ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Ψηφιακών Συστημάτων
Π.Μ.Σ. ΔΙΚΤΥΟΚΕΝΤΡΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

«ΔΗΜΙΟΥΡΓΙΑ ΣΥΣΤΗΜΑΤΟΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΚΛΙΝΩΝ ΝΟΣΟΚΟΜΕΙΟΥ»

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΝΙΚΟΛΑΟΥ Ι. ΦΙΛΙΟΥ
Α.Μ: ΜΕ/08109

Επιβλέπων : Μαρίνος Θεμιστοκλέους
Επίκουρος Καθηγητής Παν/μίου Πειραιώς

Πειραιάς, Δεκέμβριος 2010

Τίτλος Πτυχιακής Εργασίας: *Δημιουργία συστήματος ηλεκτρονικής διαχείρισης κλινών νοσοκομείου*

Συγγραφέας: *Νικόλαος Ι. Φίλιος*
A.M.: ΜΕ/08109
Email: nikosfilios@yahoo.gr

Πειραιάς, Δεκέμβριος 2010

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Θεμιστοκλέους Μαρίνο για την ανάθεση της διπλωματικής εργασίας, την πολύτιμη βοήθεια του και τη καθοδήγηση του κατά τη διάρκεια αυτής.

Επίσης, θα ήθελα να ευχαριστήσω το συνάδελφο μου από το Ίδρυμα τεχνολογίας και έρευνας καθώς και το υπαλληλικό προσωπικό των κλινικών του ΠΕΡΙΦΕΡΕΙΑΚΟ Γ.Ν. Μαιευτήριο Αθηνών «ΕΛΕΝΑ ΒΕΝΙΖΕΛΟΥ» για τις γνώσεις που μου προσέφεραν και τις συμβουλές για τη συγγραφή της διπλωματικής αυτής εργασίας.

Ακόμη, θα ήθελα να ευχαριστήσω τη κ. Παναγιωτάκη Άρτεμις για τη βοήθεια της στη συγγραφή της εργασίας.

Τέλος την εργασία την αφιερώνω στον αείμνηστο Πατερά μου Ιωάννη Φίλιο που έφυγε νωρίς.

Περίληψη

Το πληροφοριακά συστήματα βρίσκουν ολοένα και αυξανόμενες εφαρμογές στο χώρο της υγείας. Μία από αυτές αποτελεί η ηλεκτρονική διαχείριση κλινών νοσοκομείου, η οποία βοηθάει να γίνεται ταχύτερα και καλυτέρα η νοσηλεία των ασθενών στο νοσοκομείο . Η παρούσα εργασία ασχολείται με το πεδίο αυτό και συγκεκριμένα με το σχεδιασμό, τη μελέτη και την υλοποίηση ενός συστήματος που αφορά την εισαγωγή και την εξαγωγή του ασθενούς από το νοσοκομείο. Η εφαρμογή που θα υλοποιηθεί βασίζεται σε σύγχρονες τεχνολογίες όπως είναι τα Webservices, η Java και η XML. Παράλληλα δε, θα αναλυθούν από θεωρητική σκοπιά όλες οι πτυχές του πεδίου μελέτης, ενώ θα γίνει και μία προσπάθεια αναφοράς στη σημερινή πραγματικότητα στην Ελλάδα.

Σύνοψη

Η παρούσα εργασία αφορά την ανάλυση, το σχεδιασμό και την υλοποίηση μιας εφαρμογής διαχείρισης κλινών νοσοκομείων. Η εφαρμογή αυτή έχει σκοπό τη διαχείριση των κλινών στα διάφορα τμήματα που διαθέτει ένα νοσοκομείο ώστε εύκολα να μπορεί να ορίζεται το σε ποια κλίνη θα φιλοξενηθεί ο κάθε ασθενής που εισάγεται. Παράλληλα το συγκεκριμένο πληροφοριακό σύστημα θα παρέχει τη δυνατότητα εντοπισμού της κλίνης στην οποία φιλοξενείται ο κάθε ασθενής αλλά και το πότε και που είχε φιλοξενηθεί στο παρελθόν.

Η εφαρμογή αυτή θα έχει τρεις κατηγορίες χρηστών. Η πρώτη κατηγορία είναι αυτή του χρήστη που έχει δικαιώματα αναζήτησης διαθέσιμης κλίνης και αφορά συνεργαζόμενα νοσοκομεία που αναζητούν κλίνη για ασθενείς που δεν μπορούν να φιλοξενήσουν, η δεύτερη κατηγορία αφορά γιατρούς που έχουν δικαίωμα να εντοπίζουν σε ποια κλίνη βρίσκεται κάποιος ασθενής και η τελευταία κατηγορία είναι οι διαχειριστές του συστήματος που έχουν δικαίωμα πρόσβασης σε όλες τις λειτουργίες του συστήματος.

Στα πλαίσια της παραπάνω εφαρμογής ικανοποιούνται τα εξής προβλήματα:

1. Οργάνωση των κλινών των νοσοκομείων σε θαλάμους, των θαλάμων σε πτέρυγες και των πτερύγων σε τμήματα.
2. Κατηγοριοποίηση των πτερύγων σε κατηγορίες, ανάλογα με την ποιότητα των παρεχόμενων υπηρεσιών.
3. Κατηγοριοποίηση των ασθενών ανάλογα με την ποιότητα υπηρεσιών που δικαιούνται και τοποθέτηση τους σε κλίνες αντίστοιχου επιπέδου.
4. Εντοπισμό διαθεσιμότητας κλίνης μεταξύ διαφορετικών νοσοκομείων.
5. Καταχώρηση εισαγωγής σε συγκεκριμένη κλίνη.
6. Εξιτήριο από συγκεκριμένη κλίνη

Η εφαρμογή θα παρέχει επίσης τη δυνατότητα στον διαχειριστή να παράγει κάθε είδους αναφορά που ενδεχομένως του ζητηθεί, όπως στατιστικά εισαγωγών ή ιστορικό εισαγωγών συγκεκριμένου ασθενή. Το σύστημα είναι ανεπτυγμένο με χρήση Web Services και δίνει δυνατότητα διαλειτουργικότητας, δηλαδή αξιοποίησης των λειτουργιών του από τρίτες εφαρμογές.

Να σημειωθεί ότι για την υλοποίηση της εφαρμογής διαχείρισης κλινών νοσοκομείου η γλώσσα προγραμματισμού Java έκδοση 6 και τα framework Spring framework για depedency injection, JPA με την υλοποίηση Hibernate για object-relational mapping και το web services stack Metro. Η ανάπτυξη έγινε μέσω του περιβάλλοντος ανάπτυξης NetBeans 6.9 με χρήση web server GlashFish 3. Το σύτημα διαχείρισης βάσεων δεδομένων που χρησιμοποιήθηκε είναι το JavaDB, αλλά θα μπορούσε να είναι κι οποιοδήποτε άλλο σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων που διαθέτει αντίστοιχη υλοποίηση του JDBC API.

Κεφάλαιο 1: Εισαγωγή

1.1 ΠΛΑΙΣΙΟ

Η παρούσα διπλωματική εργασία αναπτύχθηκε κατά την περίοδο από Μάιο μέχρι Δεκέμβριο του 2010 και αφορά την κατασκευή ενός πληροφοριακού συστήματος για την διαχείριση νοσοκομειακών κλινών.

Για την ανάλυση και σχεδίαση του συστήματος που παρουσιάζεται στα κεφάλαια της παρούσας αναφοράς χρησιμοποιήθηκαν τα εργαλεία Microsoft® Visio και NetBeans UML Plug-in. Με την χρήση των συγκεκριμένων εργαλείων σχεδιάστηκαν διαγράμματα ροής δεδομένων, οντοτήτων συσχετίσεων και περιπτώσεων χρήσεων.

Για την υλοποίηση του συστήματος χρησιμοποιήθηκε το περιβάλλον ανάπτυξης NetBeans 6.9, που ενσωματώνει την Java 6 τον Application Server GlashFish 3 και το σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων JavaDB.

1.2 ΣΚΟΠΟΣ ΚΑΙ ΑΝΤΙΚΕΙΜΕΝΟ ΤΗΣ ΕΡΓΑΣΙΑΣ

Ο σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη ενός πληροφοριακού συστήματος για την καλύτερη διαχείριση των νοσοκομειακών κλινών. Ο σκοπός αυτός εξειδικεύεται στους εξής επιμέρους αντικειμενικούς στόχους:

1. Ανασκόπηση Βιβλιογραφίας με στόχο την κατανόηση το τι είναι ένα πληροφοριακό σύστημα νοσοκομείου καθώς και ποιες είναι οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής.
2. Επιλογή κατάλληλης ερευνητικής μεθοδολογίας
3. Συλλογή δεδομένων.
4. Η ανάλυση και σχεδίαση της εφαρμογή όπου στόχος μας είναι να κατανοήσουμε από τεχνική σκοπιά την εφαρμογή.
5. Δημιουργία σεναρίων χρήσης.

1.3 ΠΡΟΒΛΗΜΑ

Η πρώτη ενέργεια που γίνεται όταν ένας ασθενής πρόκειται να νοσηλευθεί είναι η εισαγωγή. Η διαδικασία αυτή φαντάζει απλή, αλλά στην περίπτωση ενός πραγματικού νοσοκομείου με δυνατότητα φιλοξενίας εκατοντάδων ασθενών σε δεκάδες τμήματα δεν είναι καθόλου εύκολη.

Η παρούσα εργασία ασχολείται με την ανάπτυξη ενός πληροφοριακού συστήματος που θα καλύψει αυτή την ανάγκη για οποιουδήποτε μεγέθους νοσοκομείο. Το συγκεκριμένο πληροφοριακό σύστημα θα αναπτυχθεί με χρήση

τεχνολογίας Web Services. Αυτή η τεχνολογία θα δώσει τη δυνατότητα αξιοποίησης της αναζήτησης διαθέσιμου δωματίου από διάφορα συστήματα. Μέσω αυτής της δυνατότητας το γραφείο κίνησης ενός νοσοκομείου θα μπορούσε να εντοπίζει σε πιο άλλο νοσοκομείο υπάρχουν διαθέσιμα κρεβάτια στο ζητούμενο τμήμα, ώστε να μεταφερθεί ο ασθενής σε περίπτωση που υπάρχει έλλειψη στο συγκεκριμένο νοσοκομείο.

Φυσικά δεν θα πρέπει χειριστής από διαφορετικό νοσοκομείο να έχει δικαίωμα δέσμευσης ή αποδέσμευσης κρεβατιού, αλλά μόνο αναζήτησης διαθεσιμότητας.

1.4 ΜΕΘΟΔΟΛΟΓΙΑ ΕΠΙΛΥΣΗΣ

Για την ανάπτυξη του πληροφοριακού συστήματος της παρούσας διπλωματικής εργασίας εφαρμόστηκε η μεθοδολογία του ευέλικτου μοντέλου *(agile software development).* Το ευέλικτο μοντέλο δίνει την δυνατότητα στην ομάδα ανάπτυξη μιας εφαρμογής να την αναπτύσσει σταδιακά και συνεχώς να μπορεί να προσθέτει νέες λειτουργίες που δεν είχαν αρχικά προδιαγραφεί. Η χρήση αυτού του μοντέλου ενέχει τον κίνδυνου υπερβολικής αύξησης του κόστους υλοποίησης ενός πληροφοριακού συστήματος, αλλά από την άλλη δεν ορίζει ποτέ το τέλος της εξέλιξης του. Η φιλοσοφία του ευέλικτου μοντέλου στηρίζεται σε κάποιες αρχές οι οποίες είναι οι εξής:

Υψηλότερη προτεραιότητα αποτελεί η ικανοποίηση του πελάτη η οποία επιτυγχάνεται μέσα από συνεχή παραδοτέα από τις πρώτες κιόλας φάσεις του έργου

Οι αλλαγές στις απαιτήσεις του προς ανάπτυξη συστήματος θα πρέπει να γίνονται δεκτές ακόμη και σε τελικά στάδια ανάπτυξης λογισμικού. Οι ευέλικτες μεθοδολογίες προσδίδουν ανταγωνιστικό πλεονέκτημα στην εταιρία – πελάτη.

- Παράδοση εκτελέσιμου λογισμικού συχνά (από ένα παράθυρο χρόνου δύο εβδομάδων, μέχρι δύο μηνών, με προτίμηση σε κατά το δυνατόν μικρότερες χρονικές περιόδους).

- Προγραμματιστές και άνθρωποι άλλων ειδικοτήτων που σχετίζονται με το έργο θα πρέπει να δουλεύουν μαζί σε καθημερινή βάση.

- Το έργο πρέπει να αναπτύσσεται από ανθρώπους που έχουν κίνητρα. Πρέπει να τους δίδεται το σωστό περιβάλλον, να ικανοποιούνται οι ανάγκες τους και να εμπιστευόμαστε τις ικανότητές τους.

- Ο πιο αποτελεσματικός τρόπος επικοινωνίας προς την ομάδα ανάπτυξης αλλά και μεταξύ των μελών της ομάδας ανάπτυξης είναι οι κατ’ ιδίαν συνομιλίες.

- Η πρόοδος του έργου θα πρέπει να μετριέται κυρίως με παραδοτέα λογισμικού τα οποία είναι λειτουργικά..

- Κάνοντας χρήση των ευέλικτων μεθοδολογιών επιτυγχάνουμε ρυθμό στην ανάπτυξη λογισμικού.

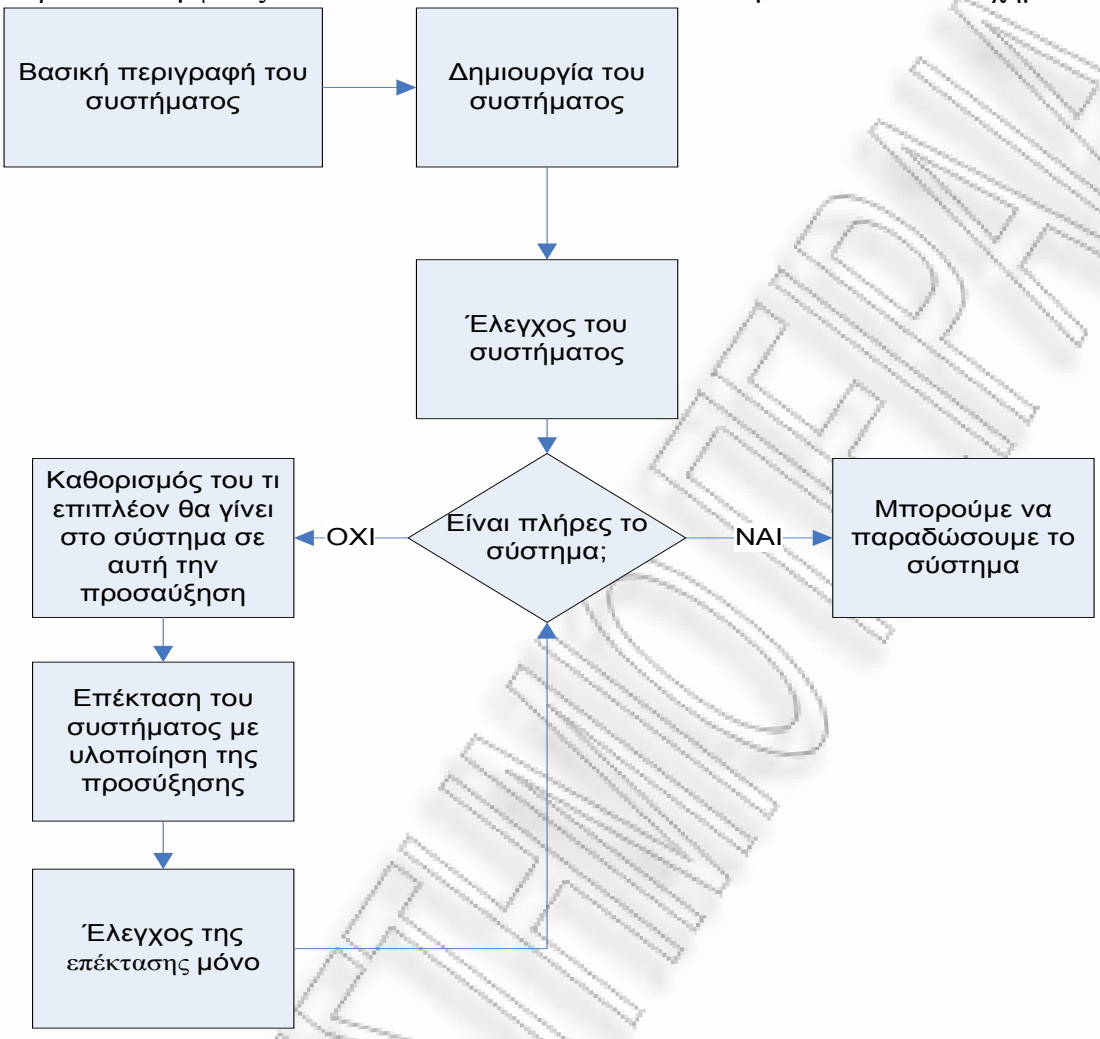
- Δίνοντας έμφαση στην τεχνική υπεροχή και τον καλό σχεδιασμό, επιτυγχάνουμε αυξημένη ευελιξία.

- Η απλότητα – η ικανότητα δηλαδή του να μπορούμε να προσδιορίσουμε το μεγαλύτερο μέρος της δουλειάς που δεν έχει γίνει – είναι υψίστης σημασίας.

- Οι καλύτερες αρχιτεκτονικές, απαιτήσεις και σχέδια προκύπτουν από αυτοοργανούμενες ομάδες.

- Κατά τη διάρκεια διαλειμμάτων, που πρέπει να συμβαίνουν συχνά, η ομάδα πρέπει να βολιδοσκοπεί το πώς μπορεί να γίνει πιο αποτελεσματική και, στη συνέχεια, να κινείται προς την κατεύθυνση αυτή.

Παρακάτω εμφανίζονται τα στάδια του ευέλικτου μοντέλου στο σχήμα 1.



Σχήμα 1: Ροή ευέλικτου μοντέλου

Τα στάδια του συγκεκριμένου μοντέλου είναι τα εξής:

- Βασική περιγραφή του συστήματος είναι το πρώτο στάδιο όπου καταγράφονται οι πλέον στοιχειώδεις απαιτήσεις του συστήματος οπότε δημιουργείται ένα διάγραμμα κλάσεων υψηλής αφαιρετικότητας, που θα χρησιμοποιηθεί ως πυρήνας του συστήματος[1][2]. Να σημειωθεί ότι δεν είναι στόχος το να προσδιορισθεί το πως θα υλοποιούνται αυτές οι περιπτώσεις χρήσεις, αλλά μόνο το ποιες θα είναι.

- Δημιουργία του συστήματος είναι η υλοποίηση του πυρήνα.

- Έλεγχος του συστήματος είναι ο έλεγχος για το κατά πόσον η υλοποίηση καλύπτει αυτά που είχαν καταγραφεί στις απαιτήσεις.

[1] Πυρήνας είναι το κομμάτι του συστήματος που κάνει όλες τις στοιχειώδεις λειτουργίες που είναι απαραίτητες για την ανάπτυξη του υπόλοιπου συστήματος.
[2] Ο πυρήνας αυτός παραμένει αμετάβλητος καθ' όλη την εξέλιξη του συστήματος

- Καθορισμός του τι επέκταση χρειάζεται να γίνει στο σύστημα είναι το στάδιο κατά το οποίο ανακαλύπτονται κάποια νέα χαρακτηριστικά που θα μπορούσαν να προστεθούν στο πυρήνα, ώστε να τείνει το σύστημα προς την πληρότητα. Επιπλέον είναι η προσθήκη των ανάλογων περιπτώσεων χρήσης στο υπάρχον διάγραμμα.

- Υλοποίηση της επέκτασης είναι το στάδιο κατά το οποίο υλοποιούνται οι επιπλέον περιπτώσεις χρήσης, με τέτοιο τρόπο ώστε να μπορούν να επεκτείνουν το υπάρχον σύστημα.

- Έλεγχος της επέκτασης είναι το στάδιο κατά το οποίο ελέγχεται το κατά πόσον η υλοποίηση που έγινε στην επέκταση ανταποκρίνεται στις περιπτώσεις χρήσεις. Αν ανταποκρίνεται τότε προστίθεται στο υπόλοιπο σύστημα.

1.5 ΔΟΜΗ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Η εν λόγω εργασία αποτελείται από τα ακόλουθα κεφάλαια:

Κεφάλαιο 1: Εισαγωγή, όπου περιγράφονται τα βασικά θέματα που αφορούν την συγκεκριμένη διπλωματική εργασία, δηλαδή τι έχει ως στόχο, τι εργαλεία χρησιμοποιήθηκαν για την ολοκλήρωσή της, τι μεθοδολογία ακολουθήθηκε προκειμένου να υπάρξει το επιθυμητό αποτέλεσμα και τέλος το πώς δομούνται τα κεφάλαια.

Κεφάλαιο 2: Η βιβλιογραφία οπού περιγράφονται οι τεχνολογίες που έχουν χρησιμοποιηθεί καθώς και το τι είναι πληροφοριακά συστήματα υγείας.

Κεφάλαιο 3 Ερεύνα του (Ολοκληρωμένου Πληροφοριακού Συστήματος Υγείας του Α΄ Περιφερειακού Συστήματος Υγείας & Πρόνοιας Αττικής» (Α΄ ΠΕΡΙΦΕΡΕΙΑΚΟ ΣΥΣΤΗΜΑ ΥΓΕΙΑΣ ΚΑΙ ΠΡΟΝΟΙΑΣ ΑΤΤΙΚΗΣ, Σύμβαση 3/11/2006) με βάση την ερευνά δράσης.

Κεφάλαιο 4: Ανάλυση και σχεδίαση, ό π υ αναλύεται το πρό βλημα σε πιο στοιχειώδη κι ευκολότερα επιλύσιμα υπό-προβλήματα , και με χρήση διαφόρων, διαγραμματικής κυρίως φύσεως, τεχνικών παρουσιάζεται η επίλυση των επιμέρους υπό-προβλημάτων αλλά και το πως συνδέονται ώστε να παραχθεί η συνολική λύση, δηλαδή το προς υλοποίηση σύστημα.

Κεφάλαιο 5 Η υλοποίηση "MED.I.S.[3]", όπου παρουσιάζεται η υλοποίηση του συστήματος προβάλλοντας το ποια είναι η αρχιτεκτονική του συστήματος και τις υποστηριζόμενες λειτουργίες. Επίσης παρουσιάζονται και παραδείγματα χρήσης που αποδεικνύουν το πως υποστηρίζονται οι λειτουργίες αυτές καθώς γίνεται και ο έλεγχος της εφαρμογής, όπου περιγράφεται η μεθοδολογία που χρησιμοποιήθηκε ώστε να επιβεβαιωθεί η σωστή λειτουργία του συστήματος σύμφωνα με αυτά που απαιτούσε το πρόβλημα.

[3] MED.I.S.:MEDical Information System

Συμπεράσματα όπου περιγράφεται το τι αποκόμισε ο συγγραφέας και προγραμματιστής από αυτήν την διπλωματική εργασία, αλλά και το τι επιπλέον επεκτάσεις μπορούν να γίνουν στο σύστημα.

Ιδέες για μελλοντικη εξέλιξη του συστήματος όπου γίνεται μία προσέγγιση για το πώς μπορεί να εξελιχθεί μελλοντικά το συστημά σύμφωνα με τις ανάγκες ενός νοσοκομείου.

Κεφάλαιο 2: Ανασκόπηση Βιβλιογραφίας

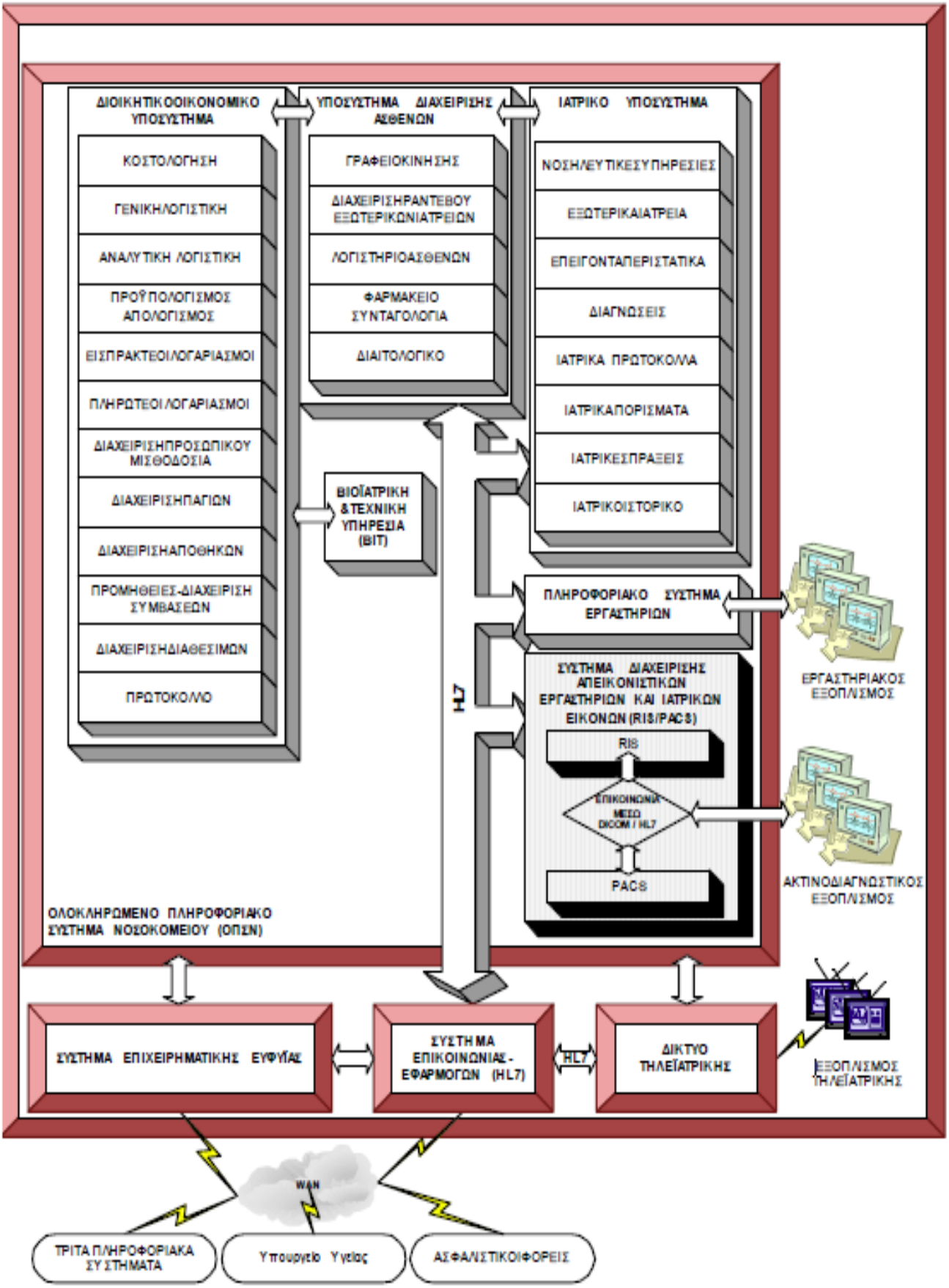
2.1 ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ ΥΓΕΙΑΣ

Τα νοσοκομεία αποτελούν ένα μέρος του Συστήματος Υγείας και Πρόνοιας το οποίο περιλαμβάνει ακόμα την πρόληψη, τη πρωτοβάθμια περίθαλψη, τη νοσηλεία στο σπίτι, τη κοινωνική ασφάλιση και την ιατρική έρευνα. Τα νοσοκομεία είναι ο ακρογωνιαίος λίθος ο οποίος στηρίζει την ομαλή λειτουργία του Συστήματος Υγείας. Η ταχύτητα που απαιτείται σήμερα στη λήψη σωστών αποφάσεων, επιβάλλει τη μηχανογράφηση του Συστήματος Υγείας και Πρόνοιας και κατ' επέκταση και του νοσοκομειακού κλάδου [27]. Η μηχανογράφηση ενός νοσοκομείου είναι μια περίπλοκη διεργασία η οποία απαιτεί τη διασύνδεση ανομοιογενών τμημάτων τα οποία ανταλλάσσουν πληροφορίες προς όφελος του Πολίτη. Ένα ολοκληρωμένο πληροφοριακό σύστημα νοσοκομείου (ΟΠΣΝ) ή Hospital Information System (HIS) αποτελείται από μια πληθώρα υποσυστημάτων. Τα κύρια υποσυστήματα φαίνονται στο Σχήμα 2.

Η μηχανογράφηση των νοσοκομείων έχει μόνο θετικά στοιχεία να προσφέρει. Η εγκατάσταση και λειτουργία ενός HIS προσβλέπει στη βελτίωση των συνθηκών νοσηλείας, στη μείωση του κόστους λειτουργίας και στην αυτοματοποίηση των διαδικασιών.

Προκειμένου να είναι σε θέση η διοίκηση ενός νοσοκομείου να παρακολουθεί με πραγματικά στοιχεία τη λειτουργία του απαιτείται η εξαγωγή έγκυρων δεδομένων σε μορφή επεξεργασμένη πληροφορίας (αναφορές, εκτυπώσεις καθημερινής εργασίας, στατιστικά δεδομένα, δείκτες ποιότητας, δείκτες αποτελεσματικότητας, δείκτες υγείας, κ.λπ.). Έτσι η διοίκηση ενός νοσοκομείου στηρίζεται στις πληροφορίες που αντλεί από τα συνεργαζόμενα συστήματα που υπάρχουν στο νοσοκομείο και συνεπώς όσο πληρέστερη είναι η ανάπτυξη της πληροφοριακής υποδομής τόσο ευκολότερο είναι το έργο της διοίκησης μιας μονάδας υγείας. Τα παρακάτω πληροφοριακά συστήματα:

- Το Πληροφοριακό Σύστημα Εργαστηρίων (ΠΣΕ, ή LIS – Laboratory InformationSystem)
- Το Πληροφοριακό Σύστημα Ακτινολογικών Εξετάσεων (RIS – Radiology Information System
- Το Σύστημα Αρχειοθέτησης και Επικοινωνίας Ιατρικών Εικόνων (PACS – PictureArchiving and Communication system)
- Διάφορα συστήματα Τηλεϊατρικής και κατ' οίκον νοσηλείας



Σχήμα 2: Ενδεικτική διάταξη υποσυστημάτων HIS [26]

Οι λόγοι που καθιστούν αναγκαία στις μέρες μας, την εισαγωγή πληροφοριακού συστήματος στα σύγχρονα νοσοκομεία, απορρέουν από τη γενικότερη ανάγκη βελτίωσης τόσο του τρόπου λειτουργίας τους, όσο και των παρεχόμενων υπηρεσιών υγείας. Οι βασικοί επιμέρους στόχοι που θα πρέπει να ικανοποιηθούν για το σκοπό αυτό είναι:

1. Η γενικότερη αναβάθμιση των υπηρεσιών του νοσοκομείου (βελτίωση της ποιότητας περίθαλψης και εξυπηρέτησης των ασθενών). Ο στόχος αυτός μπορεί να επιτευχθεί με:

- Την εισαγωγή και την διαχείριση ηλεκτρονικού φακέλου ασθενούς, που θα συγκεντρώνει και θα παρουσιάζει κατάλληλα όλα τα στοιχεία που αφορούν στους κρίσιμους παράγοντες περίθαλψης, την πορεία της πάθησης κ.λπ.

- Τον συσχετισμό των παραπάνω στοιχείων σύμφωνα με τους κανόνες της ιατρικής επιστήμης, ώστε να εξυπηρετούνται οι ιατροί στην λήψη αποφάσεων σχετικών με την προτεινόμενη αγωγή.

- Την παροχή δυνατότητας πρόσβασης σε παλαιότερα στοιχεία περίθαλψης (στο ίδιο ή /και σε άλλο νοσηλευτικό ίδρυμα) ώστε να είναι δυνατή η άμεση αναδρομή στο ιστορικό του ασθενούς.

- Τη μείωση της γραφειοκρατίας.

- Τη βελτίωση της πληροφόρησης των συναλλασσομένων και της ταχύτητας εξυπηρέτησής τους.

- Την ελαχιστοποίηση των λαθών

2. Ο περιορισμός των χειρόγραφων διαδικασιών και η βελτίωση του εργασιακού περιβάλλοντος. Ο στόχος αυτός μπορεί να επιτευχθεί με:

- Την αυτοματοποίηση των διαδικασιών

- Την διασύνδεση και την ολοκλήρωση των επί μέρους συστημάτων σε ένα πλήρες σύστημα.

- Την αναβάθμιση του εσωτερικού εργασιακού περιβάλλοντος

- Την εξασφάλιση αποτελεσματικότητας στην διεκπεραίωση καθημερινών εργασιών

- Τη διαχείριση και αξιοποίηση του ανθρωπίνου δυναμικού

- Την αξιοποίηση σύγχρονων τεχνολογιών πληροφορικής

3. Η ελαχιστοποίηση του κόστους παροχής περίθαλψης. Ο στόχος αυτός μπορεί να επιτευχθεί με:

- Την ορθολογική διαχείριση των πόρων του νοσηλευτικού ιδρύματος (έλεγχοι ανάλωσης υλικού, προγραμματισμός διαδικασιών, αυτοματοποίηση ελέγχων, κ.λ.π.).
- Την αποφυγή άσκοπων ιατρικών πράξεων (π.χ. αποφυγή επανάληψης εξετάσεων).

4. Η παροχή ικανών και αξιόπιστων πληροφοριών στη διοίκηση του νοσοκομείου. Η πληροφόρηση αυτή μπορεί να περιλαμβάνει τόσο διαχειριστικά όσο και επιστημονικά στοιχεία.

Χαρακτηριστικά αναφέρονται:

- Η πληρότητα θαλάμων, ο μέσος χρόνος νοσηλείας κ.λ.π.
- Η παρακολούθηση των ποσοτικών και οικονομικών δεικτών τόσο ανά κατηγορία, όσο και ανά κέντρο κόστους.
- Το κόστος νοσηλείας ανά διάγνωση ή ομάδα διαγνώσεων.
- Τα ποσοστά αποθεραπείας ανά διάγνωση ή ομάδα διαγνώσεων.

5. Η δημιουργία ενός ευέλικτου εργαλείου υποστήριξης στην λήψη αποφάσεων για τον καθορισμό και τον έλεγχο των διαφορετικών πολιτικών οργάνωσης της παροχής υγείας, κοστολόγησης και τιμολόγησης των υπηρεσιών της.

2.2 ΙΣΤΟΡΙΑ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΝΟΣΟΚΟΜΕΙΩΝ

Πρώτη γενιά:1960-1970

Κατά την περίοδο αυτή τα πληροφοριακά συστήματα νοσοκομείων που αναπτύχθηκαν αφορούσαν κυρίως εφαρμογές για την υποστήριξη περισσότερο των κλινικών και λιγότερο των διοικητικών διαδικασιών του νοσοκομείου. Ο στόχος ήταν η βελτίωση της παρεχόμενης περίθαλψης. Τα συστήματα αυτά ήταν ιδιαίτερα ακριβά και χρησιμοποιήθηκαν κατά κύριο λόγο από τα μεγάλα νοσοκομεία.

Δεύτερη γενιά: 1970-1980

Κατά την περίοδο αυτή, στην οποία έγινε και η εμφάνιση των μικροϋπολογιστών, τα ΠΣΝ άρχισαν να περιλαμβάνουν εφαρμογές για την υποστήριξη των οικονομικών και διοικητικών διαδικασιών του νοσοκομείου. Τα συστήματα αυτά χρησιμοποιήθηκαν και από τα νοσοκομεία μικρότερης κλίμακας μεγέθους καθώς το κόστος τους αλλά και ο όγκος τους είχε μειωθεί σημαντικά.

Επίσης, κατά την περίοδο αυτή, εκτός από την εμφάνιση των μικροϋπολογιστών, άρχισε και η χρήση των βάσεων δεδομένων η οποία έδωσε την δυνατότητα άμεσης διαθεσιμότητας των δεδομένων και παραγωγής αναφορών. Τα συστήματα αυτά ήταν κατά κύριο λόγο εφαρμογές, η λειτουργία και η χρησιμότητα των οποίων περιορίζονταν στα πλαίσια ενός συγκεκριμένου λειτουργικού τμήματος (stand-alone). Συνήθως, βασίζονταν σε τοπικές βάσεις δεδομένων ενώ η δυνατότητα σύνδεσης μεταξύ τους αντιμετωπιζόταν ως δευτερεύον θέμα.

Ένα παράδειγμα ενός stand-alone συστήματος είναι ο προσωπικός υπολογιστής στο φαρμακείο ενός νοσοκομείου στον οποίο λειτουργεί μια εφαρμογή για την καταχώρηση των ιατρικών συνταγών, την έκδοση αποδείξεων και τη διαχείριση της αποθήκης του φαρμακείου. Το σύστημα αυτό είναι stand-alone καθώς δεν υπάρχει επικοινωνία (σύνδεση) με τα κλινικά τμήματα του νοσοκομείου ούτε με το λογιστήριο στο οποίο γίνεται και η χρέωση των ασθενών. Εάν το σύστημα αυτό δεν ήταν stand-alone, δεν θα απαιτούνταν η επαναπληκτρολόγηση των συνταγών καθώς αυτές θα ήταν άμεσα διαθέσιμες (μέσω της επικοινωνίας των συστημάτων) από τη χρονική στιγμή έκδοσης τους στο κλινικό τμήμα. Επίσης, ο λογαριασμός του ασθενή θα ενημερωνόταν για οποιαδήποτε χρέωση από τη χρονική στιγμή εκτέλεσης μιας συνταγής.

Τρίτη γενιά: 1980-1991

Κατά την περίοδο αυτή έγινε η εμφάνιση των προσωπικών υπολογιστών και η χρήση των τοπικών δικτύων υπολογιστών (Local Area Networks – LAN). Έτσι, πολλοί προμηθευτές πληροφοριακών συστημάτων αναγκάστηκαν να δώσουν στα συστήματα τους τη δυνατότητα επικοινωνίας με άλλα συστήματα. Επίσης, κατά το χρονικό αυτό διάστημα άρχισε και η θεμελίωση των πρώτων προτύπων λειτουργικών συστημάτων, πρωτοκόλλων δικτύων και συστημάτων διαχείρισης αρχείων δεδομένων. Ως αποτέλεσμα, οι προμηθευτές ΠΣΝ άρχισαν να χρησιμοποιούν συστήματα διαχείρισης βάσεων δεδομένων άλλων προμηθευτών, μερικά από τα οποία συμπεριλάμβαναν και γλώσσες διαχείρισης δεδομένων μέσω των οποίων δινόταν η δυνατότητα ανάκτησης δεδομένων που διαχειρίζονταν άλλες εφαρμογές.

Τέταρτη γενιά: 1991 έως σήμερα

Από το 1991 έχει αρχίσει να εμφανίζεται μια νέα γενιά ΠΣΝ, αν και τα χαρακτηριστικά της προηγούμενης γενιάς δεν έχουν εκλείψει εντελώς. Υπάρχουν διάφοροι παράγοντες που επηρεάζουν τη γενιά αυτή, όπως η αύξηση της δυνατότητας σύνδεσης δικτύων υπολογιστών, η δυνατότητα εγκατάστασης και χρήσης ενός συστήματος διαχείρισης βάσεων δεδομένων σε περισσότερα από ένα σημεία και η αύξηση και η καθιέρωση προτύπων στη λειτουργία των πληροφοριακών συστημάτων. Με τον όρο πρότυπο, εννοούμε τον κοινό τρόπο θεώρησης και αντιμετώπισης ενός συγκεκριμένου θέματος. Έτσι, στον χώρο της πληροφορικής στο διάστημα αυτό εμφανίστηκαν πρότυπα επικοινωνίας υπολογιστών, παραγωγής δεδομένων κ.λ.π. τα οποία έδωσαν τη δυνατότητα επικοινωνίας διαφορετικών πληροφοριακών συστημάτων (στο ίδιο γεωγραφικό σημείο ή σε διαφορετικά).

Ένα πληροφοριακό σύστημα έχει σχεδιαστεί και υλοποιηθεί με βάση κάποιο μοντέλο, το οποίο αναπαριστά τη δομή του νοσοκομείου σε συγκεκριμένη χρονική στιγμή. Η πρόκληση που αντιμετωπίζει ένα νοσοκομείο είναι η επιλογή συστημάτων των οποίων το μοντέλο είναι όσο το δυνατόν περισσότερο προσαρμοσμένο στην πραγματική κατάσταση. Κάθε γενιά πληροφοριακών συστημάτων βασίζεται σε συγκεκριμένη τεχνολογία με δυνατότητες και περιορισμούς. Η δεύτερη καμπύλη στο Σχήμα 2 δείχνει την καθυστέρηση εφαρμογής ης τεχνολογίας για την υποστήριξη των αλλαγών που πραγματοποιούνται μέσα σε ένα νοσοκομείο. Ακόμη και τα νοσοκομεία που αναγνωρίζουν έγκαιρα τις αλλαγές και την ανάγκη προσαρμογής των συστημάτων τους ή την απόκτηση νέων δεν μπορούν εύκολα να ικανοποιήσουν την ανάγκη αυτή. Επίσης, εκείνα τα νοσοκομεία τα οποία βρίσκονται στη δεξιά άκρη της καμπύλης είναι καταδικασμένα να έχουν απαρχαιωμένα συστήματα.

2.3 ΤΥΠΟΙ ΟΛΟΚΛΗΡΩΜΕΝΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΥΓΕΙΑΣ

2.3.1 Νοσηλευτικά πληροφοριακά συστήματα

Τα νοσηλευτικά πληροφοριακά συστήματα είναι λογισμικό που έχει αναπτυχθεί για να χρησιμοποιείται ειδικά από τους νοσηλευτές. Τα προγράμματα αυτά είτε αφορούν ένα συγκεκριμένο χώρο της νοσηλευτικής είτε υποστηρίζουν γενικότερα τις υπηρεσίες νοσηλευτικής διοίκησης. Παραδείγματα των νοσηλευτικών τομέων που μπορούν να ωφεληθούν από τη υποστήριξη των πληροφοριακών συστημάτων είναι μεταξύ άλλων, η ψυχική υγεία, η νεογνολογία, η ουρολογία, η ογκολογία, η μαιευτική, η χειρουργική και ο έλεγχος λοιμώξεων.

Τα γενικά νοσηλευτικά πληροφοριακά συστήματα διαθέτουν πολλαπλά προγράμματα ή μοντέλα, που χρησιμοποιούνται για να επιτελούν διάφορες κλινικές, εκπαιδευτικές και διαχειριστικές λειτουργίες. Τα περισσότερα από αυτά διαθέτουν μοντέλα για την ταξινόμηση των ασθενών, τη στελέχωση, τον προγραμματισμό των υπηρεσιών, τη διοίκηση προσωπικού και τη σύνταξη εκθέσεων. Μπορούν να ενταχθούν και άλλα μοντέλα όπως η κατάρτιση προϋπολογισμών, η κατανομή πόρων, ο έλεγχος του κόστους, η διαχείριση της ποιότητας, η ανάπτυξη προσωπικού, η διαμόρφωση μοντέλων και η προσομοίωση για την λήψη αποφάσεων, ο στρατηγικός σχεδιασμός, οι βραχυπρόθεσμες ανάγκες για την πρόβλεψη και σχεδιασμό εργασίας και η αξιολόγηση προγράμματος.

Τα μοντέλα για την ταξινόμηση ασθενών, την στελέχωση, τον προγραμματισμό των υπηρεσιών, τη διοίκηση προσωπικού και τη σύνταξη εκθέσεων συχνά σχετίζονται στενά μεταξύ τους. Οι ασθενείς ταξινομούνται σύμφωνα με τα καθιερωμένα κριτήρια βαρύτητας της κατάστασης. Οι πληροφορίες για την ταξινόμηση των ασθενών αποτελούν εισροή για το μοντέλο που αφορά την απαιτούμενη στελέχωση υπηρεσιών και τα επίπεδα στελέχωσης υπολογίζονται με βάση διάφορους τύπους υπολογισμού του φόρτου εργασίας. Επίσης, η πραγματική στελέχωση αποτελεί και αυτή εισροή και μπορεί να γίνει σύγκριση της απογραφής, της βαρύτητας της κατάστασης των ασθενών, της απαιτούμενης στελέχωσης και της πραγματικής στελέχωσης. Ο προϋπολογισμός υποστηρίζεται από την απογραφή, τη βαρύτητα της κατάστασης του ασθενούς και από τα απαιτούμενα μοντέλα

στελέχωσης. Οι πληροφορίες αυτές είναι πολύτιμες στην υποστήριξη αιτημάτων για επιπλέον προσωπικό, πλήρους ή μερικής απασχόλησης. Το μοντέλο της σύνταξης εκθέσεων δίνει τη δυνατότητα ανάκλησης όλων των καταχωρημένων πληροφοριών με έγκαιρο και παρουσιάσιμο τρόπο.

Τα νοσηλευτικά πληροφοριακά συστήματα μπορούν να χρησιμοποιηθούν για να κάνουν τη φροντίδα του ασθενούς πιο αποτελεσματική και οικονομική. Τα κλινικά στοιχεία περιλαμβάνουν το ιστορικό και την εκτίμηση του ασθενούς, τα σχέδια νοσηλευτικής φροντίδας, σημειώσεις και διαγράμματα νοσηλευτικής προόδου, παρακολούθηση των ασθενών, και σχεδιασμό της εξόδου από το ίδρυμα. Αυτά όλα μπορούν να γίνουν στο σταθμό του νοσηλευτή ή σε πιο προοδευτικά συστήματα, κοντά στον ασθενή.

Οι κλινικοί νοσηλευτές μπορούν να χρησιμοποιήσουν το νοσηλευτικό πληροφοριακό σύστημα για να αντικαταστήσουν χειρόγραφα συστήματα καταγραφής δεδομένων. Αυτό μπορεί να οδηγήσει σε μείωση του κόστους, ενώ παράλληλα μπορεί να δοθεί δυνατότητα για βελτιωμένη ποιότητα φροντίδας καθώς και ποιότητας ζωής. Οι κλινικοί νοσηλευτές μπορούν να συγκεντρώνουν και να καταχωρούν κλινικά δεδομένα, να χρησιμοποιούν τους Η/Υ για να τα αναλύουν και να τα καταρτίζουν και κατά συνέπεια να λαμβάνουν αποφάσεις ώστε να υποστηρίζουν τις κλινικές κρίσεις τους. Η αυτοματοποιημένη παροχή συμβουλών μπορεί να εφαρμοστεί στην οθόνη για να διαπιστωθούν αρνητικές αντιδράσεις σε φάρμακα, αλληλεπιδράσεις και προετοιμασία των σωστών δόσεων. Οι Η/Υ μπορούν με τον κατάλληλο προγραμματισμό να απορρίπτουν εντολές που μπορούν να δημιουργήσουν προβλήματα σε αυτούς και άλλους τομείς, αποτρέποντας, έτσι, τη δημιουργία λαθών.

2.3.2 Πληροφοριακά συστήματα εργαστηρίων

Τα εργαστηριακά πληροφοριακά συστήματα (Laboratory Information Systems) είναι λογισμικό εγκατεστημένο σε ηλεκτρονικό υπολογιστή, ο οποίος είναι συνδεδεμένος με τον κατάλληλο ιατρικό εξοπλισμό. Είναι υπεύθυνα για την αποθήκευση κλινικών δεδομένων, την επαλήθευση της ακρίβειας των εξετάσεων, τη βαθμονόμηση των οργάνων, τη δημιουργία ή ενημέρωση αρχείων ασθενών, τη συλλογή πληροφοριών από ένα πλήθος συσκευών όπως συσκευές ανάλυσης αίματος. Οι ιατρικές συσκευές που πραγματοποιούν τις διάφορες μετρήσεις ονομάζονται εργαστηριακοί αναλυτές και διαθέτουν μικροεπεξεργαστές, που ελέγχουν και συντονίζουν τη σωστή λειτουργία των συσκευών. Ο χρήστης μπορεί να μεταφέρει την ίδια στιγμή ηλεκτρονικά στο εργαστηριακό πληροφοριακό σύστημα τις μετρήσεις από τις συσκευές. Οι χρησιμοποιούμενοι εργαστηριακοί αναλυτές διασυνδέονται στο όλο σύστημα μέσω ειδικών διατάξεων, που συνδέονται σε Η/Υ και το σύστημα, έτσι, αποτελεί ενιαίο κορμό παραγωγής.

Ένα ιδανικό ολοκληρωμένο πληροφοριακό σύστημα εργαστηρίων για να είναι καταξιωμένο στον ιατρικό χώρο χρειάζεται να είναι προσαρμοσμένο στις ανάγκες και τις ιδιαιτερότητες των εργαστηρίων κάθε νοσοκομείου ήδιαγνωστικού κέντρου. Γενικά χαρακτηριστικά ενός πληροφοριακούεργαστηριακού συστήματος είναι:

- Μονόδρομη και αμφίδρομη επικοινωνία με πληθώρα αυτόματων αναλυτών
- Παραγγελία εργαστηριακών εξετάσεων σε πραγματικό χρόνο
- Έγκριση και ανάγνωση αποτελεσμάτων σε πραγματικό χρόνο
- Δυνατότητα σύνδεσης αποτελεσμάτων και διαγνώσεων
- Διαχείριση ποιότητας ιατρικών συσκευών
- Παρακολούθηση αναλώσιμων
- Στατιστική ανάλυση

Σ' ένα τέτοιο σύστημα το λογισμικό είναι δομημένο με τη συλλογιστική πολλών χρηστών, που ο καθένας έχει διαφορετικές αρμοδιότητες και προσβάσεις στις διακινούμενες πληροφορίες. Διαθέτει πλήρη παραμετροποίηση επιτρέποντας το διαχωρισμό του συνόλου των εργαστηρίων σε επί μέρους τμήματα, τον καθορισμό του προσωπικού του τμήματος όπως και τις εξετάσεις που πραγματοποιεί το κάθε τμήμα. Διαχειρίζεται το ιστορικό των εξετάσεων όλων των ασθενών παρακολουθώντας τις εξετάσεις ανά ασθενή, τμήμα εργαστηρίου, κλινική, ασφαλιστικό φορέα και προαιρετικά μπορεί να εκτελεί τιμολογήσεις και να παρακολουθεί όλα τα σχετικά οικονομικά στοιχεία.

