ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Διδακτικής της Τεχνολογίας και Ψηφιακών Συστημάτων

Μεταπτυχιακό Τμήμα

Κατεύθυνση: Ψηφιακές Επικοινωνίες & Δίκτυα

Διπλωματική Εργασία

Speech (Audio)-based Perceptual Services for Context-Awareness in Intelligent Environments

ΣΥΝΤΑΚΤΗΣ:

Τσαρμπόπουλος Κυριάκος, ΜΕ/0423,

k.tsarmpopoulos@gmail.com

Νοέμβριος 2006

Περίληψη

Η παρούσα εργασία αποτελεί την προσπάθεια υλοποίησης μιας εφαρμογής η οποία προσφέρει την δυνατότητα αναγνώρισης φωνής και λειτουργεί πάνω σε μία πλατφόρμα επικοινωνίας agents με αναβαθμισμένα χαρακτηριστικά που επιτρέπουν την προσομοίωση της ανθρώπινης συμπεριφοράς κατά την λειτουργία της εφαρμογής.

Γίνεται ανάλυση του θεωρητικού υποβάθρου των χρησιμοποιουμένων τεχνολογιών που είναι η αναγνώριση φωνής και η πλατφόρμα των agents αλλά και των ίδιων των τεχνολογιών που είναι το sphinx-4 και το Jadex με σκοπό να γίνει η αποσύνθεση τους και η κατανόηση τους σε βάθος όπως ακριβώς έγινε και κατά την διαδικασία της υλοποίησης της εφαρμογής.

Η θεωρητική ανάλυση συνοδεύεται από την υλοποίηση της εφαρμογής, η οποία έχει μορφή εγγράφου προδιαγραφών λογισμικού και έχει συνταχθεί με τέτοιον τρόπο ώστε να περιφγραφτούν με τον καλύτερο δυνατό τρόπο οι διαδικασίες που ακολουθήθηκαν για την υλοποίηση αυτή. Παρουσιάζονται με αναλυτικό τρόπο οι προσαρμογές, αλλαγές και τροποποιήσεις που πραγματοποιήθηκαν στην μηχανή αναγνώρισης φωνής – sphinx4, στην πλατφόρμα επικοινωνίας με βάση agents – Jadex, στην δημιουργία του δυναμικού γραφικού περιβάλλοντος λειτουργίας και στην ενοποίηση όλων αυτών με σκοπό την δημιουργία ενός τελικού προϊόντος που να ανταποκρίνεται στις ανάγκες δημιουργίας του. Όλες οι τεχνολογίες συνεργάζονται μέσω της αντικειμενοστραφούς γλώσσας προγραμματισμού Java, που είναι ανεξάρτητη από αρχιτεκτονικές συστημάτων και έτσι μπορεί η συγκεκριμένη εφαρμογή να χρησιμοποιηθεί σε οποιοδήποτε σύστημα και αρχιτεκτονική.

Φυσικά κατά την πορεία της υλοποίησης υπάρχουν ενέργειες και ιδέες που δεν τελεσφόρησαν και οποίες αναλύονται συνοδευόμενες από τους λόγους για τους οποίους δεν ολοκληρώθηκαν.

Τέλος υπάρχουν κάποιες προτάσεις για μελλοντικές ενέργειες που μπορούν να γίνουν για να μπορέσει η παρούσα εφαρμογή να ενσωματωθεί σε μεγαλύτερα projects.

Περιεχόμενα

Κατάλογος Σχημάτων

1. Εισαγωγή

Η παρούσα εργασία είναι μια προσπάθεια υλοποίησης μιας εφαρμογής η οποία να προσφέρει αναγνώριση περιεχομένου και πιο συγκεκριμένα αναγνώριση φωνητικών εντολών μέσα σε ένα ‘έξυπνο’ περιβάλλον λειτουργίας. Το περιβάλλον αυτό θα πρέπει να προσφέρει εξελιγμένες δυνατότητες επεξεργασίας δεδομένων που να είναι πολύ κοντά σε ανθρώπινες συμπεριφορές και δραστηριότητες όπως είναι η σκέψη, η αίσθηση του περιβάλλοντος λειτουργίας και η απόφαση βάση των εξελίξεων που γίνονται.

