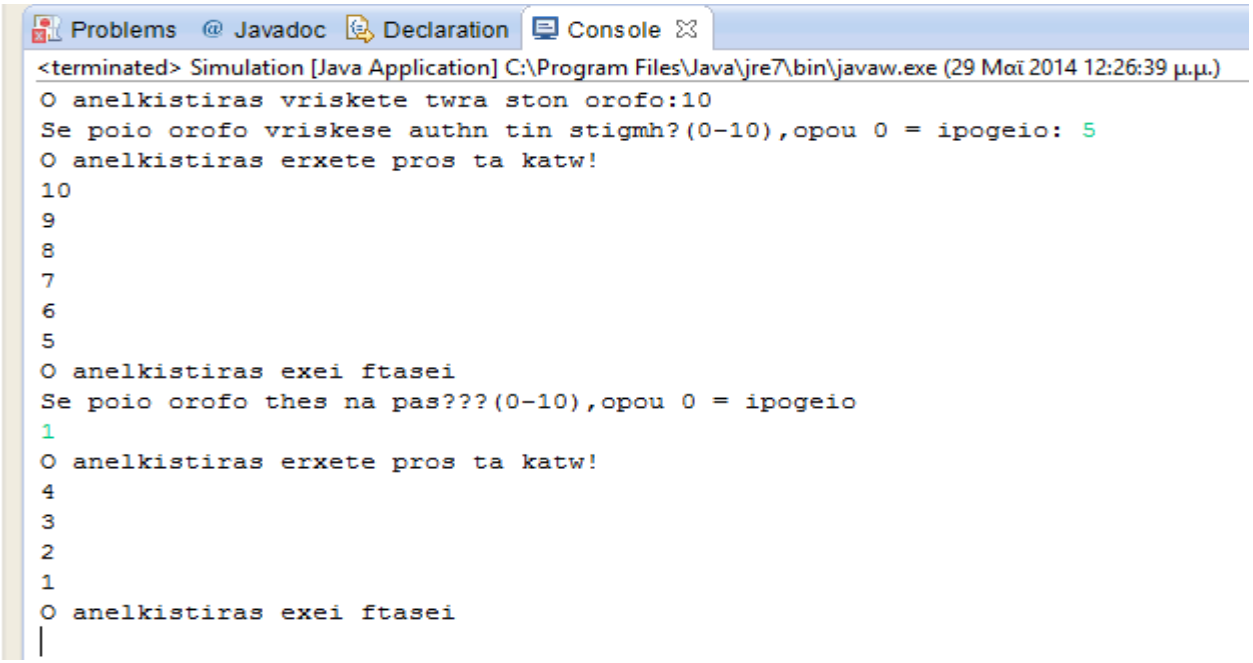
[15] P.Laplante, Real-Time Systems Design and Analysis, 2004

[16] F.Strobl and A.Wisspeitner, Spesification of an Elevator Control System

[17] P.Software, Modeling Systems with UML ©, 1998

Παράρτημα Α

Παράδειγμα εκτέλεσης και κώδικας για μελέτη περίπτωσης 1: Σύστημα ελέγχου ανελκυστήρα (Κεφάλαιο 4)




```
O anelkistiras vriskete twra ston orofo:4
Se poio orofo vriskese authn tin stigmh?(0-10),opou 0 = ipogeio: 4
Mpes mesa ston anelkistira!
Se poio orofo thes na pas???(0-10),opou 0 = ipogeio
8
O anelkistiras erxete pros ta panw!
4
5
6
7
8
O anelkistiras exei ftasei
```