Ένα δίκτυο υπολογιστών απλώνεται στα τμήματα των εργαστηρίων. Οι καθημερινές εξετάσεις εισάγονται στο σύστημα είτε από κάθε κλινική, είτε από την γραμματεία των εργαστηρίων (τμήμα παραλαβής δειγμάτων) είτε από πολλαπλές γραμματείες των εργαστηριακών τμημάτων. Σημαντικό είναι ότι ένα τέτοιο σύστημα μπορεί να υποβοηθά στην κατάργηση των πολλαπλών σημείων παραλαβής δειγμάτων καθώς και για παράδειγμα, στην κατάργηση πολλαπλών αιμοληψιών που παρατηρούνται στον ίδιο ασθενή κατά τη διάρκεια της ημέρας, για τις ανάγκες του κάθε εργαστηριακού τμήματος. Επιπλέον από τα διάφορα τερματικά που τοποθετούνται οι θεράποντες ιατροί παρακολουθούν το ιστορικό του ασθενούς, ενώ τα τρέχοντα αποτελέσματα διατίθενται στο τερματικό αμέσως μετά την ολοκλήρωση των εργαστηριακών διαδικασιών και ακολουθεί η έγκρισή τους από τους διευθυντές του κάθε εργαστηριακού τομέα, σε πραγματικό χρόνο και χωρίς καθυστερήσεις και ενδιάμεσα τηλεφωνήματα στα εργαστήρια. Οι ασθενείς πλέον δεν συγκεντρώνονται στα εργαστήρια αναμένοντας τα αποτελέσματά τους ενώ η εικόνα της πορείας του ασθενούς είναι εμφανής και ευδιάκριτη.

Οφέλη που προκύπτουν από τη χρήση του συστήματος είναι:

- Μείωση αναλωσίμων (φιαλίδια, σύριγγες, κλπ)
- Μείωση χρόνου παραδόσεως αποτελεσμάτων
- Μείωση λαθών στα αποτελέσματα (άλλου ασθενούς σε άλλον)
- Αύξηση ακρίβειας και αξιοπιστίας αποτελεσμάτων
- Μείωση του όγκου του αρχείου του Νοσοκομείου
- Μείωση του χρόνου ανευρέσεως παλιών αποτελεσμάτων
- Μείωση του κόστους συντηρήσεως των οργάνων
- Γενική οργάνωση των εργαστηρίων
- Ύπαρξη στατιστικών στοιχείων για εκτιμήσεις επενδύσεων ή προμηθειών αναλωσίμων

Υποσυστήματα αποτελούν το ολοκληρωμένο πληροφοριακό εργαστηριακό σύστημα. Το κάθε υποσύστημα του ιατρικού εργαστηρίου έχει τη δυνατότητα να διασυνδέεται τόσο με άλλα πληροφοριακά υποσυστήματα κλινικών, εξωτερικών ιατρείων κλπ ανταλλάσσοντας δεδομένα, όσο και με πληροφοριακά συστήματα τα οποία βρίσκονται εκτός νοσοκομείου. Όλα αυτά, βέβαια, προϋποθέτουν την αυτόματη ενημέρωση του ιατρικού φακέλου του ασθενούς. Για παράδειγμα, το πληροφοριακό σύστημα απεικονιστικού εργαστηρίου (ακτινολογικό, αξονικός ή μαγνητικός τομογράφος, υπέρηχοι) έχει τη δυνατότητα αποθήκευσης των εικόνων που προέρχονται από τα απεικονιστικά ιατρικά μηχανήματα στη Βάση Δεδομένων (image database). Με την ύπαρξη πληροφοριακού συστήματος, την αρχειοθετημένη εικόνα μπορούν και την βλέπουν τόσο οι εργαστηριακοί ιατροί ιδιωτικών κέντρων όσο και εργαστηριακοί ιατροί νοσοκομειακών ιδρυμάτων.

Ένα από τα υποσυστήματα του εργαστηριακού πληροφοριακού συστήματος αποτελεί το πληροφοριακό σύστημα αιμοδοσίας. Σκοπός της εφαρμογής του συστήματος αιμοδοσίας είναι η πλήρης διαχείριση όλων των εργασιών του τμήματος, καθώς επίσης και της ενσωμάτωσης όλων των χρησιμοποιούμενων διαγνωστικών συσκευών στο πληροφοριακό σύστημα. Ακολουθώντας τη δομή το πληροφοριακού εργαστηριακού συστήματος και το υποσύστημα αυτό είναι δομημένο με τη συλλογιστική πολλαπλών χρηστών. Αποτελείται από ένα δίκτυο υπολογιστών, που «απλώνεται» στο τμήμα της αιμοδοσίας και το οποίο παρέχει πλήρη δυνατότητα διασύνδεσης με το ενιαίο πληροφοριακό σύστημα ή με τις διάφορες κλινικές και τα εργαστήρια, σε κατάσταση πραγματικού χρόνου. Βασικός ρόλος του είναι να διαχειρίζεται πλήρως το ιστορικό των εξετάσεων όλων των ασθενών και αιμοδοτών. Ακόμη, εμφανίζει όλες τις εργαστηριακές εξετάσεις που έχουν πραγματοποιηθεί, τις χορηγημένες μονάδες, τις καλύψεις που έχουν γίνει είτε είναι από αιμοδότες είτε από άλλα νοσοκομεία και τέλος τις διασταυρωμένες μονάδες που υπάρχουν προς χορήγηση. Οι διαδικασίες αυτές αυτοματοποιούνται και η πρόσβαση σε αυτά τα στοιχεία γίνεται άμεσα. Παράλληλα το τμήμα διακίνησης εύκολα και γρήγορα μπορεί να έχει όλες τις πληροφορίες που του χρειάζονται όπως ποιά και πόσα είναι τα αποθέματα μονάδων, ποιές μονάδες υπάρχουν προς χορήγηση εσωτερικών ασθενών, τα υπόλοιπα των ασθενών που έχουν προκύψει από χορηγήσεις μονάδων και αιτήσεις καλύψεων καθώς επίσης και τις εκκρεμότητες που υπάρχουν για επικοινωνία με αιμοδοσίες άλλων νοσοκομείων. Παράλληλα δίνεται η δυνατότητα εκτύπωσης όλων των καταστάσεων και κινήσεων που είναι υποχρεωτικές, με αποτέλεσμα να καταργούνται όλα τα βιβλία που χωρίς το πληροφοριακό εργαστηριακό σύστημα αιμοδοσίας είναι απαραίτητο να κρατούνται χειρόγραφα. Αποθέματα μονάδων, λογιστικό έλλειμμα μονάδων ασθενών, στατιστική κίνηση μονάδων ανά κλινική και ιατρό, εισαγωγές μονάδων από άλλα νοσοκομεία, ειδοποιήσεις αιμοδοτών, απαλλάσσουν το προσωπικό από απαραίτητες μεν, χρονοβόρες δε, εργασίες παρέχοντας με ασφάλεια και αξιοπιστία όλες τις απαραίτητες πληροφορίες. Ουσιαστική μπορεί, λοιπόν, να θεωρηθεί η ύπαρξη και η χρήση του πληροφοριακού εργαστηριακού συστήματος αιμοδοσίας.

Συμπερασματικά, η διαχείριση της πληροφορίας γίνεται στιβαρή με ελαχιστοποίηση λαθών, με υποδιπλασιασμό σχεδόν του απαιτούμενου χρόνου, με δραστική μείωση του αριθμού των επανεξετάσεων μέσω ενσωματωμένου συστήματος ελέγχου ποιότητας και συνεπώς ουσιαστική μείωση του κόστους παραγωγής, τόσον από πλευράς αναλώσιμων υλικών όσο και από πλευράς χρόνου

απασχόλησης προσωπικού. Ο συνδυασμός της μείωσης του κόστους και της αύξησης της αξιοπιστίας των μετρήσεων που παρέχει ένα ολοκληρωμένο πληροφοριακό σύστημα εργαστηρίων, είναι προφανές ότι έχει τεράστια και ουσιαστικότατα οφέλη.

2.3.3 Πληροφοριακά συστήματα διαγνωστικών κέντρων

Τα διαγνωστικά κέντρα αποτελούν οργανισμούς ή επιχειρήσεις κερδοσκοπικού χαρακτήρα που δραστηριοποιούνται στον ιατρικό χώρο με επιτυχία προσφέροντας ιατρικές υπηρεσίες υψηλού ποιοτικού επιπέδου. Σκοπός τους είναι η έγκυρη και έγκαιρη διάγνωση για πρόληψη και θεραπεία προβλημάτων υγείας. Επιπλέον, στόχος τους αποτελεί η παροχή υπηρεσιών κάτω από άριστες συνθήκες, με ιδιαίτερη φροντίδα, συνέπεια και επιστημονική πληρότητα. Τα διαγνωστικά κέντρα έκαναν την εμφάνισή τους από το 1980 και μετά. Ραγδαία ήταν η ανάπτυξη τους στην Ελλάδα τα τελευταία χρόνια και πιο συγκεκριμένα στην περίοδο 1990-1995. Λειτουργούν σε άνετους χώρους, με σύγχρονα μηχανήματα και με εξειδικευμένους γιατρούς. Σήμερα ο συνολικός αριθμός των διαγνωστικών κέντρων που λειτουργούν στη χώρα μας εκτιμάται ότι αγγίζει τα 400.

Η χρήση των πληροφοριακών συστημάτων στα διαγνωστικά κέντρα είναι αναγκαία. Παρόλο που το πεδίο των υπηρεσιών τους είναι μικρότερο από αυτό των νοσοκομείων, κρίνεται απαραίτητη η ύπαρξη πληροφοριακών συστημάτων. Οι νοσηλευτικές υπηρεσίες διευκολύνονται μέσω του σύγχρονου τεχνολογικού εξοπλισμού και των πληροφοριακών συστημάτων που εφαρμόζονται. Πολλές χειρονακτικές εργασίες αυτοματοποιούνται, με αποτέλεσμα η επεξεργασία των δεδομένων και οι διάφορες διεργασίες να γίνονται πολύ ταχύτερα. Η γρηγορότερη, λοιπόν, διεκπεραίωση των εργασιών συνεπάγεται την καλύτερη οικονομική και διοικητική οργάνωση του διαγνωστικού κέντρου. Τα έσοδα και οι δαπάνες προϋπολογίζονται και υπολογίζονται με τον καλύτερο δυνατό τρόπο, συνεπώς γίνεται αποτελεσματικότερη η διαχείριση των οικονομικών του κέντρου.

Επιπλέον, στην καλύτερη εφαρμογή των πληροφοριακών διαγνωστικών συστημάτων συντελεί η καταχώρηση των προσωπικών δεδομένων των ασθενών σε ιατρικούς φακέλους με την ταυτόχρονη επικοινωνία με τους άλλους τομείς του συστήματος. Υλοποιείται σε διάφορα κέντρα ηλεκτρονική εφαρμογή που δίνει την δυνατότητα στους γιατρούς να διαχειρίζονται και να επεξεργάζονται τον Ηλεκτρονικό Ιατρικό Φάκελο ασθενών. Ο γιατροί είτε μέσω επιτραπέζιου ηλεκτρονικού υπολογιστή (desktop pc), είτε μέσω φορητού υπολογιστή (laptop pc) αλλά κυρίως μέσω υπολογιστή παλάμης (pocket pc) θα μπορούν να δουν, περισσότερο στο μέλλον, και να επεξεργαστούν το ιστορικό και τα δημογραφικά στοιχεία του ασθενούς καθώς επίσης και τα αποτελέσματα των εργαστηριακών εξετάσεων.

Ακόμη, η χρήση πληροφοριακών συστημάτων υποστηρίζει την εφαρμογή της τηλεϊατρικής και των έμπειρων συστημάτων και στα διάφορα διαγνωστικά κέντρα με τη διαφορά από τα νοσοκομειακά ιδρύματα ότι το πεδίο παροχής ιατρικών υπηρεσιών στα διαγνωστικά κέντρα είναι πιο περιορισμένο. Σε πολλά διαγνωστικά κέντρα χρησιμοποιούνται κάποιες εφαρμογές που αποτελούν μερικώς πληροφοριακά συστήματα. Αναπτύσσονται υψηλής απόδοσης μαζικής αποθήκευσης συστήματα που

συνδυάζουν την ταχύτητα των παράλληλων συστημάτων και την λειτουργικότητα της μαζικής αποθήκευσης με ιεραρχική δομή. Το αποτέλεσμα είναι συστήματα με ανοιχτή αρχιτεκτονική προσβάσιμη από οποιοδήποτε δίκτυο που υποστηρίζει γνωστά πρότυπα. Δίνεται, έτσι, η δυνατότητα ανάπτυξης συστημάτων ικανών να αποθηκεύσουν μεγάλους όγκους πληροφορίας (ιατρικός φάκελος) με δυνατότητα άμεσης ανάκτησης και αποθήκευσης δεδομένων.

Τα προγράμματα αυτά εκτός των άλλων προσφέρουν :

- Ανοιχτή αρχιτεκτονική για εύκολη πρόσβαση
- Κατασκευή συστημάτων από χαμηλού κόστους αποθηκευτικά μέσα
- Είναι εφαρμόσιμα σε διάφορα συστήματα
- Απεριόριστο αριθμό συνδέσεων
- Λιακωτό απόδοση στη διαδικασία μετάπτωσης αρχείων
- Συνεργάσιμα με τα πιο γνωστά είδη αποθηκευτικών μέσων

Οι υπηρεσίες που προσφέρονται από τέτοιου είδους εφαρμογές είναι:

- Ασφαλής αποθήκευση και ανταλλαγή ιατρικών αρχείων σε πραγματικό χρόνο.
- Ασφαλής σύνδεση με τον φάκελο του ασθενούς μέσω κινητού τηλεφώνου τρίτης γενεάς.
- Φιλικές προς τον χρήστη διαδικασίες ώστε να γίνεται προσιτό ακόμα και στον άπειρο χρήστη.

Για την υλοποίηση τέτοιων εφαρμογών, συστημάτων υπάρχει συνεργασία μεταξύ οργανισμών από Γαλλία, Ιταλία και Ελλάδα.

Η ραγδαία εξέλιξη της τεχνολογίας ανεβάζει καθημερινά τα standards σε κάθε διαγνωστικό κέντρο. Σκοπός τους, λοιπόν, είναι να είναι πρωτοπόρα και σε αυτόν τον τομέα που ονομάζεται τεχνολογία και που είναι σημαντικότατος στον χώρο της υγείας. Προγράμματα και εφαρμογές που αποτελούν μερικώς πληροφοριακά συστήματα σίγουρα βοηθάνε με τον καλύτερο τρόπο προς αυτόν τον σκοπό εφόσον η χρήση ολοκληρωμένων πληροφοριακών συστημάτων δεν είναι ακόμη διαδεδομένη και εφικτή.

2.3.4 Νοσοκομειακά πληροφοριακά συστήματα

Τα πληροφοριακά συστήματα νοσοκομείου είναι μεγάλα, περίπλοκα συστήματα υπολογιστών που έχουν σχεδιαστεί για να βοηθούν στην επικοινωνία και στη διαχείριση των αναγκών πληροφόρησης ενός νοσοκομείου. Αποτελούν εργαλεία για ενδοτομεακή και διατομεακή χρήση. Ένα πληροφοριακό σύστημα νοσοκομείου έχει εφαρμογή σε θέματα εισαγωγής ασθενών, σε ιατρικά αρχεία, σε λογιστικές πληροφορίες, επιχειρησιακές υπηρεσίες, νοσηλευτική, εργαστήρια, ακτινολογικό, φαρμακείο, κεντρικές προμήθειες, διαιτολογικές υπηρεσίες, προσωπικό και μισθοδοσία. Πολλές άλλες εφαρμογές μπορούν να υπάρξουν για κάθε τμήμα και ουσιαστικά για κάθε σκοπό.

Οι εφαρμογές που αφορούν την εισαγωγή ασθενών περιλαμβάνουν προγραμματισμό ασθενών, προεισαγωγική φάση, φάση εισαγωγής, φάση εξόδου από το νοσοκομείο, μεταφορές και διαδικασίες καταγραφής. Ορισμένες εφαρμογές που αφορούν ιατρικά αρχεία περιλαμβάνουν την τήρηση γενικού μητρώου ασθενών, έγγραφα, αλληλογραφία και διαδικασίες εντοπισμού ιατρικών αρχείων. Οι επιχειρησιακές και λογιστικές διαδικασίες περιλαμβάνουν επιβεβαίωση ασφάλειας ασθενούς, χρέωση παρεχομένων υπηρεσιών, παρακολούθηση μετά τη χρέωση, επίλυση αποριών όσον αφορά τις χρεώσεις, λογαριασμούς πληρωτέους, λογαριασμούς εισπρακτέους, διαχείριση μετρητών και τήρηση αρχείου υπηρεσιών και τρίτων φορέων.

Οι εφαρμογές σε άλλους τομείς όπως η νοσηλευτική, τα εργαστήρια, το ακτινολογικό, το φαρμακείο και το τμήμα κεντρικών προμηθειών μπορεί να είναι πολλές και περίπλοκες και να διαθέτουν δικά τους πληροφοριακά συστήματα. Τα συστήματα αυτά ξεχωρίζουν και λειτουργούν ανεξάρτητα από το πληροφοριακό σύστημα του νοσοκομείου, αλλά συνήθως συνδέονται μεταξύ τους για τη μεταβίβαση πληροφοριών.

Από τη χρήση ενός πληροφοριακού συστήματος νοσοκομείου μπορούν να εξαχθούν χρήσιμα συμπεράσματα ως προς τον τρόπο λειτουργίας του νοσοκομείου. Η εξαγωγή των συμπερασμάτων αυτών μπορεί να γίνει με την ανάλυση των στατιστικών δεδομένων του συστήματος καθώς και με τη χρήση εργαλείων τα οποία παρέχουν τη δυνατότητα προσομοίωσης της λειτουργίας του νοσοκομείου μετά την υλοποίηση μιας ή και περισσοτέρων αλλαγών. Το κύριο πλεονέκτημα των εργαλείων αυτών είναι η δυνατότητα παροχής της εικόνας της λειτουργίας του νοσοκομείου καθώς και των συνεπειών (με ένα πολύ μεγάλο βαθμό αξιοπιστίας) πριν από την πραγματική τους υλοποίηση. Επίσης, τα ΠΣΝ μπορούν να χρησιμοποιηθούν για την αναδιοργάνωση της λειτουργίας του νοσοκομείου ή του οργανισμού (όταν αυτή επιβάλλεται) με τον ανασχεδιασμό της ροής των εργασιών τους. Ακόμη, τα πληροφοριακά συστήματα μπορούν να χρησιμοποιηθούν για να δεσμεύσουν τους προμηθευτές ενός νοσοκομείου να υλοποιήσουν ηλεκτρονικές συνδέσεις μεταξύ τους, έτσι ώστε να παρέχουν βελτιωμένες σε ποιότητα υπηρεσίες.

Τα πληροφοριακά συστήματα νοσοκομείου τείνουν να αναπτύσσονται με κεντρικό υπολογιστή και τερματικά, παρόλο που σήμερα παρατηρείται μια στροφή προς τον περιορισμό του μεγέθους και τη διασπορά των δικτύων δεδομένων. Η επιλογή, η ανάπτυξη και η υλοποίηση ενός νοσοκομειακού πληροφοριακού συστήματος μπορεί να διαρκέσει χρόνια. Τα πλεονεκτήματα και τα μειονεκτήματα κάθε στρατηγικής σταθμίζονται πριν υλοποιηθεί κάποιο πληροφοριακό σύστημα. Το χρονικό διάστημα ποικίλλει ανάλογα με το σύστημα και την πολυπλοκότητα των εφαρμογών του. Στην ουσία μπορεί να είναι μια συνεχής διαδικασία. Το αρχικό κόστος για την εξασφάλιση των μηχανημάτων και του λογισμικού, καθώς και η ετήσια διαρκής συντήρηση απαιτεί την καταβολή πολύ υψηλών χρηματικών ποσών. Με την πάροδο του χρόνου, το νοσοκομείο «μαθαίνει» πώς να προσαρμοστεί και να χρησιμοποιεί μία συγκεκριμένη καινούρια τεχνολογία.

Όπως φαίνεται στο Σχήμα 3, υπάρχουν τέσσερα (4) στάδια εκμάθησης τα οποία αναφέρονται.

Σχήμα 3 : Σταδία υιοθέτησης της τεχνολογίας στον χώρο της υγείας [24]

Στάδια εκμάθησης της τεχνολογίας στον χώρο της υγείας

1.Πρώτη εφαρμογή της τεχνολογίας: Στο στάδιο αυτό η εφαρμογή της τεχνολογίας γίνεται σε μερικά τμήματα του νοσοκομείου μετά από εκτεταμένη ανάλυση και με μεγάλη προσοχή. Η επέκταση της στα υπόλοιπα τμήματα είναι αργή, καθώς το νοσοκομείο προσπαθεί να κατανοήσει την εφαρμογή της, τις δυνατότητες που παρέχει για την υποστήριξη των εργασιών του, όπως επίσης το κόστος και τα οφέλη από την επέκταση αυτή.

2.Μαζική επέκταση της τεχνολογίας: Το στάδιο αυτό χαρακτηρίζεται από τη μαζική επέκταση της τεχνολογίας στα υπόλοιπα τμήματα του νοσοκομείου. Καθώς το νοσοκομείο αρχίζει να αισθάνεται περισσότερη ασφάλεια ως προς την τεχνολογία που εφαρμόζει, ξεκινά πολλά προγράμματα επέκτασης της. Στο στάδιο αυτό απαιτείται μεγάλη προσοχή, αφού υπάρχει ο κίνδυνος απώλειας του ελέγχου της επέκτασης της, όταν γίνεται χωρίς κατάλληλη
ανάλυση, σχεδιασμό και παρακολούθηση.

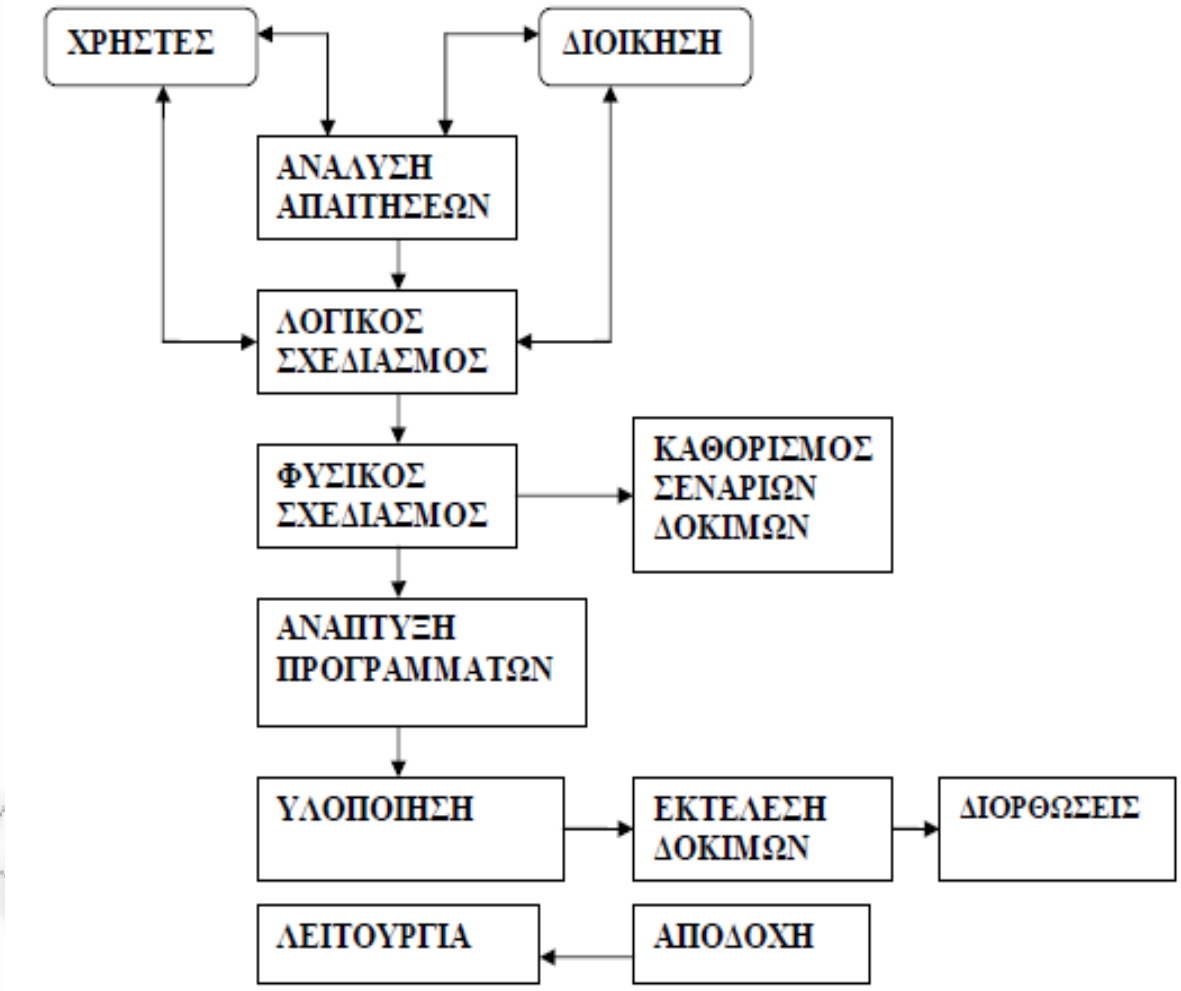
3.Ελεγχόμενη επέκταση της τεχνολογίας: Στο στάδιο αυτό οι δραστηριότητες επέκτασης της τεχνολογίας στο νοσοκομείο γίνονται κάτω από σημαντικό έλεγχο, ο οποίος περιλαμβάνει την εφαρμογή μεθοδολογιών, τεχνικών διαχείρισης έργων κ.λ.π.

4.Ωρίμανση: Το νοσοκομείο έχει μάθει πώς να διαχειρίζεται τη συγκεκριμένη τεχνολογία και την εφαρμόζει σε όλα του τα τμήματα. Γενικά, τα πληροφοριακά συστήματα στα νοσοκομεία, συνήθως, χρησιμοποιούνται για τη άμεση και σωστή προσαρμογή της λειτουργίας τους σε αλλαγές που γίνονται στο εσωτερικό ή στο εξωτερικό τους περιβάλλον (π.χ. αλλαγή του τρόπου συνταγογράφησης και χρέωσης ενός ασθενή). Ωστόσο, υπάρχει και μία άλλη προσέγγιση στη χρήση ενός πληροφοριακού συστήματος. Η προσέγγιση αυτή ορίζει ότι ένα πληροφοριακό σύστημα μπορεί να χρησιμοποιηθεί κατά τέτοιο τρόπο, ώστε να επηρεάζει τις αλλαγές που συμβαίνουν στο νοσοκομειακό περιβάλλον. Η επιρροή αυτή είναι πάντα

προς όφελος του νοσοκομείου και συνεπώς των ασθενών του. Πιο συγκεκριμένα, με βάση την προσέγγιση αυτή, το πληροφοριακό σύστημα χρησιμοποιείται ως μέσο με το οποίο μπορούν να παρατηρηθούν δυνατότητες βελτίωσης της λειτουργίας του νοσοκομείου, οι οποίες δεν υπαγορεύονται πάντα από το περιβάλλον του.

2.4 ΚΥΚΛΟΣ ΖΩΗΣ ΤΩΝ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΥΓΕΙΑΣ

Ο κύκλος ζωής πληροφοριακών συστημάτων υγείας περιλαμβάνει τις φάσεις που απαιτούνται για την ανάπτυξη, λειτουργία και συντήρηση τους. Σε κάθε φάση εκτελούνται συγκεκριμένες εργασίες σε συγκεκριμένο χρόνο και με τη χρήση των απαιτούμενων πόρων. Επίσης, από κάθε φάση παράγονται συγκεκριμένα αποτελέσματα, τα οποία πρέπει να τεκμηριώνονται επαρκώς. Ένας τυπικός κύκλος ζωής πληροφοριακών συστημάτων αποτελείται από έξι φάσεις: την ανάλυση απαιτήσεων, το λογικό σχεδιασμό, το φυσικό σχεδιασμό, την ανάπτυξη προγραμμάτων, την υλοποίηση και τη λειτουργία. Η σχέση των φάσεων αυτών φαίνεται στο ακόλουθο σχήμα.



Σχήμα 4 Κύκλος ζωής πληροφοριακού συστήματος υγείας [24]

2.4.1 Ανάλυση απαιτήσεων

Η πρώτη φάση του κύκλου ζωής αφορά τον προσδιορισμό των απαιτήσεων του πληροφοριακού συστήματος υγείας. Συγκεκριμένα, κατά τη φάση αυτή γίνονται επαναλαμβανόμενες συναντήσεις μεταξύ των αναλυτών του συστήματος και των υπευθύνων του οργανισμού έτσι ώστε να προσδιοριστεί σαφώς το πλαίσιο του συστήματος και οι δυνατότητες επικοινωνίας του με άλλα συστήματα.

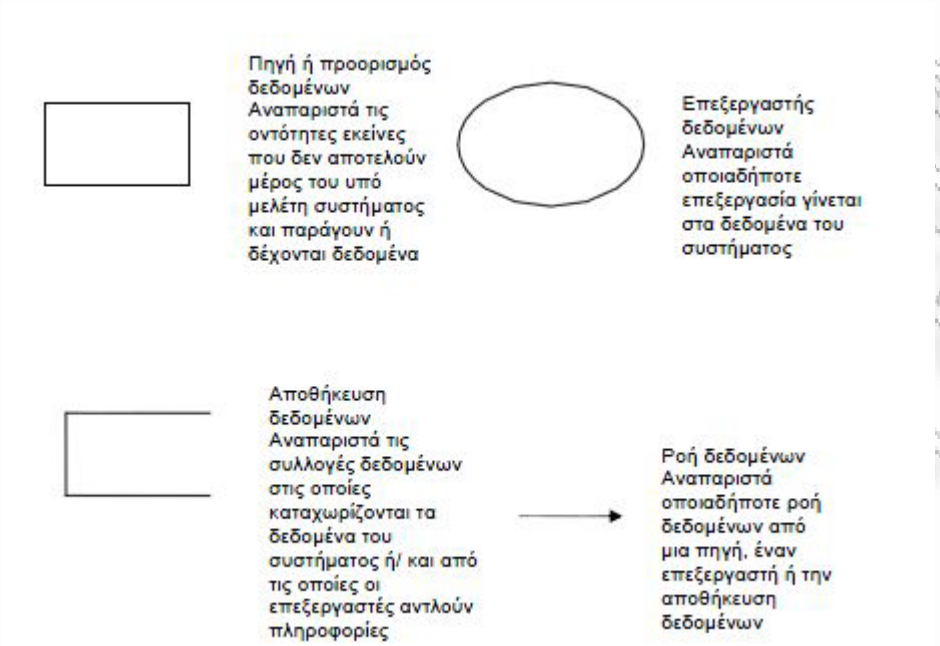
Τα αποτελέσματα των συναντήσεων αυτών μελετώνται από τους αναλυτές, οι οποίοι στη συνέχεια υποβάλλουν στον οργανισμό την πρόταση τους στην οποία δίνεται η περιγραφή του συστήματος, οι απαιτούμενοι πόροι, ο αναμενόμενος χρόνος υλοποίησης του καθώς και το κόστος του. Στη συνέχεια, και μετά την αποδοχή της πρότασης από τους υπεύθυνους του οργανισμού, εκτελούνται οι ακόλουθες εργασίες: μελέτη της τρέχουσας λειτουργίας του οργανισμού, καταγραφή εναλλακτικών λύσεων, αξιολόγηση και επιλογή της καταλληλότερης λύσης. Με βάση τις πληροφορίες αυτές κατασκευάζεται το μοντέλο λειτουργίας του νοσοκομείου το οποίο δίνει τη δυνατότητα ολοκληρωμένης και λεπτομερούς θεώρησης και μελέτης του.

Μετά την επιλογή της καταλληλότερης λύσης από τον οργανισμό, προετοιμάζεται το πρόγραμμα έργου το οποίο περιλαμβάνει τους στόχους του, τις εργασίες που θα εκτελεστούν, τα χρονικά σημεία στα οποία περατώνεται η εκτέλεση ενός συνόλου εργασιών ή ξεκινά η έναρξη ενός άλλου και γίνεται ο έλεγχος της επίτευξης των στόχων. Ο έλεγχος αυτός περιλαμβάνει τη σύγκριση του προϋπολογιζόμενου και του πραγματικού κόστους, καθώς και την ανασκόπηση της ποιότητας της εργασίας που έχει εκτελεστεί.

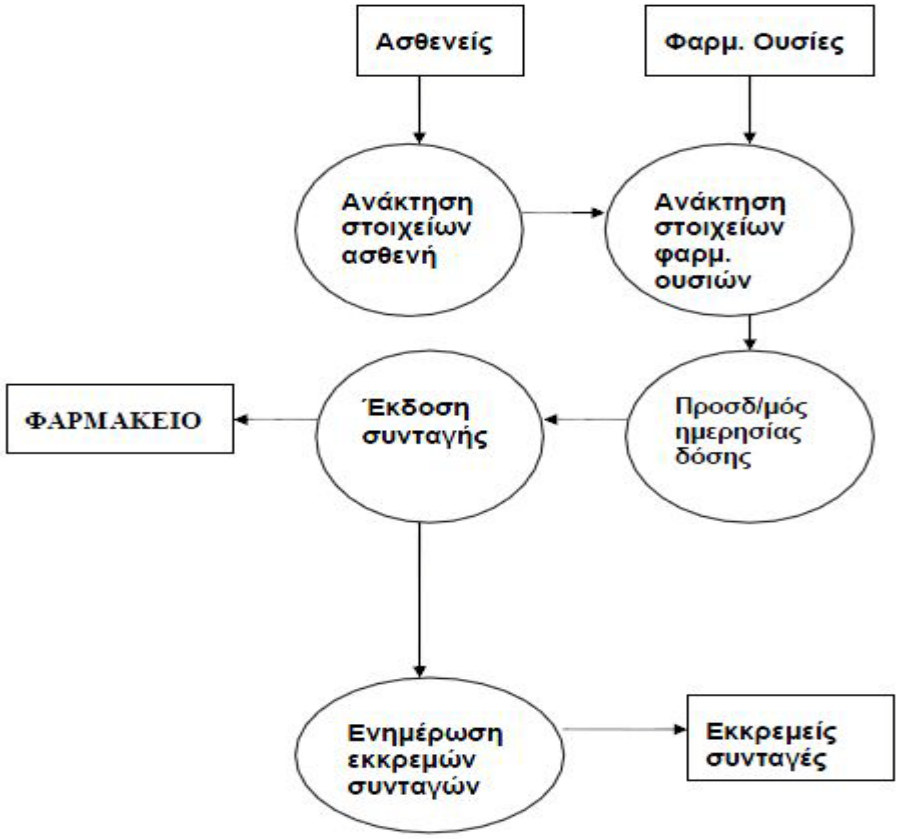
2.4.2 Λογικός σχεδιασμός

Κατά τη φάση αυτή καθορίζεται λογικά η δομή του πληροφοριακού συστήματος υγείας. Συγκεκριμένα, οι πληροφοριακές απαιτήσεις του οργανισμού μετασχηματίζονται σε ένα εννοιολογικό μοντέλο του νέου συστήματος. Για την κατασκευή του μοντέλου αυτού μπορεί να χρησιμοποιηθούν διάφορες τεχνικές, όπως τα διαγράμματα ροής δεδομένων, το λεξικό δεδομένων, τα διαγράμματα δομής δεδομένων, οι πίνακες αποφάσεων κ.λπ.

Συγκεκριμένα, τα **διαγράμματα ροής δεδομένων** είναι μία από τις πιο γνωστές μεθόδους ανάλυσης και σχεδιασμού ενός πληροφοριακού συστήματος. Χρησιμοποιώντας τη μέθοδο αυτή μπορούμε να περιγράψουμε την τρέχουσα λειτουργία ενός συστήματος, αναπαριστώντας τα επιμέρους συστατικά του σε οποιοδήποτε επιθυμητό επίπεδο λειτουργίας. Έτσι, μέσω διαγραμμάτων ροής δεδομένων σχηματίζουμε μία παραστατική εικόνα μέσω της οποίας μπορούμε να σχεδιάσουμε τις προτεινόμενες λύσεις. Τα σύμβολα που χρησιμοποιούνται είναι τα ακόλουθα



Στο παρακάτω σχήμα 5 παρουσιάζεται το διάγραμμα ροής δεδομένων της διαδικασίας «Συνταγογράφηση»



Σχήμα 5 : Συνταγογράφηση [24]

Λεξικό δεδομένων ονομάζεται το σύνολο των πληροφοριών σχετικά με τα δεδομένα που χρησιμοποιεί ή προβλέπεται ότι θα χρησιμοποιήσει το πληροφοριακό σύστημα. Οι πληροφορίες που περιλαμβάνει αφορούν το όνομα, την κατηγορία του, τη χρήση του, τον τρόπο διαχείρισης του κ.λπ. **Τα διαγράμματα δομών** χρησιμοποιούνται για τη λειτουργική διάσπαση ενός συστήματος σε υποσυστήματα και είναι ιδιαίτερα χρήσιμα σε συστήματα μεγάλου μεγέθους. **Πίνακας αποφάσεων** ονομάζεται ένας πίνακας στον οποίο καταγράφονται οι λειτουργικοί κανόνες ενός οργανισμού με τη μορφή συνθηκών και ενεργειών.

Από τη κατασκευή των μοντέλων αυτών καταγράφονται συμπεράσματα τα οποία αφορούν:

1. Τα κύρια συστατικά υποσυστήματα του οργανισμού και τους τρόπους επικοινωνίας μεταξύ τους ή και με συστήματα άλλων οργανισμών,

2. Τις εργασίες που εκτελούνται σε καθένα από τα υποσυστήματα, καθώς και τον τρόπο εκτέλεσης τους,

3. Τη ροή δεδομένων από τη πηγή μέχρι και τη διάθεση τους στους τελικούς χρήστες,

4. Τις επεξεργασίες και τους μετασχηματισμούς των δεδομένων από την εκτέλεση των εργασιών,

5. Τα είδη των αρχείων που χρησιμοποιούνται για την αποθήκευση των δεδομένων και

6. Τις απαιτήσεις σε πόρους (ανθρώπινους, υλικούς και χρηματικούς) για την εκτέλεση των εργασιών.

2.4.3 Φυσικός σχεδιασμός

Κατά τη φάση αυτή γίνεται ο σαφής καθορισμός των υποσυστημάτων, της βάσης δεδομένων και των προγραμμάτων εφαρμογών του πληροφοριακού συστήματος. Τη βάση για τον καθορισμό αυτό αποτελούν τα συμπεράσματα του λογικού σχεδιασμού. Συνήθως, χρησιμοποιούνται μέθοδοι για το σχεδιασμό των λογικών τμημάτων στο λογισμικό των εφαρμογών του πληροφοριακού συστήματος, όπως είναι τα διαγράμματα δομής δεδομένων.

Ο σχεδιασμός της βάσης δεδομένων αφορά τη λογική και φυσική δόμηση των δεδομένων και τον καθορισμό των μεθόδων προσπέλασης τους. Συγκεκριμένα, ο σχεδιασμός αυτός περιλαμβάνει τον καθορισμό των αρχείων δεδομένων, τις τεχνικές προσπέλασης τους, τα προβλεπόμενα μεγέθη αρχείων δεδομένων και ευρετηρίων, τη διαδικασία λήψης αντιγράφων, τις επιπτώσεις από την αναδιοργάνωση τη βάσης δεδομένων και το σύστημα ασφαλείας της. Τα αποτελέσματα της φάσης αυτής υποβάλλονται στη διοίκηση του οργανισμού για μελέτη και αποδοχή και χρησιμοποιούνται ως βάση για την επόμενη φάση, την ανάπτυξη των προγραμμάτων.

2.4.4 Ανάπτυξη προγραμμάτων

Κατά τη φάση αυτή τα λογικά τμήματα του λογισμικού των εφαρμογών που προσδιορίστηκαν το στάδιο του φυσικού σχεδιασμού υλοποιούνται και ενώνονται μεταξύ τους ενώ παράλληλα υλοποιείται και η βάση δεδομένων του συστήματος. Η συγγραφή των προγραμμάτων εκτελείται από την ομάδα προγραμματιστών του συστήματος και γίνεται με τη χρήση κάποιας γλώσσας προγραμματισμού (π.χ.Java, c++, Visual Basic κ.λπ.). Σε μεγάλα έργα ανάπτυξης πληροφοριακών συστημάτων η ομάδα των προγραμματιστών διασπάται σε υποομάδες, καθεμία από τις οποίες έχει ένα προϊστάμενο προγραμματιστή , οι οποίες αναλαμβάνουν τη συγγραφή ενός συνόλου εφαρμογών (που συνήθως αφορούν ένα ανεξάρτητο υποσύστημα). Το αποτέλεσμα της φάσης αυτής είναι το ολοκληρωμένο λογισμικό εφαρμογών του πληροφοριακού συστήματος και το αντίστοιχο υποστηρικτικό υλικό.

2.4.5 Υλοποίηση

Κατά τη φάση αυτή γίνεται η δοκιμή του λογισμικού των εφαρμογών, εκπαιδεύονται οι χρήστες και εγκαθίστανται το νέο σύστημα.

Η δοκιμή του λογισμικού αφορά τόσο τη δοκιμή του κώδικα όσο και τον έλεγχο της ικανοποίησης των προδιαγραφών του συστήματος, όπως ορίστηκαν στις προηγούμενες φάσεις. Για τη δοκιμή αυτή δημιουργούνται διάφορα σενάρια εκτέλεσης των εφαρμογών έτσι ώστε να ελεγχθούν όλες οι δυνατές περιπτώσεις. Για παράδειγμα, για τον έλεγχο του κώδικα τα σενάρια αυτά εξασφαλίζουν ότι θα ελεγχθεί η εκτέλεση κάθε γραμμής εντολής που περιλαμβάνει. Η δοκιμή των εφαρμογών γίνεται τόσο στο επίπεδο μονάδας όσο και στο επίπεδο ολοκληρωμένου λογισμικού. Στην πρώτη περίπτωση η κάθε εφαρμογή εξετάζεται ανεξάρτητα από τις άλλες με σκοπό να βρεθούν τυχόν λογικά ή προγραμματιστικά λάθη. Στη δεύτερη περίπτωση η δοκιμή αφορά τον έλεγχο και τον εντοπισμό τυχόν λαθών ως προς την ικανοποίηση των αρχικών προδιαγραφών και την επικοινωνία των εφαρμογών μεταξύ τους. Στο στάδιο αυτό εξετάζεται επίσης και η υλοποίηση της βάσης δεδομένων. Ελέγχεται δηλαδή η δυνατότητα του συστήματος να αντεπεξέλθει στον μέγιστο φόρτο εργασίας, ο χρόνος απόκρισής του καθώς και η δυνατότητα ανάκαμψης του συστήματος μετά από μία βλάβη.

Παράλληλα με τη δοκιμή του συστήματος γίνεται και η εκπαίδευση των τελικών χρηστών. Κάθε χρήστης πρέπει να γνωρίζει τον ακριβή ρόλο του, τον τρόπο χρήσης του συστήματος και τις δυνατότητες που αυτό του παρέχει. Η εκπαίδευση που παρέχεται δεν είναι η ίδια για όλους τους χρήστες. Ανάλογα με την ειδικότητα, τη θέση στην ιεραρχία του οργανισμού και την προβλεπόμενη χρήση, κάθε χρήστης έχει και την κατάλληλη εκπαίδευση.

Τέλος, στη φάση της υλοποίησης του συστήματος περιλαμβάνεται και η μετάβαση στο νέο σύστημα. Γενικά, υπάρχουν τέσσερις προσεγγίσεις μετάβασης : η παράλληλη, η τμηματική, η πιλοτική και η άμεση. Σύμφωνα με την παράλληλη προσέγγιση, το υπάρχον και το νέο πληροφοριακό σύστημα λειτουργούν ταυτόχρονα για ένα χρονικό διάστημα κατά το οποίο συγκρίνονται τα αποτελέσματα τους.

Ακολουθώντας την τμηματική προσέγγιση, η λειτουργία του νέου πληροφοριακού συστήματος ξεκινά σε συγκεκριμένα τμήματα του οργανισμού. Μετά την εξασφάλιση της επιτυχίας του, το σύστημα επεκτείνεται για να καλύψει και τη λειτουργία άλλων τμημάτων και στη συνέχεια εγκαθίστανται και σε αυτά. Με την πιλοτική προσέγγιση, το πληροφοριακό σύστημα υγείας υλοποιείται για ένα ή περισσότερα τμήματα του οργανισμού που είναι αντιπροσωπευτικά της όλης λειτ ργίας το υ Τέλο ς υπάρχει και η άμεση προ έγγιση η ο π ά θεωρείται συντομότερη όλων. Σύμφωνα με αυτή, το νέο πληροφοριακό σύστημα αντικαθιστά ολοκληρωτικά το παλιό σε ένα συγκεκριμένο χρονικό σημείο. Μία από τις σημαντικότερες προϋποθέσεις της προσέγγισης αυτής είναι ο καλός χρονικός προγραμματισμός.

2.4.6 Λειτουργία

Μετά την υλοποίηση του συστήματος ακολουθεί το στάδιο της λειτουργίας του κατά το οποίο πρέπει να εξασφαλιστεί ότι το σύστημα παρέχει τα αναμενόμενα οφέλη στον οργανισμό. Κατά τη διάρκεια λειτουργίας το συστήματος είναι δυνατόν να βρεθούν λάθη μικρής κλίμακας τα οποία και διορθώνονται αμέσως. Επίσης, είναι δυνατόν να ζητηθεί η βελτίωση των εφαρμογών ή και η ανάπτυξη νέων με σκοπό τη βελτίωση της αποδοτικότητας όλου του συστήματος.

Τα παραδοτέα της φάσης αυτής είναι το τεκμηριωτικό υλικό του συστήματος: το εγχειρίδιο λειτουργίας το οποίο αφορά τον τρόπο λειτουργίας του συστήματος και απευθύνεται στο προσωπικό μηχανογράφησης του οργανισμού, το εγχειρίδιο συντήρησης το οποίο περιέχει τις διαδικασίες συντήρησης του συστήματος και προορίζεται για αναλυτές και προγραμματιστές και το εγχειρίδιο χρήσης το οποίο περιέχει οδηγίες για τον τρόπο χρήσης του συστήματος και αφορά τους τελικούς χρήστες.

2.5 JAVA

Η Java είναι μία αντικειμενοστρεφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems. Αρχικός στόχος ήταν η ανάπτυξη μιας γλώσσας προγραμματισμού, κατάλληλη για ανάπτυξη εφαρμογών σε μικρο-συσκευές. Οι γλώσσες προγραμματισμού που χρησιμοποιούνταν πριν την Java ήταν η C και η C++, αλλά ειχε αποδειχθεί ότι δεν μπορούσαν να επιτύχουν τον στοχο της Java. Οπότε ο James Gosling, που εργαζόταν για την Sun, επηρεάστηκε απο την C++ για να αναπτύξει την Java.

Τα κυριότερα πλεονεκτήματα που έχει η Java, σε σχέση με την C++, είναι:

1. Ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Macintosh, αλλά και άλλες πλατφόρμες για τις οποίες υπάρχει κατάλληλο JVM (Java Virtual Machine), χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει

ο πηγαίος κώδικας[4] για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός (assembly) κώδικας που εκτελείται σε Windows είναι διαφορετικός από αυτόν που εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη της Εικονικής Μηχανής (Virtual Machine ή VM ή ΕΜ στα ελληνικά).

2. Είναι πολυνηματική, δηλαδή ένα απλό πρόγραμμα σε Java μπορεί να κάνει πολλά, διαφορετικά προγράμματα ανεξάρτητα και αλληλεπιδρώντα.

3. Είναι ασφαλής, καθώς οι κλάσεις αλλά και οι μέθοδοι μπορούν να προσδιοριστούν ως public, private ή protected ώστε να περιορίζεται το από ποια σημεία του κώδικα μπορούν να κληθούν αυτά τα κομμάτια κώδικα.

4. Έχει annotations μέσω των οποίων μπορούν να οριστούν κάποιες μεταπληροφορίες για τον κώδικα οι οποίες μπορούν να οδηγήσουν στην ελαχιστοποίηση του κώδικα. Για παράδειγμα μπορεί να οριστεί η αντιστοίχιση μιας κλάσης με έναν πίνακα χρησιμοποιώντας ένα annotation @Table().