Η εφαρμογή αυτή είναι αποτέλεσμα μελέτης διαφόρων τεχνολογιών που εκπληρώνουν τις απαιτήσεις της. Δηλαδή έχει γίνει ενδελεχής μελέτη σε πεδία όπως είναι η αναγνώριση φωνής καθώς και η πλατφόρμα εξελιγμένων δυνατοτήτων επικοινωνίας που έχουν χρησιμοποιηθεί. Με βάση το θεωρητικό υπόβαθρο που αποκτήθηκε από την μελέτη των τεχνολογιών δόθηκε η ευκαιρία για υλοποίηση της εφαρμογής χρησιμοποιώντας όλες εκείνες τις απαραίτητες αρχές των τεχνολογιών που εκπλήρωναν τις απαιτήσεις αλλά και προσαρμογή κάποιων παραμέτρων που έπρεπε να γίνουν για να προσαρμοστεί η όλη εφαρμογή με ένα γενικότερο περιβάλλον λειτουργίας.

Η εργασία αποτελείται από διάφορα τμήματα που έχουν σαν στόχο την ανάλυση τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο της δουλειάς που έγινε για την υλοποίηση της εφαρμογής αυτής αλλά και της γνώσης που αποκτήθηκε κατά την διάρκεια της υλοποίησης αυτής.

Καταρχήν αναλύεται σε μικρό βαθμό το γενικότερο περιβάλλον λειτουργίας της συγκεκριμένης εφαρμογής. Δηλαδή ποιες είναι οι ανάγκες δημιουργίας της καθώς και με ποια άλλα συστήματα πρέπει να συνεργάζεται ώστε να υπάρξουν τα επιθυμητά αποτελέσματα.

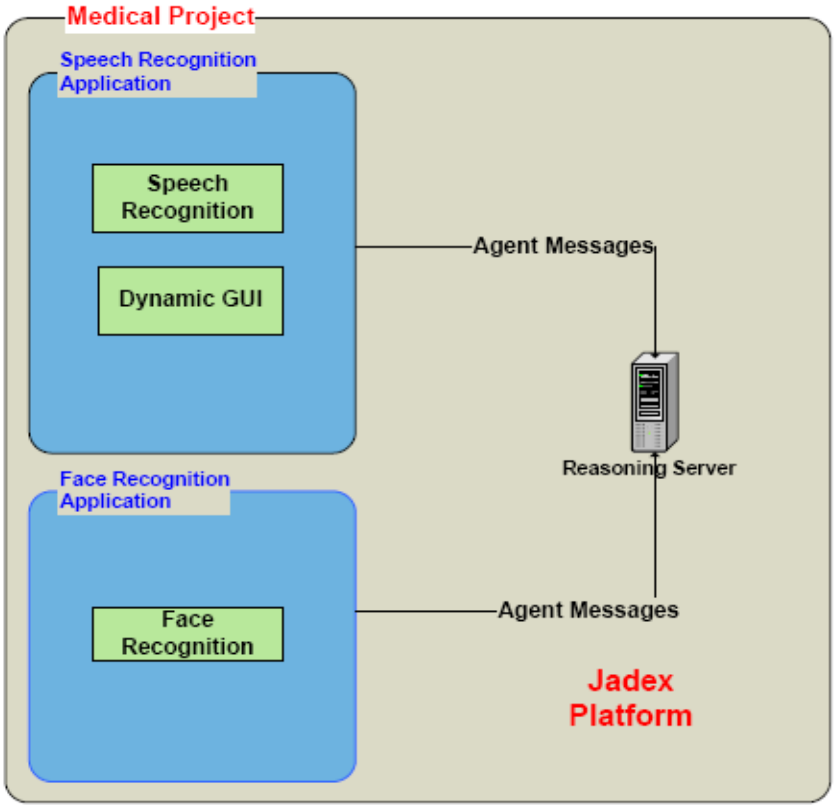
Στην συνέχεια ακολουθεί η ανάλυση του θεωρητικού υποβάθρου που χρησιμοποιήθηκε στην εργασία αυτή. Το θεωρητικό αυτό υπόβαθρο χρησιμοποιήθηκε σε όλη την φάση της υλοποίησης της εφαρμογής και είναι πολύ σημαντικό να δούμε ποιες αρχές και έννοιες διέπουν τις τεχνολογίες που χρησιμοποιήθηκαν για να μπορούμε να καταλάβουμε καλύτερα το πρακτικό κομμάτι που ακολουθεί.