Person.java

```
import java.util.*;

public class Person extends Thread {
	private int starting_floor;
	private int arrived_floor;
	private int time_entered;
	private int time_left;
	private int ID = -1;

	// represents whether Person is moving or waiting
	private boolean moving;

	// reference to Location (either on Floor or in Elevator)
	private Location location;

	 // time in milliseconds to walk to Button on Floor
	 private static final int TIME_TO_WALK = 3000;



	//constructor
	public Person(int entering_time, int starting_floor, int arrived_floor){
		time_entered=entering_time;
		time_left=-1;
		this.starting_floor=starting_floor;
		this.arrived_floor=arrived_floor;
	}

	// Person constructor set initial location
	  public Person( int identifier, Location initialLocation ){
	    super();
	    ID = identifier; // assign unique identifier
	    location = initialLocation; // set Floor Location
```

```
            moving = true; // start moving toward Button on Floor
          }


        public int get_starting_floor(){
                return starting_floor;
        }

        public int get_arrived_floor(){
                return arrived_floor;
        }

        public int get_time_entered(){
                return time_entered;
        }

        public void set_leaving_time(int leaving_time) {
           time_left=leaving_time;
         }

        public int get_waiting_time() {
           if(time_left<0)
             return 0;
           return (time_left-time_entered);
         }

// set Person Location
private void setLocation( Location newLocation ){
  location = newLocation;
}

// get current Location
private Location getLocation(){
  return location;
}
```

```
// get identifier
public int getID(){
   return ID;
}

// set if Person should move
public void setMoving( boolean personMoving ){
   moving = personMoving;
}

// get if Person should move
public boolean isMoving(){
   return moving;
}

// Person either rides or waits for Elevator
public void run(){

   // walk to Elevator
   pauseThread( TIME_TO_WALK );

   // stop walking at Elevator
   setMoving( false );

   // get Door on current Floor
   Door currentFloorDoor = location.getDoor();

   // begin exclusive access to currentFloorDoor
   synchronized ( currentFloorDoor ) {

      // check whether Floor Door is open
      if ( !currentFloorDoor.isDoorOpen() ) {

         pauseThread( 1000 );
```

```
         // press Floor's Button to request Elevator
         Button floorButton = getLocation().getButton();
        // floorButton.pressButton( getLocation() );
      }

      // wait for Floor door to open
      try {

         while ( !currentFloorDoor.isDoorOpen() )
            currentFloorDoor.wait();
      }

      // handle exception waiting for Floor door to open
      catch ( InterruptedException interruptedException ) {
         interruptedException.printStackTrace();
      }

      // Floor Door takes one second to open
      pauseThread( 1000 );
      // waiting for Elevator's Door to open takes a second
      pauseThread( 1000 );

      // walk away from Elevator
      setMoving( true );
      }
}

// pause thread for desired number of milliseconds
private void pauseThread( int milliseconds ){
   try {
      sleep( milliseconds );
   }

   // handle exception if interrupted when paused
```

```
    catch ( InterruptedException interruptedException ) {
      interruptedException.printStackTrace();
    }
  } // end method pauseThread

}
```

Floor.java

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Floor extends Location {

          private int number;
            // distance from the ground
            private float ceiling;  // relative to the floor's height
            private List carEntrances = new ArrayList();

            public Floor(int number, float height, float ceiling){
              this.number = number;
              this.ceiling = ceiling;
            }
  // Floor constructor sets name of Floor
  public Floor( String name ){
    setLocationName( name );
  }

  public int getFloorNumber(){
   return number;
  }

  public float getCeiling(){
   return ceiling;
```

```
  }

@Override
public Button getButton() {
        // TODO Auto-generated method stub
        return null;
}

@Override
public Door getDoor() {
        // TODO Auto-generated method stub
        return null;
}
}
```

Button.java

```
public class Button  {
  // metavliti gia to an exei patithei to koumpi
  private boolean pressed = false;

  // epistrefei an patithike to koumpi
  public boolean isButtonPressed(){
    return pressed;
  }

}
```

Door.java

```
import java.util.*;

public class Door  {

  // metavlites gia to an i porta einai anoixti h einai kleisti
```

```
  private boolean open = false;
  private boolean close = false;

  // xronos prin i porta klisei aytomata
  public static final int AUTOMATIC_CLOSE_DELAY = 3000;



  // h topothesia opou i porta anoigei h kleinei
  private Location doorLocation;



  // epistrefei an i porta einai anoixti
  public synchronized boolean isDoorOpen(){
    return open;
  }