5. Είναι αντικειμενοστραφής κι απλή, δηλαδή ακολουθεί την θεωρία των αντικειμενοστραφών γλωσσών προγραμματισμού χωρίς να έχει την πολυπλοκότητα της C++. Κάθε κλάση στην Java μπορεί να επεκτείνει μόνο μια κλάση, εν αντίθεση με την C++ που μπορεί να επεκτείνει πολλές.

2.5.1 Η εικονική μηχανή της Java

Αφού γραφεί κάποιο πρόγραμμα σε Java τότε μεταγλωττίζεται μέσω του εργαλείου javac, το οποίο παράγει έναν αριθμό από αρχεία .class (=bytecode). Το bytecode είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν προσπαθήσουμε λοιπόν να εκτελέσουμε την εφαρμογή μας το Java Virtual Machine που πρέπει να είναι εγκατεστημένο στο μηχάνημά μας, θα αναλάβει να διαβάσει τα αρχεία .class και να τα μεταφράσει σε γλώσσα και εντολές μηχανής (assembly) που υποστηρίζει το λειτουργικό μας σύστημα και ο επεξεργαστής μας, έτσι ώστε να εκτελεστεί (να σημειώσουμε εδώ ότι αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine) . Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν πολύχρηστα τμήματα bytecode απ' ευθείας σε εγγενή κώδικα (native code) με αποτέλεσμα να βελτιώνεται η ταχύτητα). Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Πρέπει να πούμε ότι η JVM είναι λογισμικό εξαρτημένο από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει

[4] Εξαίρεση αποτελει η περίπτωση που το πρόγραμμα χρησιμοποιεί κάποια λειτουργία που υποστηρίζεται μόνο από κάποιο συγκεκριμένο λειτουργικό σύστημα, ή επικοινωνεί με κάποια βιβλιοθήκη dll ή so.

διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Από την άλλη μεριά ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το δίκτυο και να τον εκτελέσει. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα συγχρόνως.

2.5.2 Ο συλλέκτης απορριμμάτων (Garbage Collector)

Ακόμα μία ιδέα που βρίσκεται πίσω από τη Java είναι η ύπαρξη του συλλέκτη απορριμμάτων (Garbage Collector). Συλλογή απορριμμάτων είναι μία κοινή ονομασία που χρησιμοποιείται στον τομέα της πληροφορικής για να δηλώσει την ελευθέρωση τμημάτων μνήμης από δεδομένα που δε χρειάζονται και δε χρησιμοποιούνται άλλο. Αυτή η απελευθέρωση μνήμης στη Java είναι αυτόματη και γίνεται μέσω του συλλέκτη απορριμμάτων. Υπεύθυνη για αυτό είναι και πάλι η εικονική μηχανή η οποία μόλις «καταλάβει» ότι ο σωρός (heap) της μνήμης (στη Java η συντριπτική πλειοψηφία των αντικειμένων αποθηκεύονται στο σωρό σε αντίθεση με τη C++ όπου αποθηκεύονται κυρίως στη στοίβα - stack) κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης, ούτε και για δείκτες (pointers) που αναφέρονται σε άδειο χώρο μνήμης. Αυτό είναι ιδιαίτερα σημαντικό αν σκεφτούμε ότι ένα μεγάλο ποσοστό κατάρρευσης των προγραμμάτων οφείλονται σε λανθασμένο χειρισμό της μνήμης.

2.6 WEB SERVICES

Τα web services είναι μια καινοτομική αρχιτεκτονική με την οποία παρέχεται η δυνατότητα δημιουργίας και χρήσης ηλεκτρονικών υπηρεσιών σε δίκτυο με απλό και οικονομικό τρόπο.

Η αρχιτεκτονική των web services μοιάζει σε έναν βαθμό με την κλασική αρχιτεκτονική του Client-Server, όπου ο εξυπηρετητής (Server) είναι ο πάροχος (Provider) της υπηρεσίας ενώ ο πελάτης (Client) είναι ο καταναλωτής (Consumer). Αυτή η χρήση των διαφορετικών όρων γίνεται διότι στον κόσμο των Web Services αυτό που παρέχεται είναι η υπηρεσία κι όχι η επικοινωνία. Ο consumer θέλει μια υπηρεσία και ο provider του την παρέχει.

Επίσης ο provider έχει μια περιγραφή των παρεχόμενων υπηρεσιών την οποία μπορεί ο consumer να «διαβάσει» και να κατανοήσει ώστε να ζητήσει την υπηρεσία που θέλει με τον τρόπο που πρέπει. Αυτή η περιγραφή δίδεται μέσω ενός τυποποιημένου XML που ονομάζεται WSDL (Web Service Description Language).

Το WSDL περιέχει πληροφορίες για το πως θα πρέπει να είναι ένα request. Υπάρχουν αρκετά εργαλεία παραγωγής κώδικα που λαμβάνουν ως είσοδο ένα wsdl και παράγουν ως έξοδο τον κατάλληλο κώδικα κλήσης των αντίστοιχων web services[5].

Η επικοινωνία μεταξύ παρόχου και καταναλωτή επιτυγχάνεται μέσω του πρωτοκόλλου επικοινωνία SOAP (Simple Object Access Protocol). Το πρωτόκολλο αυτό είναι πολύ πιο απλό από πρωτόκολλα παλαιότερων τεχνολογιών όπως αυτά που χρησιμοποιούνταν από τα κατανεμημένα περιβάλλοντα CORBA , DCOM, RPC. Έτσι το να δημιουργήσει κανείς μια υλοποίηση SOAP που υπόκειται στα πρότυπα (standards-compliant) είναι πολύ πιο εύκολο. Σήμερα μπορεί να βρει κανείς υλοποιήσεις του SOAP από τις μεγαλύτερες εταιρίες πληροφορικής αλλά ακόμη και από μεμονωμένους προγραμματιστές, πράγμα αδιανόητο για παλαιότερες κατανεμημένες τεχνολογίες. Με τις προηγούμενες τεχνολογίες η συνεργασία μεταξύ εταιριών ήταν ένα θέμα διότι κατανεμημένες τεχνολογίες όπως CORBA και DCOM χρησιμοποιούσαν μη πρότυπες πόρτες. Σαν αποτέλεσμα η συνεργασία σήμαινε άνοιγμα "οπών" στα τείχη προστασίας (firewalls) κάτι που πολλές φορές δεν ήταν αποδεκτό από τους ανθρώπους της πληροφορικής σε μια εταιρία αφού έθετε σε κίνδυνο στην ασφάλεια των συστημάτων. Το γεγονός αυτό δεν επέτρεπε δυναμική συνεργασία λόγω του ότι απαιτούσε μια χειροκίνητη διαδικασία για τη συνεργασία μιας εταιρίας με τους συνεργάτες της. Τα web services μπορούν να χρησιμοποιήσουν (μεταξύ άλλων) το HTTP ως πρωτόκολλο μεταφοράς και τα περισσότερα τείχη προστασίας επιτρέπουν την πρόσβαση μέσω της θύρας 80 (πρότυπη θύρα για το HTTP). Με αυτόν τον τρόπο οδηγούμαστε σε ευκολότερες και δυναμικές συνεργασίες μεταξύ των συστημάτων των εταιριών.

Οι προηγούμενες κατανεμημένες τεχνολογίες υπέφεραν από ζητήματα διαλειτουργικότητας διότι κάθε προμηθευτής (vendor) υλοποιούσε το δικό του πρότυπο για distributed object messaging. Με την XML σαν το μόνο πρότυπο στα web services, συστήματα φτιαγμένα από διαφορετικές τεχνολογίες όπως η Java και το .Net μπορούν να επικοινωνήσουν μεταξύ τους 4. Επιπλέον λόγω της απλότητας της XML είναι πολύ πιο εύκολο να γραφτούν νέες εφαρμογές σε μικρό χρονικό διάστημα.

2.6.1 Τα web services από την επιχειρηματική σκοπιά

Σε ένα υψηλότερο εννοιολογικό επίπεδο, μπορούμε να δούμε τα web services σαν μονάδες εργασίας (units of work). Ένα βήμα παραπέρα, αυτές οι μονάδες μπορούν να συνδυαστούν σε εργασίες επιχειρησιακού προσανατολισμού για να χειριστούν συγκεκριμένες επιχειρησιακές λειτουργίες. Αυτό με τη σειρά του επιτρέπει σε μη τεχνικούς ανθρώπους να σχεδιάσουν εφαρμογές οι οποίες μπορούν να χειριστούν τα επιχειρησιακά ζητήματα συνδυάζοντας τα web services σε ροές εργασίας. Σε μία αναλογία από τον χώρο των αυτοκινήτων, ο αρχιτέκτονας των επιχειρησιακών διαδικασιών μπορεί να συνδυάσει ολόκληρο τον κινητήρα με το πλαίσιο, τη μετάδοση και τα υπόλοιπα υποσυστήματα παρά να ασχοληθεί με τα εσωτερικά κομμάτια του κινητήρα. Επιπλέον η δυναμική πλατφόρμα σημαίνει ότι ο κινητήρας μπορεί να συνεργαστεί με τη μετάδοση ή άλλα υποσυστήματα διαφορετικών κατασκευαστών.

[5] Ενα τέτοιου είδου εργαλείο χρησιμοποιήθηκε και για την ανάπτυξη του consumer της παρούσας εργασίας.

Αυτό που προκύπτει από αυτή την πτυχή είναι ότι τα web services βοηθούν στη γεφύρωση του κενού που υπάρχει ανάμεσα στους ανθρώπους του επιχειρείν και τους ανθρώπους της πληροφορικής σε ένα οργανισμό. Οι άνθρωποι του επιχειρείν μπορούν να περιγράψουν γεγονότα και δραστηριότητες και οι τεχνικοί μπορούν να τα συνδέσουν με τις κατάλληλες υπηρεσίες.

Με καθολικά καθορισμένες διεπαφές και καλά σχεδιασμένες λειτουργίες, γίνεται επίσης εύκολο να επαναχρησιμοποιηθούν αυτές οι λειτουργίες άρα και οι εφαρμογές που αντιπροσωπεύουν. Η επαναχρησιμοποίηση μιας εφαρμογής λογισμικού σημαίνει καλύτερη απόδοσης της επένδυσης (return on investment) διότι μπορεί να παράγει περισσότερα με τους ίδιους πόρους. Επιτρέπει στους ανθρώπους να εξετάσουν το ενδεχόμενο χρησιμοποίησης μιας ήδη υπάρχουσας εφαρμογής με ένα διαφορετικό τρόπο ή το ενδεχόμενο προσφοράς αυτής σε ένα συνεργάτη με διαφορετικό τρόπο, αυξάνοντας ενδεχομένως τις επιχειρησιακές συναλλαγές μεταξύ των συνεργατών.

Επομένως, τα κύρια ζητήματα που τα web services προσπαθούν να λύσουν είναι τα ζητήματα ολοκλήρωσης (integration) δεδομένων και εφαρμογών και αυτά της μεταμόρφωσης των τεχνικών λειτουργιών σε υπολογιστικές λειτουργίες επιχειρησιακού προσανατολισμού.

2.6.2 Τα web services από την τεχνική σκοπιά

Ενώ τα web services προσφέρουν όλα αυτά τα δυναμικά χαρακτηριστικά ώστε να συνδυάσουμε υπηρεσίες σε εφαρμογές, πρέπει πρώτα να χτίσουμε αυτές τις υπηρεσίες. Οι γλώσσες προγραμματισμού στην πληροφορική συνεχώς εξελίσσονται. Ξεκινήσαμε δεκαετίες πριν με την ιδέα της function στην οποία παρέχουμε μερικές παραμέτρους, εκτελεί μια λειτουργία με αυτές τις παραμέτρους, και επιστρέφει μια τιμή βασισμένη στους υπολογισμούς που έγιναν. Τελικά, αυτή η πρώτη έννοια εξελίχθηκε σε αντικείμενο (object) όπου κάθε αντικείμενο είχε όχι απλώς έναν αριθμό από λειτουργίες (functions) που μπορεί να εκτελέσει αλλά και τις δικές του ιδιωτικές μεταβλητές (private data variables), αντί να στηρίζεται σε εξωτερικές μεταβλητές του συστήματος που προηγουμένως έκανα πιο περίπλοκη την ανάπτυξη εφαρμογών. Δεδομένου ότι οι εφαρμογές άρχισαν να επικοινωνούν μεταξύ τους, η έννοια του καθορισμού καθολικών διεπαφών (universal interfaces) για αντικείμενα έγινε σημαντική, επιτρέποντας αντικείμενα σε διαφορετικές πλατφόρμες να επικοινωνούν ακόμη και αν είχαν αναπτυχθεί σε διαφορετικές γλώσσες προγραμματισμού και εκτελούνταν σε διαφορετικά λειτουργικά συστήματα 1.

Στο πιο πρόσφατο βήμα, τα web services προχώρησαν μπροστά με την έννοια των διεπαφών και επικοινωνιών καθορισμένων με XML, ενώνοντας τελικά κάθε είδους εφαρμογή με οποιαδήποτε άλλη, όπως και παρέχοντας την ελευθερία στις εφαρμογές αν αλλάξουν και να εξελιχθούν με το χρόνο, αρκεί να είναι σχεδιασμένες σύμφωνα με την κατάλληλη διεπαφή. Η μεταβλητότητα της XML είναι αυτό που κάνει τα web services διαφορετικά από τεχνολογίες προηγούμενων γενεών. Επιτρέπει το διαχωρισμό της γραμματικής δομής (syntax) και της γραμματικής έννοιας (semantics), και του πώς αυτά υποβάλλονται σε επεξεργασία και κατανοούνται από μία υπηρεσία και το περιβάλλον μέσα στο οποίο υπάρχει. Έτσι λοιπόν τώρα, τα αντικείμενα μπορούν να καθοριστούν σαν υπηρεσίες, οι οποίες επικοινωνούν με άλλες υπηρεσίες σε γραμματική καθορισμένη σε XML, με την οποία κάθε υπηρεσία

μεταφράζει και αναλύει το μήνυμα σύμφωνα με μην τοπική της υλοποίησης και το περιβάλλον της. Κατά συνέπεια μια δικτυακή εφαρμογή μπορεί πραγματικά να συντεθεί από πολλαπλές οντότητες διαφόρων υλοποιήσεων και σχεδιασμών εφόσον προσαρμόζονται στους κανόνες που καθορίζονται από την προσανατολισμένη στις υπηρεσίες αρχιτεκτονική τους.

Κατά συνέπεια, με αυτά στο μυαλό, τα web services μας επιτρέπουν:

- Την αλληλεπίδραση μεταξύ υπηρεσιών σε οποιαδήποτε πλατφόρμα, γραμμένες σε οποιαδήποτε γλωσσά προγραμματισμού.
- Να αντιληφθούμε λειτουργίες εφαρμογών ως εργασίες, οδηγούμενοι σε ανάπτυξη και ροές εργασιών προσανατολισμένες σε εργασίες. Αυτό επιτρέπει μια υψηλότερη αφαίρεση του λογισμικού το οποίο μπορεί να υιοθετηθεί από λιγότερο τεχνικά καταρτισμένους χρήστες.
- Τη χαλαρή συνδεσιμότητα μεταξύ εφαρμογών, πράγμα που σημαίνει ότι αλληλεπιδράσεις μεταξύ υπηρεσιών δε θα χαλάνε κάθε φορά που υπάρχει κάποια αλλαγή το πώς μία ή περισσότερες υπηρεσίες σχεδιάζονται ή υλοποιούνται.
- Την προσαρμογή ήδη υπαρχουσών εφαρμογών στις μεταβαλλόμενες επιχειρησιακές συνθήκες και ανάγκες των πελατών.
- Να παρέχουμε υπάρχουσες εφαρμογές λογισμικού με διαπαφές υπηρεσιών χωρίς να αλλάξουμε τις αρχικές εφαρμογές, επιτρέποντάς τους να λειτουργούν πλήρως στο περιβάλλον των υπηρεσιών.
- Να εισάγουμε άλλες διοικητικές λειτουργίες ή λειτουργίες διαχείρισης διαδικασιών όπως η αξιοπιστία, υπευθυνότητα, ασφάλεια, κ.λπ., ανεξάρτητες της αρχικής λειτουργίας μιας εφαρμογής, αυξάνοντας κατά συνέπεια τη μεταβλητότητα και τη χρησιμότητά της στο επιχειρησιακό περιβάλλον.

2.6.3 Εφαρμογές των web services

Τα πρώτα web services σκόπευαν να είναι πηγές πληροφορίας τις οποίες μπορεί κανείς πολύ εύκολα να ενσωματώσει στις εφαρμογές του : τιμές μετοχών, προβλέψεις καιρού, αποτελέσματα αθλητικών παιχνιδιών κλπ. Είναι εύκολο να φανταστεί κανείς μια ολόκληρη κατηγορία εφαρμογών που μπορεί να κατασκευάσει ώστε να αναλύει και να συνδυάζει πληροφορία που τον ενδιαφέρει και να την παρουσιάζει με ποικίλους τρόπους. Για παράδειγμα, θα μπορούσαμε να έχουμε ένα λογιστικό φύλλο (spreadsheet) το οποίο συνοψίζει όλη την οικονομική μας εικόνα : μετοχές, τραπεζικούς λογαριασμούς, δάνεια κλπ. Αν αυτή η πληροφορία ήταν διαθέσιμη μέσω web services το λογιστικό φύλλο θα μπορούσε να ενημερώνεται συνεχώς. Οι περισσότερες από αυτές τις πληροφορίες είναι ήδη διαθέσιμες στον παγκόσμιο ιστό αλλά τα web services θα κάνουν την προγραμματιστική πρόσβαση σε αυτές πιο εύκολη και πιο αξιόπιστη.

Εκθέτοντας ήδη υπάρχουσες εφαρμογές σαν web services θα επιτρέψει στους χρήστες να κατασκευάσουν νέες πιο ισχυρές εφαρμογές οι οποίες χρησιμοποιούν τα web services σαν δομικά στοιχεία. Για παράδειγμα, ένας χρήστης θα μπορούσε να

αναπτύξει μια εφαρμογή προμηθειών η οποία να παίρνει αυτόματα τιμές από προμηθευτές, να επιτρέπει στο χρήστη να επιλέξει προμηθευτή, να υποβάλει την παραγγελία και να παρακολουθεί την αποστολή έως ότου να γίνει η παραλαβή της. Η εφαρμογή του προμηθευτή, εκτός από το να εκθέτει τις υπηρεσίες της στον ιστό, θα μπορούσε να χρησιμοποιήσει άλλα web services για να ελέγξει την πιστοληπτική ικανότητα του πελάτη, να χρεώσει τον τραπεζικό λογαριασμό του πελάτη και να καθορίσει την αποστολή με μια εταιρία μεταφορών.

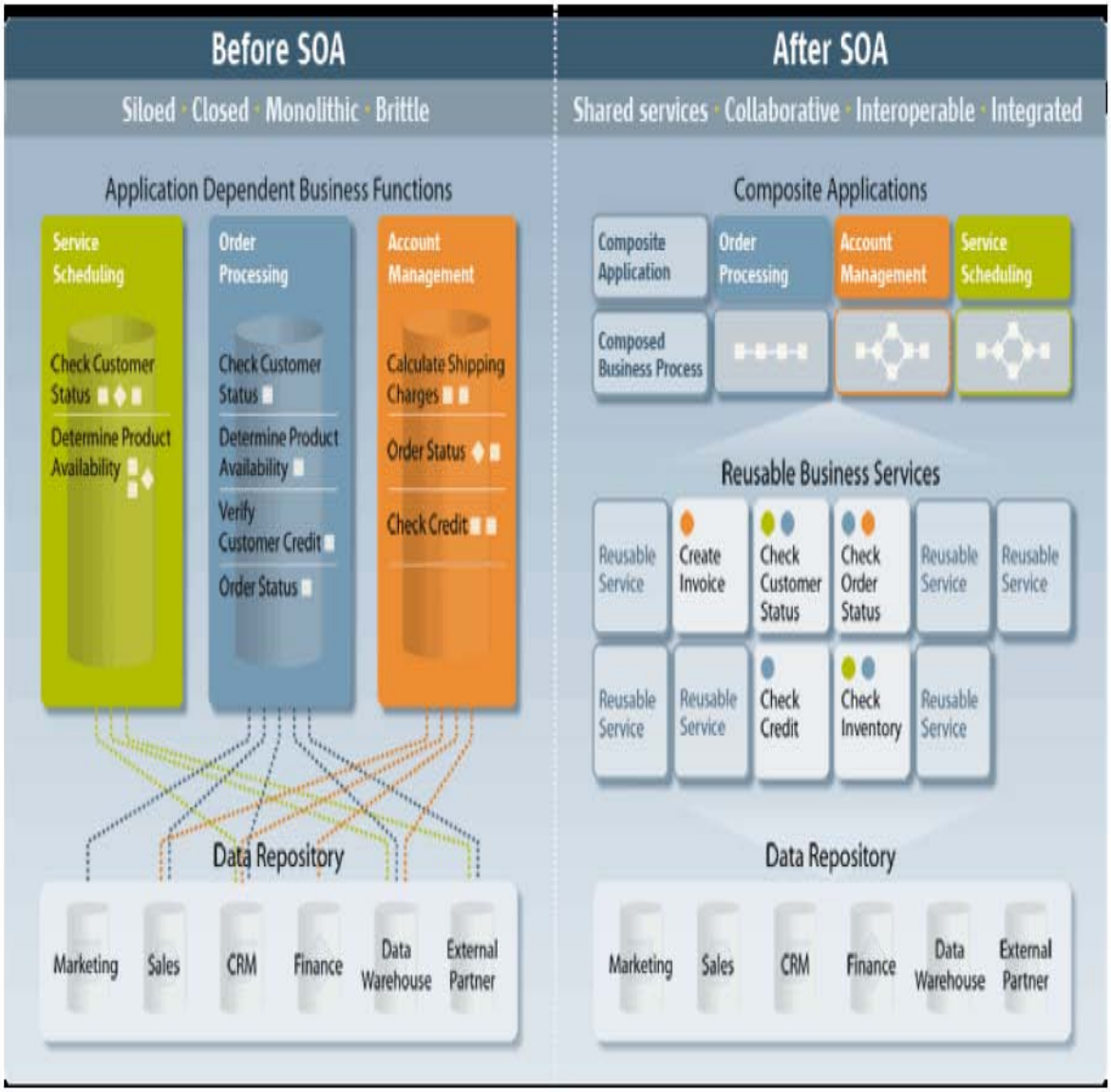
Στο άμεσο μέλλον μερικά από τα πιο ενδιαφέροντα web services θα υποστηρίζουν εφαρμογές που χρησιμοποιούν τον ιστό για να κάνουν πράγματα που δεν μπορούν να γίνουν σήμερα. Για παράδειγμα, μία από τις υπηρεσίες που τα web services θα κάνουν δυνατή είναι η υπηρεσία ημερολογίου. Αν ο οδοντίατρος ή ο μηχανικός σας εξέθεταν τα ημερολόγιά τους μέσω μιας τέτοιας web service, θα μπορούσατε να προγραμματίσετε τα ραντεβού σας με αυτούς ή θα μπορούσαν να προγραμματίσουν αυτοί τα ραντεβού κατευθείαν στο δικό σας ημερολόγιο άν θέλατε. Με λίγη φαντασία, μπορούμε να οραματιστούμε εκατοντάδες εφαρμογές οι οποίες μπορούν να κατασκευαστούν μόλις έχουμε τη δυνατότητα να προγραμματίσουμε τον ιστό.

2.7 SERVICE ORIENTED ARCHITECTURE

Τα web services είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Ένα web service είναι μια διεπαφή λογισμικού (software interface) που περιγράφει μια συλλογή από λειτουργίες οι οποίες μπορούν να προσεγγιστούν από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρότυπα βασισμένα στη γλώσσα XML για να περιγράψει μία λειτουργία (operation) προς εκτέλεση και τα δεδομένα προς ανταλλαγή με κάποια άλλη εφαρμογή. Μια ομάδα από web services οι οποίες αλληλεπιδρούν μεταξύ τους καθορίζει μια εφαρμογή web services.

Ο σχεδιασμός και η ανάπτυξη ολοκληρωμένων συστημάτων εφαρμογών καθώς επίσης και οι αυξανόμενες προσδοκίες των εκάστοτε χρηστών έχουν αλλάξει δραματικά τα τελευταία χρόνια, κυρίως όσον αφορά την τεχνολογική εξέλιξη στην ανάπτυξη προσβάσιμων επιχειρησιακών συστημάτων. Έτσι η επιτυχία που σημείωσε η εμφάνιση του διαδικτύου, οι διαδικτυακές υπηρεσίες και γενικότερα η εμφάνιση τεχνολογιών σημασιολογικού ιστού, ήταν αρκετή για να πείσει σχεδόν τους περισσότερους ότι η χρήση των τεχνολογιών αυτών είναι καθοριστικής σημασίας για τον σχεδιασμό, την ανάπτυξη αλλά και την διασύνδεση συστημάτων εφαρμογών. Κατά συνέπεια, αυτό είχε σαν αποτέλεσμα να κάνει την εμφάνισή της δειλά– δειλά η υπηρεσιοστραφής αρχιτεκτονική (service oriented architecture) η οποία συνθέτει όλες τις παραπάνω τεχνολογίες και καθορίζει ένα σύνολο κατάλληλων συνιστωσών με στόχο αφενός την διασύνδεση ετερογενών συστημάτων και αφετέρου το σχεδιασμό και την ανάπτυξη μιας νέας μορφής προσβάσιμων επιχειρησιακών συστημάτων.

Σχήμα 6 βασικές έννοιες για τη SOA [18]



Σχήμα 7 πριν και μετά τη SOA [18]

Κεφάλαιο 3: Ερεύνα και Συλλογή Δεδομένων

3.1 Ορισμός Action Research (Έρευνα δράσης)

Συνήθως περιγράφεται ως δοκιμή των ιδεών μας στην πράξη, ως ένα μέσο για την αύξηση της γνώσης ή της βελτίωσης του προγράμματος των σχολείων, της μάθησης και της διδασκαλίας (J. McNiff:1995, J. Elliott:1991).

Η έρευνα δράσης (action research) είναι όρος που καθιέρωσε ο πατέρας της έρευνας δράσης Kurt Lewin (1946). Αποτελεί ένα εναλλακτικό, δημοκρατικό και συμμετοχικό τύπο έρευνας, που βασίζεται στη δράση και που στοχεύει στην εύρεση απτών λύσεων για πραγματικά προβλήματα και όχι απλά στην παραγωγή της θεωρητικής γνώσης.

Συμβάλλει συγχρόνως στην κοινωνική θεωρία και πράξη. Το φιλοσοφικό της υπόβαθρο στηρίζεται στις ερμηνευτικές θεωρίες στις οποίες αντίθετα από το θετικισμό, ο ερευνητής είναι μέρος του κόσμου κι εστιάζεται στο προσωπικό νόημα των γεγονότων, τη μελέτη της αλληλεπίδρασης ανάμεσα στους ανθρώπους και το περιβάλλον, τις τάσεις, τις αντιλήψεις. Δεν ψάχνει για νόμους, αφού υπάρχουν πολλές πιθανές ερμηνείες. Ψάχνει να καταλάβει τον κόσμο από την πλευρά των συμμετεχόντων (ποιοτική έρευνα).

3.2 Τα κύρια χαρακτηριστικά της έρευνας δράσης:

1. Η έρευνα δράσης που πραγματοποιείται από μεμονωμένους ερευνητές εντάσσεται συνήθως σε μια επικοινωνιακή δομή. Η επικοινωνία μεταξύ συναδέλφων δίνει την ευκαιρία να συζητηθούν προβλήματα σχετικά με το περιεχόμενο, τις μεθόδους έρευνας και να ζητηθεί έμπρακτη υποστήριξη (π.χ. να μαγνητοφωνήσουν μια συνέντευξη). Μπορούν να συμμετέχουν και πρόσωπα όπως ερευνητές, καθηγητές, σχετικοί με την έρευνα κ.ά.

2. Οι ερευνητές πραγματοποιούν εργασίες έρευνας και ανάπτυξης αναφορικά με το ερευνητικό ερώτημα, συνδυάζοντας στοιχεία αναστοχασμού και πρακτικής .

3. Η έρευνα δράσης χαρακτηρίζεται από σπειροειδείς κύκλους αναγνώρισης του προβλήματος, συστηματικής αλλαγής στοιχείων, στοχασμού, ανάλυσης δράσης καθοδηγούμενης από τα στοιχεία και τέλος από επαναπροσδιορισμό του προβλήματος κάτω από το φως των νέων ευρημάτων.

4. Η έρευνα δράσης μπορεί να βασίζεται σε ερωτήματα που έχουν διατυπωθεί από επαγγελματίες, οι οποίοι τα θεωρούν ουσιώδη για την εκπαιδευτική

πράξη.Οι ερευνητές όμως είναι οι άμεσα ενδιαφερόμενοι για την κατάσταση που μελετιέται.

5. Οι ερευνητές δράσης τονίζουν ότι είναι απαραίτητο να εξετάζουν διαφορετικές θεωρήσεις της ίδιας ερευνόμενης κατάστασης. Για τη συλλογή δεδομένων χρησιμοποιούνται διάφοροι μέθοδοι, όπως η συνέντευξη, η παρατήρηση, το ερωτηματολόγιο, η ανάλυση κειμένων κ.ά. [St. Kemmis, R. Mc Taggart (eds):1988].

3.3 Τα στάδια της έρευνας δράσης

Ο Lawrence Stenhouse πρωτεργάτης στο αγγλικό κίνημα της έρευνας δράσης στα τέλη της δεκαετίας το υ 1 9 6 0 πίστευε ότι η βελτίωση της εκπ/κής πραγματικότητας περνούσε κυρίως μέσα από τη μελέτη της σχολικής πραγματικότητας.

Τα δε κριτήρια επιστημονικής εγκυρότητας, αξιοπιστίας και αποτελεσματικότητας διασφαλίζονται:

1. Με το ομαδικό πνεύμα (ομάδα που υπηρετεί στο ίδιο σχολείο).

2. Με τη συνεργασία μεταξύ εργασιακής και ακαδημαϊκής κοινότητας.

3. Με την κατασκευή γερού θεωρητικού πλαισίου (επιστημολογική στάση).

4. Με τη διαλεκτική σχέση μεταξύ θεωρίας και πράξης.

Για να αρχίσει την έρευνα δράσης ο ερευνητής πρέπει να εντοπίσει μια αφετηρία στα πλαίσια της πρακτικής του και να έχει τη θέληση να προχωρήσει στη διερεύνησή της (στάδιο Α). Στη συνέχεια, μέσα από συζητήσεις, συνεντεύξεις και άλλες μεθόδους συλλογής στοιχείων, αλλά και μέσα από την ανάλυση των πληροφοριών η κατάσταση περιγράφεται με μεγαλύτερη σαφήνεια (στάδιο Β). Η αποσαφήνιση αυτή οδηγεί στην ανάπτυξη στρατηγικών δράσης και στην εφαρμογή τους (στάδιο Γ).

Κατά κανόνα οι νέες στρατηγικές δράσης δεν αναμένεται να λύσουν αμέσως ένα πρόβλημα. Γι' αυτό το λόγο, ο ερευνητής παρακολουθεί τις νέες καταστάσεις που διαμορφώνονται και βελτιώνει τις στρατηγικές δράσης όπου χρειάζεται (κυκλική διαδικασία μεταξύ των σταδίων Β και Γ).

Ο ερευνητής δημοσιοποιεί τα αποτελέσματα της έρευνας συντάσσοντας γραπτές μελέτες περίπτωσης. Οι διαπιστώσεις τους γίνονται αντικείμενο συζήτησης και κριτικής (στάδιο Δ).

Ερευνητικές πρακτικές στην έρευνα δράσης

1. Η αποσαφήνιση μιας κατάστασης μπορεί να γίνει με:

α) την τήρηση ημερολογίου όπου ο ερευνητής γράφει τακτικά, ανάλογα με το είδος του ερευνητικού ερωτήματος που εξετάζεται (π.χ. καταγράφεται η ημερομηνία του συμβάντος, πληροφορίες για τις συνθήκες κάτω από τις οποίες έγινε το συμβάν, χρόνος, τόπος) πληροφορίες που μπορούν να βοηθήσουν στην κατανόηση μιας κατάστασης όπως παρατηρήσεις, συναισθήματα, αντιδράσεις, ερμηνείες, σκέψεις, ιδέες και εξηγήσεις.

β) με τη μαγνητοφώνηση μαθημάτων (στα κείμενα απομαγνητοφώνησης αφήνουμε ένα περιθώριο για σχόλια).
Οι ερευνητές μπορούν να ελέγξο ιν την αφετηρία της έρευνάς το ις με βάση τα ακόλουθα ερωτήματα:

- Τι είναι αυτό που θα θέλατε να βελτιώσετε; (εξαρτάται από τον ίδιο ή από ποιους άλλους;)
- Τι θα θέλατε να δοκιμάσετε;
- Τι θα θέλατε να αλλάξετε;

Είναι εύκολο να οδηγηθούμε σε εσφαλμένα συμπεράσματα από μια πρώτη εντύπωση. Γι' αυτό χρειάζεται περισσότερος χρόνος στην αποσαφήνιση της αφετηρίας. Η φάση αυτή μπορεί να ενισχυθεί με συνεντεύξεις, ερωτηματολόγια, φύλλα εργασίας, κ.ά.

2. Η ανάλυση των δεδομένων
Ο ερευνητής αναζητεί μια ερμηνεία της κατάστασης που να φαίνεται σωστή και να αποτελεί μια γερή βάση για περαιτέρω δράση.

3. Η ανάπτυξη και εφαρμογή της στρατηγικής δράσης:

Η αξιοπιστία των αποτελεσμάτων της έρευνας δε διασφαλίζεται μέσα από μια επιδέξια ανάλυση βασισμένη σε μια συγκεκριμένη θεωρία, αλλά μέσα από μια διαδικασία συσχετισμού έρευνας και δράσης.

Οι στρατηγικές δράσεις είναι ενέργειες που σχεδιάζει και εφαρμόζει ο ερευνητής για να βελτιώσει μια κατάσταση ή το πλαίσιό της.

3.4 Εμπειρικά Δεδομένα

Μελέτη περίπτωσης (Ολοκληρωμένου Πληροφοριακού Συστήματος Υγείας του Α΄ Περιφερειακού Συστήματος Υγείας & Πρόνοιας Αττικής» (Α΄ ΠΕΡΙΦΕΡΕΙΑΚΟ ΣΥΣΤΗΜΑ ΥΓΕΙΑΣ ΚΑΙ ΠΡΟΝΟΙΑΣ ΑΤΤΙΚΗΣ, Σύμβαση 3/11/2006)

ΑΝΑΘΕΤΟΥΣΑ ΑΡΧΗ	Α΄ Δ.Υ.Πε Αττικής
ΦΟΡΕΑΣ ΛΕΙΤΟΥΡΓΙΑΣ (ΓΙΑ ΤΟΝ ΟΠΟΙΟ ΠΡΟΟΡΙΖΕΤΑΙ ΤΟ ΕΡΓΟ)	Μονάδες υγείας της Α' Δ.Υ.Πε Αττικής όπως προσδιορίζονται στη διακήρυξη του διαγωνισμού
ΠΡΟΫΠΟΛΟΓΙΣΜΟΣ	4.180.000 € , (συμπεριλαμβανομένου ΦΠΑ)
ΧΡΟΝΟΣ ΥΛΟΠΟΙΗΣΗΣ – ΔΙΑΡΚΕΙΑ ΕΡΓΟΥ	24 μήνες από την υπογραφή της σύμβασης (υπάρχει παράταση)
ΑΝΑΔΟΧΟΣ ΕΡΓΟΥ	Ένωση εταιρειών SIEMENS – ITE – Computer Solutions
Διάρκεια Σύμβασης SLA (εντός προϋπολογισμού)	6 μήνες
Αριθμός Μονάδων Υγείας (Νοσοκομεία + Κεντρική Υπηρεσία)	10 Νοσοκομεία + Κεντρική Υπηρεσία ΔΥΠΕ

3.4.1 Στόχοι ΟΠΣ

Στόχος του παρόντος Έργου είναι η δημιουργία ενός Ολοκληρωμένου Πληροφορικού Συστήματος Υγείας της (τέως) Α' ΔΥΠε Αττικής (ΟΠΣ) που θα βελτιώσει το βαθμό αυτοματοποίησης των διαδικασιών, θα αξιοποιήσει πλήρως την επιχειρησιακή πληροφορία και θα βελτιώσει την ποιότητα των παρεχόμενων προς τον πολίτη υπηρεσιών υγείας.

Η εισαγωγή του ΟΠΣ εκτιμάται ότι θα αποτελέσει έργο οδηγό για την αναδιοργάνωση της λειτουργίας των Νοσοκομείων. Ειδικότερα, το ΟΠΣ της (τέως) Α' ΔΥΠε Αττικής θα λειτουργήσει σαν το ***κύριο εργαλείο*** πάνω στο οποίο θα στηριχθούν οι προσπάθειες για αναδιάρθρωση των οργανωτικών δομών και λειτουργιών των Νοσοκομείων.

Με την ανάπτυξη του ΟΠΣ στοχεύεται η αποτελεσματικότερη διαχείριση της οικονομικής και διοικητικής πληροφορίας των Μονάδων Υγείας και της Κεντρικής Υπηρεσίας της (τέως) Α' ΔΥΠε Αττικής. Αυτός ο στόχος επιτυγχάνεται με την υλοποίηση του Πληροφορικού Συστήματος Διοικητικό-Οικονομικής Διαχείρισης, το οποίο περιλαμβάνει 3 Υποσυστήματα (Διοικητικό-Οικονομικό Υποσύστημα, Υποσύστημα Διαχείρισης Ασθενών και Υποσύστημα Επιχειρηματικής Ευφυΐας (BI)) και την υλοποίηση του Υποσυστήματος Βιοϊατρικής Τεχνολογίας στην Κεντρική Υπηρεσία και στα παρακάτω 10 Νοσοκομεία:

- Γ.Ν. Αθηνών «ΑΛΕΞΑΝΔΡΑ»
- Γ.Ν. Μαιευτήριο Αθηνών «ΕΛΕΝΑ ΒΕΝΙΖΕΛΟΥ»
- Γ.Ν. Αθηνών ΠΟΛΥΚΛΙΝΙΚΗ
- Γ.Ν. ΠΑΤΗΣΙΩΝ
- ΟΦΘΑΛΜΙΑΤΡΕΙΟ ΑΘΗΝΩΝ
- Νοσοκομείο Αφροδίσιων & Δερματικών Νόσων Αθηνών «ΑΝΔΡΕΑΣ ΣΥΓΓΡΟΣ»
- Αντικαρκινικό – Ογκολογικό Ν.Α. «Ο Άγιος Σάββας»
- Γ.Ν.Ν. της Θειας Πρόνοιας «Η ΠΑΜΜΑΚΑΡΙΣΤΟΣ»
- ΝΟΜ. ΠΑΘΟΛ. ΝΟΣΟΚ. Αθηνών Σπηλιοπούλειο «Η ΑΓ. ΕΛΕΝΗ»
- Γ.Ν. Αθηνών «Η ΕΛΠΙΣ»

Η υλοποίηση του Έργου αναμένεται να έχει τα παρακάτω αναμενόμενα οφέλη για την (τέως) Α' ΔΥΠε Αττικής και τον Πολίτη, τα οποία και αποτελούν τους στόχους του Έργου:

- Ολοκλήρωση της εισαγωγής Τεχνολογιών Πληροφορικής και Επικοινωνιών στις Μονάδες Υγείας
- Βελτίωση της Πληροφόρησης και της Εξυπηρέτησης του Πολίτη
- Συστηματικός Έλεγχος και Εξορθολογισμός Δαπανών κυρίως μέσω της υλοποίησης Διπλογραφικού
- Αποδοτική Διαχείριση και Αξιοποίηση της Επιχειρησιακής Πληροφορίας
- Υποστήριξη στη λήψη στρατηγικών αποφάσεων τόσο των διοικήσεων των Νοσοκομείων όσο και της αρμόδιας Κεντρικής Υπηρεσίας.
- Μέτρηση αποτελεσματικότητας και απόδοσης παραγωγής Μονάδων Υγείας
- Παρακολούθηση δεικτών ποιότητας, αποτελέσματος, κόστους, ανάπτυξης

- Βέλτιστη διαχείριση υποδομών
- Εξορθολογισμός διαδικασιών στην Κεντρική Υπηρεσία και στις Μονάδες Υγείας
- Ανάπτυξη ανθρώπινου δυναμικού

3.4.2 Πρόλογος

Η παρούσα μελέτη περίπτωσης αφορά την έρευνα στο έργο της Α ΠΕΣΥΠ για το ποσό βοήθησε η ύπαρξη ενός Υποσύστηματός Διαχείρισης Ασθενών σε ένα από τα 10 νοσοκομεία και συγκεκριμένα στο ΠΕΡΙΦΕΡΕΙΑΚΟ Γ.Ν. Μαιευτήριο Αθηνών «ΕΛΕΝΑ ΒΕΝΙΖΕΛΟΥ» όπου υλοποιήθηκε το έργο . Σκοπός, επίσης είναι κατά πόσο με αυτό διευκολύνονται οι εργασίες όλων των υπαλλήλων του νοσοκομείου, ιατρών, νοσηλευτών κ.λ.π. Μέρος της μελέτης αποτελεί η εφαρμογή των διεθνών κωδικοποιήσεων για ασθένειες κ.λ.π. καθώς και άλλα ειδικότερα θέματα. Στην περίπτωση μη ύπαρξης ολοκληρωμένου πληροφοριακού συστήματος εξετάζεται πώς τα διάφορα τμήματα του νοσοκομείου λειτουργούν και πώς εκτελούνται οι εργασίες τους. Επιπλέον, μελετάται ανάλογα με την περίπτωση αν υπάρχουν προοπτικές για εξέλιξη ή εγκατάσταση ολοκληρωμένου πληροφοριακού συστήματος. Η παρούσα έρευνα στο νοσοκομείο έγινε ύστερα από συνέντευξη με τους υπεύθυνους κάθε τμήματος λόγω του ότι εργαζόμουνα σε μία από τις ανάδοχες εταιρίες του έργου. Η προσωπική επαφή με το προσωπικό του ιδρύματος διευκόλυνε στη συγκέντρωση και στην επαλήθευση πληροφοριών που είχαν συγκεντρωθεί από άλλες πηγές. Οι ιδέες και οι απόψεις των εργαζομένων υπήρξαν κατατοπιστικές για την διεξαγωγή της έρευνας. Η πλήρης προετοιμασία μας βοήθησε στο να αντιληφθούμε τις αντιδράσεις του προσωπικού καθώς και την υποστήριξη του απέναντι στο ισχύον σύστημα. Η βασική δυσκολία της τεχνικής της συνέντευξης που ακολουθήσαμε ήταν η ‘’πίεση’’ μας για ανάλωση χρόνου του προσωπικού του κάθε τμήματος, που καθυστέρησε τις εργασίες του προκειμένου να μας εξυπηρετήσει.

3.4.3 Παρουσίαση της έρευνας

Το νοσοκομείο ΕΛΕΝΑ ΒΕΝΙΖΕΛΟΥ επελέγη για την εκπόνηση της μελέτης μας για το λόγο ότι στο στήσιμο του συστήματος διαχείρισης ασθενών είχα αναλάβει την εκπαίδευση των χρηστών του συστήματος πράγμα το οποίο μας διευκόλυνε στην επαφή μας με τους υπεύθυνους του νοσοκομείου. Το σύστημα που χρησιμοποιείται ονομάζεται ICS και είναι του Εργαστήριου Βιοϊατρικής Πληροφορικής του Ιδρύματος τεχνολογίας και έρευνας, η έδρα της οποίας είναι στο Ηράκλειο Κρήτης. Η λύση που παρέχει το ICS προσαρμόζεται στις ανάγκες του νοσοκομείου του Πύργου, δεδομένου ότι στηρίζεται στην ανάλυση που πραγματοποίησαν οι ειδικοί του Εργαστήριο Βιοϊατρικής Πληροφορικής του Ιδρύματος τεχνολογίας και έρευνας σε συνεργασία με επιλεγμένα στελέχη του νοσοκομείου. Το σύστημα είναι εύχρηστο και μερικώς εναρμονισμένο στις ανάγκες και στις ιδιομορφίες του κάθε τμήματος. Η εκπαίδευση των χρηστών του συστήματος πραγματοποιήθηκε από ειδικούς της εταιρείας.

3.4.3.1 Τμήμα κίνησης ασθενών

Εισαγωγή ασθενή

Η διαδικασία εισαγωγής στην γενική της περίπτωση έχει ως εξής:

- Ο ασθενής παρουσιάζεται στο **Γραφείο Κίνησης** για να εισαχθεί σε κάποια κλινική. Η εντολή εισαγωγής έχει δοθεί από κάποιον γιατρό στα εξωτερικά ιατρεία ή ο ασθενής έρχεται ύστερα από κάποια προγραμματισμένη εισαγωγή από προηγούμενη επαφή του με το νοσοκομείο. Ο ασθενής έχει μαζί του την **Κάρτα Εισόδου Ασθενή** (ή οποιαδήποτε άλλο **παραπεμπτικό εισαγωγής,** πχ για εισαγωγή στην χειρουργική έχει μαζί του το **Δελτίο προγραμματισμού ασθενών προς χειρουργείο** που του δόθηκε από την χειρουργική κλινική κατά τον προγραμματισμό της εγχείρισης).

- Ο διοικητικός υπάλληλος στο Γραφείο Κίνησης δημιουργεί τον φάκελο του ασθενή, και καταγράφει τα δημογραφικά δεδομένα του ασθενή. Ταυτόχρονα ανοίγεται και ο «οικονομικός φάκελος» όπου θα αποθηκευτούν όλα τα δεδομένα για την χρέωση του ασθενή
- Επίσης ο υπάλληλος του ΓΚ γράφει το **Εισιτήριο**. Αυτό δίνεται στον ασθενή και ο ασθενής προσέρχεται στην κλινική για την εισαγωγή.
- Σε ορισμένες περιπτώσεις όπως **επείγουσας εισαγωγής** ο ασθενής πηγαίνει ή μεταφέρεται **απ΄ ευθείας στην κλινική** από τα επείγοντα χωρίς να περάσει πρώτα από το Γραφείο Κίνησης. Το εισιτήριο γράφεται στην κλινική και σε μελλοντική στιγμή αποστέλλεται στο Γραφείο Κίνησης. Εναλλακτικά το εισιτήριο μπορεί να γραφτεί στα επείγοντα και να σταλθεί στο ΓΚ σε μελλοντική στιγμή.
- Επίσης σε άλλες περιπτώσεις **ο ασθενής πηγαίνει από μόνος του απ΄ ευθείας στην κλινική** και το εισιτήριο γράφεται στην κλινική και στην συνέχεια αποστέλλεται στο Γραφείο Κίνησης

Κατά την άφιξη του ασθενή στην κλινική γίνονται όλες οι κάποιες από τις ακόλουθες ενέργειες :

- <u>Νοσηλευτικές ενέργειες</u>
  - Διαχείριση εισιτηρίου
  - Καταγραφή ασθενή στο **μητρώο ασθενών της κλινικής**
  - Άνοιγμα **φακέλου ασθενή**
  - Τοποθέτηση ασθενή σε κλίνη
  - Ενημέρωση **πλάνου ορόφου**
  - Εκτέλεση αρχικών οδηγιών γιατρού και λογοδοσία (χορήγηση φαρμάκων και άλλες αρχικές ενέργειες)
- <u>Ιατρικές ενέργειες</u>
  - Λήψη ιατρικού ιστορικού και καταγραφή στο **φύλλο νοσηλείας**

- Κλινική εξέταση ασθενή και καταγραφή στο **φύλλο νοσηλείας**
- Αρχικές οδηγίες διαχείρισης ασθενή προς το νοσηλευτικό προσωπικό (φάρμακα, σίτιση, διενέργεια εξετάσεων, μετρήσεις, άλλες ενέργειες).