Στην συνέχεια υπάρχει η ανάλυση της ίδιας της εφαρμογής. Αναλύονται στοιχεία όπως είναι οι απαιτήσεις της εφαρμογής όπως αυτές δημιουργήθηκαν είτε από εξωτερικούς παράγοντες όπως είναι για παράδειγμα η συνεργασία με άλλα συστήματα, είτε από εσωτερικούς παράγοντες όπως είναι κάποιοι περιορισμοί που θέτει ο συνδυασμός των χρησιμοποιούμενων τεχνολογιών. Υπάρχει επίσης ανάλυση της αρχιτεκτονικής της εφαρμογής με κατάλληλα σχεδιαγράμματα που βοηθούν στην κατανόηση της δομής της εφαρμογής. Ακολουθεί ένα μεγάλο κομμάτι που αφορά την ίδια την υλοποίηση της εφαρμογής και περιλαμβάνει στοιχεία όπως είναι η περιγραφή διαφόρων σημαντικών κομματιών ακόμα και σε επίπεδο κώδικα. Εδώ περιλαμβάνονται και μεθοδολογίες που χρησιμοποιήθηκαν μαζί με τα αποτελέσματα που προσφέρουν.

Τέλος υπάρχει ένα είδος οδηγού εγκατάστασης και εκτέλεσης που συνοδεύεται με διάφορες φωτογραφίες της εφαρμογής κατά την ώρα της εκτέλεσης ώστε να υπάρχει και οπτική αναφορά στις περιγραφές. Εδώ βρίσκονται και μερικές συμβουλές και αναφορές για μελλοντική εξέλιξη και χρήση της εφαρμογής για κάποιες ιδιαίτερες συνθήκες που μπορούν να είναι χρήσιμες για την ύπαρξη αυτής της εφαρμογής σε κάποια άλλα συστήματα με ελαφρώς διαφορετικές απαιτήσεις καθώς και αναφορές για μεθοδολογίες που χρησιμοποιήθηκαν και δεν ήταν επιτυχείς μαζί με τα αποτελέσματα τους ή τους λόγους που οδήγησαν στην αποτυχία.

2. Γενικότερο περιβάλλον λειτουργίας

Το γενικότερο περιβάλλον λειτουργίας είναι ένα μεγαλύτερο ιατρικό project που απαρτίζεται από τρία επιμέρους στοιχεία-εφαρμογές. Τα στοιχεία αυτά φαίνονται στο παρακάτω σχεδιάγραμμα:



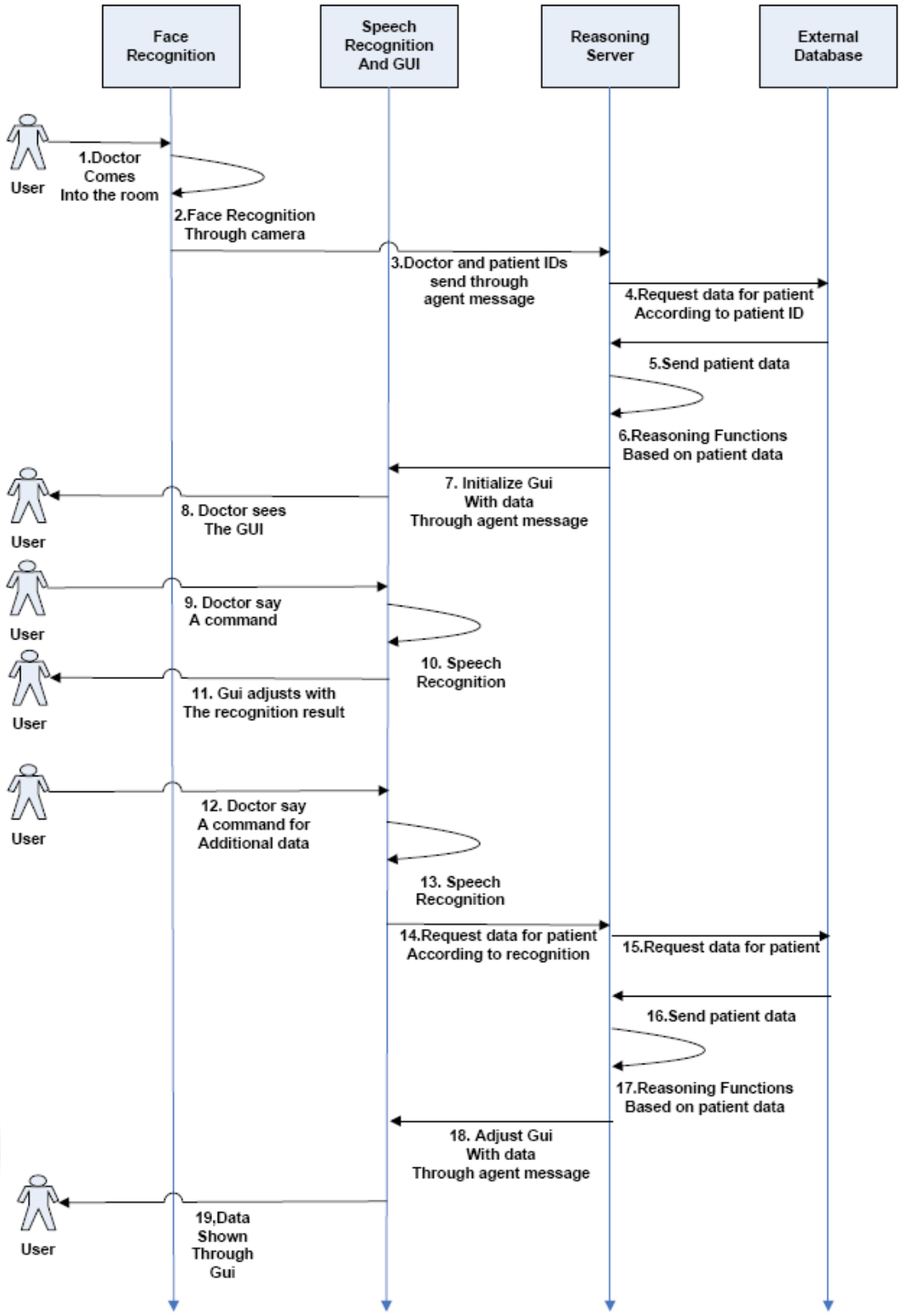
Σχήμα 2.1. Δομικά στοιχεία του γενικότερου project

Τα τρία λοιπόν στοιχεία που απαρτίζουν το γενικότερο ιατρικό project όπως φαίνονται και από το παραπάνω σχεδιάγραμμα είναι :

- **Speech Recognition Application:** Είναι μια εφαρμογή που είναι υπεύθυνη για αναγνώριση φωνητικών εντολών και λειτουργεί πάνω στην Jadex πλατφόρμα. Συνοδεύεται η όλη διαδικασία από δυναμικό γραφικό περιβάλλον που προσαρμόζεται στις φωνητικές εντολές που αναγνωρίζονται. Είναι η εφαρμογή που έχει υλοποιηθεί σε αυτήν την εργασία.
- **Face Recognition Application:** Είναι μια εφαρμογή που είναι υπεύθυνη για αναγνώριση προσώπων από κάποια κάμερα και λειτουργεί πάνω στην Jadex πλατφόρμα.
- **Reasoning Server:** Είναι ένας server ο οποίος εκτελεί τις εξελιγμένες δυνατότητες επεξεργασίας δεδομένων που αφορούν τις άλλες δύο εφαρμογές και αλληλεπιδρά μαζί τους μέσω agent μηνυμάτων αφού τρέχει και αυτό πάνω στην Jadex πλατφόρμα.