// epistrefei an i porta einai kleisti
  public synchronized boolean isDoorClose(){
    return close;
  }

}
```

Simulation.java

```
import java.awt.geom.*;
import java.util.Scanner;
public class Simulation {

        static int floor = 0;
        static int choice1;
        static int person = 0;

        public static void main(String args[])
        {
```

```
        Scanner keyboard = new Scanner(System.in);
        floor = ((int) (Math.random() * 10 + 1));

        System.out.println("O anelkistiras vriskete twra ston orofo:" +floor);
        System.out.print("Se poio orofo vriskese authn tin stigmh?(0-10),opou 0 = ipogeio: ");
        choice1 =keyboard.nextInt();

        if(floor == choice1){
            System.out.println("Mpes mesa ston anelkistira!");
        }

        else if(floor > choice1){
            ElevatorDown();
        }

        else if(floor < choice1){
            ElevatorUp();
        }

        System.out.println("Se poio orofo thes na pas???(0-10),opou 0 = ipogeio");
        choice1 = keyboard.nextInt();

        if(floor > choice1){
            ElevatorDown();
        }

        else if(floor < choice1){
            ElevatorUp();
        }

    }

    public static void ElevatorUp(){
        System.out.println("O anelkistiras erxete pros ta panw!");
```

```
            for (person = choice1; choice1>=floor; floor++)

                System.out.println(floor);

            System.out.println("O anelkistiras exei ftasei");
        }

        public static void ElevatorDown(){
            System.out.println("O anelkistiras erxete pros ta katw!");
            for (person = choice1; choice1<=floor; floor--)

                System.out.println(floor);

            System.out.println("O anelkistiras exei ftasei");
        }
    }
```

Location.java

```
public abstract class Location {

  // name of Location
  private String locationName;

  // set name of Location
  protected void setLocationName( String name ){
    locationName = name;
  }

  // return name of Location
  public String getLocationName(){
    return locationName;
  }
```

```
   // return Button at Location
   public abstract Button getButton();

   // return Door object at Location
   public abstract Door getDoor();
}
```

Light.java

```
public class Light{

   // Light state (on/off)
   private boolean lightOn;
   // time before Light turns off automatically (3 seconds)
   public static final int AUTOMATIC_TURNOFF_DELAY = 3000;
   // location where Light turned on or off
   private Location lightLocation;

   // turn on Light
   public void turnOnLight( Location location )
   {
      if ( !lightOn ) {
         lightOn = true;
         lightLocation = location;

         // declare Thread that ensures automatic Light turn off
         Thread thread = new Thread(
            new Runnable() {

               public void run()
               {
                  // turn off Light if on for more than 3 seconds
                  try {
                     Thread.sleep( AUTOMATIC_TURNOFF_DELAY );
                     turnOffLight( lightLocation );
```

```
            }

            // handle exception if interrupted
            catch ( InterruptedException exception ) {
              exception.printStackTrace();
            }
          }
        } // end anonymous inner class
      );

      thread.start();
    }
  } // end turnOnLight

  // turn off Light
  public void turnOffLight( Location location ){
    if ( lightOn ) {

      lightOn = false;
    }
  } // end method turnOffLight

  // return whether Light is on or off
  public boolean isLightOn(){
    return lightOn;
  }
}
```

Elevator.java

```
import java.util.*;

public class Elevator {

  // diaxirizetai to thread tou elevator
```

```
private boolean elevatorRunning = false;

// epistrefoun tin katastash tou elevator (idle or moving)
private boolean moving = false;
private boolean idle=false;

// current Floor
private Location currentFloorLocation;

// destination Floor
private Location destinationFloorLocation;

// Elevator needs to service other Floor
private boolean summoned;

private Button elevatorButton;
public static final int ONE_SECOND = 1000;

//o xronos pu xreiazetai gia na taxidepsei anamesa stous orofous (5 seconds)
private static final int TRAVEL_TIME = 5 * ONE_SECOND;

// Elevator's thread to handle asynchronous movement
private Thread thread;