Έντυπα

- Μητρώο Ασθενών κλινικής
- Κάρτα Εισόδου Ασθενή
- Εισιτήριο
- Πλάνο ορόφου
- Φύλλο νοσηλείας

Επικοινωνίες

Οι περισσότερες επικοινωνίες γίνονται με έντυπα που μεταφέρει ο ίδιος ο ασθενής. Οι επικοινωνίες αφορούν τα ακόλουθα:

- Ροή δεδομένων μεταξύ Γραφείου Κίνησης-Εξωτερικών Ιατρείων (μέσω ασθενή): **Κάρτα Εισόδου Ασθενή (ή άλλο παραπεμπτικό εισαγωγής).**

- Ροή δεδομένων μεταξύ Γραφείου Κίνησης-κλινικής: **Εισιτήριο** (σε πολλές περιπτώσεις το εισιτήριο μεταφέρεται από την κλινική στο Γραφείο Κίνησης)

Μεταφορά ασθενή

Η μεταφορά ασθενή περιλαμβάνει δύο περιπτώσεις

- την μεταφορά σε άλλο κρεβάτι της ίδιας κλινικής
- την μεταφορά σε άλλη κλινική

Η διαδικασία μεταφοράς σε άλλη κλινική έχει ως εξής:

- Ο υπεύθυνος γιατρός για τον ασθενή αποφασίζει την μεταφορά του σε άλλη κλινική και δίνει την αντίστοιχη εντολή
- Ο ειδικευόμενος (συνήθως) ή το νοσηλευτικό προσωπικό καταγράφουν τα δεδομένα της μεταφοράς
- Γίνεται αποστολή των δεδομένων μεταφοράς στο ΓΚ και στην κλινική όπου μεταφέρεται ο ασθενής
- Γίνεται ενημέρωση του πλάνου ορόφου της κλινικής
- Ο ασθενής μεταφέρεται συνήθως μαζί με τον φάκελο του ο οποίος σε μελλοντική στιγμή επιστρέφει πίσω στην κλινική

Έξοδος ασθενή

Η διαδικασία έχει ως εξής:

- Ο γιατρός που είναι υπεύθυνος για τον ασθενή αποφασίζει την έξοδο του ασθενή από την κλινική. Δίνει την εντολή σε κάποιον υφιστάμενο ο οποίος αναλαμβάνει να διεκπεραιώσει την διαδικασία. (η διαδικασία μπορεί να διεκπεραιωθεί και από τον ίδιο).
- Αυτός που τελικά αναλαμβάνει να διεκπεραιώσει την διαδικασία γράφει **το ενημερωτικό εξόδου** το οποίο δίνει στον ασθενή και επίσης το **εξιτήριο** το οποίο αποστέλλεται μέσω του ασθενή (ή συγγενή του) **στο ΓΚ και στο Λογιστήριο Ασθενών.**
- Στην συνέχεια, ο συγγενής του ασθενή (ή ο ασθενής) πηγαίνει στο Λογιστήριο Ασθενών για να πληρώσει ή να τακτοποιήσει τις οικονομικές εκκρεμότητες. Η κλινική έχει ήδη αποστείλει στο Λογιστήριο Ασθενών όλα τα δεδομένα που απαιτούνται για την χρέωση του ασθενή για φάρμακα και ενέργειες εκτός νοσηλίου. Μετά το Λογιστήριο Ασθενών ο ασθενής παρουσιάζεται στο ΓΚ (που μπορεί να είναι στον ίδιο χώρο). Το ΓΚ υπογράφει το εξιτήριο, το παραδίδει στον ασθενή με το βιβλιάριό του και ο ασθενής φεύγει.

Έντυπα

- Εξιτήριο
- Ενημερωτικό εξόδου

Επικοινωνίες

- Ροή δεδομένων μεταξύ κλινικής-Γραφείου Κίνησης-Λογιστηρίου Ασθενών : **εξιτήριο**
- Ροή δεδομένων μεταξύ κλινικής-ασθενή : **ενημερωτικό εξόδου**

3.4.3.2 Προβλήματα Νοσοκομείου

Κατά τη διάρκεια της έρευνάς μας εντοπίσαμε διάφορα προβλήματα όσον αφορά τη λειτουργία του πληροφοριακού συστήματος. Ορισμένα από τα προβλήματα που εντοπίσαμε είναι τα εξής:

1. Ο υψηλός χρόνος απόκρισης του συστήματος με αποτέλεσμα να καθυστερεί η εκτέλεση των εργασιών ορισμένες από τις οποίες αποτελούν σημαντικό παράγοντα για την διεξαγωγή άλλων απαραίτητων λειτουργιών.

2. Το σύστημα του Γραφείου Κίνησης παρουσιάζει διακοπές λειτουργίας με αποτέλεσμα να σταματούν οι εργασίες του δεδομένου ότι δεν υπάρχει υποδομή για να λειτουργήσει με μη ηλεκτρονικό τρόπο.

3. Οι προγραμματισμένες λειτουργίες δεν προσαρμόζονται έγκαιρα στις νέες απαιτήσεις. Για παράδειγμα, ο υπολογισμός αναμονών δημιουργεί προβλήματα τόσο στους ασθενείς όσο και στη διοίκηση του νοσοκομείου.

4. Η διαδικασία τήρησης του αρχείου ασθενών εμφανίζει ορισμένα προβλήματα πέρα από αυτά που σχετίζονται με την έλλειψη μηχανοργάνωσης και ηλεκτρονικού φακέλου. Πιο συγκεκριμένα η ταξινόμηση των καρτελών γίνεται με βάση τον Α.Μ. του ασθενή. Το ιδιαίτερα συχνό φαινόμενο της απώλειας της ατομικής κάρτας από τους ασθενείς άρα και του εμποδίζει την ανεύρεση του ασθενή σε επόμενες επισκέψεις, οδηγεί σε έκδοση νέου Α.Μ. και νέας καρτέλας, σε διόγκωση του αρχείου (υπάρχει ήδη πρόβλημα χώρου) και το πιο σημαντικό, σε ασυνέχεια και κατακερματισμό στο ιστορικό του ασθενή.

5. Πολλοί ιατροί ιδίως όταν πρόκειται για επιβαρημένα περιστατικά δεν επιστρέφουν τις καρτέλες των ασθενών στο τέλος του ιατρείου προκειμένου να τις επεξεργαστούν, με συνέπεια πολλές φορές να μην επιστρέφουν ποτέ στο αρχείο. Επιπλέον, κάποια ιατρεία που λόγω των περιστατικών διαχειρίζονται ογκώδεις φακέλους ασθενών, ζητούν από το αρχείο να τηρεί ξεχωριστά τις καρτέλες που αφορούν τα ιατρεία αυτά με αποτέλεσμα την διάσπαση της ταξινόμησης.

6. Στην πραγματικότητα ο αριθμός των επισκέψεων που υπολογίζεται από το πληροφοριακό σύστημα δεν είναι ο πραγματικός. Πραγματοποιούνται επισκέψεις χωρίς ραντεβού καθώς και ακυρώσεις που δεν υπολογίζονται. Σε καμιά περίπτωση όμως οι ακυρώσεις δεν αντισταθμίζονται με τις επισκέψεις χωρίς ραντεβού. Έτσι, ο πραγματικός αριθμός των επισκέψεων υπερβαίνει κατά κανόνα τον υπολογιζόμενο. Το γεγονός αυτό αποτελεί αρνητικό παράγοντα για την ποιότητα των παρεχόμενων υπηρεσιών.

3.4.3.3 ΣΥΜΠΕΡΑΣΜΑΤΑ

Το σύγχρονο νοσοκομείο σαν μονάδα παροχής υπηρεσιών υγείας για να λειτουργήσει σωστά χρειάζεται αξιόπιστες πληροφορίες κατάλληλα επεξεργάσιμες και κυρίως τεκμηριωμένες. Η διοίκηση λοιπόν ενός νοσηλευτικού ιδρύματος χρειάζεται ένα αξιόπιστο, ορθολογικό και δυναμικά εξελισσόμενο σύστημα διακίνησης της πληροφορίας ανάμεσα στα τμήματα του. Το νοσοκομείο Έλενα Βενιζέλου έχοντας προχωρήσει σε ένα ενιαίο ολοκληρωμένο πληροφοριακό σύστημα διοίκησης έχει τα εξής οφέλη:

- Καλύτερη οργάνωση διοικητικό – οικονομικών υπηρεσιών

- Απλούστευση των διαδικασιών και ταχύτερη διεκπεραίωση των εργασιών ποιοτική εξυπηρέτηση των νοσηλευομένων
- Ταχύτερη και αποτελεσματική διαχείριση των εσόδων του νοσοκομείου
- Έλεγχος των δαπανών και του κόστους λειτουργίας
- Συλλογή στοιχείων – στατιστικών – πληροφοριών

Η σημερινή κατάσταση στο νοσοκομείο Έλενα Βενιζέλου με την υλοποίηση του συγκεκριμένου έργου έχει επιφέρει σημαντικές αλλαγές στην οργάνωση του νοσοκομείου όμως οι παραπάνω ελλείψεις που περιγράφονται παραπάνω έπρεπε να ληφθούν υπόψη από το υπουργείο ώστε να επιφέρουν στο μέγιστο τα παραπάνω οφέλη.

Κεφάλαιο 4: Ανάλυση και Σχεδίαση (MED.I.S)

4.1 ΑΝΑΛΥΣΗ

4.1.1 Σενάρια χρήσης

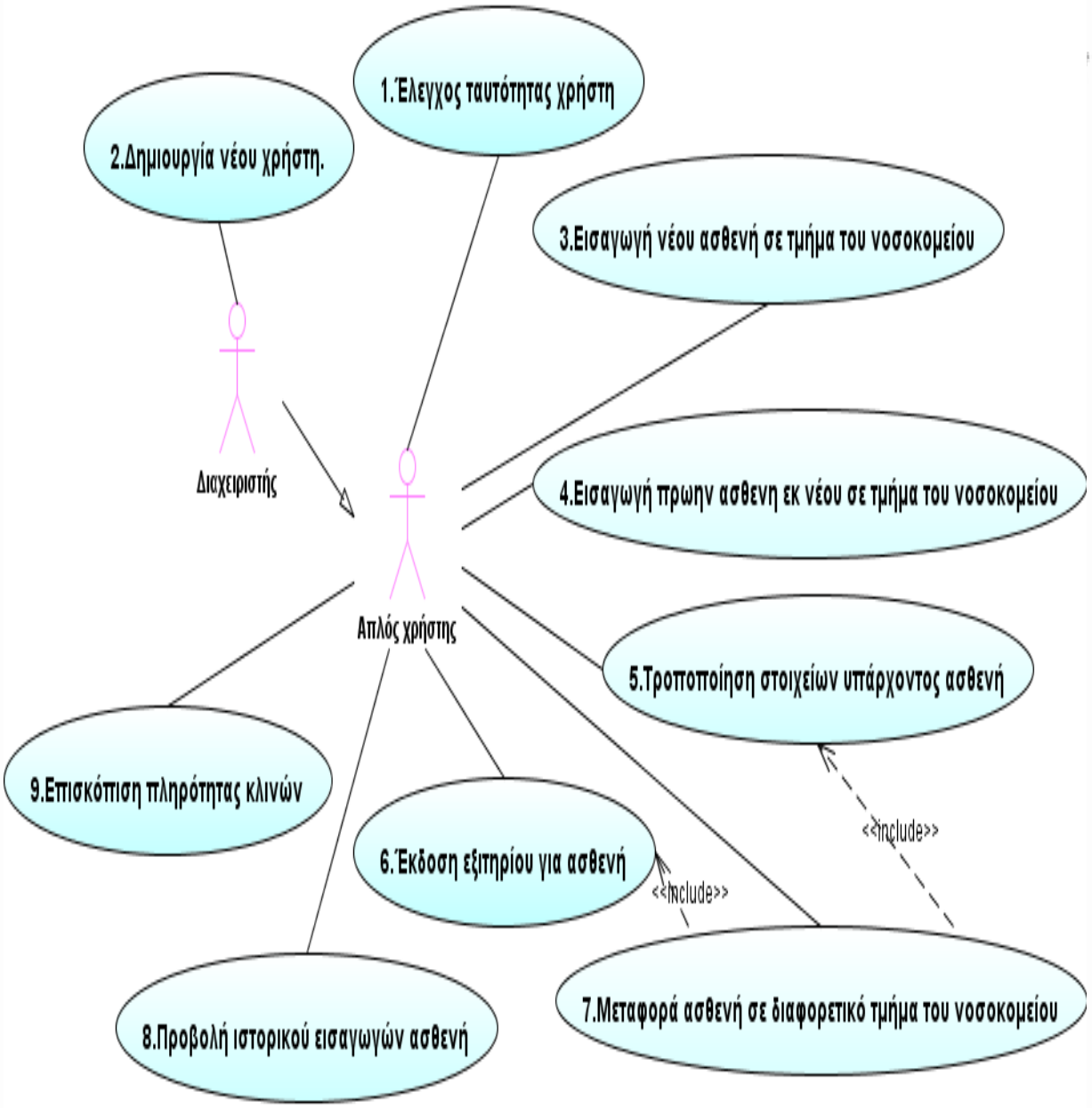
Το αρχικό πρόβλημα είναι αρκετά περίπλοκο ώστε να το αντιμετωπίσουμε συνολικά. Για το λόγο αυτό θα το διαιρέσουμε σε υπό-προβλήματα τα οποία ονομάζονται σενάρια χρήσης και θα μας βοηθήσουν ώστε να λύσουμε ευκολότερα και αποδοτικότερα το αρχικό πρόβλημα. Ακολουθώντας τη κλασσική λογική του «διαίρει και βασίλευε» θα έχουμε τα εξής σενάρια χρήσης τα οποία θα πρέπει αρχικά να καλύπτει το σύστημα μας.

Τα σενάρια χρήσης είναι:

1. Έλεγχος ταυτότητας χρήστη
2. Δημιουργία νέου χρήστη.
3. Εισαγωγή νέου ασθενή σε τμήμα του νοσοκομείου
4. Εισαγωγή πρώην ασθενή εκ νέου σε τμήμα του νοσοκομείου.
5. Τροποποίηση στοιχείων υπάρχοντος ασθενή.
6. Έκδοση εξιτηρίου για ασθενή.
7. Μεταφορά ασθενή σε διαφορετικό τμήμα του νοσοκομείου.
8. Προβολή ιστορικού εισαγωγών ασθενή.

4.1.2 Διάγραμμα περιπτώσεων χρήσης (Use Case Diagram)

Σύμφωνα με την παραπάνω παράγραφο προκύπτει το ακόλουθο διάγραμμα περιπτώσεων χρήσης, το οποίο υλοποιήθηκε με το πρόγραμμα :



Εικόνα 1 Διάγραμμα περιπτώσεων χρήσης (USE CASE DIAGRAM)

4.1.3 Ζητήματα

Αναλύοντας το πρόβλημα σε υπό-προβλήματα ανακύπτουν αρκετά ζητήματα ως προς την υλοποίηση της εφαρμογής που θα δώσει λύση στο πρόβλημα. Στη παράγραφο αυτή επισημαίνονται τα ζητήματα αυτά και προτείνονται τρόποι επίλυσής τους.

Τα κυριότερα προβλήματα είναι αυτά της ταχύτητας και της απλότητας. Η εφαρμογή πρέπει να μην απασχολεί το χρήστη για θέματα που μπορούν να υπολογιστούν, όπως τον εντοπισμό διαθέσιμης κλίνης. Για τον λόγο αυτό ο χρήστης θα πρέπει κατά την εισαγωγή ασθενή να καταχωρεί μόνο τα στοιχεία του και το τμήμα στο όποιο θα εισαχθεί, ενώ το σύστημα θα αναλαμβάνει να εντοπίσει την διαθέσιμη κλίνη και να τον εισαγάγει σε αυτήν. Πέραν όμως από την απλοποίηση κατά την εισαγωγή πρέπει να απλοποιηθεί και η διαδικασία της μεταφοράς του ασθενή σε διαφορετικό τμήμα. Ο χρήστης κανονικά θα έπρεπε να εκδώσει εξιτήριο και εν συνεχεία να εισαγάγει τον χρήστη στο νέο τμήμα. Το σύστημα προκειμένου να το διευκολύνει του δίνει την δυνατότητα στην αλλαγή στοιχείων να αλλάξει μέχρι και το τμήμα στο οποίο έχει γίνει η εισαγωγή και αυτόματα να εκδοθεί εξιτήριο και να γίνει εισαγωγή στο νέο τμήμα χωρίς να χρειαστεί κάτι επιπλέον από τον χρήστη. Άρα υπάρχουν δύο ζητήματα που χρήζουν αντιμετώπισης:

1. Αυτόματος εντοπισμός διαθέσιμης κλίνης.

2. Αυτόματη έκδοση εξιτηρίου και εισαγωγή στο νέο τμήμα όταν γίνεται αλλαγή τμήματος.

4.2 ΣΧΕΔΙΑΣΗ

Στη φάση του σχεδιασμού το λογισμικό διαιρείται σε ενότητες ,οι οποίες σχεδιάζονται και αργότερα κατασκευάζονται ξεχωριστά. Οι δομικές αυτές μονάδες θα διαιρεθούν με τη σειρά τους σε άλλες απλούστερες, έτσι ώστε το πολύπλοκο πρόβλημα της κατασκευής πληροφοριακού συστήματος διαχείρισης νοσοκομειακών κλινών να μετατραπεί σε απλούστερα.

4.2.1 Διάγραμμα οντοτήτων συσχετίσεων

Επιγραμματικά

Στο σύστημα μας υπάρχουν οι εξής οντότητες:

1. Χρήστης
2. Τμήμα
3. Πτέρυγα
4. Θάλαμος
5. Κρεβάτι
6. Ασθενής
7. Φιλοξενία ασθενή

Που σχετίζονται μέσω των ακόλουθων σχέσεων:

1. Κάθε ασθενής έχει πολλές φιλοξενίες, αλλά το πολύ μία που δεν έχει ημερομηνία εξαγωγής
2. Κάθε κρεβάτι έχει πολλές φιλοξενίες ασθενή αλλά το πολύ μία που δεν έχει ημερομηνία εξαγωγής
3. Κάθε κρεβάτι ανήκει αποκλειστικά και μόνο σε ένα θάλαμο. Κάθε θάλαμος μπορεί να έχει πολλά κρεβάτια.
4. Κάθε θάλαμος ανήκει σε μια και μόνο πτέρυγα. Κάθε πτέρυγα μπορεί να έχει πολλούς θαλάμους
5. Κάθε πτέρυγα ανήκει σε ένα μόνο τμήμα. Κάθε τμήμα μπορεί να έχει πολλές πτέρυγες.

Αναλυτικά

Πιο αναλυτικά παρουσιάζονται στον ακόλουθο πίνακα οι οντότητες και οι σχέσεις:

ΟΝΤΟΤΗΤΕΣ - ΠΕΔΙΑ		
Α/Α	**ΟΝΤΟΤΗΤΕΣ ΠΙΝΑΚΩΝ**	**ΠΕΔΙΑ**
1	Χρήστης	Κωδικός **(P[6])** Όνομα Επώνυμο Ιδιότητα [Διαχειριστής/Απλός χρήστης]
2	Τμήμα	Κωδικός**(P)** Ονομασία
3	Πτέρυγα	Κωδικός**)** Ονομασία Τμήμα**(FK[7])** Κατηγορία πολυτελείας[8]
4	Θάλαμος	Κωδικός**(P)** Ονομασία Πτέρυγα**(FK)**
5	Κρεβάτι	Κωδικός**(P)**

[6] Πρωτεύον κλειδί
[7] Ξένο κλειδί
[8] Η θέση πολυτελείας καθορίζει το σε ποια πτέρυγα θα φιλοξενηθεί ο ασθενής ανάλογα με την θέση που έχει ως ασφαλισμένος.

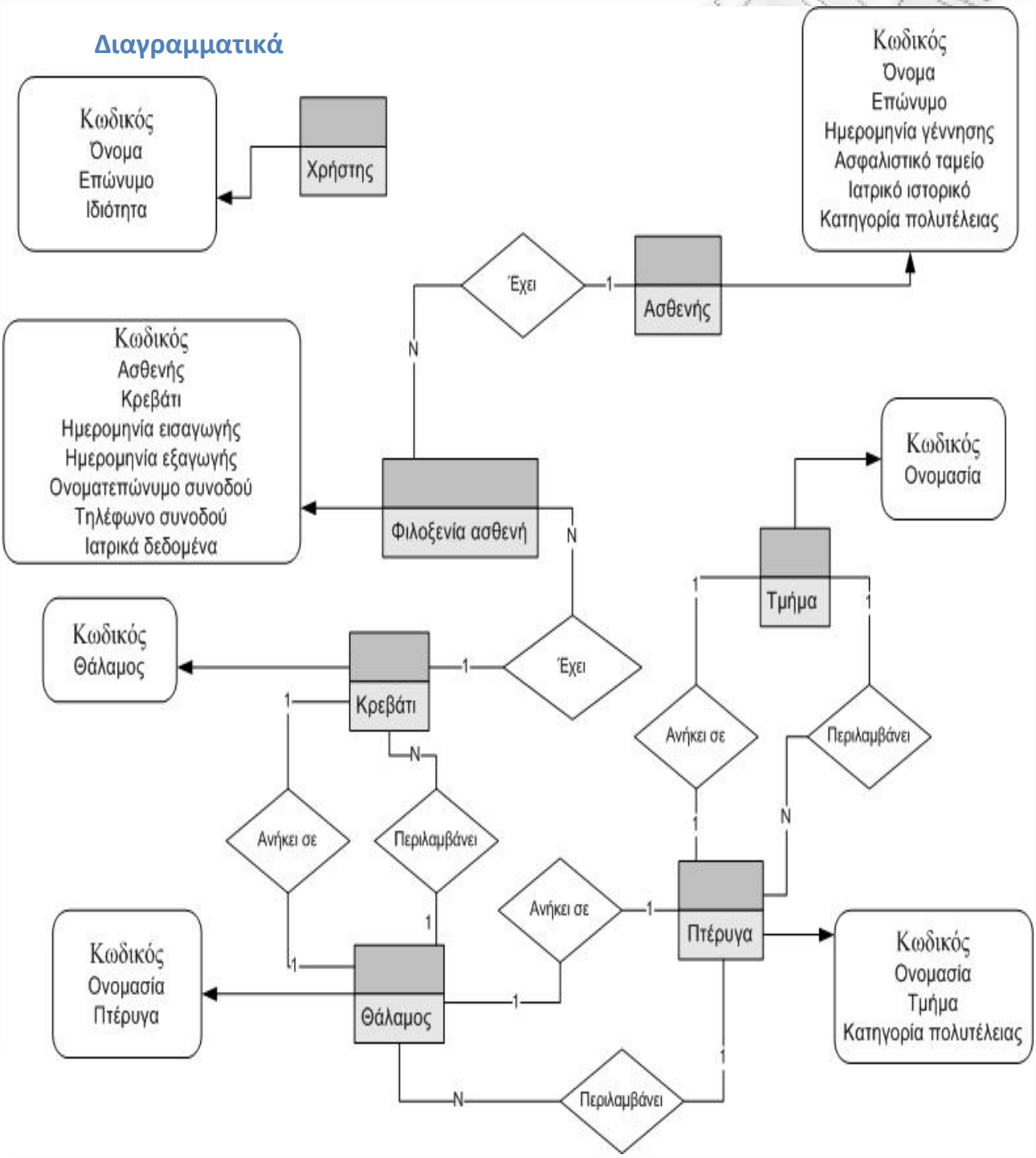
		Θάλαμος**(FK)**
6	Ασθενής	Κωδικός**(P)** Όνομα Επώνυμο Ημερομηνία γέννησης Ασφαλιστικό ταμείο Ιατρικό ιστορικό Κατηγορία πολυτελειας[8]
7	Φιλοξενία ασθενή	Κωδικός**(P)** Ασθενής**(FK)** Κρεβάτι**(FK)** Ημερομηνία εισαγωγής Ημερομηνία εξαγωγής Ονοματεπώνυμο συνοδού Τηλέφωνο συνοδού Ιατρικά δεδομένα

ΣΥΣΧΕΤΙΣΕΙΣ ΟΝΤΟΤΗΤΩΝ		
Α/Α	**ΣΥΣΧΕΤΙΣΗ-ΟΝΤΟΤΗΤΕΣ**	**ΠΕΡΙΓΡΑΦΗ ΚΑΙ ΠΕΔΙΑ**
1	Κάθε ασθενής έχει πολλές φιλοξενίες, αλλά το πολύ μία που δεν έχει ημερομηνία εξαγωγής[9]	**1 προς Ν**. Ένας ασθενή έχει πολλές φιλοξενείς Πεδία: Κωδικός, Ασθενής
2	Κάθε κρεβάτι έχει πολλές φιλοξενίες ασθενή αλλά το πολύ μία που δεν έχει ημερομηνία εξαγωγής	**1 προς Ν.** Ένα κρεβάτι έχει πολλές φιλοξενίες ασθενή. Πεδία: Κωδικός, Κρεβάτι
3	Κάθε κρεβάτι ανήκει αποκλειστικά και μόνο σε ένα θάλαμο. Κάθε θάλαμος μπορεί να έχει πολλά κρεβάτια.	**Ν προς 1.** Πολλά κρεβάτια ανήκουν σε έναν θάλαμο. Πεδία: Θάλαμος, Κωδικός
4	Κάθε θάλαμος ανήκει σε μια και μόνον πτέρυγα. Κάθε πτέρυγα μπορεί να	**Ν προς 1.** Πολλοί θάλαμοι ανήκουν σε μια πτέρυγα. Πεδία Πτέρυγα, Κωδικός

[9] Η ύπαρξη μοναδικής εγγραφής με κενή την ημερομηνία εξαγωγής ανά ασθενή και ανά κρεβάτι θα ελέγχεται απο την εφαρμογή κι όχι απο την βάση δεδομένων

	έχει πολλούς θαλάμους	
5	Κάθε πτέρυγα ανήκει σε ένα μόνον τμήμα. Κάθε τμήμα μπορεί να έχει πολλές πτέρυγες	**Ν προς 1.** Πολλές πτέρυγες ανήκουν σε ένα τμήμα. Πεδία Τμήμα, Κωδικός

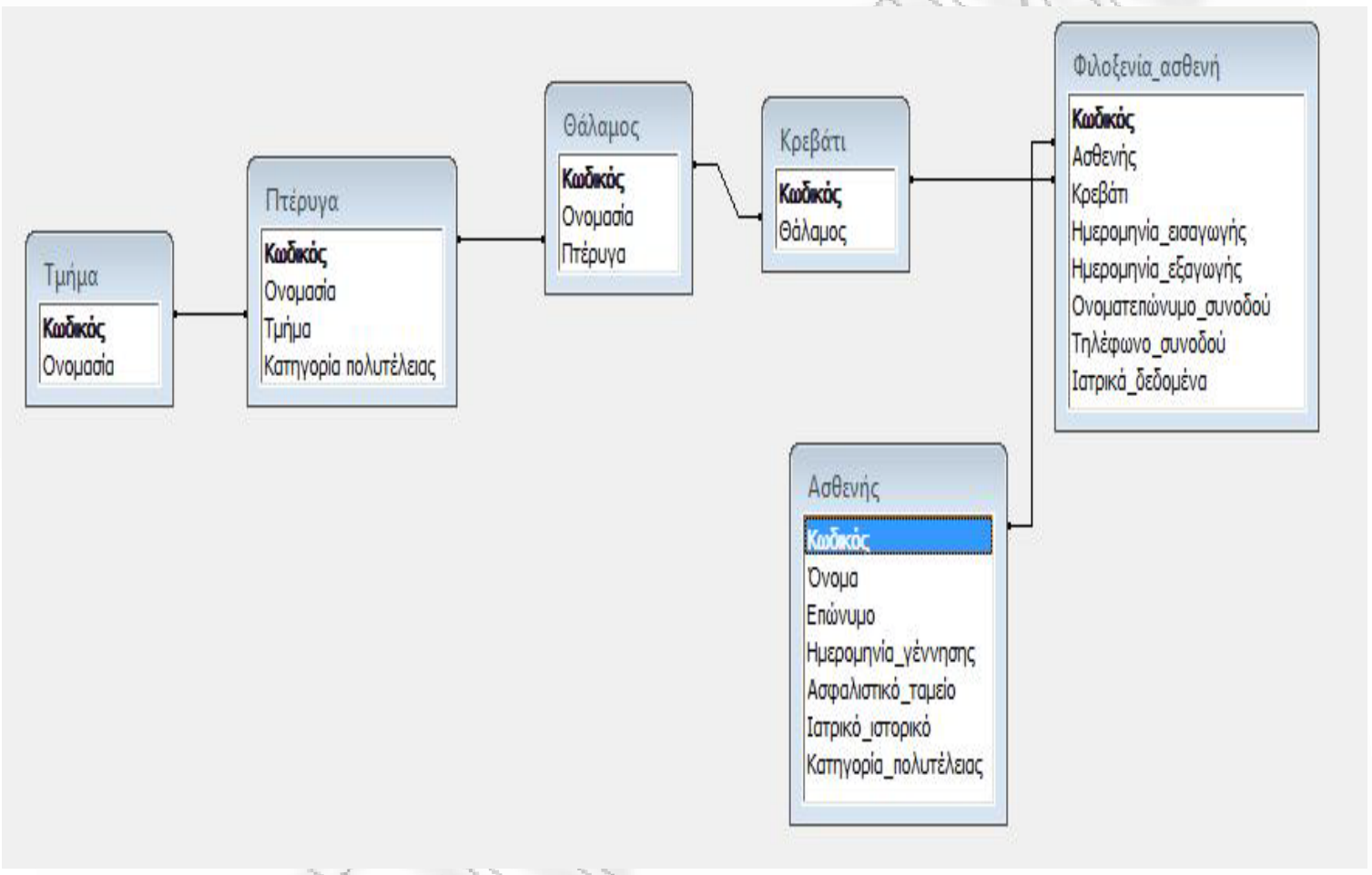
Διαγραμματικά



Σχήμα 8 Διάγραμμα οντοτήτων συσχετίσεων

4.2.2 Σχεσιακό σχήμα βάση δεδομένων

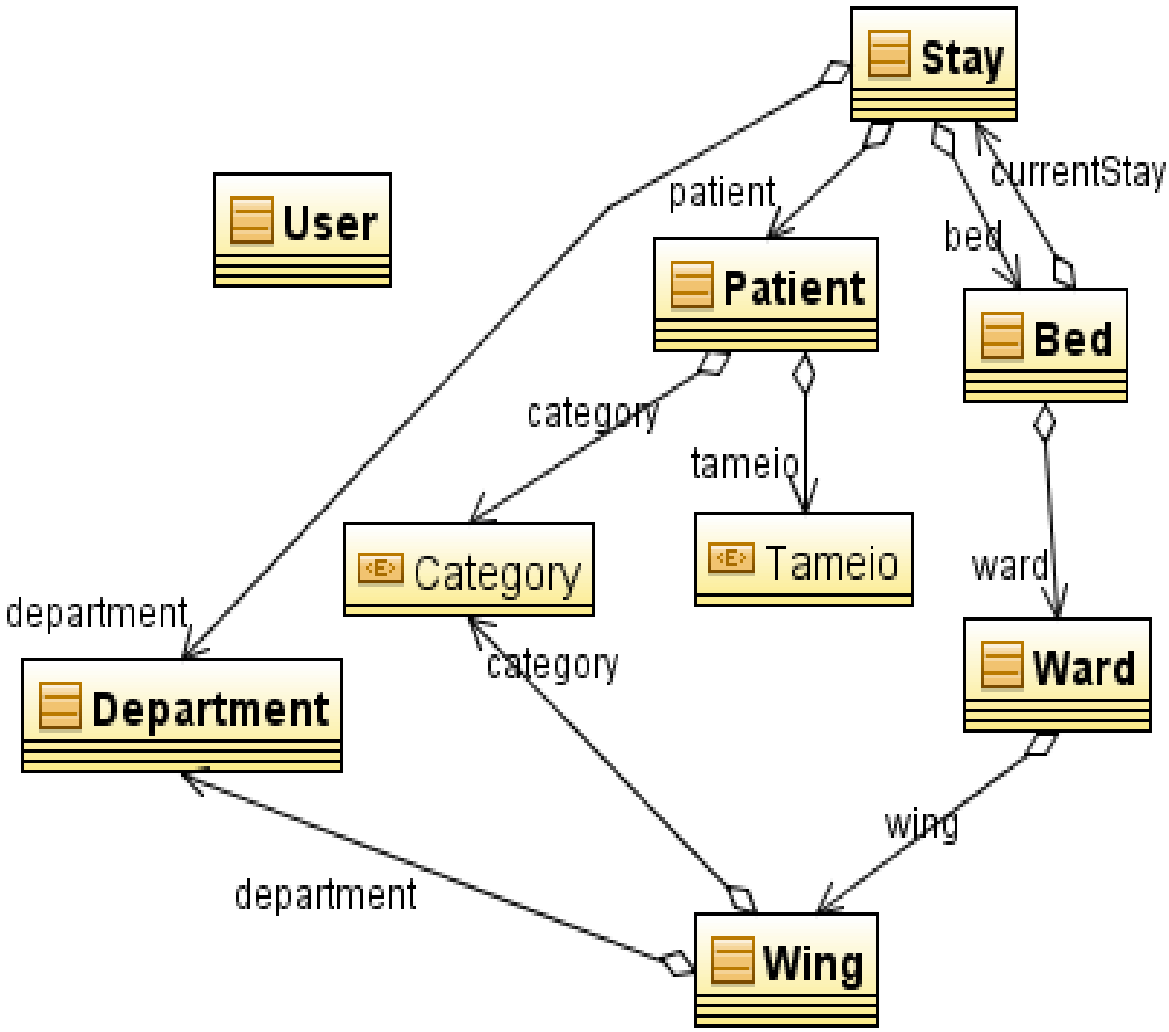
Σύμφωνα με την δομή των οντοτήτων και των μεταξύ τους συσχετίσεων δημιουργείται η σχεσιακή β



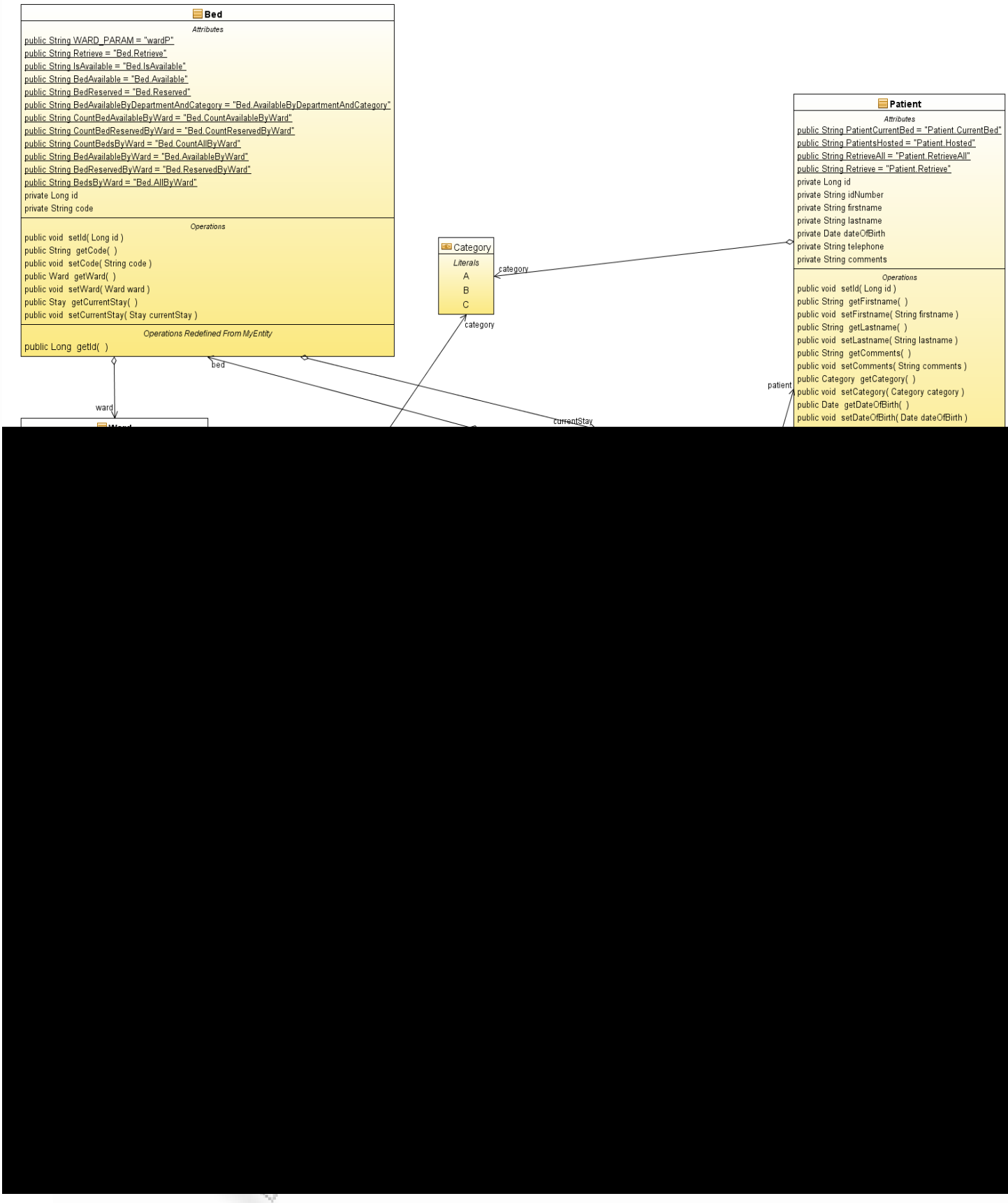
Σχήμα 9 Σχεσιακό σχήμα βάσεως δεδομένων

Διάγραμμα κλάσεων μοντέλου

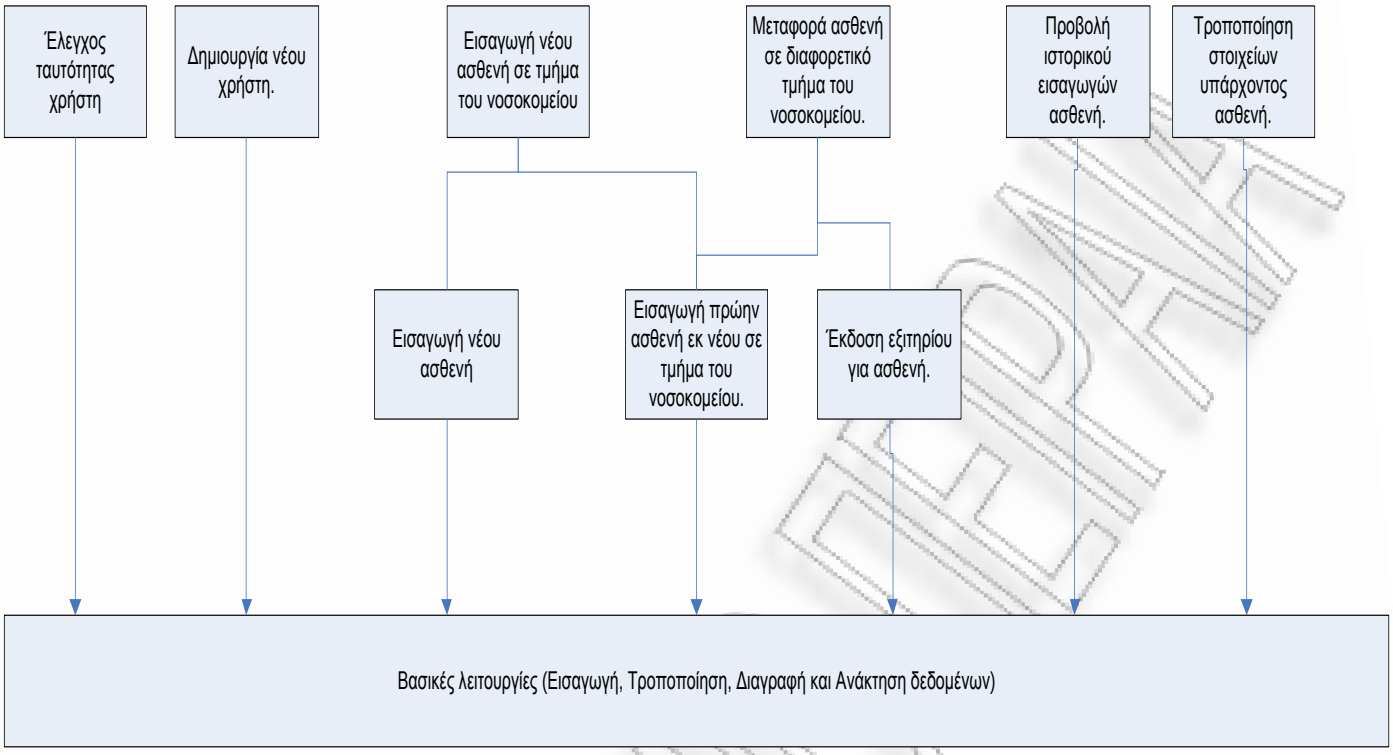
Προκειμένου να γίνει η υλοποίηση της εφαρμογής με χρήση Java, χρησιμοποιήθηκε το JPA (Java Persistence API). Το JPA είναι ένα πρότυπο για δημιουργία αντικείμενο σχεσιακών βάσεων δεδομένων, δηλαδή για την αξιοποίηση σχεσιακών βάσεων δεδομένων με χρήση αντικειμενοστραφούς λογικής. Με τη χρήση του συγκεκριμένου προτύπου δίδεται η δυνατότητα συσχέτισης κλάσεων με πίνακες. Για τη συγκεκριμένη περίπτωση δημιουργήθηκε το αντικειμενοστραφές μοντέλο που φαίνεται στο ακόλουθο σχήμα.



Σχήμα 10 Διάγραμμα κλάσεων (συνοπτικό)



Σχήμα 11 Διάγραμμα κλάσεων (αναλυτικό)



Σχήμα 12 Ιεραρχικό Διάγραμμα Υπηρεσιών Medis

4.2.3 Σχεδίαση λύσεων για τις βασικές περιπτώσεις χρήσεως

1. Έλεγχος ταυτότητας χρήστη

Μια από τις πλέον κοινές λειτουργίες των πληροφοριακών συστημάτων είναι αυτή του ελέγχου της ταυτότητας του χρήστη. Ο πλέον απλός τρόπος, ο οποίος αξιοποιήθηκε για το MED.I.S., είναι με την χρήση ενός συνδυασμού δύο λέξεων. Η μια λέξη χαρακτηρίζεται ως «όνομα χρήστη» και η άλλη «κωδικός πρόσβασης». Το «όνομα χρήστη» είναι μια λέξη που προσδιορίζει μονοσήμαντα κάποιον χρήστη, δηλαδή κάθε χρήστης έχει διαφορετικό «όνομα χρήστη». Ο «κωδικός πρόσβασης» είναι μια λέξη που πρέπει να γνωρίζει μόνον ο χρήστης και αξιοποιείται ώστε να επιβεβαιώνεται η ταυτότητα του.

Άρα κάθε χρήστης που θέλει να εισέλθει στο πληροφοριακό σύστημα MED.I.S. πρέπει να διαθέτει κάποιο προφίλ. Το προφίλ αυτό είναι η ταυτότητα του. Η ανάκτηση της ταυτότητας γίνεται μέσω του δοθέντος «ονόματος χρήστη», ενώ η επιβεβαίωση της ταυτότητας γίνεται μέσω ταυτοποίησης του «κωδικού πρόσβασης» που δίνει ο χρήστης. Αυτό επιτυγχάνεται με χρήση μιας απλής οθόνης που ζητάει αυτά τα δύο στοιχεία. Να σημειωθεί πως για λόγους ασφάλειας ο «κωδικός πρόσβασης» δεν θα πρέπει να φαίνεται ακόμα και κατά την δακτυλογράφηση.

2. Δημιουργία νέου χρήστη.

Η εφαρμογή θα έχει μια λειτουργία για προσθήκη επιπλέον χρηστών. Η

λειτουργία αυτή θα είναι διαθέσιμη αποκλειστικά σε μια ειδική κατηγορία χρηστών που χαρακτηρίζονται «Διαχειριστές». Η δημιουργία νέου χρήστη απαιτεί την αναλυτική καταχώριση των διαφόρων στοιχείων και απαραίτητα του «ονόματος χρήστη» και του «κωδικού χρήστη».

3. Εισαγωγή νέου ασθενή σε τμήμα του νοσοκομείου

Η βασικότερη λειτουργία του MED.I.S. είναι αυτή της εισαγωγής ασθενών στα διάφορα τμήματα του νοσοκομείου. Κατά την εισαγωγή νέου ασθενή θα πρέπει να γίνεται καταχώριση όλων των στοιχείων του, σε ποιο τμήμα θα εισαχθεί και ποια είναι τα στοιχεία του συνοδού του. Το σύστημα θα εντοπίζει αυτόματα κλίνη στο συγκεκριμένο τμήμα και σύμφωνα με την κατηγορία της ασφάλισης του ασθενή. Επίσης θα ορίζεται αυτόματα και η ημερομηνία εισαγωγής.

4. Εισαγωγή πρώην ασθενή εκ νέου σε τμήμα του νοσοκομείου.

Όταν γίνεται εισαγωγή ασθενή που είχε εισαχθεί κατά το παρελθόν θα δίδεται δυνατότητα ανάκτησης των στοιχείων του με χρήση του αριθμού ταυτότητας. Εφόσον γίνει η ανάκτηση θα μπορούν να αλλαχτούν τα στοιχεία. Προφανώς και σε αυτήν την περίπτωση θα πρέπει να προσδιοριστεί το τμήμα εισαγωγής και τα στοιχεία του συνοδού, κατά αντιστοιχία με το σενάριο «Εισαγωγή νέου ασθενή σε τμήμα του νοσοκομείου».

5. Τροποποίηση στοιχείων υπάρχοντος ασθενή.

Όταν ένας ασθενής έχει εισαχθεί στο νοσοκομείο, τότε μπορεί να γινει ανάκτηση της καρτέλας που αφορά την εισαγωγή του ώστε να γίνει η οποιαδήποτε μεταβολή στοιχείων. Η ανάκτηση θα γίνεται δίνοντας στο σύστημα τον αριθμό ταυτότητας του ασθενή.

6. Έκδοση εξιτηρίου για ασθενή.

Μια από τις «τροποποιήσεις» των στοιχείων του ασθενή θα είναι και αυτή της έκδοσης εξιτηρίου. Η έκδοση εξιτηρίου απλά ορίζει μια ημερομηνία εξιτηρίου στην καρτέλα της εισαγωγής του ασθενή. Η ημερομηνία εξιτηρίου ορίζεται αυτόματα.

7. Μεταφορά ασθενή σε διαφορετικό τμήμα του νοσοκομείου.

Μια άλλη περίπτωση «τροποποίησης» των στοιχείων του ασθενή ειναι αυτή της μεταφοράς του σε διαφορετικό τμήμα του νοσοκομείου. Ο χρήστης απλά πηγαίνει στην καρτέλα της «εισαγωγής» και επιλέγει διαφορετικό τμήμα. Με το που κάνει αποθήκευση, το σύστημα εντοπίζει την αλλαγή του τμήματος και εκδίδει αυτόματα εξιτήριο από το προηγούμενο τμήμα και κάνει νέα εισαγωγή στο καινούριο. Με αυτόν τον τρόπο αποφεύγει ο χρήστης να κάνει χειροκίνητα έκδοση εξιτηρίου και επαναεισαγωγή σε διαφορετικό τμήμα.

8. Προβολή ιστορικού εισαγωγών ασθενή.

Έχοντας αποθηκευμένες όλες τις εισαγωγές κάθε ασθενή είναι εφικτό να ανακτήσουμε ολόκληρό το ιστορικό του. Οι πληροφορίες αυτές θα εμφανίζονται υπό μορφή πίνακα, ταξινομημένες βάση ημερομηνίας ώστε να φαίνεται η σειρά των εισαγωγών.

Κεφάλαιο 5: Η ΥΛΟΠΟΙΗΣΗ MED.I.S.

Το πληροφοριακό σύστημα MED.I.S. αποτελείται από δύο εφαρμογές που επικοινωνούν μεταξύ τους. Η μια εφαρμογή είναι για τη παροχή των υπηρεσιών που καλύπτουν τα σενάρια χρήσης, ενώ η άλλη είναι για το καταναλωτή. Η εφαρμογή για το καταναλωτή είναι μια γραφική εφαρμογή ιδιαίτερα φιλική στον χρήστη.

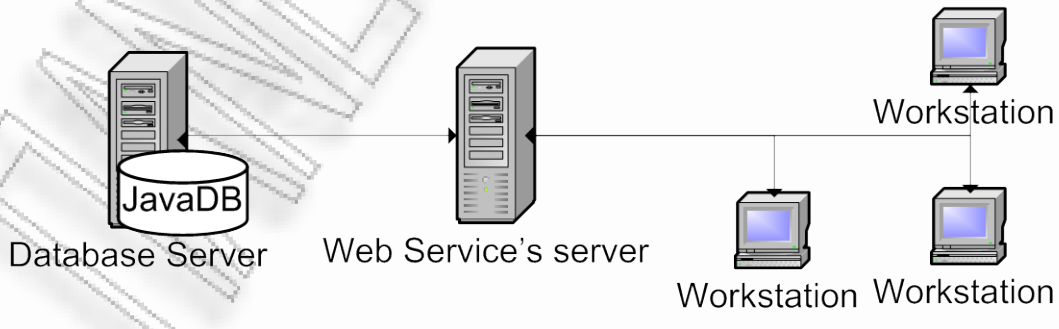
5.1 ΔΙΚΤΥΑΚΗ ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Όπως όλα τα σύγχρονα πληροφοριακά συστήματα, έτσι και το MED.I.S. λειτούργει σε περιβάλλον δικτύου. Η δικτυακή δομή του συγκεκριμένου πληροφοριακού συστήματος παρουσιάζεται στο ακόλουθο σχήμα. Τα μέρη που αποτελούν αυτό το δίκτυο είναι ο Database Server, ο Web Service's Server και τα Workstations.

Ο Database Server είναι ο κεντρικός υπολογιστής όπου τρέχει το σύστημα διαχείρισης βάσεων δεδομένων. Συγκεκριμένα για την περίπτωση του MED.I.S. τρέχει το ΣΣΔΒ JavaDB.

Ο Web Service's Server είναι ο κεντρικός υπολογιστής που επικοινωνεί με τον Database server και παρέχει με τυποποιημένο τρόπο κάποιες υπηρεσίες. Οι υπηρεσίες αυτές αφορούν τις διάφορες περίπτωσης χρήσης που καλύπτει η εφαρμογή. Η παροχή αυτών των υπηρεσιών γίνεται μέσω της τεχνολογίας των "Web Services". Αυτή η τεχνολογία έχει μια τυποποιημένη γλώσσα περιγραφής των διαθέσιμων υπηρεσιών, που ονομάζεται Web Services Definition Language (WSDL), κι αξιοποιεί και μια άλλη τυποποιημένη γλώσσα περιγραφής της δομής των δεδομένων που μεταφέρονται, που ονομάζεται XML Schema Definition (XSD). Χρησιμοποιώντας αυτές τις δύο γλώσσες παρέχεται στον καταναλωτή (consumer) των υπηρεσιών η γνώση του πως πρέπει να επικοινωνήσει με τον web service's server και τι είδους απάντηση θα πρέπει να περιμένει.