Έτσι λοιπόν το ιατρικό project αυτό είναι μια συνολική υλοποίηση για μια ολοκληρωμένη ιατρική εφαρμογή που θα μπορεί να προσφέρει σε γιατρούς πολύ εξελιγμένες δυνατότητες με έναν διαφανή τρόπο ώστε να γίνονται όλες οι διαδικασίες χωρίς ο τελικός χρήστης να καταλαβαίνει τι ακριβώς γίνεται αλλά να βλέπει μόνο τα αποτελέσματα που τον ενδιαφέρουν. Ο σκοπός της δημιουργίας αυτού του project είναι να απλοποιηθούν ιατρικές διαδικασίες όπως είναι η γραφειοκρατία και η απόλυτη γνώση του φακέλου κάποιου ασθενή ανά πάσα στιγμή από τον γιατρό καθώς και διάφορες άλλες ενέργειες όπως η αυτόματη ενημέρωση με νέες εξετάσεις και θεραπείες που αφορούν τους ασθενείς.

Για να αντιληφθούμε όμως την λειτουργικότητα του όλου project ας δούμε παρακάτω μια περίπτωση χρήσης με όλα τα δομικά του στοιχεία και πως αυτά αλληλεπιδρούν μεταξύ τους.



Σχήμα 2.2 Περίπτωση χρήσης του γενικότερου project

Η εφαρμογή που έχει σαν στόχο να υλοποιήσει η παρούσα εργασία είναι το κομμάτι Speech Recognition Application. Θα παρουσιάσουμε στην συνέχεια της εργασίας τις απαιτήσεις, την αρχιτεκτονική της, τον τρόπο υλοποίησης της καθώς και τον τρόπο λειτουργίας της και μελλοντικές ενέργειες που μπορούν να γίνουν για να μπορέσει να προσαρμοστεί σε νέες απαιτήσεις που τυχόν προκύψουν.

Στο σημείο όμως αυτό θα πρέπει να αναφέρουμε το θεωρητικό υπόβαθρο της εργασίας και την ανάλυση των τεχνολογιών που χρησιμοποιήθηκαν για να μπορέσουμε να καταλάβουμε καλύτερα τον ρόλο τους αλλά και τις δυνατότητες τους στα πλαίσια της εφαρμογής μας.

3. Θεωρητικό υπόβαθρο

3.1 Πώς λειτουργεί η λεκτική αναγνώριση

3.1.1 Επισκόπηση

Παρακάτω ακολουθεί μια σύντομη περιγραφή της λεκτικής αναγνώρισης. Περιγράφονται οι θεμελιώδεις αρχές και διαδικασίες που πρέπει να ακολουθούνται από κάθε σύστημα που προσφέρει λεκτική αναγνώριση. Αυτό γίνεται για να μπορέσουμε να καταλάβουμε καλύτερα τον τρόπο που κάνει την αναγνώριση το sphinx που είναι η χρησιμοποιούμενη μηχανή αναγνώρισης στην υλοποίηση της εφαρμογής, καθώς και να δούμε τα δομικά του στοιχεία μέσα από ευρύτερες έννοιες.