// constructor dimiourgia metavliton
public Elevator( Floor firstFloor, Floor secondFloor ){

  elevatorButton = new Button();

  // start Elevator on first Floor
  currentFloorLocation = firstFloor;
  destinationFloorLocation = secondFloor;

} // end constructor
```

```
// swaps current Floor Location with opposite Floor Location
private void changeFloors(){
  Location location = currentFloorLocation;
  currentFloorLocation = destinationFloorLocation;
  destinationFloorLocation = location;
}

// start Elevator thread
public void start(){
  elevatorRunning = true;
  thread.start();
}

// stop Elevator thread
public void stopElevator(){
  elevatorRunning = false;
}

// Elevator thread's run method
public void run(){
  while ( isElevatorRunning() ) {

    // remain idle until awoken
    while ( !isMoving() )
      pauseThread( 10 );

    // pause while passenger exits (if one exists)
    pauseThread( ONE_SECOND );

    // closing Door takes one second
    pauseThread( ONE_SECOND );

    // Elevator needs 5 seconds to travel
    pauseThread( TRAVEL_TIME );
```

```
      // stop Elevator
      setMoving( false );

      // swap Floor Locations
      changeFloors();

   } // end while

} // end run

// pause concurrent thread for number of milliseconds
private void pauseThread( int milliseconds ){
   try {
      Thread.sleep( milliseconds );
   }

   // handle if interrupted while sleeping
   catch ( InterruptedException exception ) {
      exception.printStackTrace();
   }
} // end pauseThread

// return Button on Elevator
public Button getButton(){
   return elevatorButton;
}

// set if Elevator should move
private void setMoving( boolean elevatorMoving ){
   moving = elevatorMoving;
}

// is Elevator moving?
public boolean isMoving(){
   return moving;
```

```
  }

  // is Elevator thread running?
  private boolean isElevatorRunning(){
    return elevatorRunning;
  }

  // get the currentFloorLocation of the Elevator
  public Location getCurrentFloor(){
    return currentFloorLocation;
  }
}
```

ElevatorController.java

```
import java.util.*;

public class ElevatorController {
  //statheres
  private static final int DEFAULT_HOW_MANY_FLOORS = 9;
  private static final int DEAFAULT_HOW_MANY_PEOPLE_MAX = 30;
  private static final int DEAUFULT_DELAY_BETWEEN_STEPS_MILLI = 600;
  private static final int RANDOM_SEED=100;
  private static final int PEOPLE_INCREMENT_COUNT=3;
  private static final int INCREMENT_INTERVAL=10;
  private static final int ELEVATOR_CAPACITY=8;


  private int clockTicks;            //clock ticks passed till now
  private int elevatorFloor;         //current floor of the elevator
  private int howManyFloors;         //number of floors in the building
  private int howManyPeopleMax;      //maximum number of people in simulation
  private int delayBetweenStepsMilli; //delay for clock ticks
  private Random rgenerator;         //random number generator to assist in simulation
  private ArrayList allPeopleList;   //list of all the people entered in simulation till now
```

```
private ArrayList floors;    //an arraylist of arraylists - to keep an account of people
                             // waiting at each particular floor. ith element of this arraylist
                             // is an arraylist which is a list of people waiting at the (i+1) floor
private ArrayList elevatorList;     //list of people currently in the elevator


/** get the current floor that the elevator is on */
public int getCurrentFloor() {
  return elevatorFloor;
}

/** get the highest floor number */
public int getMaxFloor() {
  return howManyFloors;
}

/** get the lowest floor number */
public int getMinFloor() {
  return 1;
}

/** Get the current clockTicks */
public int getClockTicks() {
  return clockTicks;
}

/** get the number of people still in the system
 * This is the number of people in the elevator and number
 * of people waiting in queues on different floors*/
public int getRemainingPeopleCount() {
  int count = elevatorList.size();    // number of people in elevator
  for(int i=1; i <= howManyFloors; i++) {
    ArrayList people = (ArrayList)(floors.get(i));
    count += people.size();    // number of people waiting on floors
  }
```