Τα workstations είναι μηχανήματα που έχει δικτυακή επικοινωνία με τον Web Service's server και εκτελούν την client εφαρμογή του MED.I.S.



5.2 Η ΕΦΑΡΜΟΓΗ MED.I.S. SERVER

Το σκέλος της εφαρμογής που εκτελείτε στον Server έχει δύο σκέλη. Το ένα σκέλος είναι το Data Access Object (DAO) και το άλλο είναι το Service, που η περιγραφή και παροχή των διαθέσιμων υπηρεσιών.

Προκειμένου η επικοινωνία με την βάση να απλοποιηθεί, έγινε χρήση του Java Persistence API (JPA) και συγκεκριμένα της υλοποίησης Hibernate. Επίσης έγινε χρήση του Framework Spring που δίνει την δυνατότητα απλοποίησης του κώδικα, μέσω αξιοποιήσεις annotations[10] που περιγράφουν εάν ο κώδικας της μεθόδου πρέπει να γίνει σε ένα transaction (@Transaction), αλλά και με χρήση απλών μεθόδων wrapper που κάνουν τις πλέον συνηθισμένες λειτουργίες.

Παράδειγμα κώδικα που εκτελείται στο ίδιο transaction:

```
@Transactional
    public void saveUser(User user) {
        User u = retrieveUser(user.getUserName(),
user.getPassword());
        if (u != null) {
            user.setId(u.getId());
            getJpaTemplate().merge(user);
        } else {
            getJpaTemplate().persist(user);
        }
    }
```

Για το σκέλος που αφορά την περιγραφή και παροχή υπηρεσιών έγινε χρήση του Metro WS Stack. Και αυτό το πακέτο δίνει την δυνατότητα περιγραφής του κώδικα με χρήση annotations. Με αυτό τον τρόπο χρησιμοποιείται απλά το annotation "@WebService" προκειμένου το αξιοποιηθεί η συγκεκριμένη κλάση ως περιγραφή παρό χ υ υπηρεσιών, αλλά και η χρήση το υ "@WebMethod" για να προσδιορίσει ότι κάποια μέθοδος ειναι μέθοδος της υπηρεσία αλλά και το annotation "@WebParam" το οποίο χρησιμοποιείται για να προσδιοριστεί το όνομα κάποιας παραμέτρου. Να σημειωθεί οτι το WSDL και τα XSD παράγονται αυτόματα αξιοποιώντας τις πληροφορίες που παρέχουν κύριως αυτά τα τρία annotations.

```
@WebService
@Transactional
public class Service {
```

[10] Annotaions ειναι σημανσεις στον κώδικα οι οποίες αναγνωρίζονται απο τον compiler ή άλλα σημεία του κώδικα προκειμένου να χρήζουν ειδικής αντιμετώπισης

```
    protected Dao dao = (Dao)
ApplicationContextBeanProvider.getBean("dao");

    @WebMethod()

    public ArrayList<Stay> getStayHistory(@WebParam(name =
"patientIdNumber") String patientIdNumber){

        return dao.getStayHistory(patientIdNumber);

    }

}
```

5.3 Η ΕΦΑΡΜΟΓΗ MED.I.S. CLIENT

Κατ' αντιστοιχία η client εφαρμογή αποτελείται κι αυτή από δύο σκέλη. Το ένα είναι ο web services consumer και το άλλο είναι η γραφική επαφή.

Ο web services consumer παράγεται αυτόματα αξιοποιώντας την πληροφορία που παρέχει το wsdl και τα xsd. Είναι ουσιαστικά μια κλάση που έχει όλες τις διαθέσιμες μεθόδους του wsdl προκειμένου ο χρήστης να καλεί το web service οπως θα καλούσε οποιαδήποτε μέθοδο java.

Η γραφική διεπαφή αποτελείται από ένα κεντρικό διαχειριστή της εφαρμογής «MEDISClientApp» και 8 διαφορετικά views. Ο κεντρικός διαχειριστής είναι αυτός που αναλαμβάνει την μετάβαση από τη μια οθόνη στην άλλη, αλλά και την επικοινωνία με τον web services consumer. Για την ευκολότερη ανάπτυξη έγινε αξιοποίηση του Swing Application Framework. Το Swing Application Framework είναι ένα πακέτο βιβλιοθηκών που στόχο έχει τη τυποποίηση του τρόπου ανάπτυξης γραφικών εφαρμογών Java.

5.4 ΈΛΕΓΧΟΣ – ΣΕΝΑΡΙΑ ΧΡΗΣΗΣ

Η φάση αυτή είναι ιδιαιτέρως σημαντική. Ο κώδικας πρέπει να ελεγχθεί μέσα στο λειτουργικό σύστημα. Οι δομικές μονάδες ,όπως προέκυψαν από τη παραπάνω σχεδίαση του λογισμικού, ελέγχονται στην αρχή ξεχωριστά και έπειτα γίνεται η συνένωση και ο έλεγχος του συνόλου. Αυτή η φάση είναι κατά κάποιον τρόπο η επιβεβαίωση ότι ο στόχος επιτεύχθηκε ή ότι κάποιο συγκεκριμένο κομμάτι θέλει διόρθωση ώστε να οδηγηθούμε σε ένα άρτιο αποτέλεσμα κι έναν ευχαριστημένο τελικό χρήστη, που πρέπει να είναι ο στόχος κάθε συστήματος που αναπτύσσεται.

5.4.1 Μεθοδολογία

Η μεθοδολογία η οποία θα χρησιμοποιηθεί για τον έλεγχο μιας εφαρμογής είναι ίσως η πιο σημαντική απόφαση, καθώς μπορεί να εκτινάξει εύκολα το κόστος ολόκληρου του έργου ή απλά να μην οδηγήσει στα σωστά αποτελέσματα, δηλαδή να μην αποκαλύψει τα λειτουργικά λάθη που υπάρχουν, αλλά αντιθέτως να τα καλύψει.

Για τον λόγο αυτό πρέπει η δοκιμή να μη γίνεται από τον δημιουργό-προγραμματιστή του συστήματος, αλλά από κάποιον που πλησιάζει όσο το δυνατόν περισσότερο την κατάσταση του τελικού χρήστη. Αυτό γίνεται προκειμένου να αναπαραχθούν τα σενάρια με τον πλέον ρεαλιστικό τρόπο κι έτσι να προκύψουν τα οποιαδήποτε προβλήματα αναμένεται να αντιμετωπίσει ο τελικός χρήστης πριν καν φτάσει το σύστημα στα χέρια του.

Για τον λόγο αυτό εξετάστηκαν διάφορες εναλλακτικές λύσεις ώστε να εντοπιστεί η πλέον κατάλληλη για την περίπτωση του παρόντος συστήματος. Ακολούθως θα παρουσιαστούν οι πιο σημαντικές από αυτές τις μεθόδους.

5.4.2 Ad hoc testing

Ο πλέον απλός τρόπος να δοκιμάσεις ένα σύστημα είναι να αρχίσεις να το χρησιμοποιείς χωρίς να υπάρχει κάποια συγκεκριμένη καταγραφή των προς εξέταση σεναρίων, αλλά ούτε και των αποτελεσμάτων. Όταν η εφαρμογή είναι αρκετα μικρή η μέθοδος αυτή αποτελεί την πλέον κατάλληλη μέθοδο, αλλά όταν το επίπεδο πολυπλοκότητας ανέβει τότε τα πράγματα αλλάζουν. Αυξάνεται δραματικά ο κίνδυνος να οδηγήσει σε λανθασμένα σε συμπεράσματα ότι το σύστημα λειτουργεί ενώ στην πραγματικότητα μπορεί να υπάρχουν σημαντικά λειτουργικά κενά.

5.4.3 User Acceptance Testing

Μια καλή μέθοδος ελέγχου ενός συστήματος είναι αυτή η οποία προσομοιώνει κατά το δυνατόν πιο ρεαλιστικά το περιβάλλον όπου θα λειτουργεί κανονικά. Μια τέτοια μέθοδος είναι και η UAT. Στην UAT η ομάδα που ελέγχει τη σωστή λειτουργία είναι ειδικοί πάνω στο αντικείμενο, που καλείται να εξυπηρετήσει το σύστημα, δηλαδή έχουν το «know how». Πολλές ο ίδιος ο πελάτης αναλαμβάνει τον ρόλο του testing. Όπως λέει και το όνομα της συγκεκριμένης μεθοδολογίας, στόχος είναι η αποδοχή από τον χρήστη. Η αποδοχή ότι το σύστημα καλύπτει όλα όσα περιγράφτηκαν στην τεκμηρίωση των απαιτήσεων του χρήστη.

Το βασικό πλεονέκτημα είναι ότι τα σενάρια που ελέγχονται είναι ρεαλιστικά, ενώ το μειονέκτημα είναι ότι γίνεται μόνο στο τελικό στάδιο και δεν βοηθάει στην ανάπτυξη ενός συστήματος όπως έχει παραγραφεί για το παρόν. Πέραν αυτού του μειονεκτήματος υπάρχουν και τα μειονεκτήματα ότι δεν ελέγχονται ενδεχομένως οι ειδικές περιπτώσεις αλλά και οι περιπτώσεις υψηλού φόρτου και λοιπών ακραίων τεχνικών καταστάσεων.

5.4.4 Test driven development

Η μεθοδολογία του test driven development αφορά όχι μόνο την τελική φάση ελέγχου του συστήματος αλλά και τη σταδιακή ανάπτυξη μέσω του αρχικού καθορισμού του αποτελέσματος κι εν συνεχεία η δημιουργία του κομμάτιου του κώδικα που θα το παράγει. Είναι λίγο διαφορετική από τη κλασική ροή ανάπτυξης λογισμικού, αλλά οδηγεί σε αρκετά καλύτερα αποτελέσματα. Οι δοκιμές γενικότερα που πρέπει να γίνονται σε έναν έλεγχο βασίζονται στις απαιτήσεις που έχουν καταγραφεί οπότε ίσως είναι πιο αποτελεσματικό πριν δημιουργηθεί κάθε κομμάτι του συστήματος να υπάρχει κάποιος ο οποίος να ελέγχει το αναμενόμενο αποτέλεσμα, δηλαδή να υπάρχει κατά κάποιον τρόπο η θεωρητική προσέγγιση προ της πρακτικής, ώστε να μπορεί να επιβεβαιωθεί.

5.4.5 Unit testing

Το Unit testing είναι μια ειδική περίπτωση test driven development που βασίζεται στην βασική αρχή του «διαιρεί και βασίλευε». Συγκεκριμένα η όλη λογική είναι ότι ένα σύστημα διαιρείται σε τμήματα, σενάρια χρήσης, τα οποία για υλοποιηθούν χρειάζονται κάποια συγκεκριμένα κομμάτια κώδικα (Units). Με τη συγκεκριμένη μεθοδολογία, πριν να γραφτεί κάποιο κομμάτι κώδικα, γράφουμε το αντίστοιχο κομμάτι που ελέγχει την ορθότητα του. Δηλαδή δημιουργούμε κώδικα ο οποίος καλεί το πραγματικό κομμάτι του συστήματος, με κατάλληλες παραμέτρους που δοκιμάζουν τα διάφορα σενάρια που θέλουμε να λειτουργούν και ελέγχουμε αν τα αποτελέσματα είναι τα αναμενόμενα. Ουσιαστικά με αυτό το τρόπο δημιουργούμε ένα πρόγραμμα που ελέγχει συγκεκριμένα σενάρια και το οποίο θα ελέγχει το σύστημα για εμάς. Η συγκεκριμένη τεχνική έχει το μειονέκτημα ότι ο έλεγχος γίνεται μέσω της ανάπτυξης λογισμικού κι ενέχει το κίνδυνο ο κώδικας που κάνει τον έλεγχο να έχει επίσης κενά και να καταλήγουμε πάλι σε λάθος συμπεράσματα!

Στην παρούσα διπλωματική, δυστυχώς, δεν υπήρχε η πολυτέλεια να δοκιμαστεί από πραγματικούς χρήστες, οπότε ήταν αδύνατον να έχουμε τυπική εφαρμογή της μεθοδολογίας UAT, οπότε χρησιμοποιήθηκε η μέθοδος του test driven developement, όπως παρουσιάζεται και στην μεθοδολογία ανάπτυξης του συστήματος. Επίσης για να είναι πιο εύκολα παρουσιάσιμα τα αποτελέσματα των ελέγχων στην παρούσα αναφορά, ακολουθείται και μια παραλλαγή της UAT, όπου ο σχεδιαστής των προς έλεγχο σεναρίων αλλά και οι χρήστες δεν είναι αυτοί που θα χρησιμοποιήσουν όντως την εφαρμογή.

5.4.6 Εφαρμογή μεθοδολογίας

Στις ακόλουθες παραγράφους θα παρουσιαστούν τα σενάρια ελέγχου (Test cases) που ελέχθησαν καθώς και τα αποτελέσματα που προέκυψαν από αυτούς τους ελέγχους. Τα συγκεκριμένα αποτελέσματα αφορούν μόνον τους ελέγχους σύμφωνα με την τροποποιημένη μεθοδολογία UAT.

Κατ’ αρχήν πρέπει να ορίσουμε το ποιοι και τι θα ελέγξουν

Συμμετέχοντες στον έλεγχο

Ρόλος	*Τμήμα*	*Όνομα*
Διαχειριστής	*Γενικής διαχείρισης*	*Νικόλαος Φίλιος*
Ιατρός	*Αναζήτηση κλίνης ασθενή*	*Νικόλαος Φίλιος*
Εξωτερικός	*Αναζήτηση διαθέσιμης κλίνης*	*Νικόλαος Φίλιος*

Σενάρια

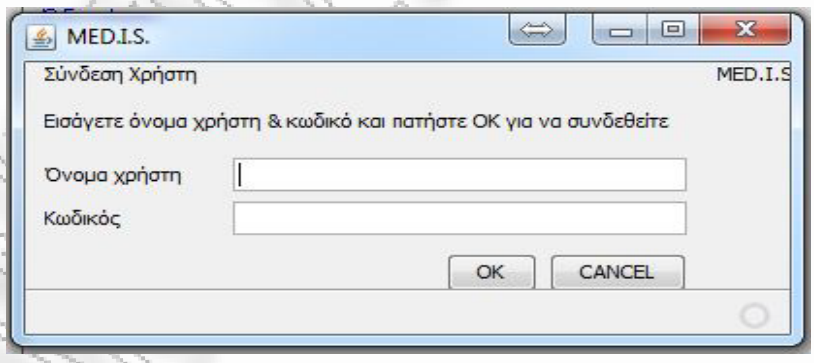
Ρόλος	*Όνομα*	*Σενάριο*
Διαχειριστής	*Νικόλαος Φίλιος*	*1.*
Ιατρός	*Νικόλαος Φίλιος*	*1. Αναζήτηση κλίνη ασθενή*
Εξωτερικός	*Νικόλαος Φίλιος*	*1. Αναζήτηση διαθέσιμης κλίνης*

Σενάριο 1 :

Έλεγχος ταυτότητας χρηστή

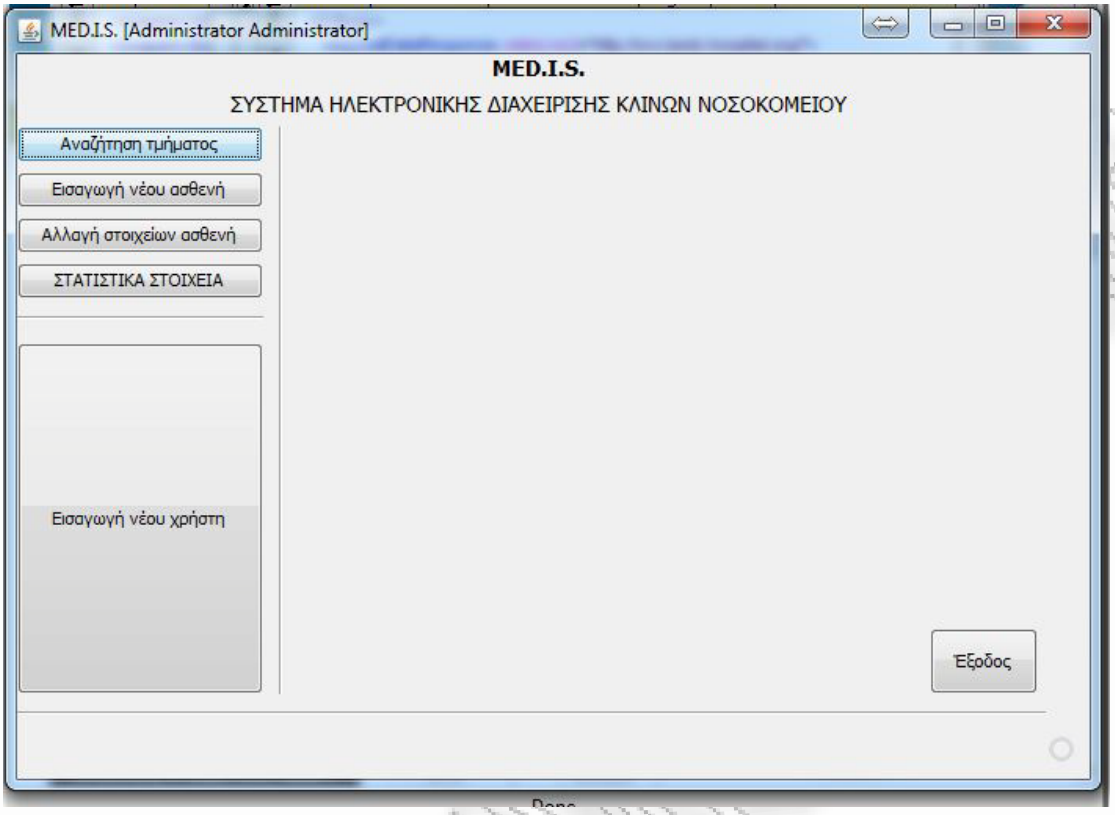
Περιγραφή:

Βήμα 1 Τρέχουμε την εφαρμογή και εμφανίζεται η οθόνη εισαγωγής στο σύστημα



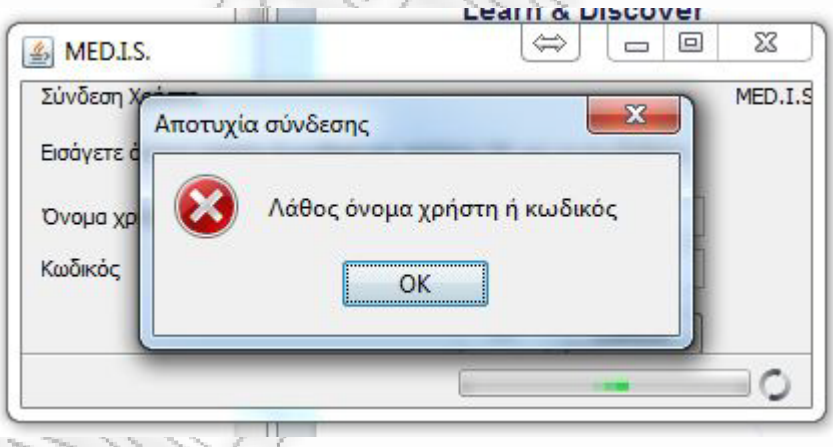
Είσοδος στο σύστημα 1

Βήμα 2 Πληκτρολογούμε ένα έγκυρο όνομα χρηστή (π.χ. admin) και έγκυρο κωδικό (π.χ. admin) έτσι έχουμε την εισαγωγή του administrator του συστήματος και εμφανίζεται η παρακάτω οθόνη



Αρχική οθόνη συστήματος 2

Βήμα 3 Αν πληκτρολογήσουμε λανθασμένα στοιχεία εμφανίζεται η παρακάτω οθόνη



Μήνυμα Λάθους 3

Αναμενόμενο αποτέλεσμα:

Τα στοιχεία η θα είναι σωστά και θα γίνετε εισαγωγή στο σύστημα η θα είναι λάθος και θα εμφανίζεται μήνυμα λάθους.

Αποτέλεσμα:

Επιτυχία

Σχόλια:

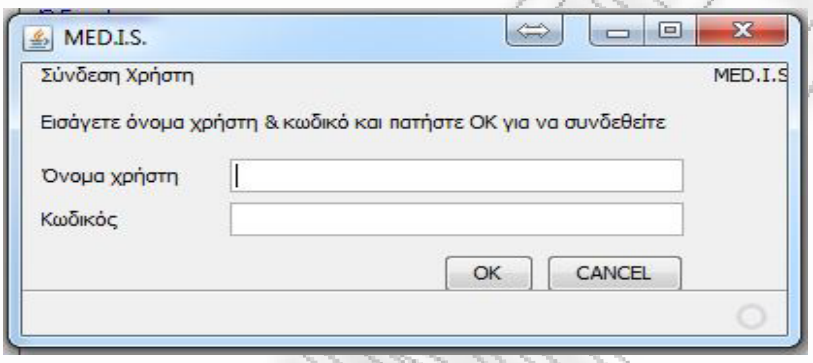
Μελλοντική εκδοση στο μήνυμα λάθους θα μπορούσε να μας λέει πιο ακριβώς είναι το λάθος δηλαδή αν είναι λάθος ο κωδικός πρόσβασης να εμφανίζει λάθος κωδικό και όχι και όνομα χρηστή.

Σενάριο 2 :

Δημιουργία νέου χρήστη

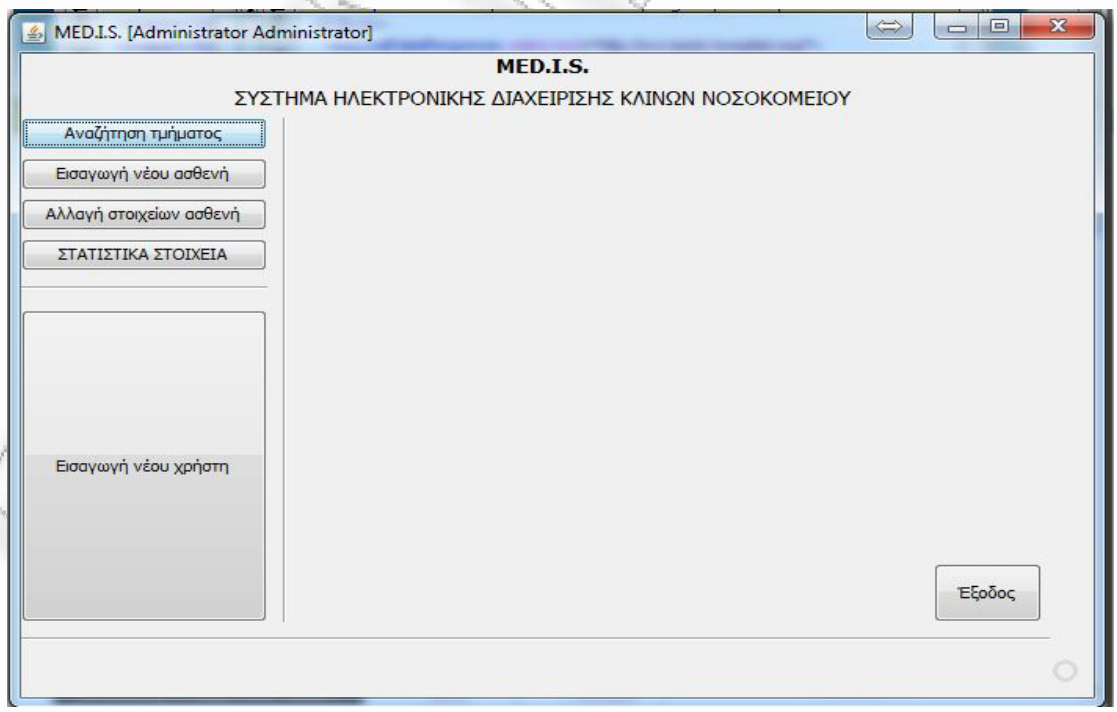
Περιγραφή:

Βήμα 1 Τρέχουμε την εφαρμογή και εμφανίζεται η οθόνη εισαγωγής στο σύστημα



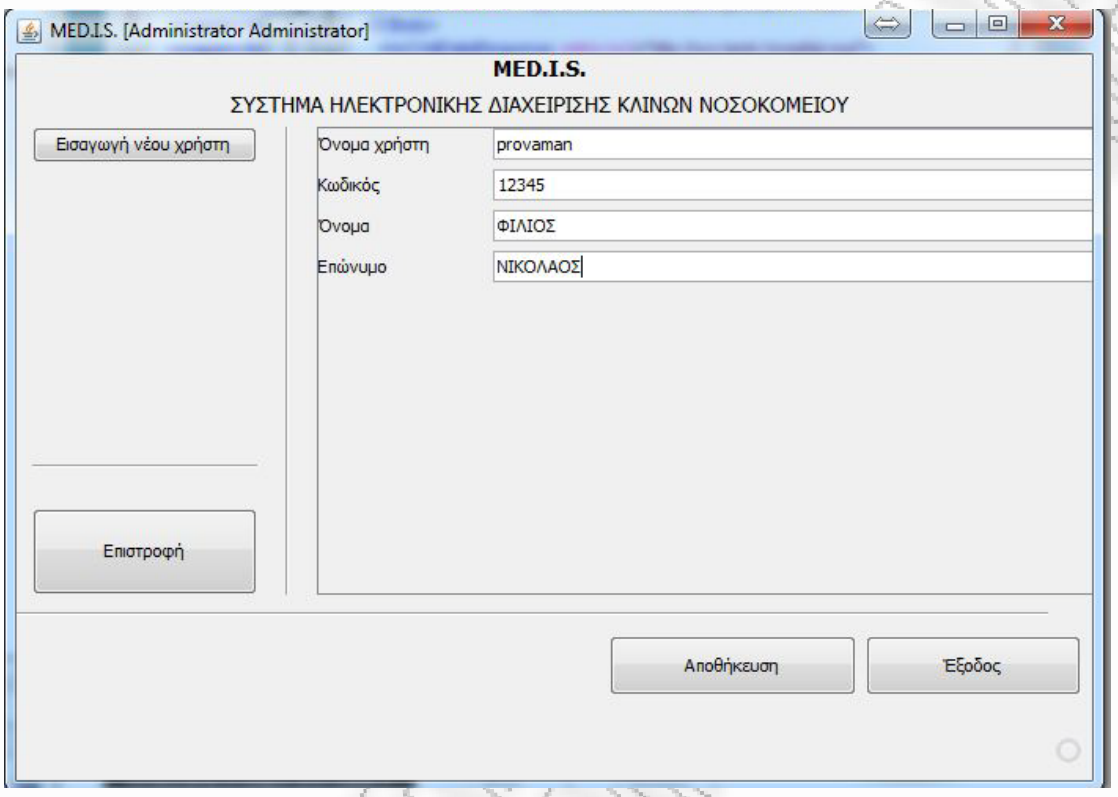
Είσοδος στο σύστημα 1

Βήμα 2 Πληκτρολογούμε ένα έγκυρο όνομα χρηστή (π.χ. admin) και έγκυρο κωδικό (π.χ. admin) έτσι έχουμε την εισαγωγή του administrator του συστήματος και εμφανίζεται η παρακάτω οθόνη



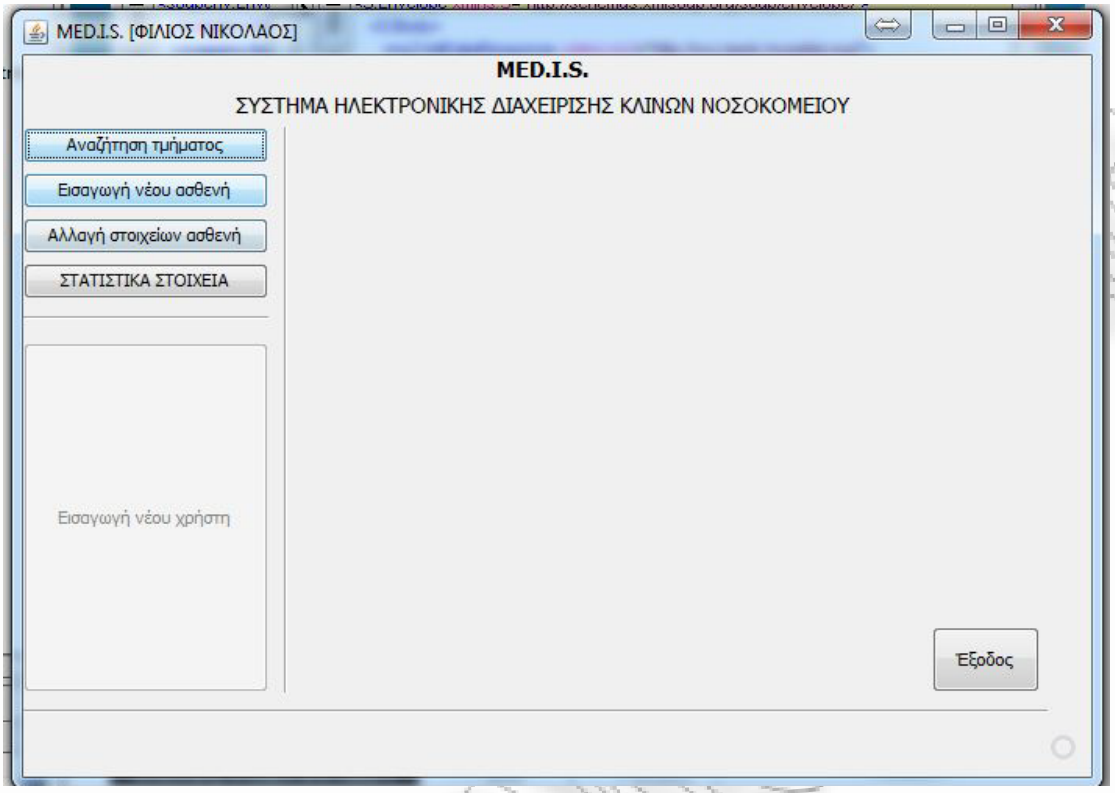
Αρχική οθόνη συστήματος με δικαιώματα administrator 2

Βήμα 3 Επιλεγούμε Εισαγωγή νέου χρηστή και εμφανίζεται η παρακάτω οθόνη και έπειτα πληκτρολογούμε το χρηστή που θέλουμε και πατάμε αποθήκευση



Οθόνη εισαγωγής νέου χρηστή 3

Βήμα 4 Έπειτα κάνουμε login στο σύστημα με το χρηστή που μόλις δημιουργήσαμε δηλαδή με Όνομα χρήστη provaman και Κωδικό 12345 και εμφανίζεται η παρακάτω οθόνη όπου παρακολουθούμε ότι δεν είναι ενεργή η δημιουργία νέου χρηστή.



Αρχική οθόνη συστήματος απλού χρηστή 4

Αναμενόμενο αποτέλεσμα:

Αν έχουμε δικαιώματα administrator στο σύστημα μπορούμε να εισαγάγουμε νέο χρήστη ενώ αν όχι δεν μπορούμε.

Αποτέλεσμα:

Επιτυχία

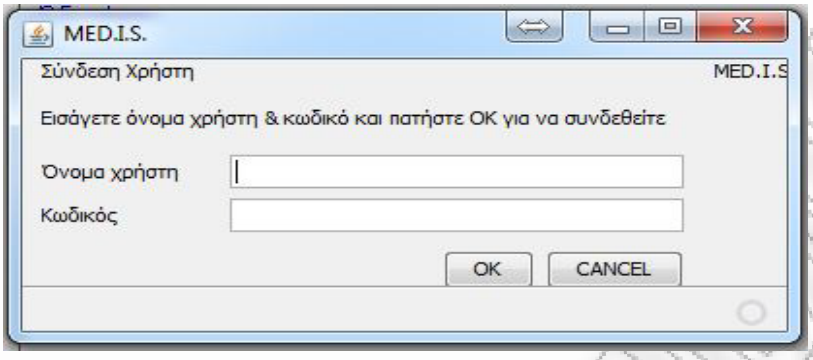
Σχόλια: Μελλοντική έκδοση θα μπορούσε στη δημιουργία χρηστών να υπήρχαν περισσότερες κατηγορίες χρηστών.

Σενάριο 3:

Εισαγωγή νέου ασθενή σε τμήμα του νοσοκομείου.

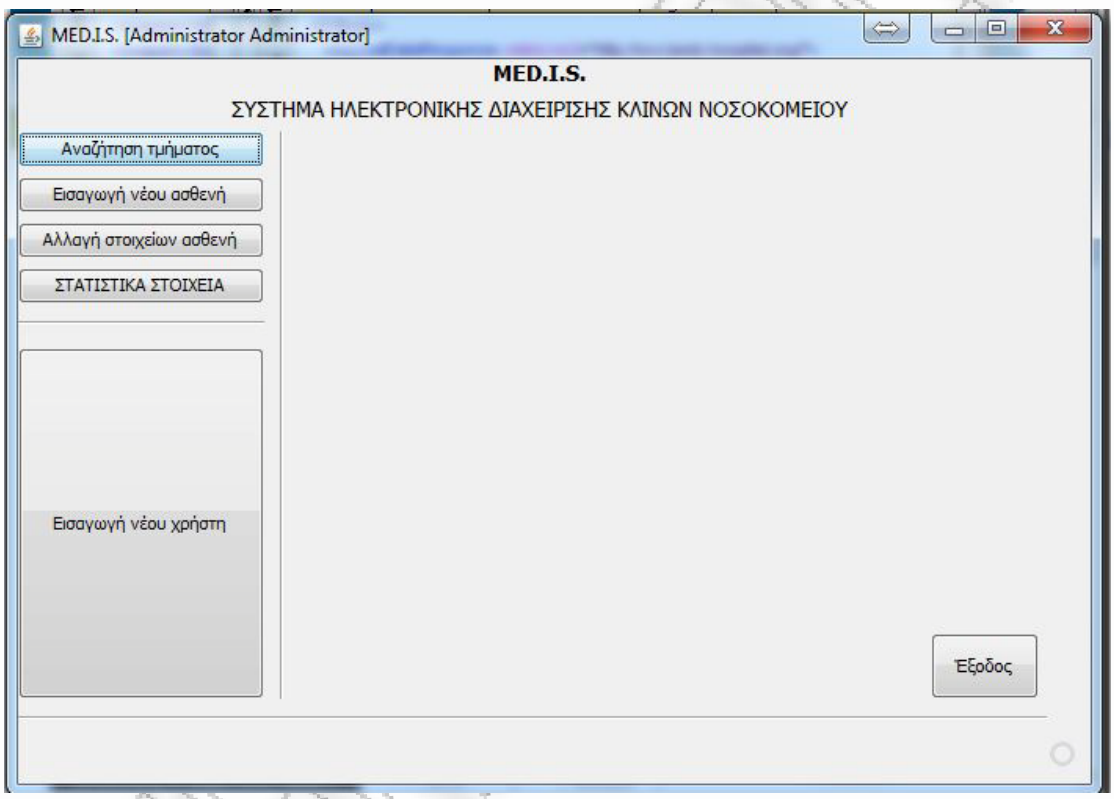
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



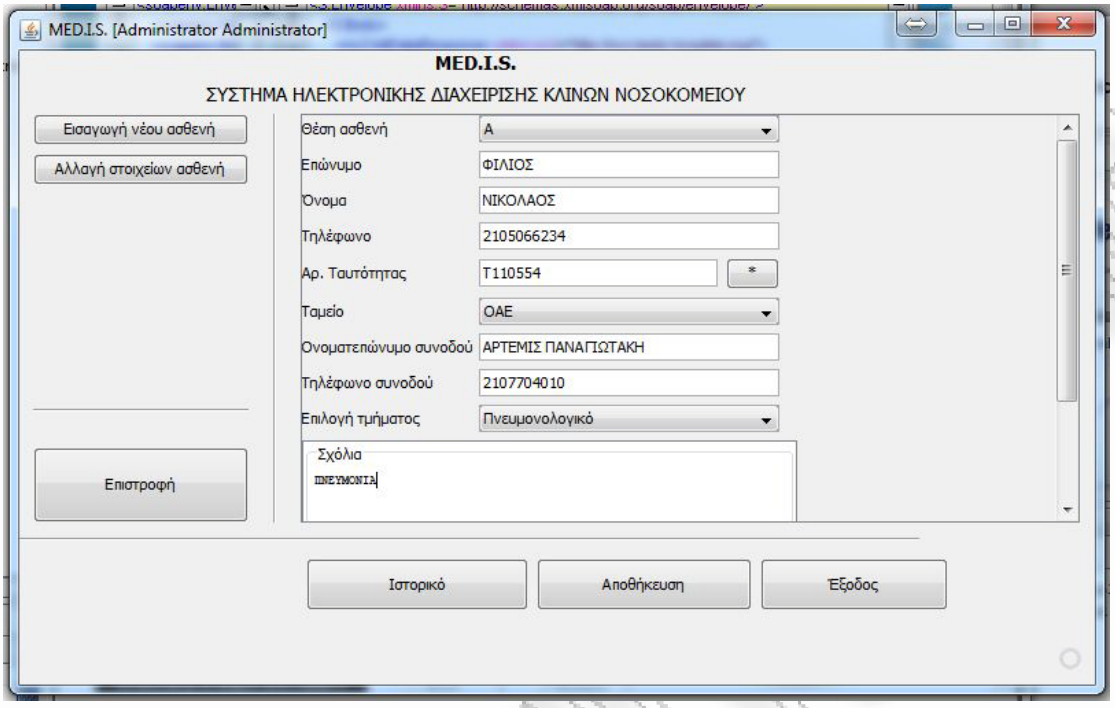
Είσοδος στο σύστημα 2

Βήμα 2 Επιλεγούμε εισαγωγή νέου ασθενή στην αρχική μας οθόνη.



Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τα στοιχεία του ασθενή που θέλουμε να κάνουμε εισαγωγή και επιλεγούμε τμήμα και πατάμε αποθήκευση.



Οθόνη εισαγωγής νέου ασθενή 3

Αναμενόμενο αποτέλεσμα:

Εισαγωγή ασθενή στο πληροφοριακό μας σύστημα.

Αποτέλεσμα:

Επιτυχία

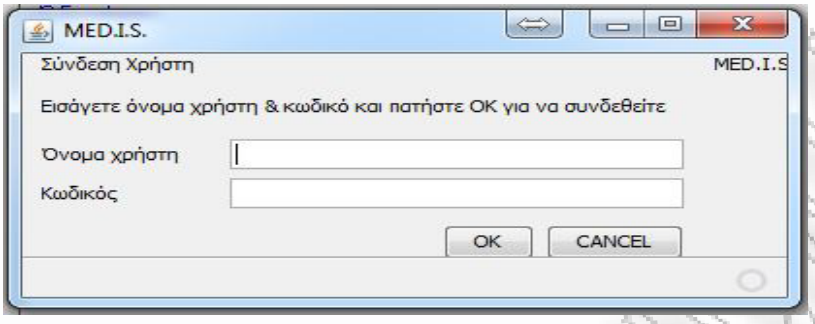
Σχόλια:

Σενάριο 4:

Εισαγωγή πρώην ασθενή εκ νέου σε τμήμα του νοσοκομείου

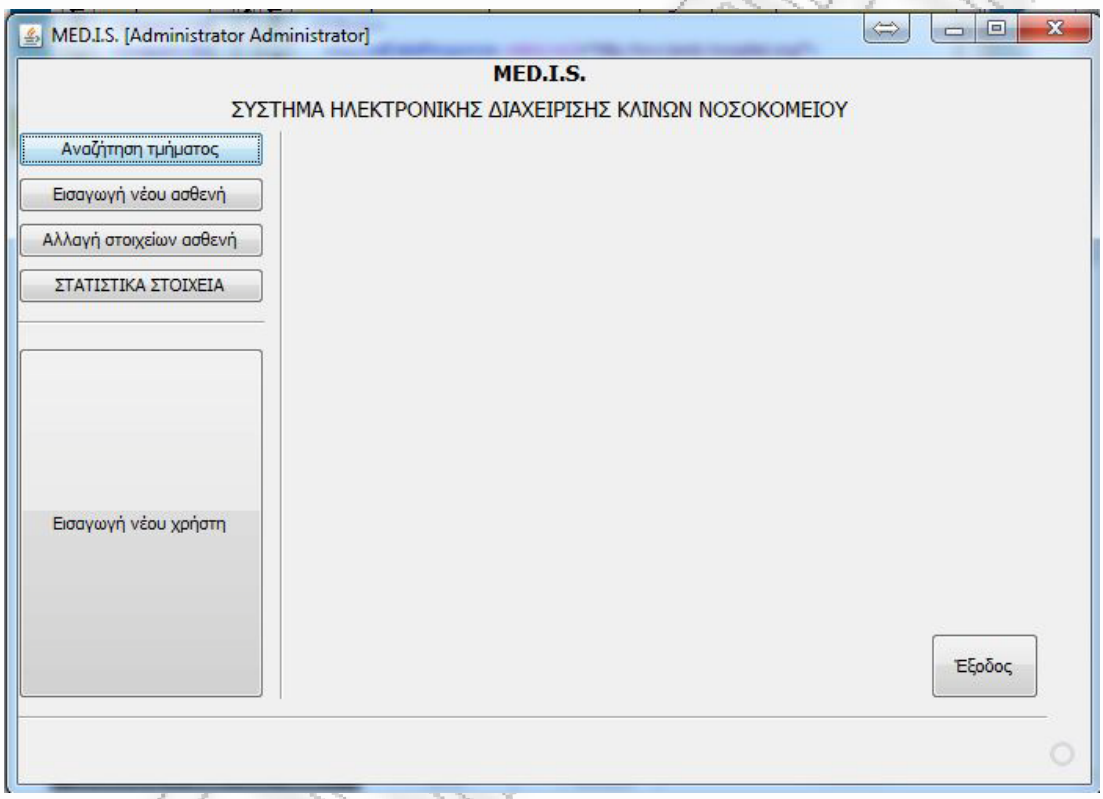
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



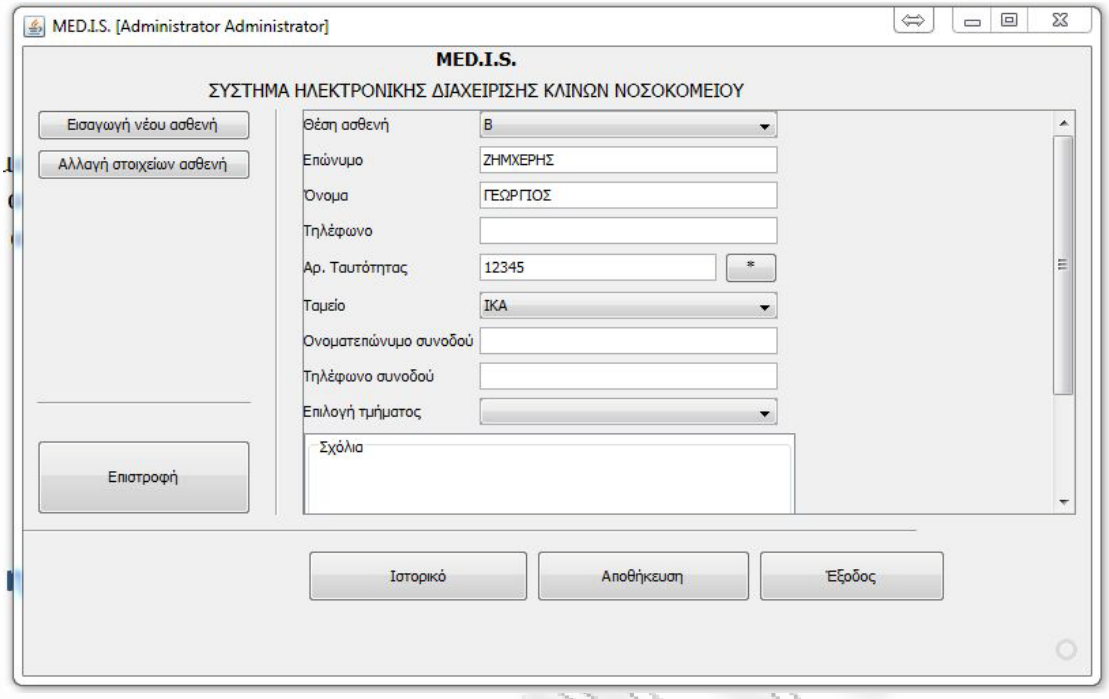
Είσοδος στο σύστημα 3

Βήμα 2 Επιλεγούμε εισαγωγή νέου ασθενή στην αρχική μας οθόνη.



Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τον αριθμό ταυτότητας του παλιού μας ασθενή και πατάμε τον αστερίσκο και εμφανίζονται τα στοιχειά του και κάνουμε τη καινούρια εισαγωγή του ασθενή στο τμήμα που θέλουμε.



Οθόνη εισαγωγής νέου ασθενή 3

Αναμενόμενο αποτέλεσμα:

Εύρεσή στοιχείων παλαιότερου ασθενή.

Αποτέλεσμα:

Επιτυχία

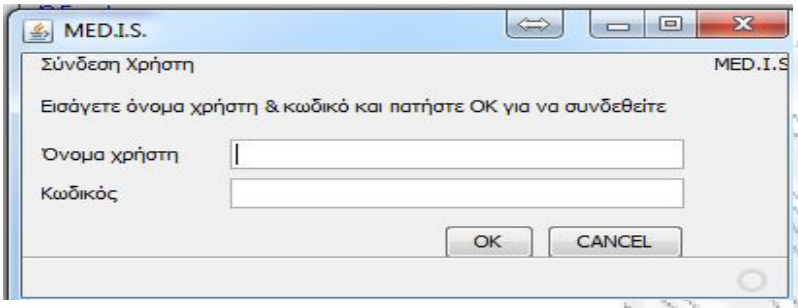
Σχόλια:

Σενάριο 5:

Τροποποίηση στοιχείων υπάρχοντος ασθενή.

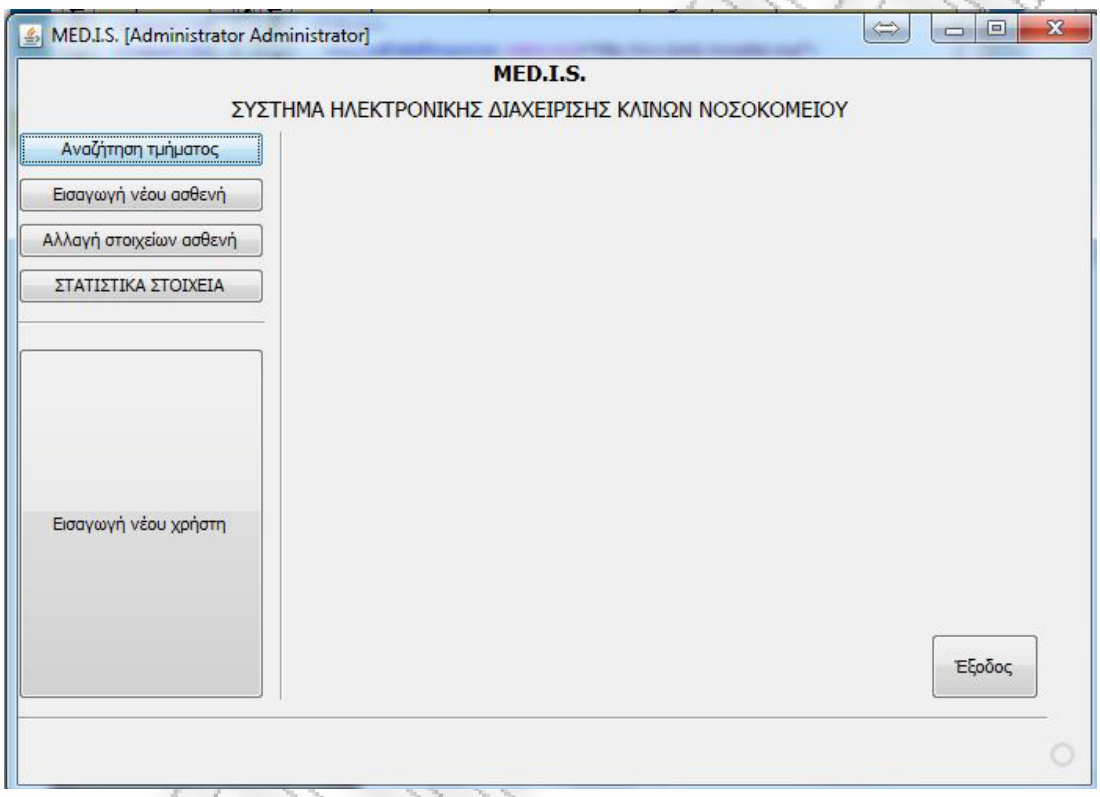
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



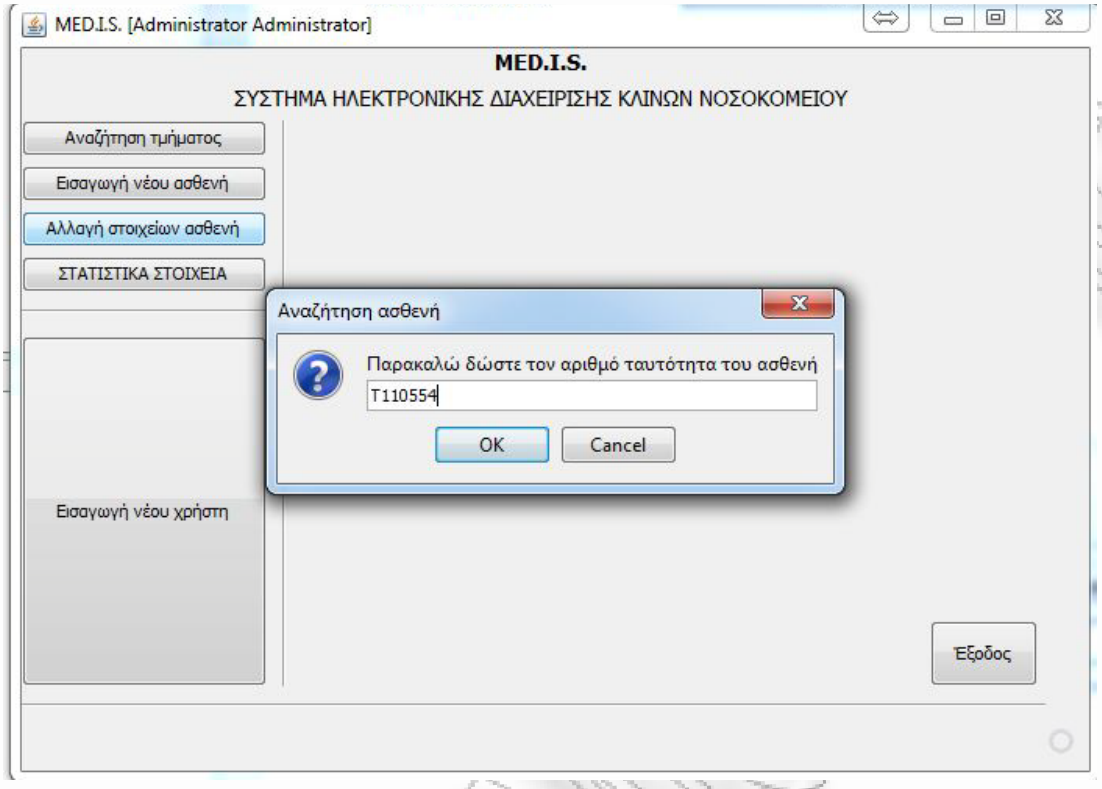
Είσοδος στο σύστημα 4

Βήμα 2 Επιλεγούμε αλλαγή στοιχείων ασθενή στην αρχική μας οθόνη.