Η λεκτική αναγνώριση λειτουργεί με μετατροπή του ψηφιακού ήχου PCM (Pulse Code Modulation) που παίρνουμε μέσω κάποιας κάρτας ήχου από κάποιον ομιλητή και μια ακολουθία ενεργειών που πρέπει να γίνουν για να έχουμε την επιθυμητή αναγνώριση. Η ακολουθία αυτή ενεργειών είναι:

1. Μετασχηματισμός του ψηφιακού ήχου PCM σε μια καλύτερη ακουστική αντιπροσώπευση. Δηλαδή κάποιας μορφής encoding ώστε να μπορέσουμε να το χρησιμοποιήσουμε στην συνέχεια.
2. Δημιουργία μιας "γραμματικής" έτσι ώστε το λεκτικό σύστημα αναγνώρισης να ξέρει ποια φωνήματα να αναμένει. Η γραμματική αυτή θα μπορούσε να είναι μια λίστα από όλες τις δυνατές λέξεις που μπορεί να αναγνωρίσει το σύστημα μας. Για παράδειγμα σαν ένα λεξικό που πρέπει να χρησιμοποιηθεί.

Φυσικά εδώ περιλαμβάνεται και η αντιστοίχηση των πιθανών λέξεων με τα δομικά τους στοιχεία. Τα δομικά αυτά στοιχεία είναι τα απλά γράμματα, τα διπλά γράμματα όπως δίφθογγοι και τα τριπλά γράμματα που είναι για παράδειγμα κάποια συνδυασμοί τριών γραμμάτων που όμως έχουν κάποια ιδιαιτερότητα όταν προφέρονται μαζί. Τα παραπάνω ονομάζονται φωνήματα (ή phonemes).

3. Εύρεση και αναγνώριση των φωνημάτων από τον ψηφιακό ήχο.
4. Μετατροπή των φωνημάτων σε λέξεις με την χρήση των βημάτων 2 και 3. Δηλαδή γίνεται η αναγνώριση των φωνημάτων και η αντιστοίχηση με το μοντέλο της γραμματικής για να μπορέσει να επιλεχθεί ή καλύτερη λέξη που βρίσκεται σε αυτό. Όσο καλύτερο είναι το γραμματικό μοντέλο και η διαδικασία της αναγνώρισης των φωνημάτων τόσο καλύτερη είναι η αναγνώριση των λέξεων που γίνεται.

Παρακάτω αναλύονται περισσότερο οι παραπάνω ενέργειες.

3.1.2 Μετασχηματισμός του ψηφιακού ήχου PCM

Το πρώτο στοιχείο της μετατροπής είναι να γίνει ένας μετασχηματισμός του ψηφιακού ήχου που προέρχεται από την κάρτα ήχου σε μια πιο αντιπροσωπευτική μορφή που μοιάζει αρκετά με τον τρόπο που αντιλαμβάνεται το ανθρώπινο αυτί του ήχους σε φυσικό επίπεδο. Ο ψηφιακός ήχος είναι μια ροή πλατών, που είναι δειγματοληπτημένα με ρυθμό 16.000 δείγματα το δευτερόλεπτο από την φυσική πηγή ήχου. Εάν απεικονίσουμε τώρα τον ψηφιακό αυτόν ήχο, κοιτάζοντας την απεικόνιση αυτή βλέπουμε μια κυματιστή γραμμή που επαναλαμβάνεται περιοδικά ενώ ο χρήστης μιλά. Με αυτήν την μορφή, τα δεδομένα δεν είναι χρήσιμα στη λεκτική αναγνώριση επειδή είναι πάρα πολύ δύσκολο να προσδιοριστούν οποιαδήποτε μοτίβα που να μπορούν να συσχετίσουν αυτό που ειπώθηκε πραγματικά.

Για να επιτύχουμε καλύτερη αναγνώριση μοτίβων, ο ψηφιακός ήχος PCM θα πρέπει να μετασχηματιστεί από το πεδίο του χρόνου που είχαμε πριν (με τα πλάτη) στο πεδίο της συχνότητας. Οι μετασχηματισμοί αυτοί γίνονται χρησιμοποιώντας έναν μετασχηματισμό Windowed Fast-Fourier. Στο πεδίο της συχνότητας, μπορούμε να προσδιορίσουμε τα τμήματα συχνότητας ενός ήχου. Από τα τμήματα συχνότητας,

είναι δυνατό να προσεγγιστεί ο τρόπος που το ανθρώπινο αυτί αντιλαμβάνεται τον ήχο.