```
  return count;
}

/** returns a list of the people waiting at the specified floor f */
public ArrayList getPeopleAtFloor(int f) {
  return (ArrayList)floors.get(f);
}

/** returns a list of the all the people waiting at any floor in the building */
public ArrayList getAllPeopleWaiting() {
  ArrayList waiting = new ArrayList();  //create a new list and put all
                                        //people waiting on the floors in list
  for(int i=1; i<=howManyFloors; i++) {
    Iterator itr = ((ArrayList)floors.get(i)).iterator();
    while(itr.hasNext()) {
      waiting.add((Person)itr.next());
    }
  }
  return waiting;
}

/** This method takes an arrayList of persons waiting for the elevator and
 * returns the person who has had the longest waiting time for the elevator */
public Person getLongestWaitingPerson(ArrayList waitingPeople) {
  Person mostWaiting = null;
  int time;
  int lowestEnteringTime = getClockTicks();
  Iterator itr = waitingPeople.iterator();
  // go through all people waiting and find person who entered the simulation
  // the longest time ago
  while(itr.hasNext()) {
    Person person = (Person)itr.next();
      mostWaiting = person;
    }
```

```
    return mostWaiting;
  }

  /** load person - takes a person object and loads it into the elevator if the
   * person is waiting on the current floor -it returns true if the person was
   * successfully loaded into the elevator, else returns false*/
  public boolean loadPerson(Person person) {
    ArrayList people = (ArrayList)floors.get(elevatorFloor);
    //this means that the person can not be loaded into the elevator as he/she is not in the
waiting queue
    //of the current floor
    if(!people.contains(person)){
      return false;
    }
    people.remove(person);
    elevatorList.add(person);
    // loading a person takes time
    incrementClock();
    delay(1);
    return true;
  }

  /** take the elevator to the specified floor */
  public void moveToFloor(int f) {
    if(f > howManyFloors) { // if floor is not in range, force it to be in range
      f = howManyFloors;
    }
    if(f <= 0) {
      f = 1;
    }

 int direction;
 if(f > elevatorFloor) {
   direction = 1 ;
 } else {
```

```
  direction = -1;
 }

   //moving takes time and the clock is correspondingly incremented.
 //for going one floor up/down, the clok takes one unit of time
 while(elevatorFloor != f) {
 elevatorFloor = elevatorFloor + direction;
 incrementClock();
    System.out.print("Elevator moved to floor:" + elevatorFloor);
    delay(1);
  }
}

 void incrementClock(int units) {
   for(int i=1; i<=units; i++) {
     incrementClock();
   }
 }

 /** increments the clock by one unit*/
 void incrementClock() {
   clockTicks++;
   if(clockTicks%INCREMENT_INTERVAL==0) {
     generatePeople();
   }
 }

 /**calling the elevator algorithm, till there are any people left
  * in the system
  */
 public void generatePeople() {

   if(allPeopleList.size() <= howManyPeopleMax) {
     // A randomly chosen number of people are added to
     // the the queues at one of the randomly chosen floors
```

```
    int newPeopleCount = rgenerator.nextInt(PEOPLE_INCREMENT_COUNT)+1;
    //add the new people to appropriate floor queues
    for(int i=1; i<=newPeopleCount; i++) {
      int floor = rgenerator.nextInt(howManyFloors)+1;
      int targetFloor = rgenerator.nextInt(howManyFloors)+1;
      //we discard this random generation if the waiting floor is
      //same as the targetfloor
      if(floor!=targetFloor) {
        Person newArrival = new Person(floor,targetFloor,getClockTicks());
        ArrayList people = (ArrayList)(floors.get(floor));

        //add the current arrival to appropriate floor list
        people.add(newArrival);
        allPeopleList.add(newArrival);
      }
    }
  }
  delay(1);
 }
 /**introduces i times a fixed amount of sleep */
 public void delay(int i) {
  try {         // 142 students -- you may ignore the try/catch, but
                // this is something you have to look forward to in 143
    Thread.sleep(i * this.delayBetweenStepsMilli);
  } catch (Throwable e) {
  }
 }
}
```