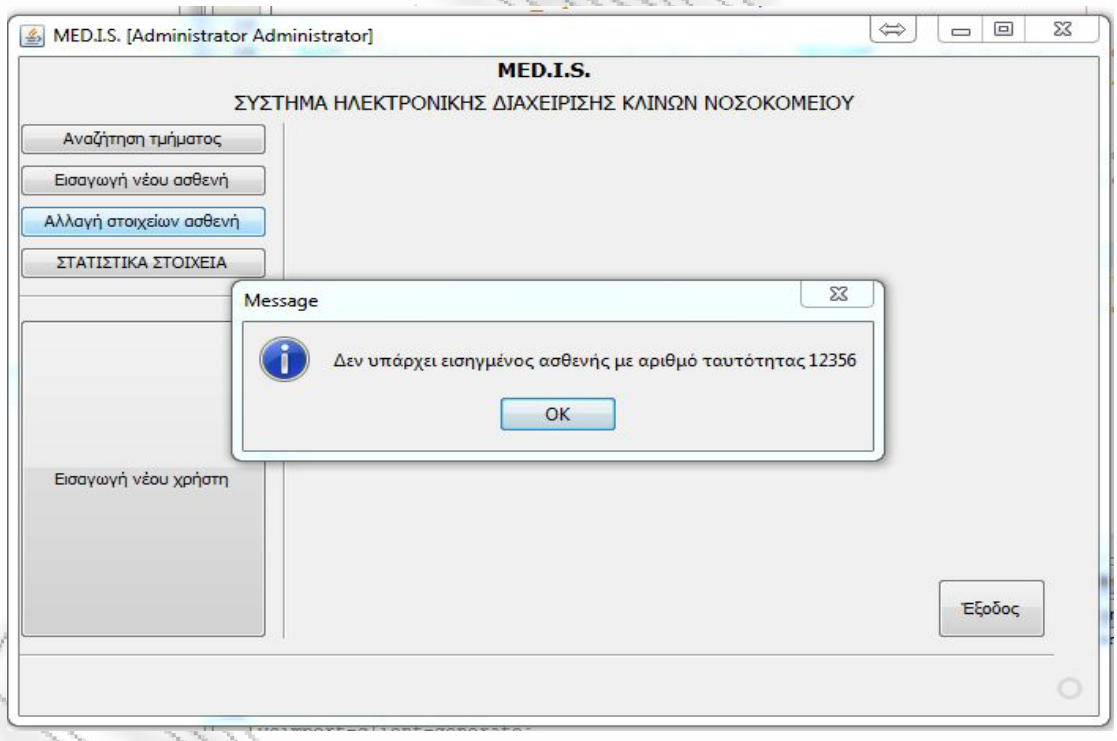


Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τον αριθμό ταυτότητας του ασθενή αν βάλουμε λάθος ταυτότητα μας εμφανίζει μήνυμα λάθους .

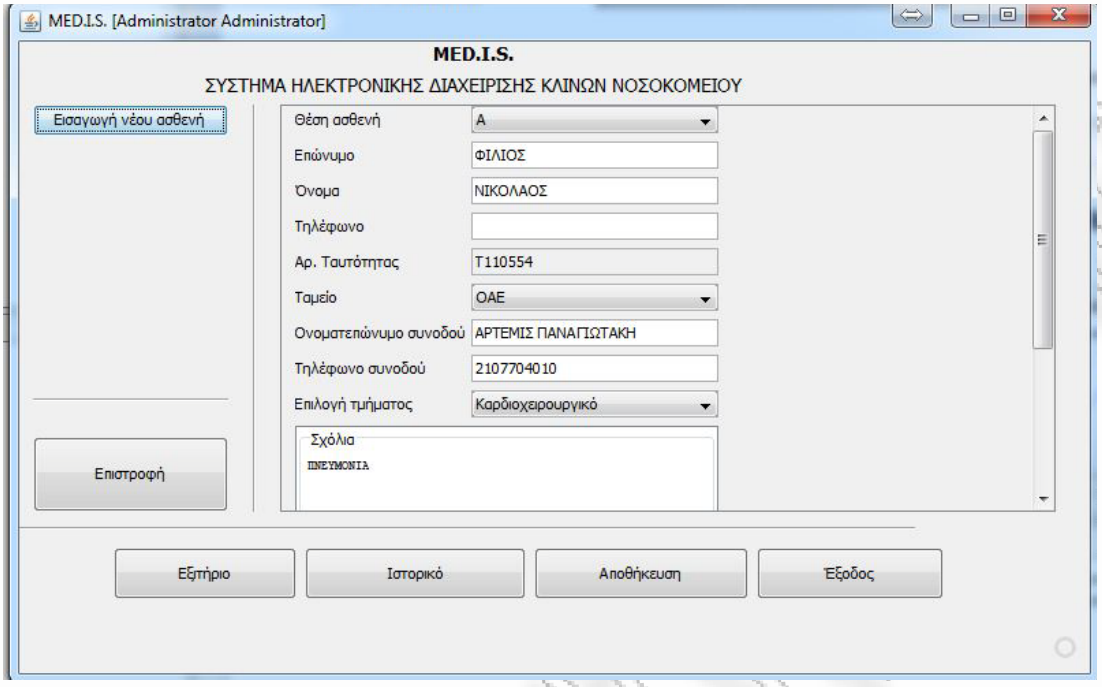


Οθόνη αναζήτησης ασθενή 3



Μήνυμα λάθους ταυτότητας 4

Βήμα 4 Αλλάζουμε τα στοιχειά του ασθενή μας και πατάμε αποθήκευση.



Οθόνη αλλαγής στοιχείων 5

Αναμενόμενο αποτέλεσμα:

Αλλαγή στοιχείων υπάρχοντος ασθενή .

Αποτέλεσμα:

Επιτυχία

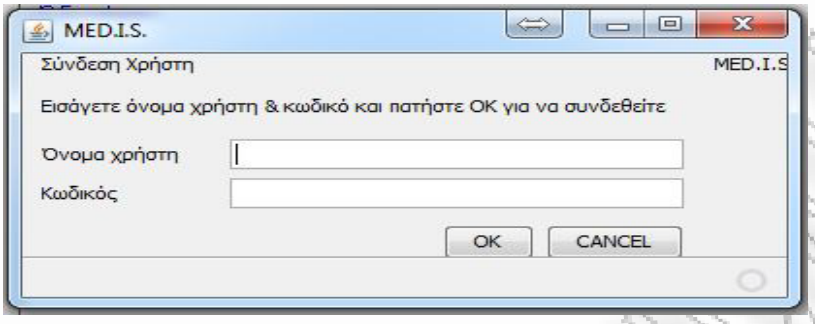
Σχόλια:

Σενάριο 6:

Έκδοση εξιτηρίου για ασθενή.

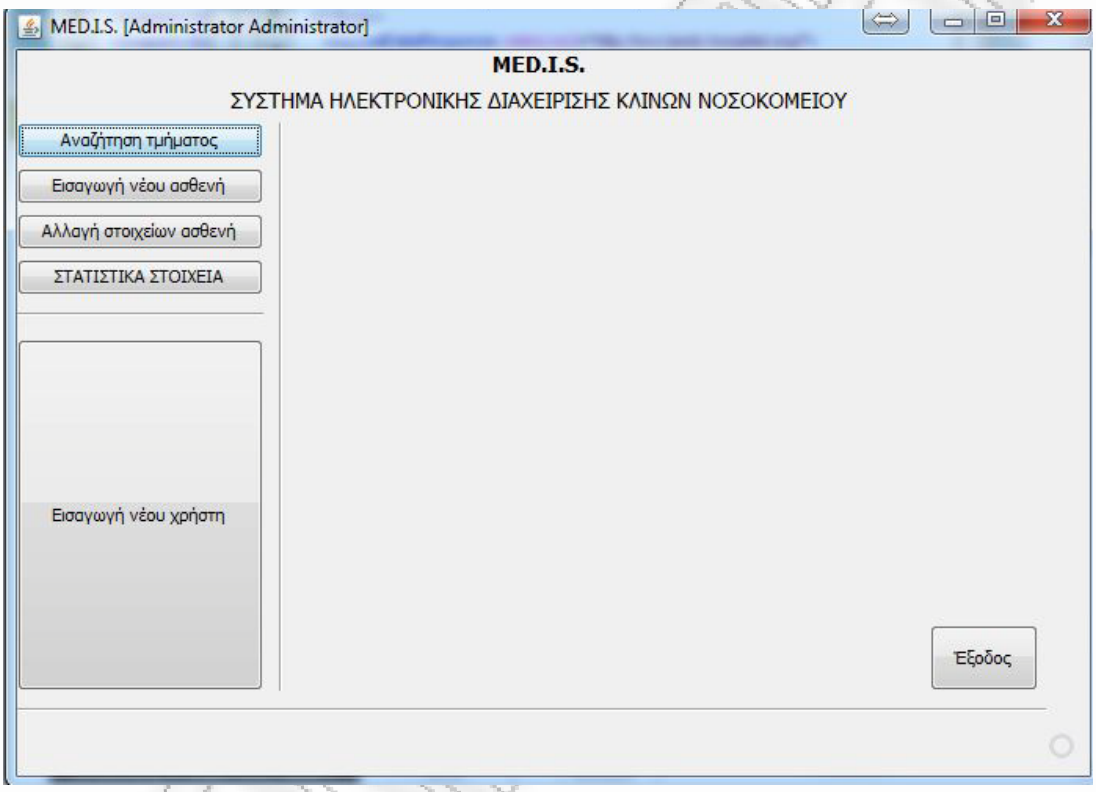
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



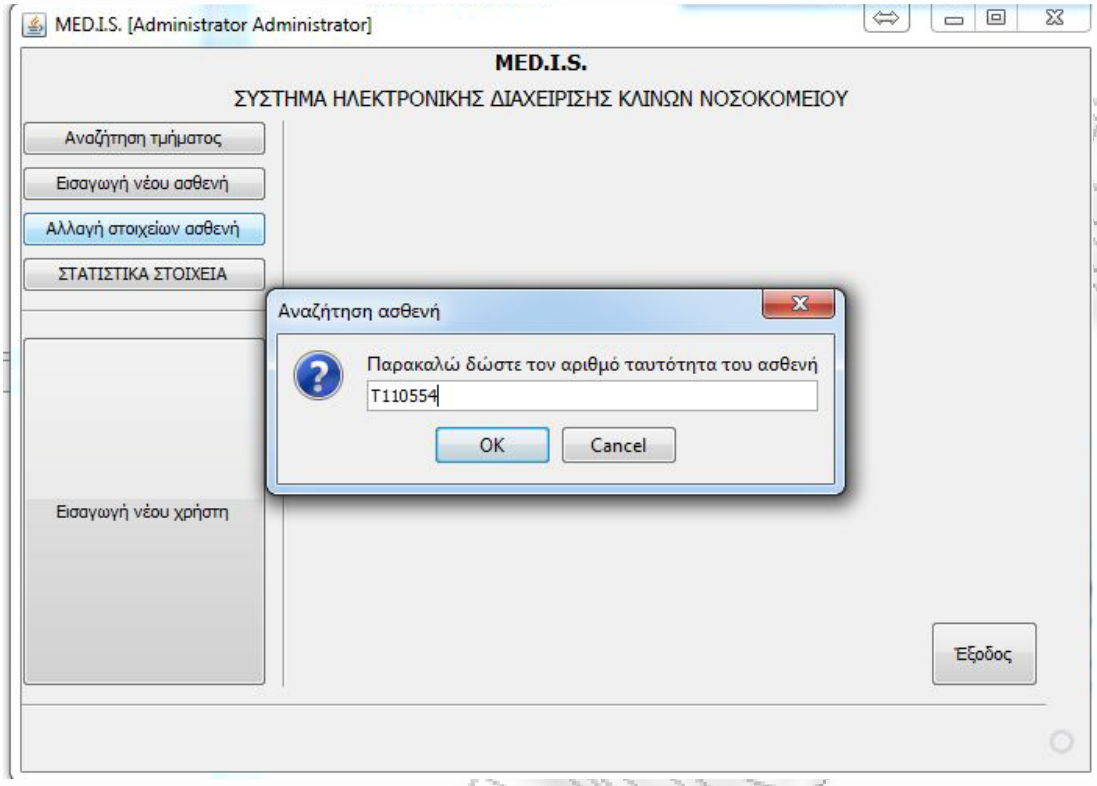
Είσοδος στο σύστημα 5

Βήμα 2 Επιλεγούμε αλλαγή στοιχείων ασθενή στην αρχική μας οθόνη.



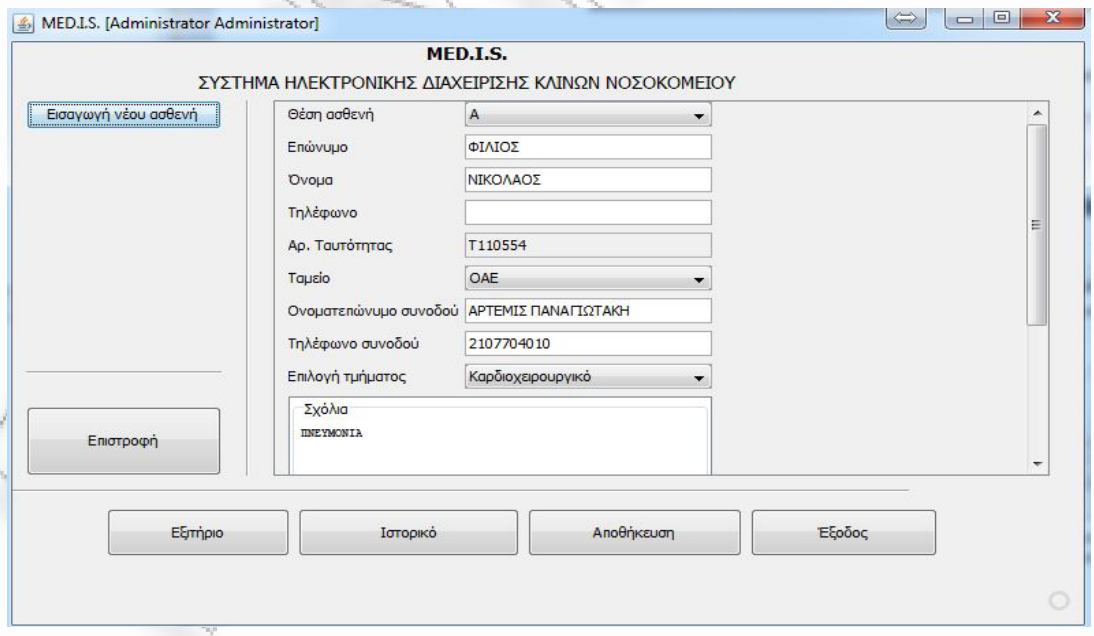
Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τον αριθμό ταυτότητας του ασθενή.



Οθόνη αναζήτησης ασθενή 3

Βήμα 4 Πατάμε το κουμπί εξιτήριο όπου γίνεται και η έξοδος του ασθενή από το σύστημα και επιστρέφουμε στην αρχική μας οθόνη.



Οθόνη αλλαγής στοιχείων 5

Αναμενόμενο αποτέλεσμα:

Έξοδος του ασθενή μας από το σύστημα .

Αποτέλεσμα:

Επιτυχία

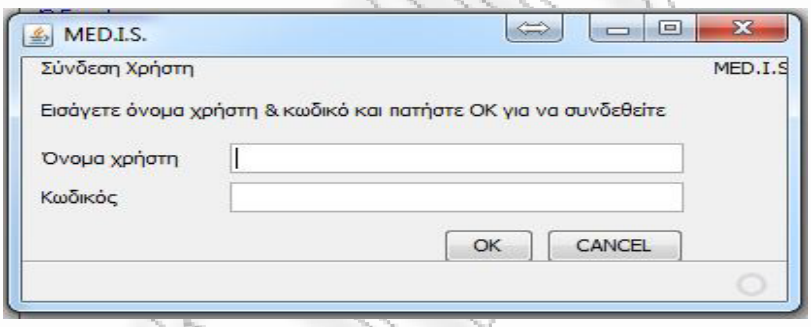
Σχόλια: Μελλοντική έκδοση θα μπορεί να υπάρχει κάποιο πλήκτρο εκτύπωσης για να παίρνει ο ασθενής το εξιτήριο του την ώρα εξόδου από το θάλαμο.

Σενάριο 7:

Μεταφορά ασθενή σε διαφορετικό τμήμα του νοσοκομείου.

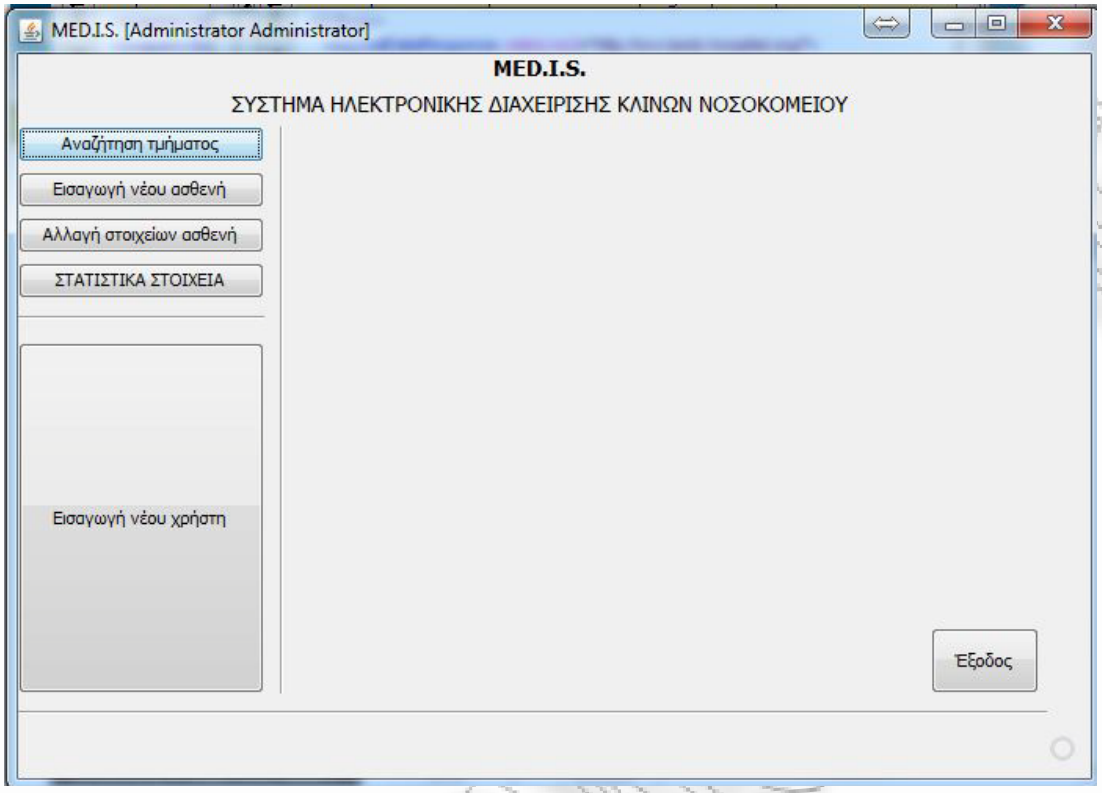
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



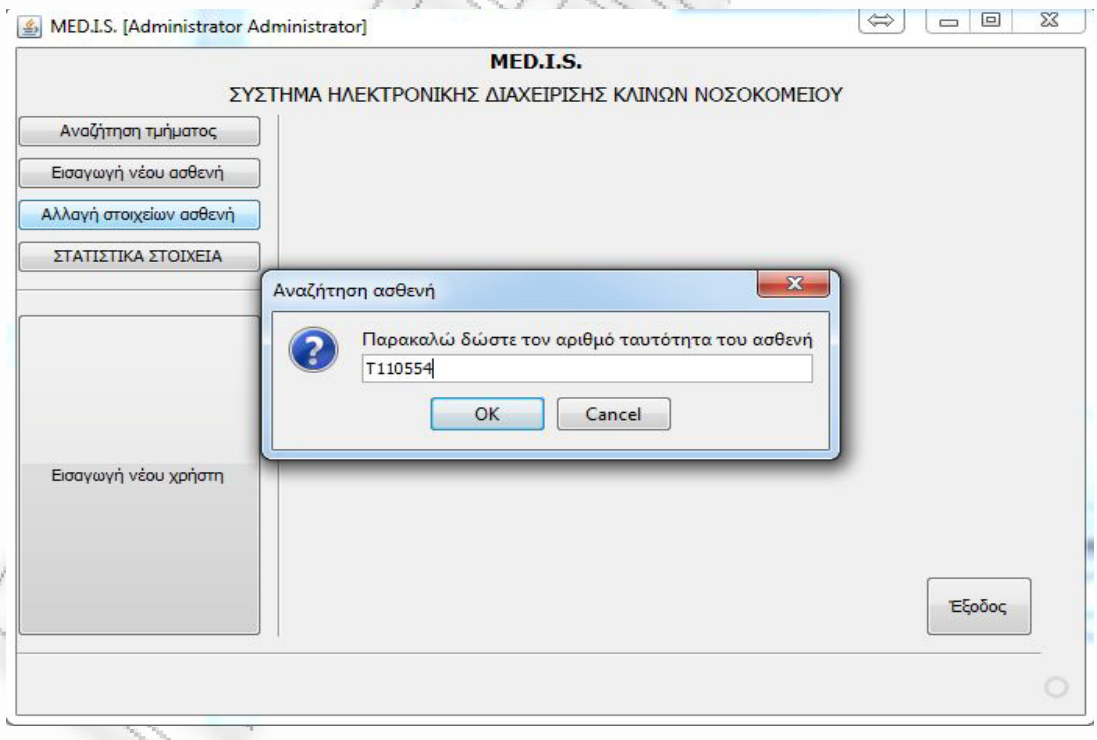
Είσοδος στο σύστημα 6

Βήμα 2 Επιλεγούμε αλλαγή στοιχείων ασθενή στην αρχική μας οθόνη.



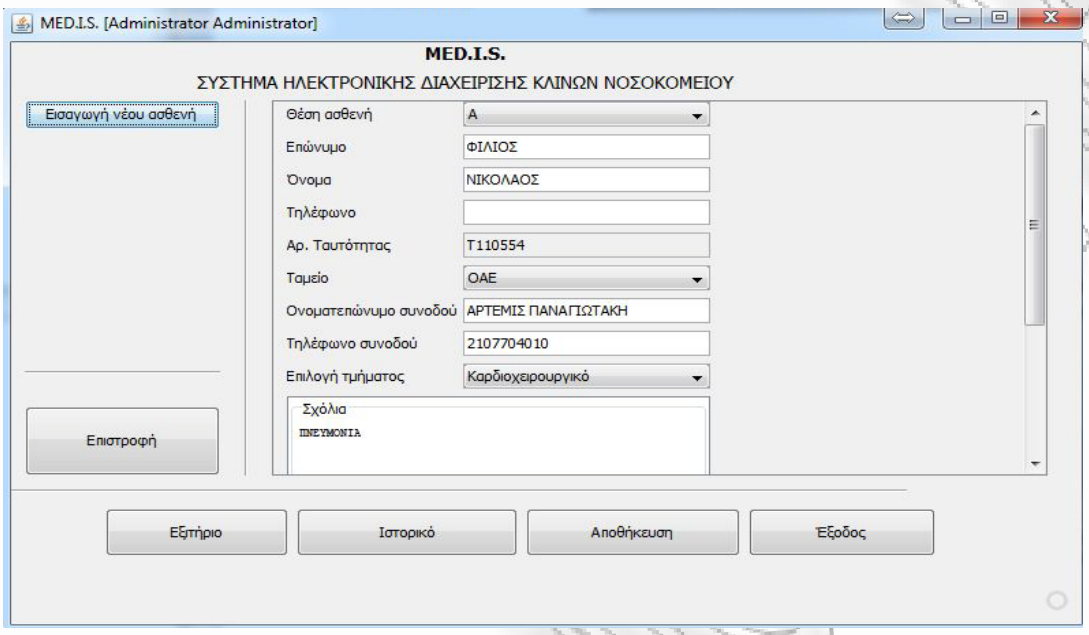
Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τον αριθμό ταυτότητας του ασθενή.

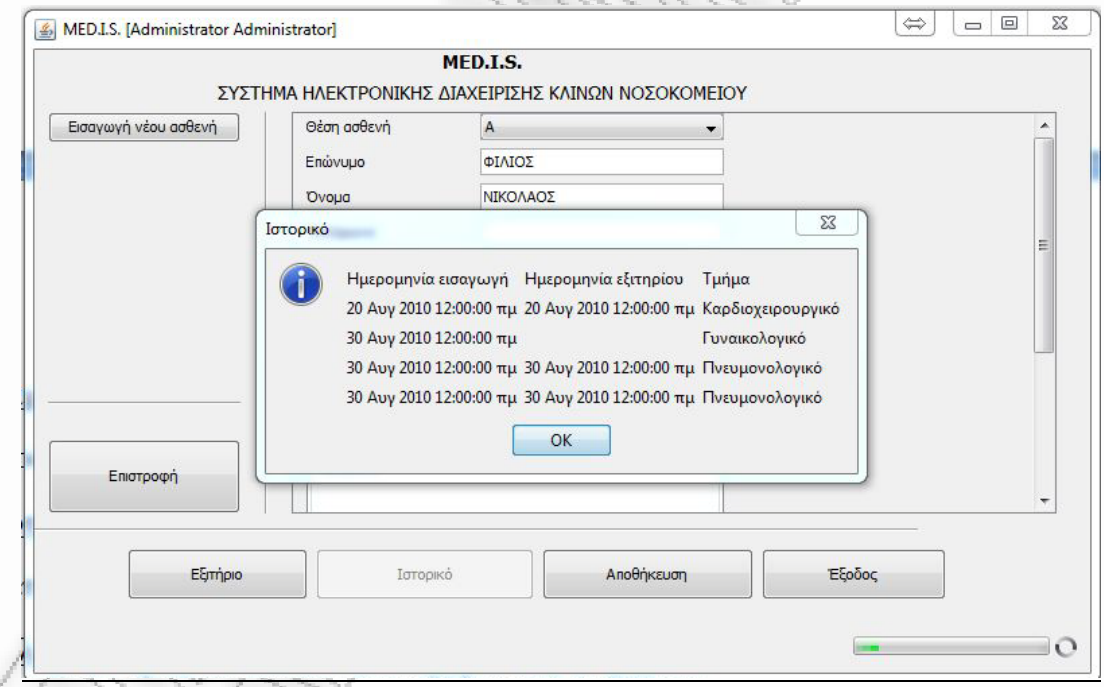


Οθόνη αναζήτησης ασθενή 3

Βήμα 4 Αλλάζουμε τμήμα στον ασθενή και πιέζοντας ιστορικό εισαγωγών παρακολουθούμε ότι έχει γίνει έξοδος από το τμήμα που βρισκόταν και είσοδος στο νέο τμήμα.



Οθόνη αλλαγής στοιχείων 4



Οθόνη ιστορικού ασθενή 5

Αναμενόμενο αποτέλεσμα:

Μεταφορά του ασθενή μας σε άλλο τμήμα .

Αποτέλεσμα:

Επιτυχία

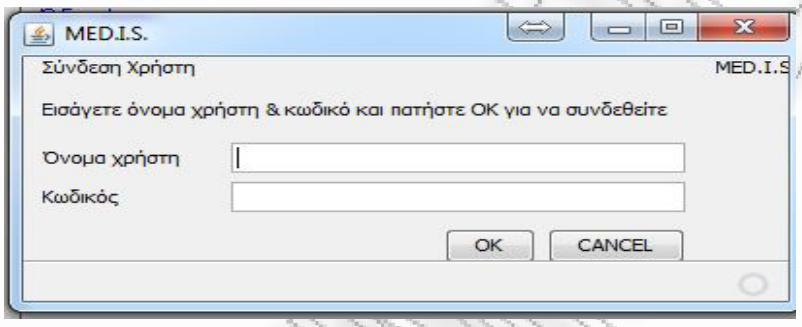
Σχόλια:

Σενάριο 8:

Προβολή ιστορικού εισαγωγών ασθενή.

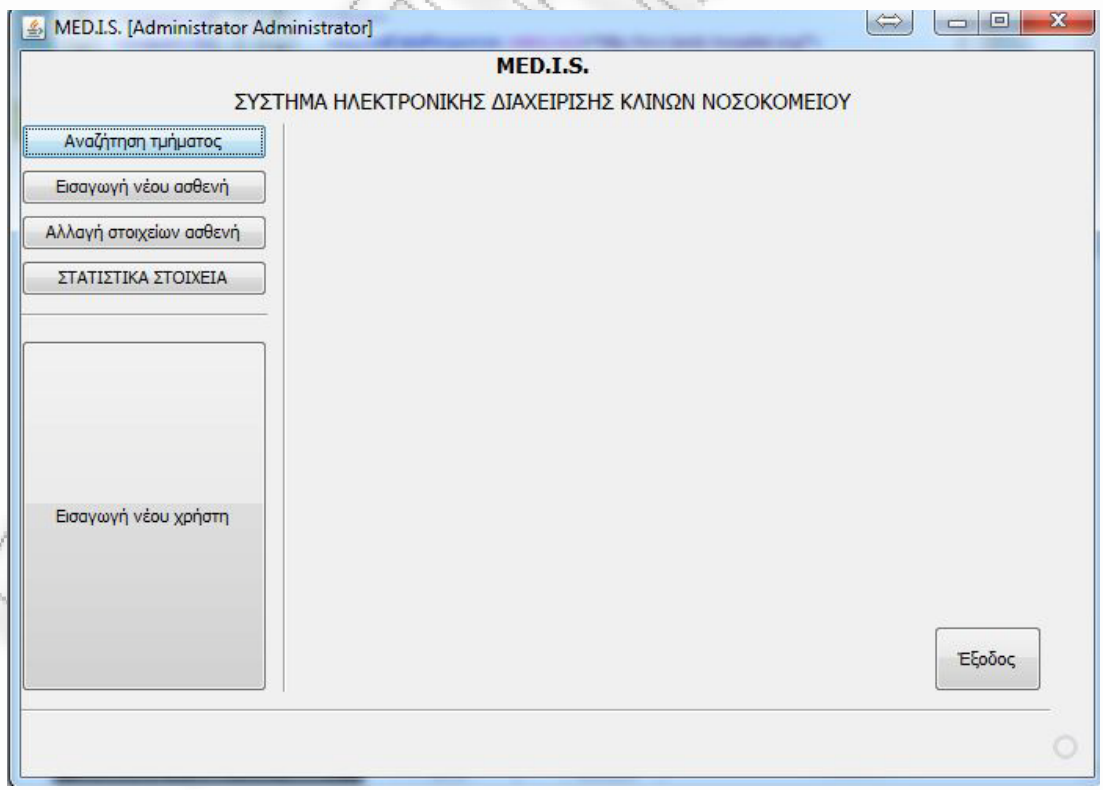
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



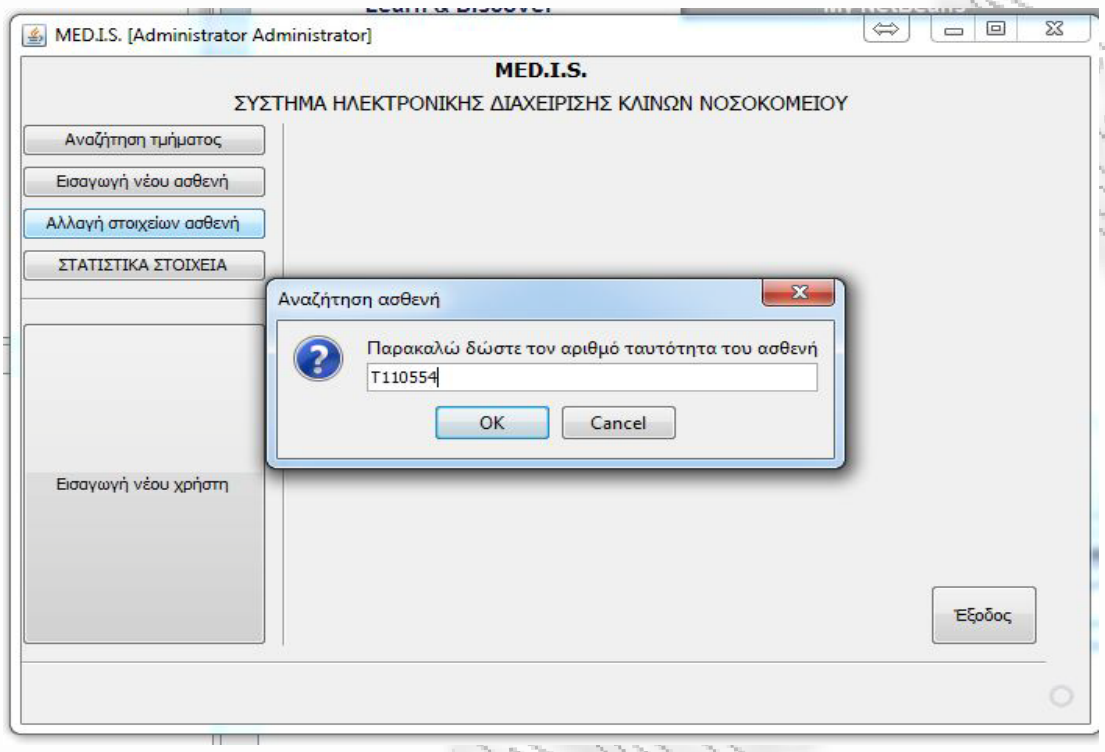
Είσοδος στο σύστημα 7

Βήμα 2 Επιλεγούμε αλλαγή στοιχείων ασθενή στην αρχική μας οθόνη.



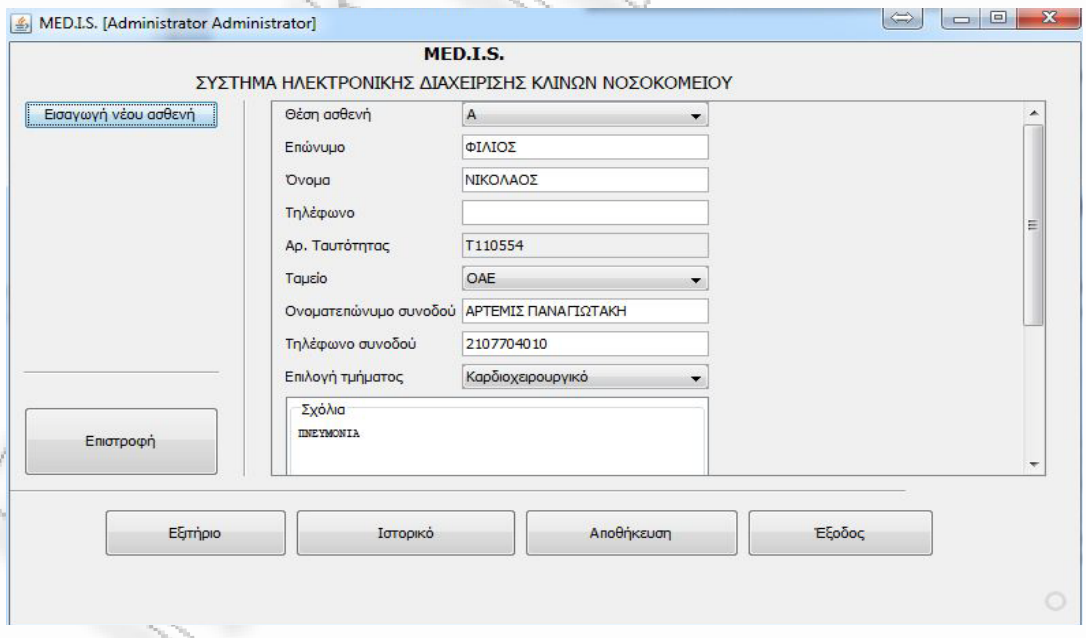
Αρχική οθόνη συστήματος 2

Βήμα 3 Πληκτρολογούμε τον αριθμό ταυτότητας του ασθενή.

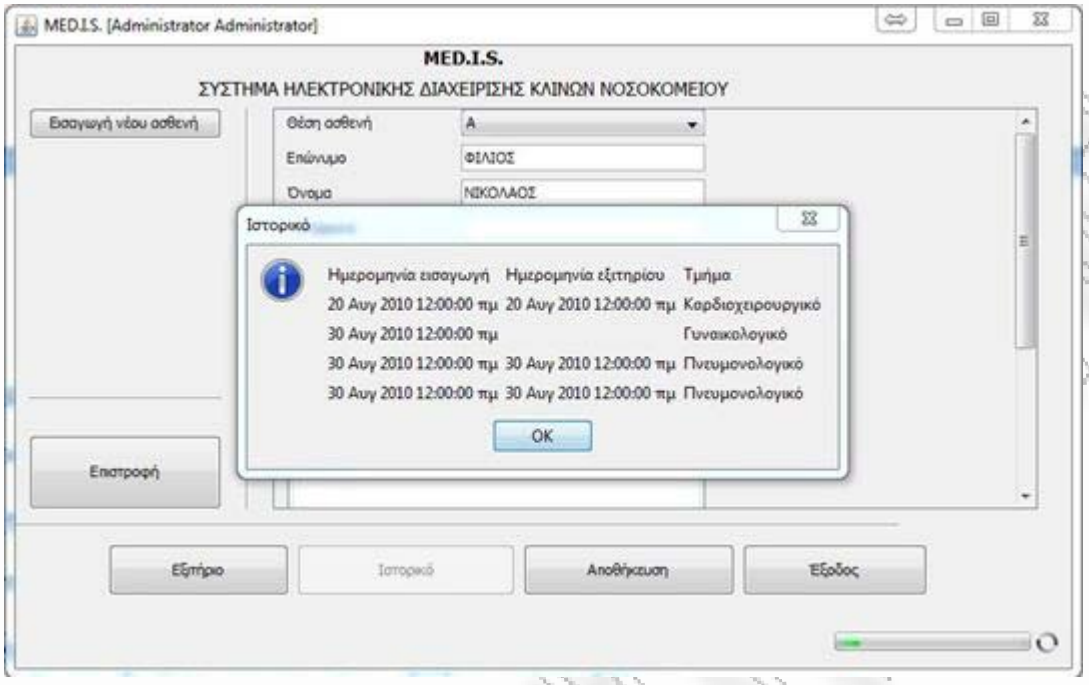


Οθόνη αναζήτησης ασθενή 3

Βήμα 4 Πιέζουμε ιστορικό εισαγωγών για να δούμε το ιστορικό. Αυτό το κουμπί αυτό υπάρχει σε πολλές οθόνες του συστήματος μας.



Οθόνη αλλαγής στοιχείων 4



Οθόνη ιστορικού ασθενή 5

Αναμενόμενο αποτέλεσμα:

Εμφάνιση ιστορικού ασθενή .

Αποτέλεσμα:

Επιτυχία

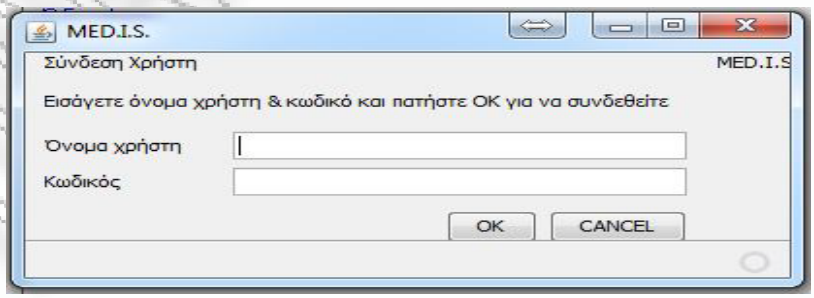
Σχόλια:

Σενάριο 9:

Επισκόπηση πληρότητας κλινών.

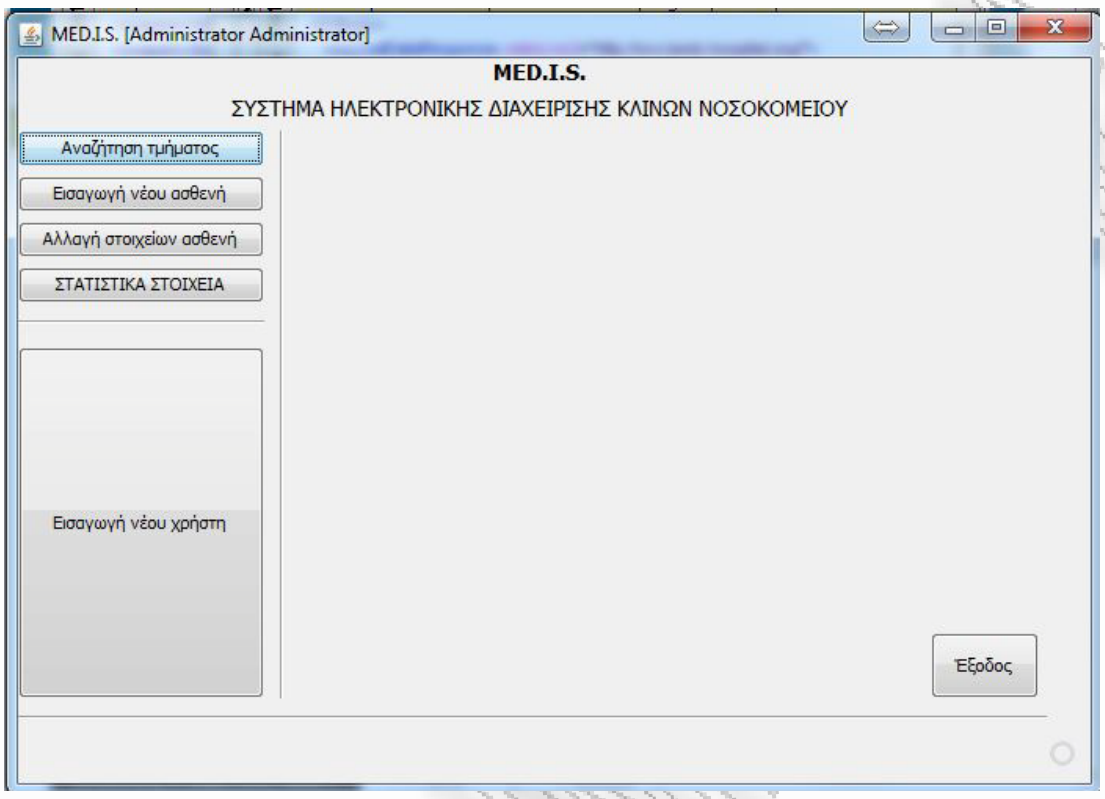
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



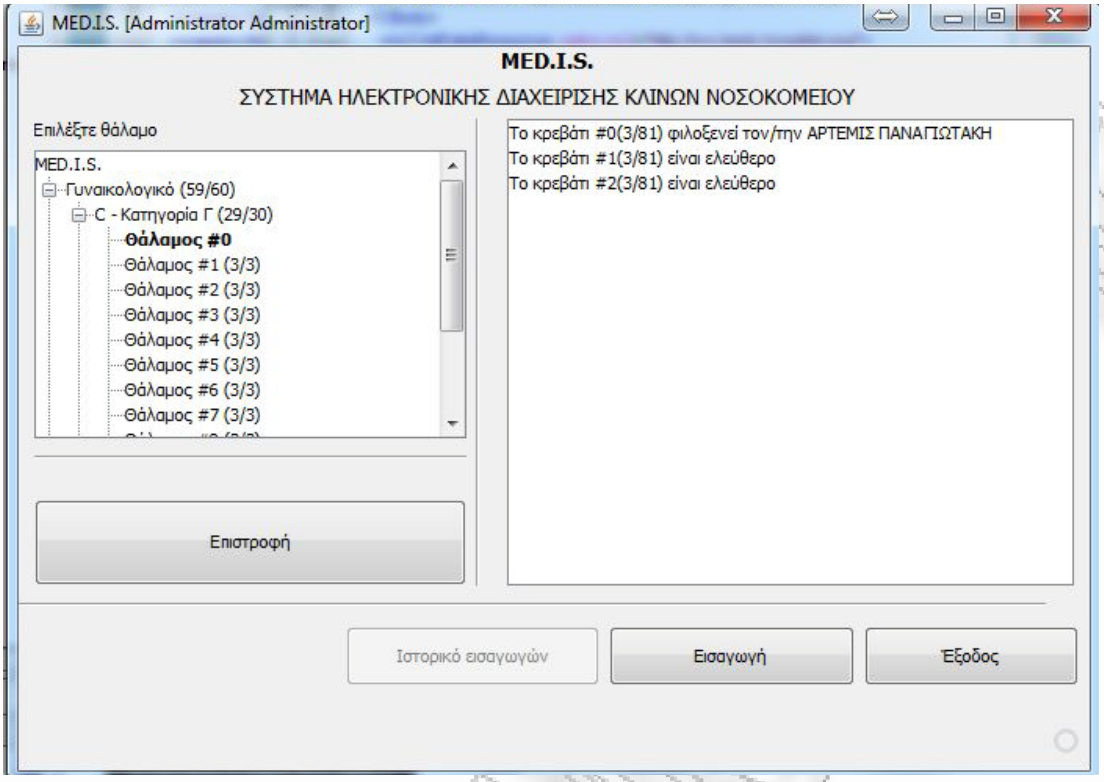
Είσοδος στο σύστημα 8

Βήμα 2 Επιλεγούμε Αναζήτηση τμήματος ασθενή στην αρχική μας οθόνη.



Αρχική οθόνη συστήματος 2

Βήμα 3 Εμφανίζεται η οθόνη που μπορούμε να επιλέξουμε τα τμήματα και μπορούμε να δούμε πόσοι διαθέσιμοι θάλαμοι υπαρχούν και ποιοι ασθενείς είναι εισαγμένοι στις αντίστοιχες θέσεις.



Οθόνη επισκόπησης πληρότητας κλινών 3

Αναμενόμενο αποτέλεσμα:

Επισκόπηση πληρότητας κλινών .

Αποτέλεσμα:

Επιτυχία

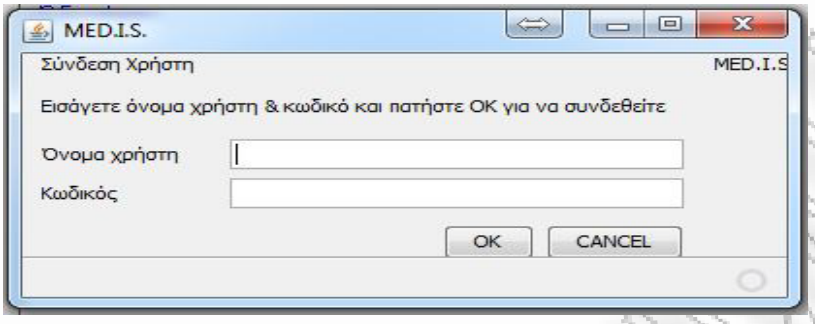
Σχόλια:

Σενάριο 10:

Στατιστικά στοιχεία του συστήματος.

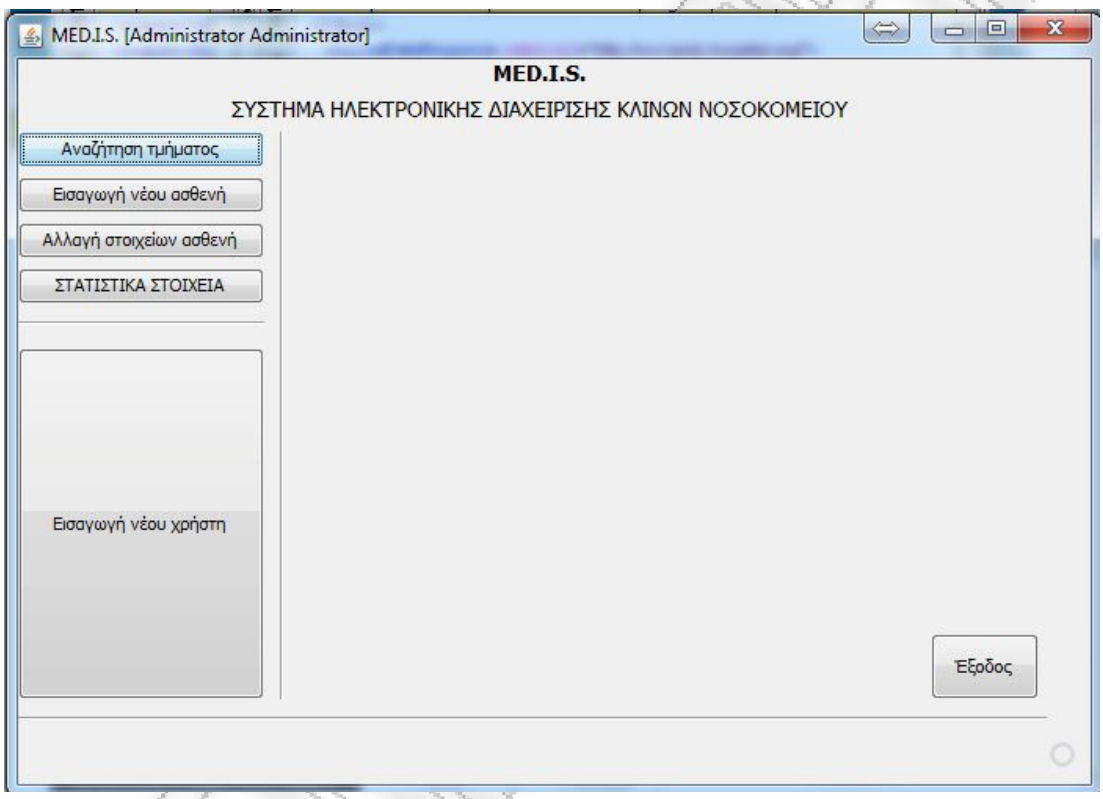
Περιγραφή:

Βήμα 1 Κάνουμε εισαγωγή στο σύστημα χρησιμοποιώντας όνομα χρηστή και κωδικό πρόσβασης.



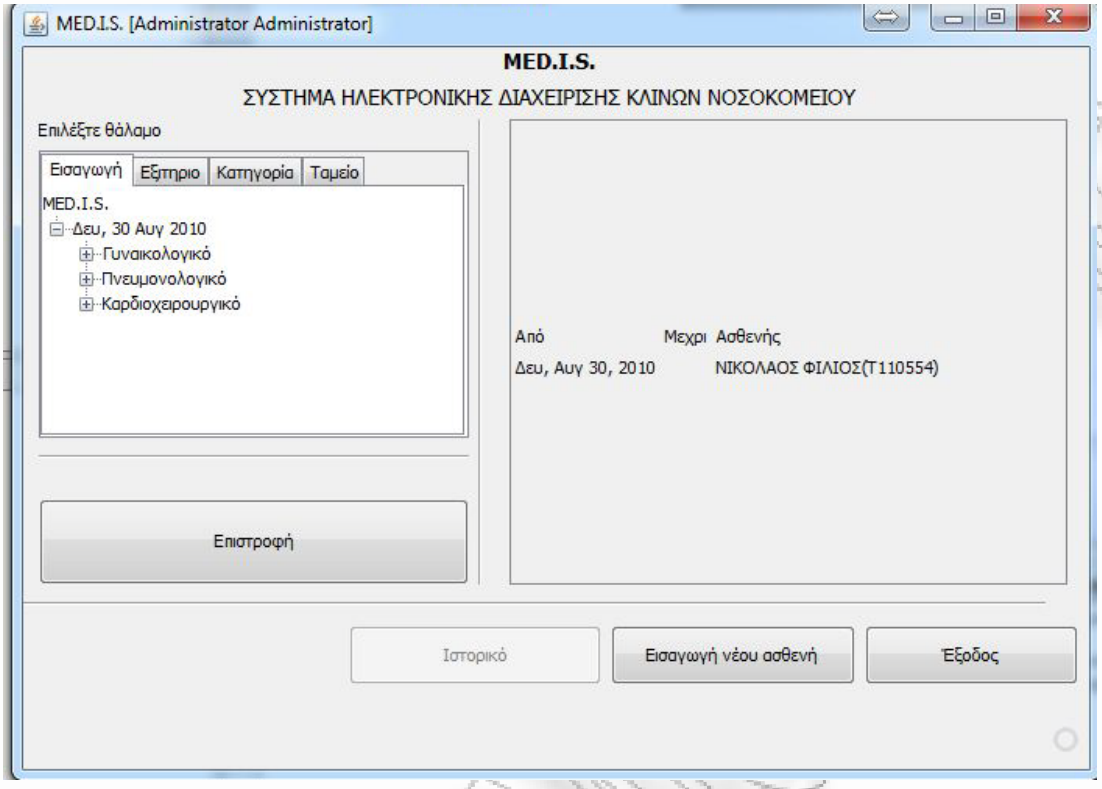
Είσοδος στο σύστημα 9

Βήμα 2 Επιλεγούμε ΣΤΑΤΙΣΤΙΚΑ ΣΤΟΙΧΕΙΑ στην αρχική μας οθόνη.



Αρχική οθόνη συστήματος 2

Βήμα 3 Μπορούμε να βρούμε στατιστικά στοιχεία για τους ασθενείς με βάση το εισιτήριο, το εξιτήριο, τη θέση και το ασφαλιστικό ταμείο.



Οθόνη στατιστικών στοιχείων 3

Αναμενόμενο αποτέλεσμα:

Στατιστικά στοιχεία για τους ασθενείς .

Αποτέλεσμα:

Επιτυχία

Σχόλια: Μελλοντική έκδοση θα μπορούσε να υπάρχει και η επιλογή συγκεντρωτικά στατιστικά στοιχεία.

5.5 ΧΡΗΣΙΜΟΠΟΙΗΘΕΝΤΑ ΕΡΓΑΛΕΙΑ

Για την υλοποίηση του πληροφοριακού συστήματος MED.I.S. έγινε χρήση της γλώσσας προγραμματισμού Java.

Η ανάπτυξη έγινε μέσω του ενοποιημένου περιβάλλοντος ανάπτυξης εφαρμογών (IDE) NetBeans. Το NetBeans είναι ένα πρόγραμμα που αναπτύσσεται από την ιδία την Sun Microsystems και διατίθεται δωρεάν. Είναι από τα κορυφαία του χώρου και παρέχει ιδιαίτερες διευκολύνσεις για την ανάπτυξη τυποποιημένων εφαρμογών. Έχει ενσωματωμένες διάφορες βιβλιοθήκες για ταχύτερη ανάπτυξη γραφικών εφαρμογών, αλλά και γραφικών περιβάλλον για την σχεδίαση τους. Πέραν των διευκολύνσεων για γραφικές εφαρμογές παρέχει και ένα ιδιαίτερα εύχρηστο περιβάλλον για δημιουργία και δοκιμή Web Services, καθώς ενσωματώνει την σουίτα ανάπτυξη web services “Metro”.

Η σουίτα ανάπτυξης web services είναι ένα ακόμα open source προϊόν της Sun Microsystems και αποτελεί μια από τις πρώτες επιλογές στην ανάπτυξη εφαρμογών που βασίζονται σε web services.

Ως σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων (ΣΣΔΒΔ) χρησιμοποιήθηκε το JavaDB. Το JavaDB είναι ένα μοντέρνο ΣΣΔΒΔ που είναι υλοποιημένο εξ' ολοκλήρου σε Java κι αυτό δίνει το πλεονέκτημα της φορητότητας. Επιπλέον χαρακτηριστικά που έχει το JavaDB και οδήγησαν στην επιλογή του, είναι ότι είναι openSource, δεν απαιτεί ειδική εγκατάσταση και είναι προϊόν της Sun Microsystems.

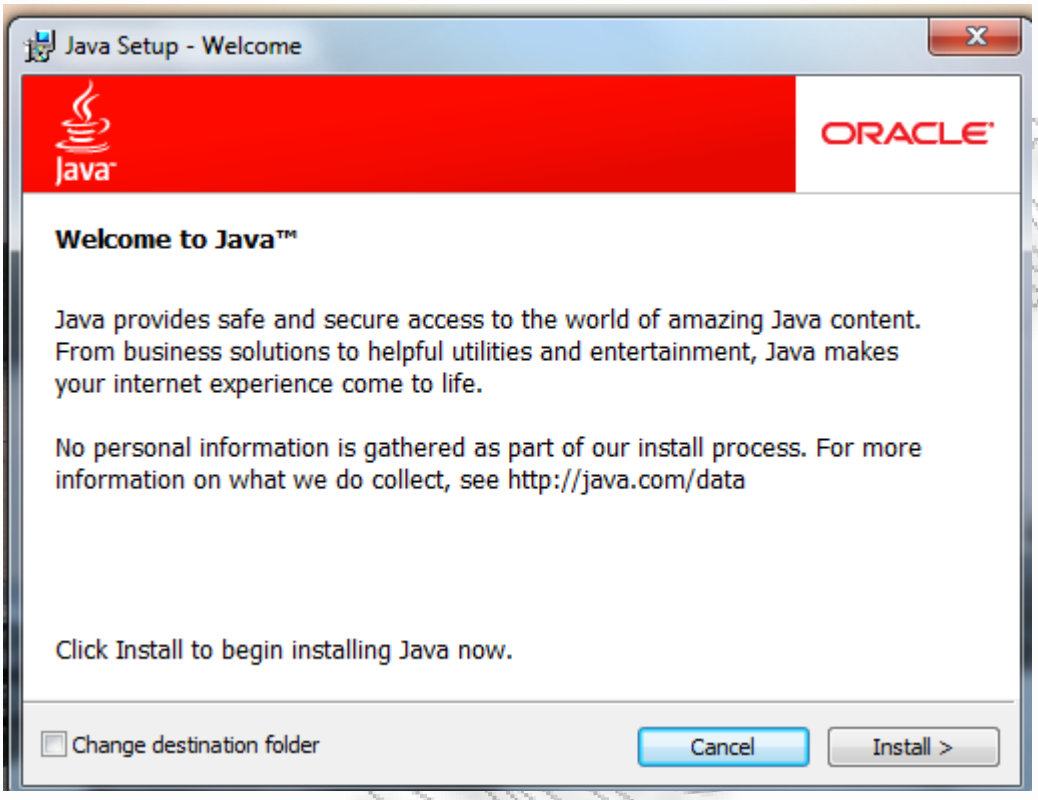
Για ακόμα ευκολότερη πρόσβαση στην βάση δεδομένων έγινε χρήση του προτύπου JPA (Java Persistent API) και συγκεκριμένα της υλοποίηση που προσφέρει η jBoss και ονομάζεται Hibernate. Το Hibernate είναι μια από τις πρώτες επιλογές στον χώρο της αντικείμενο-σχεσιακή σύνδεσης. Η χρήση ενός τέτοιου επιπέδου απαλλάσσει τον προγραμματιστή από την ανάγκη να σκέφτετε σχεσιακά και του δίνει την ελευθερία και τις διευκολύνσεις που προσφέρει ο αντικειμενοστραφής προγραμματισμός. Ουσιαστικά αποκρύπτει από τον προγραμματιστή της εφαρμογής οτιδήποτε έχει να κάνει με SQL. Φυσικά αυτή η «απόκρυψη» επιτυγχάνεται μέσω αυτόματης μετάφρασης των αντικειμενοστραφών εννοιών σε σχεσιακές. Αυτός ο αυτοματισμός μερικές φορές μπορεί να οδηγήσει σε δυσάρεστα αποτελέσματα για αυτό και η εφαρμογή πρέπει πάντοτε να ελέγχεται ότι λειτουργεί με τον αναμενόμενο τρόπο.

Να σημειωθεί ότι έχει γίνει χρήση και της τεχνολογίας Spring για συγγραφή πιο ευανάγνωστου κώδικα στην μεριά του web services server.

5.6 ΕΓΚΑΤΑΣΤΑΣΗ ΕΦΑΡΜΟΓΗΣ

1) Εγκατάσταση JRE 1.6

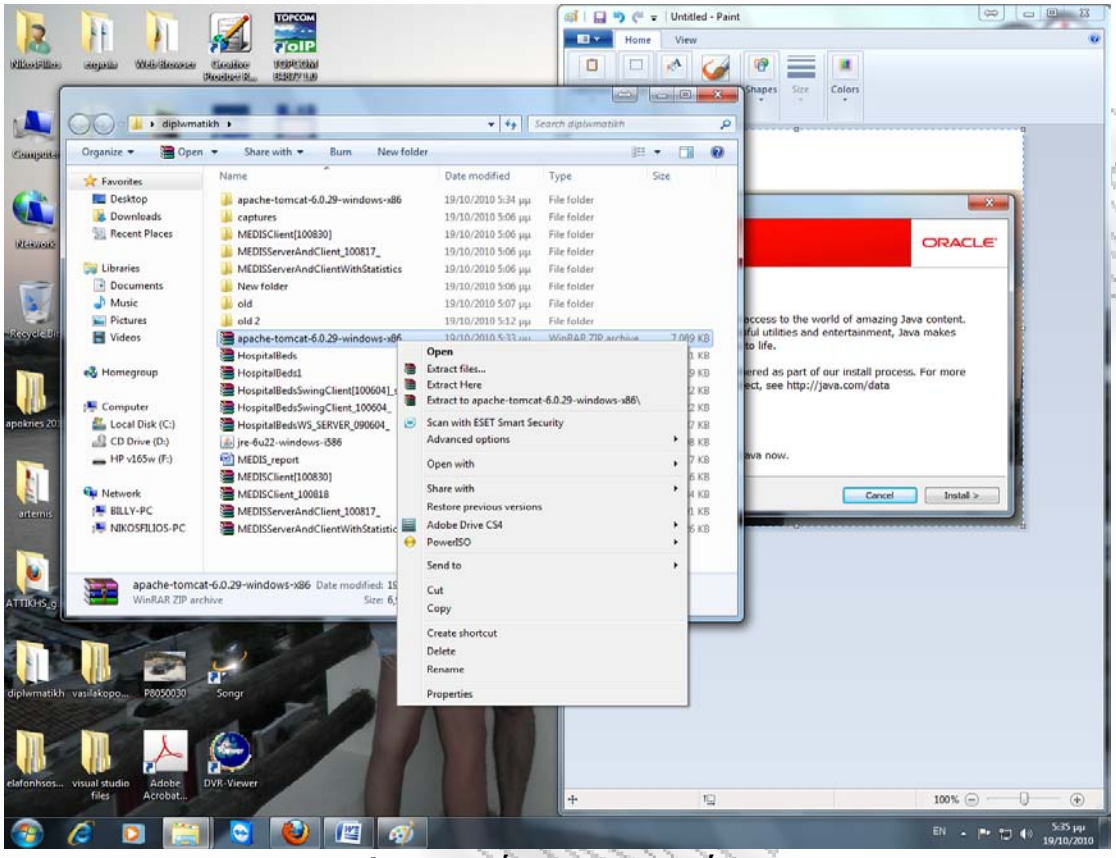
Κάνουμε εγκατάσταση την Java SE Runtime Environment 6u22 την οποία τη βρισκούμε στην ιστοσελίδα www.oracle.com/technetwork/java/javase/downloads/index.html Όπως φαίνεται στη παρακάτω εικόνα.



Εγκατάσταση JRE 1.6

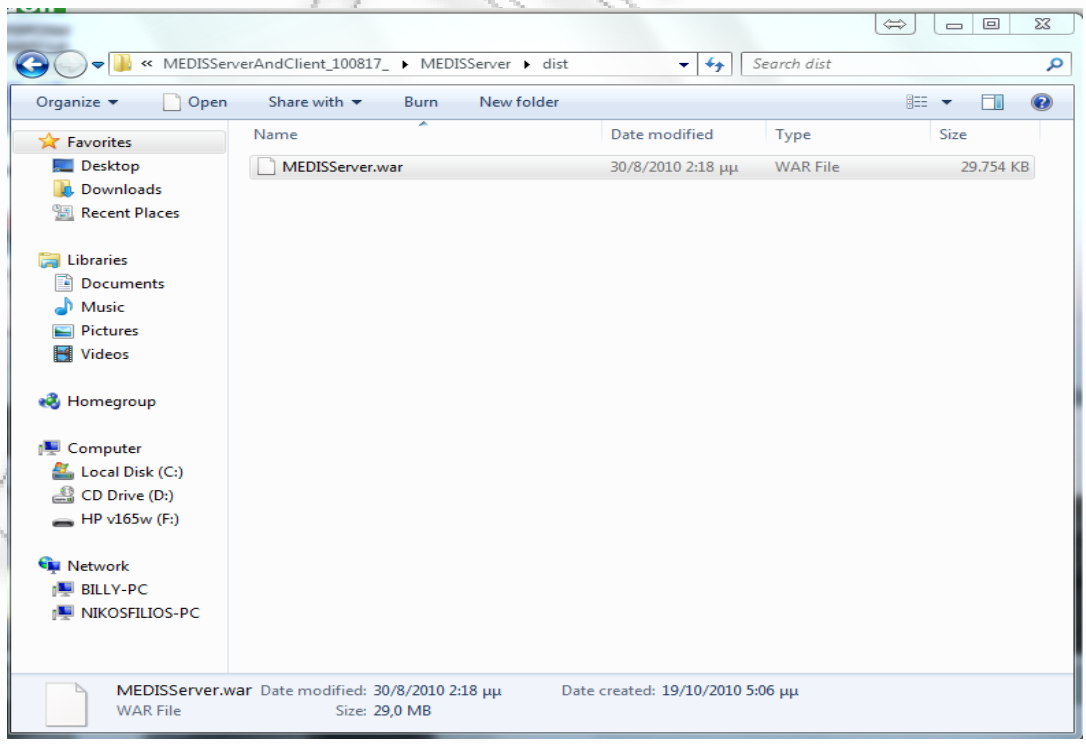
2) Εγκατάσταση Apache Tomcat 6

Κάνουμε αποσυμπίεση το αρχείο το apache-tomcat-6.0.29-windows-x86 οποίο βρίσκουμε στην ιστοσελίδα http://tomcat.apache.org/download-60.cgi



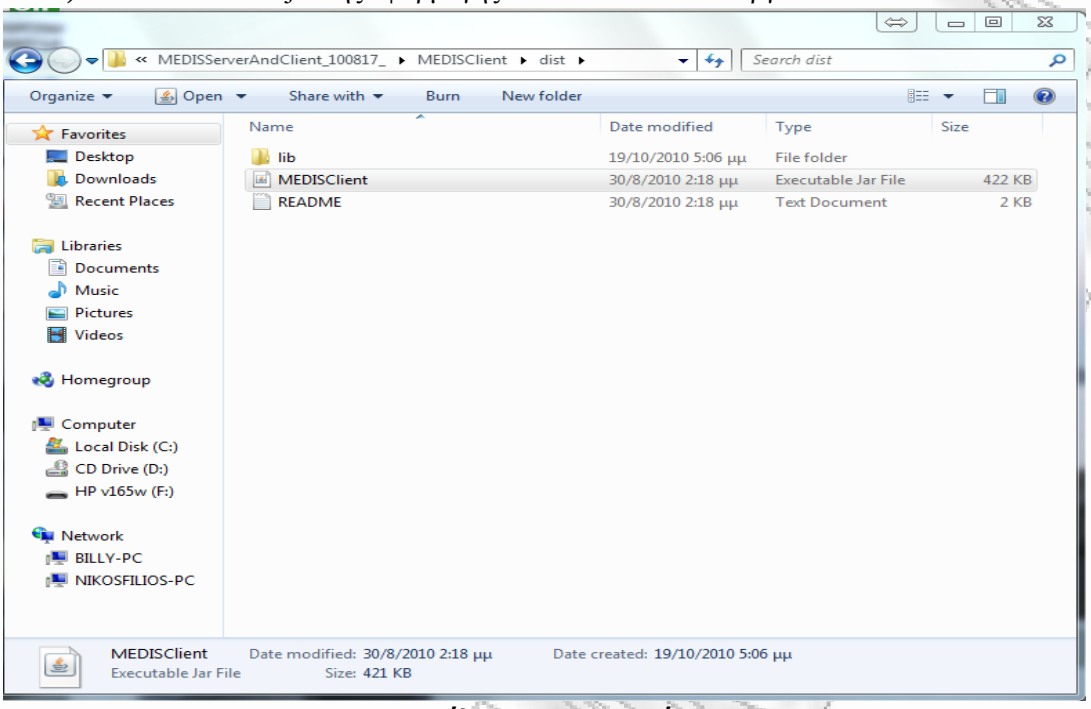
Αποσυμπίεση του αρχείου

3) Τοποθέτηση του war της εφαρμογής στον φάκελο webapps του Tomcat. (παράγεται στον υποφάκελλο dist του project με τα web services).



Αντιγραφή επικόλληση του αρχείου

4) Εκκίνηση του tomcat.
5) Διπλό κλικ στο jar της εφαρμογής του client και λειτουργεί.



Τρέξιμο του αρχείου

ΣΥΜΠΕΡΑΣΜΑΤΑ

Η διαχείριση της γνώσης, η συνεχής μάθηση και η αναβάθμιση των δεξιοτήτων του ανθρώπινου δυναμικού, καθώς και η προσαρμογή σε νέες μεθόδους εργασίας και η μεταβολή των εσωτερικών διαδικασιών στις δημόσιες υπηρεσίες και κατ' επέκταση στα νοσοκομεία αποτελούν τα βασικότερα στοιχεία για την πιο αποδοτική λειτουργία μιας σύγχρονης δημόσιας διοίκησης.

Η αξιοποίηση των τεχνολογιών πληροφορικής και επικοινωνιών δίνει τη δυνατότητα για μια αποτελεσματική και ορθολογική διοίκηση καθώς το ιατρικό και παράλληλα και όλο το υπαλληλικό προσωπικό των νοσοκομείων να έχουν ευκολότερη πρόσβαση στις πληροφορίες και στις υπηρεσίες, με αποτέλεσμα την καλύτερη εξυπηρέτηση των πολιτών και το μειωμένο διοικητικό κόστος.

Ωστόσο, πρέπει να πούμε ότι απλά η χρησιμοποίηση των ΟΠΣ δεν είναι πανάκεια. Η επιτυχημένη εφαρμογή μιας ηλεκτρονικής υπηρεσίας, στο πλαίσιο της ηλεκτρονικής διακυβέρνησης, απαιτεί σωστό σχεδιασμό ο οποίος θα πρέπει να βασίζεται στις απαιτήσεις των χρηστών, να λαμβάνει υπόψη του τις πραγματοποιούμενες αλλαγές στο οργανωτικό επίπεδο και να εστιάζεται στα κόστη και στη βελτίωση της απόδοσης.

Στην Ελλάδα είναι γεγονός ότι τα τελευταία χρόνια έχουν γίνει σημαντικά βήματα για τον τεχνολογικό εκσυγχρονισμό της, αν και με κάποια καθυστέρηση. Με αργό αλλά σταθερό ρυθμό γίνεται προσπάθεια να φτάσει τις σύγχρονες, τεχνολογικά αναπτυγμένες χώρες, όπου πληθώρα ηλεκτρονικών υπηρεσιών διευκολύνει την καθημερινότητα των πολιτών τους, και είναι πηγή ανάπτυξης και ευημερίας. Με δεδομένη την παγκοσμιοποίηση της οικονομίας, η τεχνολογική ανάπτυξη θα πρέπει να θεωρείται μονόδρομος για την παρούσα και πολύ περισσότερο την μελλοντική οικονομική ευημερία της χώρας. Η Ελλάδα έχει την ευκαιρία αλλά και την υποχρέωση να αναλάβει ενεργό ρόλο στην λειτουργία της παγκόσμιας αγοράς, χρησιμοποιώντας την τεχνολογία ως μοχλό για την ανάπτυξη της.

Σημαντικό ρόλο για την επιτυχία των προωθούμενων αλλαγών αναμένεται να διαδραματίσει η ολοκλήρωση του σύγχρονου τρόπου ζωής στην συμπεριφορά και την κουλτούρα του ελληνικού λαού. Με άλλα λόγια, είναι απαραίτητο να γίνει κατανοητό από η χρησιμότητα των παρεχόμενων ηλεκτρονικών υπηρεσιών, και πόσο πολύ μπορούν αυτές να διευκολύνουν την καθημερινότητα και τον τρόπο εργασίας. Θα πρέπει, η επόμενη γενιά επαγγελματιών υγείας να είναι αρκετά ενημερωμένη για τα πολλαπλά οφέλη που θα προκύψουν από την εισαγωγή των ηλεκτρονικών υπολογιστών ,των ολοκληρωμένων πληροφορικών συστημάτων στην υγεία τόσο για τους πολίτες αυτής της χώρας όσο και για τους ίδιους, και εκπαιδευμένη στην χρήση των νέων τεχνολογιών Ωστόσο, στην πορεία προς τεχνολογική ανάπτυξη, ιδιαίτερη προσοχή θα πρέπει να δοθεί στην διατήρηση και ανάδειξη των ιδιαίτερων χαρακτηριστικών της Ελλάδος, αποφεύγοντας την εύκολη αποδοχή ξενόφερτων τάσεων για χάρη του εκσυγχρονισμού

ΙΔΕΕΣ ΓΙΑ ΜΕΛΛΟΝΤΙΚΗ ΕΞΕΛΙΞΗ ΤΟΥ ΣΥΣΤΗΜΑΤΟΣ

Η εφαρμογή που υλοποιήθηκε περιορίζεται στο αντικείμενο της εισαγωγής και εξαγωγής καθώς επίσης και της μεταφοράς ασθενών από το ένα ιατρικό τμήμα στο άλλο. Οι μελλοντικές προοπτικές μίας τέτοιας εφαρμογής περιλαμβάνουν να γίνει ένα ολοκληρωμένο σύστημα διαχείρισης ασθενών.

Δηλαδή να εξελιχθεί η εφαρμογή σε εφαρμογή γραφείου κινήσεως.

Γραφείο Κίνησης – Εσωτερικοί Ασθενείς (Κοινό Υποσύνολο με το διοικητικό –οικονομικό υποσύστημα σε ορισμένες περιπτώσεις). Σκοπός της εφαρμογής είναι η διαχείριση και παρακολούθηση της πορείας του νοσηλευόμενου ασθενή από την εισαγωγή έως και την έκδοση του εξιτηρίου. Περιλαμβάνει την καταγραφή των δημογραφικών και ασφαλιστικών στοιχείων ασθενή κατά την προσέλευση του στο νοσοκομείο καθώς και τη στατιστική επεξεργασία των δεδομένων του ασθενή για επιθυμητές χρονικές περιόδους από τη διοίκηση αλλά και τα τμήματα - κλινικές του νοσοκομείου. Συλλειτουργεί απόλυτα με το υποσύστημα της διαχείρισης ραντεβού και συνδέεται με τη λίστα αναμονής ασθενών προκείμενου να γίνεται ο σωστός προγραμματισμός των πόρων του νοσοκομείου. Μεταξύ των βασικών χαρακτηριστικών είναι η παρακολούθηση της τακτικής και έκτακτης εισαγωγής ασθενή, η διαχείριση των επειγόντων περιστατικών που εισάγονται για νοσηλεία και η παρακολούθηση των εισιτηρίων του νοσοκομείου. Θα πρέπει επίσης να έχει δυνατότητα μελλοντικής υποστήριξης κάρτας υγείας (π.χ. με χρήση bar code, smart card) για την γρήγορη και χωρίς σφάλματα εισαγωγή του ασθενή. Επίσης θα πρέπει να διαχειρίζεται την έκδοση των εξιτηρίων καθώς και των πάσης φύσεως πιστοποιητικών (για ασφαλιστικά ταμεία, βεβαιώσεις εισαγωγής, εξαγωγής κλπ). Επίσης η εφαρμογή θα δίνει σαφή εικόνα της πληρότητας του νοσοκομείου (ανά θέση / όροφο / κλινική) ώστε να γίνεται ο σωστός προγραμματισμός των εισαγωγών ασθενών. Τέλος, η εφαρμογή του γραφείου κίνησης, πρέπει να παρέχει δυνατότητες διαχείρισης κλινών και θέσεων νοσηλείας με εύκολο και γραφικής απεικόνισης τρόπο.

Θα μπορούσε βέβαια να εξελιχθεί και σε ένα ΟΠΣΝ (Ολοκληρωμένο πληροφοριακό σύστημα νοσοκομείου) που περιγράφεται στο Κεφαλαίο 2 της εργασίας.

ΠΑΡΑΡΤΗΜΑ

1. ΓΛΩΣΣΑ XML

Η eXtensible Markup Language (XML) είναι μια γλώσσα ανεξάρτητη από σύστημα και υλικό για την αναπαράσταση δεδομένων και της μορφής τους σε ένα έγγραφο XML (XML document). Ένα έγγραφο XML στην πιο απλή του μορφή είναι ένα αρχείο κειμένου το οποίο περιέχει δεδομένα μαζί με σήμανση η οποία καθορίζει τη δομή των δεδομένων

Η XML είναι μια παγκοσμίως συμφωνημένη μεταγλώσσα σήμανσης που χρησιμοποιείται πρώτιστα για την ανταλλαγή πληροφοριών. Η ομορφιά της XML βρίσκεται στο γεγονός ότι είναι επεκτάσιμη. Απλά, η XML είναι ένα σύνολο προκαθορισμένων κανόνων (συντακτικό πλαίσιο) που πρέπει να ακολουθήσουμε κατά τη δόμηση των δεδομένων μας.

Για ένα μεγάλο χρονικό διάστημα, οι προγραμματιστές και οι προμηθετές εφαρμογών κατασκεύαζαν εφαρμογές και συστήματα εγκατεστημένα σε μια επιχείρηση τα οποία επεξεργάζονταν δεδομένα τα οποία μπορούσαν να με το δικό τους ιδιωτικό τρόπο. Αλλά καθώς η ανταλλαγή πληροφορίας μεταξύ εφαρμογών και συστημάτων στις επιχειρήσεις επικρατούσε, έγινε πολύ δύσκολο να αναταλλάσεις δεδομένα διότι τα συστήματα δε σχεδιάστηκαν ώστε να δέχονται δεδομένα από εξωτερικά, άγνωστα συστήματα.

Η XML παρέχει μία πρότυπη και κοινή δομή για τη διανομή δεδομένων μεταξύ ανόμοιων συστημάτων. Επιπλέον, η XML έχει ενσωματωμένο ένα μηχανισμό επικύρωσης δεδομένων, ο οποίος εγγυάται ότι η δομή των δεδομένων που λαμβάνεται είναι έγκυρη [5].

1.1 Παράδειγμα αναπαράστασης δεδομένων με XML

Ας δούμε πώς αναπαριστούμε δεδομένα με τη βοήθεια της XML :

```
<employee>
    <shift id= "counter" time="8-12">
        <phone id = "1"> All phone information
            <number>3444333</number >
        </phone>
    </shift >
    <shift id="help_desk" time="1-5">
        <phone id = "2"> All phone information
                <number>332333</number >
        </phone>
    </shift >
    ...
    <home-address>
            <street>3434 Norwalk street</street>
            <city>New York</city>
            <state>NY</state>
    </home-address>
</employee>
```

Στο παραπάνω παράδειγμα, αναπαριστούμε τις προσωπικές πληροφορίες και τις πληροφορίες σχετικά με τις βάρδιες ενός υπαλλήλου σε ένα οργανισμό. Βλέπουμε πώς η XML χρησιμοποιεί τις διακριτικές ετικέτες "<>" και "</>" παρόμοια με τις ετικέτες που χρησιμοποιούνται στην HTML. Αυτό συμβαίνει γιατί η XML είναι μια γλώσσα σήμανσης σαν την HTML. Η κύρια διαφορά της XML με την HTML είναι ως προς τον σκοπό της κάθε μίας:

- Η XML σχεδιάστηκε για να περιγράφει δεδομένα και να εστιάσει στο τί είναι αυτά τα δεδομένα.
- Η HTML σχεδιάστηκε για να προβάλει δεδομένα και να εστιάσει στο πώς φαίνονται αυτά τα δεδομένα.

Στο παραπάνω παράδειγμα βλέπουμε ότι έχουμε να κάνουμε με καθαρά δεδομένα : ότι ένας υπάλληλος (employee) έχει παραπάνω από μία βάρδιες (shift), για παράδειγμα το πρωί εργάζεται στο ταμείο (counter) και το μεσημέρι στο γραφείο βοήθειας (help desk) και διευθύνσεις.

Οι δύο αρχικές δομικές μονάδες XML που χρησιμοποιούνται στο προηγούμενο παράδειγμα είναι τα elements (στοιχεία) και τα attributes (ιδιότητες) .

1.2 Elements ή Στοιχεία

Τα στοιχεία (elements) είναι ετικέτες, όπως και στην HTML, και περιέχουν τιμές. Επιπλέον τα elements είναι δομημένα σαν δένδρο. Ως εκ τούτου έχουμε τα στοιχεία οργανωμένα σε ένα ιεραρχικό τρόπο με ένα στοιχείο-πατέρα και στοιχεία-παιδιά. Τα στοιχεία-παιδιά μπορούν να περιέχουν και αυτά άλλα στοιχεία-παιδιά και ούτω καθ'εξής..

Στο προηγούμενο παράδειγμα, το στοιχείο <employee> είναι το γονικό στοιχείο και έχει το στοιχείο <shift> σαν στοιχείο-παιδί. Παρακάτω το στοιχείο

<phone> είναι στοιχείο-παιδί του γονικού στοιχείου <shift>. Τα στοιχεία έχουν συγκεκριμένα χαρακτηριστικά. Ορισμένα από αυτά είναι [5]:

- Τα στοιχεία μπορεί να περιέχουν δεδομένα, οπως το στοιχείο <number> στο παράδειγμα.
- Αντίστροφα, τα στοιχεία μπορεί να μην περιέχουν δεδομένα αλλά μόνο ιδιότητες, όπως το στοιχείο <shift>.
- Εναλλακτικά, τα στοιχεία μπορεί να περιέχουν ταυτόχρονα και ιδιότητες αλλά και δεδομένα, αλλά επίσης και στοιχεία-παιδιά, όπως το στοιχείο <phone>.

Τα στοιχεία έχουν κάποιους κανόνες :

- Όλα τα στοιχεία πρέπει να έχουν ετικέτα κλεισίματος αντίθετα με την HTML όπου υπάρχουν και ετικέτες που δε χρειάζονται κλείσιμο όπως για παράδειγμα η
.
- Οι ετικέτες των στοιχείων είναι case sensitive δηλαδή υπάρχει διαχωρισμός μεταξύ κεφαλαίων και πεζών και τα ονόματά τους υπακούουν σε κανόνες ονοματολογίας.
- Τα στοιχεία πρέπει να είναι τοποθετημένα σωστά αντίθετα με την HTML. HTML : This text is bold and italic XML : This text is bold and italic
- Τα έγγραφα της XML πρέπει να έχουν ακριβώς ένα αρχικό στοιχείο (root element).

1.3 Attributes ή Ιδιότητες

Οι ιδιότητες (attributes) μας βοηθούν να δώσουμε περισσότερο νόημα και να περιγράψουμε τα στοιχεία μας πιο αποτελεσματικά και με σαφήνεια. Στο προηγούμενο παράδειγμα, το στοιχείο <shift> έχει μία ιδιότητα "id" με τιμές "counter" και "help_desk" . Με τη χρήση τέτοιων ιδιοτήτων, μπορούμε να ξέρουμε άν ένας υπάλληλος εργάζεται στο ταμείο ή στο γραφείο βοήθειας. Αυτό βοηθάει στο να κάνουμε τα δεδομένα σε ένα έγγραφο XML αυτοπεριγραφικά. Πρέπει πάντα να θυμόμαστε ότι ο κύριος σκοπός των ιδιοτήτων είναι να παρέχουν περισσότερη πληροφορία σχετική με ένα στοιχείο και δεν πρέπει να χρησιμοποιούνται για να περιέχουν τα ίδια τα δεδομένα .Όπως και τα στοιχεία έτσι και οι ιδιότητες έχουν κάποιους κανόνες :

- Οι τιμές των ιδιοτήτων πρέπει να είναι εσωκλείωνται σε "" ή σε ''.
- Τα ονόματα των ιδιοτήτων ακολουθούν τους ίδιους κανόνες με αυτά των ετικετών.

1.4 Κανόνες ονομασίας

Τα στοιχεία και οι ιδιότητες στην XML πρέπει να ακολουθούν στους παρακάτω κανόνες [8]:

- Τα ονόματα μπορούν να περιέχουν γράμματα, αριθμούς και άλλους χαρακτήρες.

- Τα ονόματα δεν πρέπει να ξεκινούν με αριθμό ή χαρακτήρα στίξης.
- Τα ονόματα δεν πρέπει να ξεκινούν με τα γράμματα xml (ή XML, ή Xml κλπ.).
- Τα ονόματα δεν μπορούν να περιέχουν κενά.

1.5 Καλά διαμορφωμένα έγγραφα (well formed documents)

Ένα «καλά διαμορφωμένο» έγγραφο XML είναι ένα έγγραφο που υπακούει στους κανόνες σύνταξης της XML που αναφέραμε προηγουμένως [8]:

- Τα έγγραφα XML πρέπει να περιέχουν ένα αρχικό στοιχείο.
- Τα στοιχεία XML πρέπει να έχουν ετικέτες κλεισίματος.
- Στις ετικέτες XML υπάρχει διαχωρισμός κεφαλαίων και πεζών.
- Τα στοιχεία XML πρέπει να είναι σωστά τοποθετημένα.
- Οι ιδιότητες XML πρέπει να εσωκλείονται πάντα σε "" ή ''.

1.6 DTD

Όπως σε μία γλώσσα προγραμματισμού πρέπει να ξέρουμε τις προδιαγραφές της γλώσσας, με παρόμοιο τρόπο το Document Type Definition (DTD) είναι μία προδιαγραφή, η οποία πρέπει να ακολουθηθεί όταν δημιουργούμε ένα έγγραφο XML. Επίσης, όπως μία από τις εργασίες του μεταγλωττιστή για κάθε γλώσσα προγραμματισμού είναι να ελέγξει αν η προδιαγραφές ακολουθήθηκαν, με παρόμοιο τρόπο υπάρχουν XML parsers οι οποίοι χρησιμοποιούν το DTD για να ελέγξουν την εγκυρότητα ενός εγγράφου XML .

Ένα DTD μας βοηθάει να καθορίσουμε τη δομή ενός εγγράφου XML. Μας παρέχει ένα αυστηρό πλαίσιο και κανόνες οι οποίοι θα ακολουθηθούν όταν δημιουργούμε έγγραφα XML. Επιπρόσθετα, το DTD μπορεί να χρησιμοποιηθεί για τον έλεγχο της εγκυρότητας και της ακεραιότητας των δεδομένων που περιέχονται σε ένα έγγραφο XML.

Μερικά χαρακτηριστικά του DTD είναι τα παρακάτω :

- Το DTD χρησιμοποιείται για να καθορίσει έγκυρα στοιχεία και ιδιότητες που μπορούν να χρησιμοποιηθούν σε ένα έγγραφο XML.
- Με ένα DTD μπορούμε να καθορίσουμε μια ιεραρχική δομή στοιχείων.
- Σε ένα DTD μπορεί επίσης να καθοριστεί η διαδοχική οργάνωση μιας συλλογής στοιχείων-παιδιών τα οποία μπορούν να υπάρχουν σε ένα έγγραφο XML.

Ένα DTD μπορεί να χρησιμοποιηθεί απευθείας μέσα σε ένα έγγραφο XML ή μπορεί να υπάρχει εκτός του εγγράφου XML. Στη δεύτερη περίπτωση θα αναφέρεται με ένα δεσμό μέσα στο έγγραφο XML που δείχνει σε αυτό το DTD.

Βασικά το DTD αποτελείται από τα παρακάτω στοιχεία :

Στοιχείο	Περιγραφή
DTD Element	Μεταδεδομένα για ένα στοιχείο. Καθορίζει τι είδους δεδομένα θα έχει το στοιχείο, τον αριθμό των περιστατικών κάθε στοιχείου, τις σχέσεις μεταξύ των στοιχείων και ούτω καθ'εξής.
DTD Attributes	Καθορίζει διάφορους κανόνες και ορισμούς που σχετίζονται με τα δεδομένα.
DTD Entities	Χρησιμοποιείται για να αναφέρει ένα εξωτερικό αρχείο ή για να παρέχει συντομεύσεις σε κοινό κείμενο.

Παράδειγμα : <!ELEMENT employee(shift+,home-address, hobbies*)>

Ένα στοιχείο employee μπορεί να περιέχει ένα ή περισσότερα στοιχεία shift και πρέπει να έχει ένα στοιχείο home-address και μπορεί να έχει μηδέν ή περισσότερα στοιχεία hobbies.

Παράδειγμα : <!ATTLIST shift id CDATA #REQUIRED>
Το στοιχείο shift πρέπει να έχει μία ιδιότητα id.

Εν ολίγοις, ένα DTD χρησιμοποιείται για να καθορίσει μια δομή εγγράφων με τη διευκρίνιση των λεπτομερειών σχετικά με όλα τα στοιχεία και τις ιδιότητες που πρόκειται να χρησιμοποιηθούν σε ένα έγγραφο XML. Ως εκ τούτου μπορεί να χρησιμοποιηθεί για να ελέγξει την εγκυρότητα ενός εγγράφου XML που υποτίθεται ότι ακολουθεί τους κανόνες που καθορίζονται από αυτό το DTD [5].

1.7 XML Schema

Το XML Schema είναι μια πιο προηγμένη έκδοση του DTD. Το DTD έχει πολλά μειονεκτήματα σε σχέση με το schema, όπως το ότι δεν υποστηρίζει ισχυρούς τύπους δεδομένων, έχει σύνταξη διαφορετική από την XML και δεν είναι επεκτάσιμο. Το XML Schema παρουσιάστηκε για να υπερνικήσει αυτά τα μειονεκτήματα [5].

Οι δύο κύριοι στόχοι του W3c XML Schema working group κατά τη διάρκεια του σχεδιασμού του προτύπου του XML Schema ήταν [4]:

- Να μπορέσουν να εκφράσουν μέσα στο πρότυπο αρχές αντικειμενοστραφούς σχεδιασμού οι οποίες μπορούν να βρεθούν σε όλες τις αντικειμενοστραφείς γλώσσες προγραμματισμού.
- Να παρέχουν υποστήριξη για σύνθετους τύπους δεδομένων παρόμοια με την υποστήριξη που υπάρχει στις περισσότερες σχεσιακές βάσεις δεδομένων.

Τα κυριότερα χαρακτηριστικά του XML Schema είναι τα παρακάτω :

- Η σύνταξη είναι όμοια με της XML. Αυτό σημαίνει ότι μπορούμε να επεξεργαστούμε το schema με οποιοδήποτε επεξεργαστή XML.
- Δεν καθορίζουμε μόνο βασικούς τύπους δεδομένων όπως αλφαριθμητικό, ακέραιος, πραγματικός και ούτω καθεξής αλλά μπορούμε επίσης να καθορίσουμε δικούς μας τύπους δεδομένων. Για παράδειγμα:

```
<xs:element name="name" type="xs:string" />
```

- Οι νέοι τύποι που μπορούμε να καθορίσουμε μπορεί να είναι απλοί ή σύνθετοι. Οι σύνθετοι τύποι μπορεί να περιέχουν και άλλα στοιχεία ή και ιδιότητες, ενώ οι απλοί τύποι όχι. Αντίθετα μπορούν να περιέχουν μόνο δεδομένα
- Το XML Schema παρέχει επικύρωση βασισμένη στο περιεχόμενο (content-based validation) δηλαδή μπορεί να ορίσει την σειρά με την οποία τα στοιχεία-παιδιά εμφανίζονται. Επίσης παρέχει επικύρωση στους ίδιους τους τύπους δεδομένων.

  Για παράδειγμα μπορούμε να ορίσουμε έναν απλό τύπο "year" με τιμές μεταξύ 2000 και 2100:

```
<xsd:simpleType name="year">
    <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="2000"/>
        <xsd:maxInclusive value="2100"/>
</xsd:restriction>
</ xsd:simpleType>
```

- Παρομοίως οι σύνθετοι τύποι μπορεί να ορίσουν τη σειρά με την οποία τα στοιχεία-παιδιά θα εμφανίζονται:

```
<xsd:complexType name="Employee">
    <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" />
    <xsd:element name="Address" type="xsd:string" />
    <xsd:element name="Phone" type="xsd:string" />
</sd:sequence>
</xsd:complexType>
```

- Το XML Schema μας παρέχει τη δυνατότητα να επεκτείνουμε άλλα έγγραφα το οποίο δεν είναι τίποτα άλλο παρά κληρονομικότητα. Αυτό σημαίνει ότι μπορούμε να παράγουμε νέους τύπους δεδομένων βάσει παλαιών τύπων.
- Το XML Schema παρέχει υποστήριξη για Namespaces (χρησιμοποιώντας URI). Παρέχει σε κάθε στοιχείο ένα μοναδικό αναγνωριστικό, με το οποίο αποφεύγονται συγκρούσεις ονομάτων μεταξύ των στοιχείων. Αυτό θα μπορούσε να συμβεί, για παράδειγμα, όταν δύο έγγραφα συγχωνεύονταν, και περιείχαν και τα δύο στοιχεία με όνομα "name" τα οποία όμως είχαν διαφορετικό νόημα σε κάθε έγγραφο. Εν ολίγοις βοηθάει στο να ξεχωρίζουν στοιχεία και ιδιότητες με ίδιο όνομα και διαφορετικό νόημα. Μία απλή

αναλογία στις γλώσσες προγραμματισμού είναι η χρήση καθολικών και τοπικών μεταβλητών. Μια τοπική μεταβλητή είναι μοναδική μέσα στο πεδίο ισχύος της ενώ μια καθολική μεταβλητή πρέπει να είναι μοναδική σε ολόκληρο το πρόγραμμα. Παρομοίως, με το namespace έχουμε την ελευθερία να να ορίσουμε τύπους χωρίς να ανησυχούμε για συγκρούσεις ονομάτων [4].

- Τέλος το XML Schema είναι εύκολα επεκτάσιμο για να ενσωματώσει και άλλες λειτουργίες στο μέλλον.

1.8 Σύγκριση του XML Schema με το DTD

Οι κυριότερες διαφορές συνοψίζονται ως εξής :

- Το XML Schema είναι επέκταση του DTD.
- Το XML Schema υποστηρίζει namespaces ενώ το DTD όχι.
- Το XML Schema χρησιμοποιεί σύνταξη XML η οποία είναι εύκολη να την κατανοήσεις ενώ το DTD χρησιμοποιεί ειδική σύνταξη.
- Το XML Schema υποστηρίζει πρότυπους τύπους δεδομένων καθώς επίσης και τύπους ορισμένους από το χρήστη (user-defined) ενώ το DTD παρέχει μόνο τύπους κειμένου.
- Το XML Schema υποστηρίζει κληρονομικότητα ενώ το DTD όχι.

1.9 Παράδειγμα XML Schema

Ας δούμε ένα παράδειγμα ενός XML Schema για τον καθορισμό της δομής ενός εγγράφου XML στο οποίο θα τηρούμε πληροφορίες για ένα βιβλιοπωλείο

BookStore) το οποίο περιέχει πολλά βιβλία (Book) :

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.books.org"
                    xmlns=http://www.books.org>  A
<xsd:element name="BookStore">  B
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Book">  C
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
      <xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="Title" type="xsd:string"/>  D
<xsd:element name="Author" type="xsd:string"/>
<xsd:element name="Date" type="xsd:string"/>
<xsd:element name="ISBN" type="xsd:string"/>
<xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

Είναι εύκολο να καταλάβουμε τα αναγνωριστικά. Ένα στοιχείο αναπαριστάται με την ετικέτα "element". Η ετικέτα "comlexType" μας παρέχει το μηχανισμό με τον οποίο μπορούμε να ορίσουμε σύνθετους τύπους δεδομένων.

Για εύκολη αναφορά στο παραπάνω έγγραφο XML, το έγγραφο έχει χαρακτηριστεί σε διάφορα τμήματα. Στο τμήμα A του εγγράφου, η πρώτη γραμμή περιέχει μια τιμή "xsd" σαν ένα namespace. Κάθε αναφορά σε namespace απεικονίζεται χρησιμοποιώντας αυτή την τιμή σαν πρόθεμα και η σύνταξη για να αναφερθούμε σε ένα στοιχείο είναι namespace:elementname (για παράδειγμα xsd:element). Η ιδιότητα targetNamespace καθορίζει το namespace στο οποίο θα ανήκουν οι νέοι καθορισμένοι τύποι. Στο παραπάνω παράδειγμα, οι καθορισμένοι τύποι BookStore, Book, Title, Author, ISBN και Publisher ανήκουν στο namespace http://www.books.org.

Η ιδιότητα xmlns ορίζει τον προεπιλεγμένο namespace. Με άλλα λόγια, από αυτή τη θέση και έπειτα αν βρεθεί κάποιο στοιχείο στο οποίο δεν ορίζεται πρόθεμα για namespace τότε αυτό ανήκει στο default namespace.

Στο τμήμα B ένας νέος τύπος BookStore, ορίζεται σαν complexType (σύνθετος τύπος), ο οποίος περιέχει μια sequence (ακολουθία) από στοιχεία τύπου Book. Ένας complexType χρησιμοποιείται για να ορίσουμε ένα τύπο καθορισμένο από τον χρήστη, ο οποίος μπορεί να περιέχει πολλαπλά στοιχεία και ιδιότητες. Το τμήμα C ορίζει τον τύπο Book, ο οποίος είναι ο ίδιος ένας σύνθετος τύπος και

περιέχει μια ακολουθία από τα στοιχεία Title, Author, Date, ISBN και Publisher. Οι ιδιότητες minOccur και maxOccur καθορίζουν τον περιορισμό στην εμφάνιση των στοιχείων. Η σειρά των στοιχείων που εμφανίζονται μέσα σε ένα στοιχείο Book πρέπει να ακολουθεί τη σειρά της προδιαγραφής. Με άλλα λόγια, ο τύπος Book θα περιέχει ένα στοιχείο Title ακολουθούμενο από ένα στοιχείο Author και ούτω καθ'εξής.

Το τμήμα D ορίζει τους τύπους των στοιχείων του Book. Κάθε ένα από αυτά τα στοιχεία είναι ένας simpleType (απλός τύπος) string.

Μπορούμε να ορίσουμε δικούς μας απλούς τύπους, αλλά ένας απλός τύπος δεν αποτελείται από στοιχεία ή ιδιότητες. Ένας απλός τύπος συνήθως καθορίζεται για να αναπαραστήσει ένα περιορισμό σε ένα βασικό τύπο. Για παράδειγμα, παρακάτω έχουμε τον ορισμό ενός απλού τύπου με όνομα elevation ο οποίος μπορεί να έχει έγκυρες τιμές μεταξύ 50 και 12000.

```
<xsd:simpleType name="elevation">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="50"/>
    <xsd:maxInclusive value="12000"/>
  </xsd:restriction>
</xsd:simpleType>
```

Έχοντας εξετάσει το έγγραφο του XML Schema παραπάνω, ας δούμε ένα έγγραφο XML που ακολουθεί το παραπάνω schema.

```
<?xml version="1.0"?>
<BookStore xmlns ="http://www.books.org"  1
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  2
           xsi:schemaLocation="http://www.books.org/
                               BookStore.xsd">  3
    <Book>  4
           <Title>Web Services Security</Title>
           <Author>Ravi Trivedi</Author>
           <Date>Dec, 2002</Date>
           <ISBN>1861007655</ISBN>
           <Publisher>Wrox Publishing</Publisher>
       </Book>
</BookStore>
```

Ο πρώτος ορισμός του xmlns χρησιμοποιεί τη δήλωση του προεπιλεγμένου namespace, και αυτό υπαγορεύει στον ελεγκτή του schema ότι όλα τα στοιχεία που χρησιμοποιούνται σε αυτό το έγγραφο έρχονται από το namespace Book.

Ο δεύτερος ορισμός λέει στον ελεγκτή του schema ότι η ιδιότητα schemaLocation έρχεται από το namespace XMLSchema.

Ο τρίτος ορισμός, με το schemaLocation, λέει στον ελεγκτή ότι το namespace http://www.books.org είναι ορισμένο στο BookStore.xsd. Αυτό το αρχείο θα ανατρέχει από τον ελεγκτή του schema για να επικυρώσει την ορθότητα του τρέχοντος εγγράφου XML.

Τέλος, στον τέταρτο ορισμό, δηλώνεται ένα στοιχείο Book το οποίο περιέχει τις λεπτομέρειες ενός βιβλίου. Άξιο προσοχής είναι το γεγονός ότι η σειρά των στοιχείων-παιδιών του διατηρείται όπως στον ορισμό του complexType Book.

Ένα από τα μεγαλύτερα πλεονεκτήματα του XML Schema είναι ότι χρησιμοποιεί XML για τη σύνταξή του. Χρησιμοποιώντας XML, υπάρχοντες XML parsers μπορούν να χρησιμοποιηθούν σε συνδυασμό με ελεγκτές schema για να παρέχουν υπηρεσίες ελέγχου καλής διαμόρφωσης (well-formedness) και επικύρωσης (validation).

Το XML Schema προσφέρει έναν αυτοματοποιημένο μηχανισμό για επικύρωση των εγγράφων XML. Έχει παρατηρηθεί ότι σε τυπικό πρόγραμμα, μέχρι και 60% του κώδικα ξοδεύεται σε έλεγχο δεδομένων. Αν τα δεδομένα μας είναι δομημένα σαν XML, και υπάρχει και ένα schema, μπορούμε να περάσουμε τον έλεγχο των δεδομένων σε ένα ελεγκτή schema (schema validator). Κατά συνέπεια, μπορούμε να μειώσουμε τον κώδικά μας μέχρι και 60%. Επίσης, την επόμενη φορά που θα αλλάξουν οι περιορισμοί των δεδομένων μας ή προσθέσουμε και άλλα στοιχεία, δεν θα χρειαστεί να γράψουμε νέο κώδικα για τον έλεγχο των δεδομένων μας αλλά απλώς να αλλάξουμε το schema.

Η προδιαγραφή του XML Schema παίζει σημαντικό ρόλο στο σχεδιασμό και την υλοποίηση των web services. Τα αρχεία WSDL είναι επίσης κατασκευασμένα χρησιμοποιώντας τη σύνταξη του XML Schema.

2. ΚΩΔΙΚΑΣ SERVER

ApplicationContextBeanProvider.java

```
package org.hospital.beds.application;


import org.springframework.context.ApplicationContextAware;


import org.springframework.beans.BeansException;

import org.springframework.context.ApplicationContext;


public class ApplicationContextBeanProvider implements ApplicationContextAware
{


   private static ApplicationContext applicationContext;
```

```
    public void setApplicationContext(ApplicationContext ctx) throws BeansException
{
        applicationContext = ctx;
    }

    public static Object getBean(String beanName) {
        return applicationContext.getBean(beanName);
    }
}
```

2.1 Dao.java

```
package org.hospital.beds.control;

import java.util.ArrayList;
import java.util.Date;
import org.hospital.beds.exceptions.NotAvailableBeds;

import org.hospital.beds.model.*;

public interface Dao {

    public String initData();

    public java.util.ArrayList<Bed> getAvailableBeds();

    public java.util.ArrayList<Bed> getReservedBeds();

    public Bed getCurrentBed(Patient patient);
```

```
public ArrayList<Patient> getHostedPatients();

public ArrayList<Patient> getPatients();

public void persist(MyEntity entity);

public void merge(MyEntity entity);

public void destroy(MyEntity entity);

public User retrieveUser(String username, String password);

public void saveUser(User user);

public Patient retrievePatient(String patientId);

public Department retrieveDepartment(String departmentId);

public Bed retrieveBed(String bedCode);

public ArrayList<Bed> getAvailableBedsByDepartmentAndCategory(Department
department, Category category);

public ArrayList<Department> getDepartments();

public void reserve(String bedCode, String patientIdNumber);

public Stay getCurrentStay(String patientId);
```

```
    public ArrayList<Stay> getStayHistory(String patientId);

    public void closeStay(String patientIdNumber);

    public Stay saveStay(Stay stay) throws NotAvailableBeds;

    public void updateStay(String patientIdNumber, String comments);

    public ArrayList<Ward> getAllWards();

    public int countAvaibleBedsInWard(Ward ward);

    public int countAllBedsInWard(Ward ward);

    public Ward retrieveWard(Long code);

    public ArrayList<Bed> getBedsInWard(Ward ward);

    public Stay getCurrentStayIfAny(Bed bed);

    public ArrayList<Stay> retrieveStayStatistics(Date from,Date to,boolean
isForStartDate);
}
```

2.2 DaoImpl.java

```
package org.hospital.beds.control;

import java.io.PrintWriter;
import java.io.StringWriter;
```

```
import java.io.Writer;
import java.util.*;
import java.util.ArrayList;
import org.hospital.beds.exceptions.NotAvailableBeds;

import org.hospital.beds.model.Bed;
import org.hospital.beds.model.Category;
import org.hospital.beds.model.Department;
import org.hospital.beds.model.MyEntity;
import org.hospital.beds.model.Patient;
import org.hospital.beds.model.Stay;
import org.hospital.beds.model.User;
import org.hospital.beds.model.Ward;
import org.hospital.beds.model.Wing;
import org.springframework.orm.jpa.support.JpaDaoSupport;
import org.springframework.transaction.annotation.Transactional;

public class DaoImpl extends JpaDaoSupport implements Dao {

   @Override
   public ArrayList<Bed> getAvailableBeds() {
      List<Bed> b = getJpaTemplate().findByNamedQuery(Bed.BedAvailable);
      return b == null ? new ArrayList<Bed>() : new ArrayList<Bed>(b);
   }

   @Override
   public ArrayList<Bed> getReservedBeds() {
      List<Bed> b = getJpaTemplate().findByNamedQuery(Bed.BedReserved);
      return b == null ? new ArrayList<Bed>() : new ArrayList<Bed>(b);
```

```
    }


    @Override
    public Bed getCurrentBed(Patient patient) {
        List<Bed> currentBed =
getJpaTemplate().findByNamedQueryAndNamedParams(
                Patient.PatientCurrentBed, Collections.singletonMap("patient", patient));
        if (currentBed.isEmpty()) {
            return null;
        }
        if (currentBed.size() == 1) {
            return currentBed.get(0);
        }
        throw new RuntimeException(
                "There more than one bed for the same patient!!");
    }


    @Override
    public ArrayList<Patient> getHostedPatients() {
        List<Patient> p =
getJpaTemplate().findByNamedQuery(Patient.PatientsHosted);
        return p == null ? new ArrayList<Patient>() : new ArrayList<Patient>(p);
    }


    @Override
    @Transactional
    public void persist(MyEntity entity) {
        getJpaTemplate().persist(entity);
    }
```

```
    @Override
    @Transactional
    public void destroy(MyEntity entity) {
        getJpaTemplate().remove(entity);
    }

    @Override
    @Transactional
    public void merge(MyEntity entity) {
        getJpaTemplate().merge(entity);
    }

    @Override
    public User retrieveUser(String username, String password) {
        Map<String, Object> parameters = new HashMap<String, Object>();
        parameters.put(User.USERNAME_PARAM, username);
        parameters.put(User.PASSWORD_PARAM, password);
        List<User> users =
getJpaTemplate().findByNamedQueryAndNamedParams(User.UserRetrieve,
parameters);
        return (users == null || users.isEmpty()) ? null : users.get(0);
    }

    @Override
    @Transactional
    public void saveUser(User user) {
        User u = retrieveUser(user.getUserName(), user.getPassword());
        if (u != null) {
            user.setId(u.getId());
            getJpaTemplate().merge(user);
```

```
        } else {
            getJpaTemplate().persist(user);
        }
    }

    @Override
    public Patient retrievePatient(String patientId) {
        List<Patient> result = getJpaTemplate().findByNamedQuery(Patient.Retrieve,
patientId);
        return (result == null || result.isEmpty()) ? null : result.get(0);
    }

    @Override
    public Bed retrieveBed(String bedCode) {
        List<Bed> result = getJpaTemplate().findByNamedQuery(Bed.Retrieve,
bedCode);
        return (result == null || result.isEmpty()) ? null : result.get(0);
    }

    @Override
    public ArrayList<Bed> getAvailableBedsByDepartmentAndCategory(Department
department, Category category) {
        Map<String, Object> parameters = new HashMap<String, Object>();
        parameters.put("department", department);
        parameters.put("category", category);
        List<Bed> beds =
getJpaTemplate().findByNamedQueryAndNamedParams(Bed.BedAvailableByDepart
mentAndCategory, parameters);
        return beds == null ? new ArrayList<Bed>() : new ArrayList<Bed>(beds);
    }
```

```
@Override
@Transactional
public String initData() {
   try {
      User user = new User();
      user.setAdmin(true);
      user.setUserName("admin");
      user.setPassword("admin");
      user.setFirstName("Administrator");
      user.setLastName("Administrator");
      getJpaTemplate().persist(user);



      List<Department> departments = new ArrayList<Department>();

      Department depA = new Department();
      depA.setTitle("Καρδιοχειρουργικό");
      depA.setCode("kardioxeirourgiko");
      getJpaTemplate().persist(depA);
      departments.add(depA);

      Department depB = new Department();
      depB.setTitle("Πνευμονολογικό");
      depB.setCode("pneumonologiko");
      getJpaTemplate().persist(depB);
      departments.add(depB);

      Department depC = new Department();
      depC.setTitle("Γυναικολογικό");
```

```
        depC.setCode("gunaikologiko");
        getJpaTemplate().persist(depC);
        departments.add(depC);

        for (Department dep : departments) {
           addWings(dep);
        }

        return "Completed!!!";
    } catch (Exception e) {
        return getStackTrace(e);
    }
}

private void addWings(Department depA) {
    List<Wing> wings = new ArrayList<Wing>();

    Wing wingA = new Wing();
    wingA.setDepartment(depA);
    wingA.setCategory(Category.A);
    wingA.setTitle("Κατηγορία Α");
    getJpaTemplate().persist(wingA);
    wings.add(wingA);

    wingA = new Wing();
    wingA.setDepartment(depA);
    wingA.setCategory(Category.B);
    wingA.setTitle("Κατηγορία Β");
```

```
        getJpaTemplate().persist(wingA);
        wings.add(wingA);

        wingA = new Wing();
        wingA.setDepartment(depA);
        wingA.setCategory(Category.C);
        wingA.setTitle("Κατηγορία Γ");
        getJpaTemplate().persist(wingA);
        wings.add(wingA);

        for (Wing wing : wings) {
            for (int wardId = 0; wardId < 10; wardId++) {
                Ward ward = new Ward();
                ward.setWing(wing);
                ward.setTitle("Θάλαμος #" + wardId);
                getJpaTemplate().persist(ward);
                for (int i = 0; i
                        < (wing.getCategory().equals(Category.A) ? 1
                        : wing.getCategory().equals(Category.B) ? 2
                        : 3); i++) {
                    Bed bed = new Bed();
                    bed.setWard(ward);
                    bed.setCode("#" + i + "(" + depA.getId() + "/" + ward.getId() + ")");
                    getJpaTemplate().persist(bed);
                }
            }
        }
    }
```

```
    public static String getStackTrace(Throwable aThrowable) {
        final Writer result = new StringWriter();
        final PrintWriter printWriter = new PrintWriter(result);
        aThrowable.printStackTrace(printWriter);
        return result.toString();
    }

    @Override
    public ArrayList<Department> getDepartments() {
        List<Department> departments =
getJpaTemplate().findByNamedQuery(Department.RetrieveAll);
        return departments == null ? new ArrayList<Department>() : new
ArrayList<Department>(departments);
    }

    @Override
    public ArrayList<Patient> getPatients() {
        List<Patient> p = getJpaTemplate().findByNamedQuery(Patient.RetrieveAll);
        return p == null ? new ArrayList<Patient>() : new ArrayList<Patient>(p);
    }

    @Override
    public Department retrieveDepartment(String departmentId) {
        List<Department> d =
getJpaTemplate().findByNamedQuery(Department.RetrieveByCode, departmentId);
        return (d == null || d.isEmpty()) ? null : d.get(0);
    }

    @Transactional
    @Override
```

```
    public Stay getCurrentStay(String patientId) {
        List<Stay> s =
getJpaTemplate().findByNamedQuery(Stay.RetriveNotClosedByPatientIdNumber,
patientId);
        if (s == null || s.isEmpty()) {
            return null;
        }
        Stay stay = s.get(0);
        stay.getBed().getWard().getWing().getDepartment();
        stay.getBed().getWard().getWing().getCategory();
        stay.getPatient().getCategory();
        return stay;
    }

    @Override
    @Transactional
    public void reserve(String bedCode, String patientIdNumber) {
        Bed bed = retrieveBed(bedCode);
        if (bed == null) {
            throw new RuntimeException("There isn't any bed having code " + bedCode);
        }
        Patient patient = retrievePatient(patientIdNumber);
        if (patient == null) {
            throw new RuntimeException("There is not any patient having id number " +
patientIdNumber);
        }
        Bed currentBed = getCurrentBed(patient);
        if (currentBed != null) {
            throw new RuntimeException("Patient having id number " + patientIdNumber
+ " is already in bed " + currentBed.getCode());
```

```
    }

    Stay stay = new Stay();
    stay.setBed(bed);
    stay.setPatient(patient);
    stay.setCheckIn(new Date());
    getJpaTemplate().persist(stay);
}

@Transactional
@Override
public void closeStay(String patientIdNumber) {
    Stay stay = getCurrentStay(patientIdNumber);
    if (stay == null) {
        throw new RuntimeException("Stay doesn't exist or is closed!");
    }
    stay.setCheckOut(new java.util.Date());
    merge(stay);
}

@Transactional
@Override
public void updateStay(String patientIdNumber, String comments) {
    Stay stay = getCurrentStay(patientIdNumber);
    if (stay == null) {
        throw new RuntimeException("Stay doesn't exist or is closed!");
    }
    stay.setComments(comments);
    merge(stay);
```

```
    }

    @Transactional
    @Override
    public ArrayList<Ward> getAllWards() {
        List<Ward> b = getJpaTemplate().findByNamedQuery(Ward.RetrieveWards);
        return b == null ? new ArrayList<Ward>() : new ArrayList<Ward>(b);
    }

    public int countAvaibleBedsInWard(Ward ward) {
        return ((Number)
getJpaTemplate().findByNamedQueryAndNamedParams(Bed.CountBedAvailableBy
Ward, Collections.singletonMap(Bed.WARD_PARAM, ward)).get(0)).intValue();
    }

    public int countAllBedsInWard(Ward ward) {
        return ((Number)
getJpaTemplate().findByNamedQueryAndNamedParams(Bed.CountBedsByWard,
Collections.singletonMap(Bed.WARD_PARAM, ward)).get(0)).intValue();
    }

    @Transactional
    public Stay getCurrentStayIfAny(Bed bed) {
        List stay =
getJpaTemplate().findByNamedQueryAndNamedParams(Stay.RetrieveNotClosedBy
Bed, Collections.singletonMap(Stay.BED_PARAM, bed));
        if (stay == null || stay.isEmpty()) {
            return null;
        }
        Stay _stay = (Stay) stay.get(0);
        _stay.setBed(bed);
```

```
        _stay.getPatient().getCategory();
        return _stay;
    }

    public Ward retrieveWard(Long code) {
        return getJpaTemplate().find(Ward.class, code);
    }

    @Transactional
    public ArrayList<Bed> getBedsInWard(Ward ward) {
        List<Bed> beds =
getJpaTemplate().findByNamedQueryAndNamedParams(Bed.BedsByWard,
Collections.singletonMap(Bed.WARD_PARAM, ward));
        if (beds == null) {
            return new ArrayList<Bed>();
        }
        for (Bed bed : beds) {
            bed.setCurrentStay(getCurrentStayIfAny(bed));
        }
        return new ArrayList<Bed>(beds);
    }

    @Transactional
    public Stay saveStay(Stay stay) throws NotAvailableBeds {
        Stay lastStay = getCurrentStay(stay.getPatient().getIdNumber());
        if (lastStay != null && lastStay.getCheckOut() == null) {
            if
(lastStay.getBed().getWard().getWing().getDepartment().equals(stay.getDepartment()
)) {
                stay.setId(lastStay.getId());
```

```
            stay.setBed(lastStay.getBed());
            stay.setCheckIn(lastStay.getCheckIn());
            getJpaTemplate().merge(stay);
            return stay;
         } else {
            closeStay(stay.getPatient().getIdNumber());
         }
      }
      stay.setDepartment(retrieveDepartment(stay.getDepartment().getCode()));
      List<Bed> beds =
getAvailableBedsByDepartmentAndCategory(stay.getDepartment(),
stay.getPatient().getCategory());
      if (beds == null || beds.isEmpty()) {
         throw new NotAvailableBeds();
      }
      stay.setCheckIn(new Date());
      stay.setBed(beds.get(0));
      stay.setDepartment(null);
      getJpaTemplate().persist(stay);
      return stay;
   }

   @Transactional
   public ArrayList<Stay> getStayHistory(String patientId) {
      List<Stay> s =
getJpaTemplate().findByNamedQuery(Stay.RetrieveByPatientIdNumber, patientId);
      if (s == null) {
         return new ArrayList<Stay>();
      }
      for (Stay ss : s) {
```

```
            ss.getBed().getCode();
            ss.getPatient().getCategory();
        }
        return new ArrayList<Stay>(s);
    }

    @Transactional
    public ArrayList<Stay> retrieveStayStatistics(Date from, Date to, boolean
isForStartDate) {
        List<Stay> s;

        if (isForStartDate) {
            Map<String, Date> parameters = new HashMap<String, Date>();
            parameters.put(Stay.CHECK_IN_FROM_PARAM, from);
            parameters.put(Stay.CHECK_IN_TO_PARAM, to);
            s =
getJpaTemplate().findByNamedQueryAndNamedParams(Stay.RetrieveStatisticsChec
kIn, parameters);
        } else {
            Map<String, Date> parameters = new HashMap<String, Date>();
            parameters.put(Stay.CHECK_OUT_FROM_PARAM, from);
            parameters.put(Stay.CHECK_OUT_TO_PARAM, to);
            s =
getJpaTemplate().findByNamedQueryAndNamedParams(Stay.RetrieveStatisticsChec
kOut, parameters);
        }
        if (s == null) {
            return new ArrayList<Stay>();
        }
        for (Stay ss : s) {
            ss.getBed().getCode();
```

```
            ss.getPatient().getCategory();
        }
        return new ArrayList<Stay>(s);
    }
}
```

2.3 NotAvailableBeds.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.hospital.beds.exceptions;


public class NotAvailableBeds extends Exception{

}
```

2.4 Bed.java

```
package org.hospital.beds.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
```

```
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.PostLoad;
import javax.persistence.Transient;
import org.hospital.beds.application.ApplicationContextBeanProvider;
import org.hospital.beds.control.Dao;
import static org.hospital.beds.model.Bed.*;

import java.util.*;

/**
 * Κλίνη
 *
 */
@Entity
@NamedQueries({
   @NamedQuery(name = BedAvailable, query = "SELECT a FROM Bed a WHERE
a.id NOT IN (SELECT b.bed FROM Stay b WHERE b.checkOut is null)"),
   @NamedQuery(name = IsAvailable, query = "SELECT a FROM Bed a WHERE
a.code=:code AND a.id NOT IN (SELECT b.bed FROM Stay b WHERE b.checkOut
is null)"),
   @NamedQuery(name = BedAvailableByDepartmentAndCategory, query =
"SELECT a FROM Bed a WHERE a.ward.wing.department=:department AND
a.ward.wing.category=:category "),
   @NamedQuery(name = BedReserved, query = "SELECT b.bed FROM Stay b
WHERE b.checkOut is null"),
   @NamedQuery(name = Retrieve, query = "SELECT a FROM Bed a WHERE
a.code=?"),

   @NamedQuery(name = CountBedAvailableByWard, query = "SELECT count(a)
FROM Bed a WHERE a.ward=:"+WARD_PARAM+" AND a.id NOT IN (SELECT
b.bed FROM Stay b WHERE b.checkOut is null)"),
```

```
    @NamedQuery(name = CountBedReservedByWard, query = "SELECT
count(b.bed) FROM Stay b WHERE b.bed.ward=:"+WARD_PARAM+" AND
b.checkOut is null"),

    @NamedQuery(name = CountBedsByWard, query = "SELECT count(a) FROM
Bed a WHERE a.ward=:"+WARD_PARAM),



    @NamedQuery(name = BedAvailableByWard, query = "SELECT a FROM Bed a
WHERE a.ward=:"+WARD_PARAM+" AND a.id NOT IN (SELECT b.bed FROM
Stay b WHERE b.checkOut is null)"),

    @NamedQuery(name = BedReservedByWard, query = "SELECT b.bed FROM
Stay b WHERE b.bed.ward=:"+WARD_PARAM+" AND b.checkOut is null"),

    @NamedQuery(name = BedsByWard, query = "SELECT a FROM Bed a WHERE
a.ward=:"+WARD_PARAM)



})

public class Bed extends MyEntity {



    public static final String WARD_PARAM="wardP";



    public static final String Retrieve = "Bed.Retrieve";

    public static final String IsAvailable = "Bed.IsAvailable";

    public static final String BedAvailable = "Bed.Available";

    public static final String BedReserved = "Bed.Reserved";

    public static final String BedAvailableByDepartmentAndCategory =
"Bed.AvailableByDepartmentAndCategory";



    public static final String CountBedAvailableByWard =
"Bed.CountAvailableByWard";

    public static final String CountBedReservedByWard =
"Bed.CountReservedByWard";

    public static final String CountBedsByWard = "Bed.CountAllByWard";



    public static final String BedAvailableByWard = "Bed.AvailableByWard";
```

```
public static final String BedReservedByWard = "Bed.ReservedByWard";
public static final String BedsByWard = "Bed.AllByWard";

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id")
private Long id;
@Column(unique = true, nullable = false)
private String code;
@JoinColumn(name = "Ward", referencedColumnName = "ID", nullable = false)
@ManyToOne(optional = false, fetch = FetchType.EAGER)
private Ward ward;

@Transient
private Stay currentStay;

public Long getId() {
   return id;
}

public void setId(Long id) {
   this.id = id;
}

public String getCode() {
   return code;
}

public void setCode(String code) {
```

```
        this.code = code;
    }

    public Ward getWard() {
        return ward;
    }

    public void setWard(Ward ward) {
        this.ward = ward;
    }

    public Stay getCurrentStay() {
        return currentStay;
    }

    public void setCurrentStay(Stay currentStay) {
      if(currentStay!=null)
        currentStay.setBed(null);
        this.currentStay = currentStay;
    }
}
```

2.5 Category.java

```
package org.hospital.beds.model;
public enum Category {
    A,B,C
}
```

2.6 Department.java

```
package org.hospital.beds.model;

import static org.hospital.beds.model.Department.*;
import javax.persistence.*;

/**
 * Τμήμα
 *
 */
@Entity
@NamedQueries({
@NamedQuery(name = RetrieveAll, query = "SELECT a FROM Department a"),
@NamedQuery(name = RetrieveByCode, query = "SELECT a FROM Department a
WHERE a.code=?")
})
public class Department extends MyEntity {

    public static final String RetrieveAll = "Department.RetrieveAll";
    public static final String RetrieveByCode="Department.RetrieveByCode";
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(unique=true)
    private String code;
```

```
private String title;
private static final long serialVersionUID = 1L;

public Department() {
    super();
}

public String getCode() {
    return code;
}

public void setCode(String code) {
    this.code = code;
}

public Long getId() {
    return this.id;
}

public void setId(Long id) {
    this.id = id;
}

public String getTitle() {
    return this.title;
}

public void setTitle(String title) {
    this.title = title;
```

```
    }
}
```

2.7 MyEntity.java

```
package org.hospital.beds.model;

import java.io.Serializable;

public abstract class MyEntity implements Serializable
{
        public abstract Long getId();

}
```

2.8 Patient.java

```
package org.hospital.beds.model;

import java.util.Date;
import javax.persistence.*;
import static org.hospital.beds.model.Patient.*;

/**
 * Ασθενής
*/
@Entity
@NamedQueries({
   @NamedQuery(name = PatientCurrentBed, query = "SELECT a.bed FROM Stay a
WHERE a.checkOut is null AND a.patient=:patient"),
```

```
   @NamedQuery(name = PatientsHosted, query = "SELECT a.patient FROM Stay a
WHERE a.checkOut is null"),
   @NamedQuery(name = RetrieveAll, query = "SELECT a.patient FROM Stay a
WHERE a.checkOut is null"),
   @NamedQuery(name = Retrieve, query = "SELECT a FROM Patient a WHERE
a.idNumber=?")
})
public class Patient extends MyEntity {

   public static final String PatientCurrentBed = "Patient.CurrentBed";
   public static final String PatientsHosted = "Patient.Hosted";
   public static final String RetrieveAll = "Patient.RetrieveAll";
   public static final String Retrieve = "Patient.Retrieve";
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   /**
    * Αριθμός ταυτότητας
    */
   @Column(nullable=false,unique=true)
   private String idNumber;
   /**
    * Ονομα
    */
   private String firstname;
   /**
    * Επώνυμο
    */
   private String lastname;
   /**
```

```
 * Ημερομηνία γέννησης
 */
@Temporal(TemporalType.DATE)
private Date dateOfBirth;
/**
 * ασφαλιστικό ταμείο
 */
@Enumerated(EnumType.STRING)
private Tameio tameio;
/**
 * τηλέφωνο
 */
private String telephone;
/**
 * Ιατρικό ιστορικο
 */
private String comments;
/**
 * Κατηγορία  πολυτελείας
 */
@Enumerated(EnumType.STRING)
@Basic(optional = false)
private Category category;

public Long getId() {
    return id;
}

public void setId(Long id) {
```

```
    this.id = id;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

public String getComments() {
    return comments;
}

public void setComments(String comments) {
    this.comments = comments;
}

public Category getCategory() {
    return category;
```

```
}

public void setCategory(Category category) {
   this.category = category;
}

public Date getDateOfBirth() {
   return dateOfBirth;
}

public void setDateOfBirth(Date dateOfBirth) {
   this.dateOfBirth = dateOfBirth;
}

public String getIdNumber() {
   return idNumber;
}

public void setIdNumber(String idNumber) {
   this.idNumber = idNumber;
}

public Tameio getTameio() {
   return tameio;
}

public void setTameio(Tameio tameio) {
   this.tameio = tameio;
}
```

```
    public String getTelephone() {
        return telephone;
    }

    public void setTelephone(String telephone) {
        this.telephone = telephone;
    }
}
```

2.9 Stay.java

```
package org.hospital.beds.model;

import java.util.Date;

import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
```

```
import javax.persistence.Transient;

import static org.hospital.beds.model.Stay.*;

/**

 * Διαμονή

 */

@Entity

@NamedQueries({

   @NamedQuery(name = RetrieveByPatientIdNumber,

   query = "SELECT a FROM Stay a WHERE a.patient.idNumber=? ORDER BY
a.checkIn"),

   @NamedQuery(name = RetriveNotClosedByPatientIdNumber,

   query = "SELECT a FROM Stay a WHERE a.patient.idNumber=? AND
a.checkOut is null"),

   @NamedQuery(name = RetrieveNotClosedByBed,

   query = "SELECT a FROM Stay a WHERE a.bed=:"+BED_PARAM+" AND
a.checkOut is null"),

   @NamedQuery(name = RetrieveStatisticsCheckOut,

   query = "SELECT a FROM Stay a WHERE a.checkOut between
:"+CHECK_OUT_FROM_PARAM+

         " AND :"+CHECK_OUT_TO_PARAM),

   @NamedQuery(name = RetrieveStatisticsCheckIn,

   query = "SELECT a FROM Stay a WHERE a.checkIn between
:"+CHECK_IN_FROM_PARAM+

         " AND :"+CHECK_IN_TO_PARAM)


})

public class Stay extends MyEntity {


   public static final String RetrieveStatisticsCheckOut =
"Stay.RetrieveStatisticsCheckOut";

   public static final String RetrieveStatisticsCheckIn =
"Stay.RetrieveStatisticsCheckIn";
```

```
    public static final String RetriveNotClosedByPatientIdNumber =
"Stay.LastNotClosedByPatientIdNumber";

    public static final String RetrieveNotClosedByBed="Stay.LastNotClosedByBed";

    public static final String RetrieveByPatientIdNumber="Stay.ByPatientIdNumber";


    public static final String BED_PARAM="bedParam";

    public static final String CHECK_OUT_FROM_PARAM="checkOutFromParam";

    public static final String CHECK_OUT_TO_PARAM="checkOutToParam";

    public static final String CHECK_IN_FROM_PARAM="checkInFromParam";

    public static final String CHECK_IN_TO_PARAM="checkInToParam";


    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String comments;

    @JoinColumn(name = "PATIENT", referencedColumnName = "ID", nullable =
false)

    @ManyToOne(optional = false, fetch = FetchType.LAZY)

    private Patient patient;

    @Basic(optional = false)

    @JoinColumn(name = "bed", referencedColumnName = "id", nullable = false,
updatable = false)

    @ManyToOne(optional = false, fetch = FetchType.LAZY)

    protected Bed bed;

    /**

     * Ιατρικά δεδομένα

     */

    @Column(length = 500)

    private String information;

    /**
```

```
 * Ημερομηνία εισαγωγής
 */
@Temporal(TemporalType.DATE)
private Date checkIn;
/**
 * Ημερομηνία εξαγωγής
 */
@Temporal(TemporalType.DATE)
private Date checkOut;

/**
 * Τηλέφωνο συνοδού
 */
private String partenerPhone;

/**
 * Ονοματεπώνυμο συνοδού
 */
private String partenetName;

/**
 * Τμήμα στο οποίο θα νοσηλευτεί ο ασθενής
 */
@Transient
private Department department;

public Department getDepartment() {
   return department;
}
```

```
public void setDepartment(Department department) {
    this.department = department;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getComments() {
    return comments;
}

public void setComments(String comments) {
    this.comments = comments;
}

public Patient getPatient() {
    return patient;
}

public void setPatient(Patient patient) {
    this.patient = patient;
}
```

```
public Bed getBed() {
    return bed;
}

public void setBed(Bed bed) {
    this.bed = bed;
}

public Date getCheckIn() {
    return checkIn;
}

public void setCheckIn(Date checkIn) {
    this.checkIn = checkIn;
}

public Date getCheckOut() {
    return checkOut;
}

public void setCheckOut(Date checkOut) {
    this.checkOut = checkOut;
}

public String getInformation() {
    return information;
}

public void setInformation(String information) {
```

```
        this.information = information;
    }

    public String getPartenerPhone() {
        return partenerPhone;
    }

    public void setPartenerPhone(String partenerPhone) {
        this.partenerPhone = partenerPhone;
    }

    public String getPartenetName() {
        return partenetName;
    }

    public void setPartenetName(String partenetName) {
        this.partenetName = partenetName;
    }
}
```

2.10 Tameio.java

```
package org.hospital.beds.model;

public enum Tameio {

    IKA,
```

```
   OAE,
   DHMOSIO
}
```

2.11 User.java

```
package org.hospital.beds.model;

import javax.persistence.*;
import static org.hospital.beds.model.User.*;



@Entity
@Table(
name="A_User",
uniqueConstraints =
@UniqueConstraint(columnNames = "userName"))
@NamedQueries(
   @NamedQuery(name = UserRetrieve, query = "SELECT a FROM User a WHERE
a.userName=:"+USERNAME_PARAM+" AND
a.password=:"+PASSWORD_PARAM)
)
public class User extends MyEntity {


   public static final String UserRetrieve = "User.Retrieve";
   public static final String USERNAME_PARAM = "username";
   public static final String PASSWORD_PARAM = "password";



   @Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private long id;
@Column(nullable = false)
private String userName;
private String password;
private String firstName;
private String lastName;
private boolean isAdmin=false;

public boolean getAdmin() {
   return isAdmin;
}

public void setAdmin(boolean admin) {
   this.isAdmin = admin;
}

public String getFirstName() {
   return firstName;
}

public void setFirstName(String firstName) {
   this.firstName = firstName;
}

public void setId(long id) {
   this.id = id;
}
```

```
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    @Override
    public Long getId() {
        return id;
    }
}
```

2.12 Ward.java

```
package org.hospital.beds.model;

import javax.persistence.*;
import org.hospital.beds.application.ApplicationContextBeanProvider;
import org.hospital.beds.control.Dao;

/**
 * Θάλαμος
  */
@Entity
@NamedQueries(
{
   @NamedQuery(name=Ward.RetrieveWards,query="SELECT object(a) FROM
Ward a")
}
)
public class Ward extends MyEntity {

   public static final String RetrieveWards="Ward.All";
   /**
    * Κωδικός
    */
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;
   /**
    * Πτέρυγα
```

```
     */
    @Basic(optional = false)
    @JoinColumn(name = "wing", referencedColumnName = "id", nullable = false,
updatable = false)
    @ManyToOne(optional = false, fetch = FetchType.EAGER)
    private Wing wing;
    /**
     * Ονομασία
     */
    private String title;

    @Transient
    private int totalBeds;

    @Transient
    private int availableBeds;

    private static final long serialVersionUID = 1L;

    public Ward() {
        super();
    }

    public Long getId() {
        return this.id;
    }

    public void setId(Long id) {
        this.id = id;
```

```
}

public Long getCode(){
   return id;
}

public void setCode(Long code){}

public String getTitle() {
   return this.title;
}

public void setTitle(String title) {
   this.title = title;
}

public Wing getWing() {
   return wing;
}

public void setWing(Wing wing) {
   this.wing = wing;
}

public int getAvailableBeds() {
   return availableBeds;
}

public void setAvailableBeds(int availableBeds) {
```

```
        this.availableBeds = availableBeds;
    }

    public int getTotalBeds() {
        return totalBeds;
    }

    public void setTotalBeds(int totalBeds) {
        this.totalBeds = totalBeds;
    }

    @PostLoad
    public void afterLoad(){
        Dao dao=(Dao)ApplicationContextBeanProvider.getBean("dao");
         setAvailableBeds(dao.countAvaibleBedsInWard(this));
         setTotalBeds(dao.countAllBedsInWard(this));
    }
}
```

2.13 Wing.java

```
package org.hospital.beds.model;

import javax.persistence.*;

/**
 * Πτέρυγα
 */
@Entity
```

```
public class Wing extends MyEntity {


    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    /**
     * Τμήμα όπου ανήκει
     */
    @Basic(optional = false)
    @JoinColumn(name = "department", referencedColumnName = "id", nullable =
false, updatable = false)
    @ManyToOne(optional = false, fetch = FetchType.EAGER)
    private Department department;
    /**
     * Ονομασία
     */
    private String title;
    /**
     * Κατηγορία  πολυτελείας
     */
    @Enumerated(EnumType.STRING)
    @Basic(optional = false)
    private Category category;
    private static final long serialVersionUID = 1L;


    public Wing() {
        super();
    }
```

```
public Long getId() {
    return this.id;
}

public void setId(Long id) {
    this.id = id;
}

public Long getCode() {
    return id;
}

public void setCode(Long code) {
}

public String getTitle() {
    return this.title;
}

public Department getDepartment() {
    return department;
}

public void setDepartment(Department department) {
    this.department = department;
}

public void setTitle(String title) {
    this.title = title;
```

```
    }

    public Category getCategory() {
        return category;
    }

    public void setCategory(Category category) {
        this.category = category;
    }
}
```

2.14 Service.java

```
package org.hospital.beds.ws;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import org.hospital.beds.application.ApplicationContextBeanProvider;
import org.hospital.beds.control.Dao;
import org.hospital.beds.exceptions.NotAvailableBeds;
import org.hospital.beds.model.*;
import org.springframework.transaction.annotation.Transactional;


@WebService
@Transactional
```

```
public class Service {

    protected Dao dao = (Dao) ApplicationContextBeanProvider.getBean("dao");

    @WebMethod
    public ArrayList<Bed> getAvailableBeds() {
        return dao.getAvailableBeds();
    }

    @WebMethod
    public Patient savePatient(@WebParam(name = "patient")Patient patient) {
        String idNumber=patient.getIdNumber();
        Patient existingPatient = dao.retrievePatient(idNumber);
        if (existingPatient != null) {
            patient.setId(existingPatient.getId());
            dao.merge(patient);
        } else {
            dao.persist(patient);
        }
        return patient;
    }

    @WebMethod
    public Patient retrievePatient(@WebParam(name = "patientId") String patientId) {
        return dao.retrievePatient(patientId);
    }

    @WebMethod
    public ArrayList<Bed> getAvailableBedsByDepartmentAndCategory(
```

```
        @WebParam(name = "departmentId") String departmentId,
        @WebParam(name = "category") Category category) {
    Department department = dao.retrieveDepartment(departmentId);
    return dao.getAvailableBedsByDepartmentAndCategory(department, category);
}

@WebMethod
public ArrayList<Bed> getReservedBeds() {
    return dao.getReservedBeds();
}

@WebMethod
public Bed getCurrentBed(@WebParam(name = "patientId") String patientId) {
    Patient patient = dao.retrievePatient(patientId);
    return dao.getCurrentBed(patient);
}

@WebMethod
public Stay getCurrentStay(@WebParam(name = "patientId") String patientId) {
    return dao.getCurrentStay(patientId);
}

public ArrayList<Department> getDepartments() {
    return dao.getDepartments();
}

@WebMethod
public ArrayList<Patient> getHostedPatients() {
    return dao.getHostedPatients();
```

```
    }

    @WebMethod
    public ArrayList<Patient> getPatients() {
        return dao.getPatients();
    }

    @WebMethod
    public User retrieveUser(@WebParam(name = "username") String userName,
@WebParam(name = "password") String password) {
        return dao.retrieveUser(userName, password);
    }

    @WebMethod
    public void saveUser(@WebParam(name = "user") User user) {
        dao.saveUser(user);
    }

    @WebMethod
    public void reserve(
            @WebParam(name = "bedCode") String bedCode,
            @WebParam(name = "patientIdNumber") String patientIdNumber) {
        dao.reserve(bedCode, patientIdNumber);
    }

    @WebMethod
    @Transactional
    public Stay saveStay(@WebParam(name = "stay") Stay stay) throws
NotAvailableBeds {
        stay.setPatient(savePatient(stay.getPatient()));
```

```
        return dao.saveStay(stay);
    }

    @WebMethod
    public void updateStay(@WebParam(name = "patientIdNumber") String patientId,
            @WebParam(name = "comments") String comments) {
        dao.updateStay(patientId, comments);
    }

    @WebMethod
    public void closeStay(@WebParam(name = "patientIdNumber") String
patientIdNumber) {
        dao.closeStay(patientIdNumber);
    }

    @WebMethod
    public String initData() {
        return dao.initData();
    }

    /**
     * Web service operation
     */
    @WebMethod()
    public ArrayList<Ward> getAllWards() {
        return dao.getAllWards();
    }

    @WebMethod()
```

```
    public ArrayList<Bed> getBedsInWard(@WebParam(name = "wardCode") Long
wardCode){

        Ward ward=dao.retrieveWard(wardCode);

        return dao.getBedsInWard(ward);

    }


    @WebMethod()

    public ArrayList<Stay> getStayHistory(@WebParam(name = "patientIdNumber")
String patientIdNumber){

       return dao.getStayHistory(patientIdNumber);

    }


    @WebMethod()

    public ArrayList<Stay> retrieveStayStatistics(@WebParam(name = "from")Date
from,@WebParam(name = "to") Date to,@WebParam(name = "isForStartDay")
boolean isForStartDay){

     Calendar cal=Calendar.getInstance();

     cal.set(Calendar.YEAR,1900);

     Date minFrom=cal.getTime();

     cal.set(Calendar.YEAR,2900);

     Date maxTo=cal.getTime();


        return dao.retrieveStayStatistics(from==null?minFrom:from,
to==null?maxTo:to, isForStartDay);

    }

}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Σημειώσεις μαθήματος «ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ»
2. Ανάπτυξη και διαχείριση πληροφοριακών συστημάτων, Διομήδης Σπινέλλης (http://dmst.aueb.gr/dds/ism/index.htm)
3. Unified Modeling Language (UML), Wikipedia (http://en.wikipedia.org/wiki/Unified_Modeling_Language)
4. IBM (2007), "New to SOA and Web services" (http://www-128.ibm.com/developerworks/webservices/newto/websvc.html)
5. MSDN (2001), "XML Web Services Basics" (http://msdn2.microsoft.com/en-us/library/ms996507.aspx)
6. Roseindia.net (2007), "Web Services" (http://www.roseindia.net/webservices/webservices.shtml)
7. Developer.com (2003), "Web Services Tutorial: Understanding XML and XML Schema"
   "Part 1" http://www.developer.com/services/print.php/2195981
   "Part 2" http://www.developer.com/services/print.php/10928_2195981_2
8. Developer.com (2002), "Introduction to Web Services"
   "Part 1" http://www.developer.com/services/article.php/1485821
   "Part 2: Architecture"
   http://www.developer.com/services/article.php/1495091
   "Part 3: Understanding XML"
   http://www.developer.com/services/article.php/1557871
9. ONJava.com (2001), "Java and Web Services Primer"
   http://www.onjava.com/lpt/a/1025
10. Horton I. (2003) "Beginning Java 2" : Wrox
11. W3schools.com (2007), "XML Tutorial"
    http://www.w3schools.com/xml/default.asp
12. W3C (2001), "Web Service Definition Language (WSDL)"
    http://www.w3.org/TR/wsdl
13. XML.com (2001), "A Web Services Primer"
    http://webservices.xml.com/lpt/a/760
14. UDDI.org (2004), "UDDI Version 3.0.2"
    http://uddi.org/pubs/uddi_v3.htm
15. W3C (2006), "SOAP Version 1.2"
    "Part 0: Primer" http://www.w3.org/TR/2006/PER-soap12-part0-20061219/
    "Part 1: Messaging Framework" http://www.w3.org/TR/soap12-part1/

"Part 2: Adjuncts" http://www.w3.org/TR/2003/REC-soap12-part2-20030624/

16. Borgen B. (2001) "SOAP Programming with Java" : Sybex Inc
17. W3schools.com (2007), "SOAP Tutorial" http://www.w3schools.com/soap/default.asp
18. Newcomer E., Lomow G (2005) "Understanding SOA with Web Services" : Addison-Wesley Professional
19. Δημητριάδης Αντώνης, **"Διοίκηση-Διαχείριση πληροφοριακών συστημάτων"**, Εκδόσεις Νέων Τεχνολογιών, Αθήνα 1998.
20. Κιουντούζης Ευάγγελος, **"Ανάλυση και Σχεδιασμός Συστημάτων"**, Εκδόσεις Μπένου, Αθήνα 1993.
21. Βασιλακόπουλος Γ.-Χρυσικόπουλος Β., **"Πληροφοριακά Συστήματα Διοίκησης"**, Σταμούλης, Πειραιάς 1990.
22. Παιδαγωγικό Ινστιτούτο, **"Πληροφοριακά Συστήματα"**, Λιβάνη, Αθήνα 2000.
23. Κιουντούζης Ευάγγελος, **"Ασφάλεια πληροφοριακών Συστημάτων"**, Εκδόσεις Μπένου, Αθήνα 1993.
24. Γρίβας Β., Κουκούμας Ν., Ξανθόπουλος Κ., Σφυρής Ν., Χρυσοχοΐδης Ι., "**Οικονομική και Χρηματοδοτική Διαχείριση Υπηρεσιών Υγείας"**, Ελληνικό Ανοικτό Πανεπιστήμιο.
25. Δελημπάσης Κων., Νικηφορίδης Γεώρ., **" Ιατρική Πληροφορική"**, Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα 2001.
26. Ινστιτούτο έρευνας και τεχνολογίας τμήμα Πληροφορικής, "Ε1 - Εισαγωγή στο έργο και το ΟΠΣΥ Α ΔΥΠΕ ΑΤΤΤΙΚΗΣ" www.ics.forth.gr

Internet Sites

27. www.ahima.org/coding/
28. http://alpha.mpl.uoa.gr/greektel/greek%20Handbook%20part%201html
29. www.mednet.gr/greek/depts/plomari/EHCR_Description_Greek.htm
30. www.datamed.gr/up/files/medicoslong.pdf
31. www.datamed_gr/news/index.asp?CAT_10=22
32. www.iatroclub.gr/enc.htm
33. www.multimedia.mmm.com/mws/mediawebserver.dyn?88888zmdpi18Rb

34. www.eicd.com/icd-9-cm.htm
35. www.ohiobwc.com/worker/brochureware/genforms/ICD-9CodingDescription.asp
36. www.hiercode.com/icd9cm.html
37. www.mcis.duke.edu/standards/HL7/hl7.htm
38. www.cdc.gov/nip/registry/download/hl723.pdf
39. www.healthcare-informatics.com/issues/2000/04_00/hl7.htm
40. www.iwayoftware.com/solutions/healthcare/pdf/HL7_techbrier.pdr
41. www.medicalcomputing.today.com/Oophl7.html
42. www.va.gov/publ/standard/health/HL7.htm
43. www.iatriko.com
44. www.docmem.net
45. www.imds.com
46. www.ccs.gr/iatrikh/proionta/e-AIMA/index.asp
47. www.sarnoff.com/products_services/healthcare_lifesci/diagnostic/index.as€
48. www.kyanousstavros.gr
49. www.4peiraias.gr/pages/Nosokomia.asp
50. www.computersolution.gr
51. www.altec.gr
52. www.guide.pathfinder.gr/guide?category=339&parent=1
53. www.who.org
54. www.cpri.org
55. www.nema.org
56. www.mri.org
57. www.biomed.ntua.gr
58. www.biomed.ntua.gr/ippokratis/activities/Hmerides/hmerida1/programma/programma.html
59. www.esy.gr
60. www.apollo.gr
61. www.techmed.teiher.gr/cd%20PSE
62. http://medlab.cs.uoi.gr/RISE/RISEoffice/Isexamples.htm
63. www.cc.uoa.gr/health/socmed/hygien/iatrplirof/main-gr.htm#1
64. www.ekdd.gr/docs/ESDD/Sem_Erg_ID/Koin_Dioik/Ypir_Ygeias/s0204d.pd
65. www.datamed.gr/up/files/xcartapdf.
66. www.ypyp.gr
67. www.findarticles.com
68. www.pliktro.gr
69. www.conta.uom.gr/conta/ekpaideysh/metaptyxiaka/technologies_diktywn
70. www.ifet.gr/e_list/atc.htm
71. http://www.scribd.com/doc/6471701/Rethinking-the-hospital-The-value-of-business-models-for-hospitals