Έτσι λοιπόν σύμφωνα με τα παραπάνω πρέπει να γίνει μετατροπή του ψηφιακού ήχου όπως αυτός λαμβάνεται από κάποιον ομιλητή από την κάρτα ήχου. Η μετατροπή αυτή ουσιαστικά είναι η αναπαράσταση του ψηφιακού ήχου από το πεδίο του χρόνου στο πεδίο της συχνότητας με στόχο να μπορούμε να αναγνωρίσουμε μοτίβα ήχων. Τα μοτίβα αυτά που ονομάζονται χαρακτηριστικά συχνότητας αποτελούν τα φωνήματα που θα περιγράψουμε στη συνέχεια και αυτά με την σειρά τους αποτελούν λέξεις και μπορούμε έτσι διαδοχικά να φτάσουμε στο τελικό μας στόχο που είναι η αναγνώριση φωνής.

3.1.3 Εύρεση των φωνημάτων

Σε αυτό το σημείο καλό είναι να δούμε πως ο recognizer του κάθε συστήματος βρίσκει ποια είναι τα φωνήματα που λέει ο ομιλητής.

Σε έναν ιδανικό κόσμο, θα μπορούσαμε να ταιριάξουμε ένα αριθμό χαρακτηριστικών γνωρισμάτων σε ένα φώνημα. Εάν για παράδειγμα ένα τμήμα του ήχου αντιστοιχούσε στο χαρακτηριστικό συχνότητας με αριθμό 52, θα μπορούσε πάντα να σημάνει ότι ο χρήστης πρόφερε το "K". Αυτό πρακτικά σημαίνει ότι όταν κάποιος χρήστης μιλούσε και αναγνωριζόταν το χαρακτηριστικό γνώρισμα στο πεδίο της συχνότητας με τον αριθμό 52 θα είχε προφέρει το φώνημα “K”. Αυτό θα ήταν πολύ αποδοτικό γιατί θα μπορούσαμε να αντιστοιχίσουμε όλα τα φωνήματα με τα χαρακτηριστικά του ήχου στο πεδίο της συχνότητας και να κάνουμε μια ασφαλή αναγνώριση φωνής.

Κάτι τέτοιο όμως δεν συμβαίνει και οι λόγοι είναι οι εξής:

- Κάθε φορά που μιλά ένας χρήστης μια λέξη, ηχεί διαφορετική. Οι χρήστες δεν παράγουν ακριβώς τον ίδιο ήχο για το ίδιο φώνημα με αποτέλεσμα να υπάρχει μια μεταβλητότητα.
- Ο θόρυβος εξαιτίας των μικροφώνων και του περιβάλλοντος του χρήστη αναγκάζει μερικές φορές το σύστημα αναγνώρισης να αναγνωρίσει ένα διαφορετικό γνώρισμα από αυτό που θα είχε αναγνωρίσει εάν ο χρήστης ήταν

σε ένα ήσυχο περιβάλλον και χρησιμοποιούσε ένα υψηλής ποιότητας μικρόφωνο.

- Ο ήχος ενός φωνήματος αλλάζει ανάλογα με ποια φωνήματα το περιβάλλουν. Για παράδειγμα το φώνημα t είναι διαφορετικό στη λέξη "talk" και διαφορετικό στη λέξη "attack".
- Ο ήχος που παράγεται κατά την διάρκεια προφοράς ενός φωνήματος αλλάζει από την αρχή ως το τέλος του φωνήματος, και δεν είναι σταθερός. Αυτό σημαίνει ότι το γνώρισμα που θα αναγνωριστεί από το σύστημα δεν θα είναι ποτέ ακριβώς το ίδιο ακόμα και αν το φώνημα προφέρεται από τον ίδιο ομιλητή.

Τα προβλήματα του θορύβου που βάζει το περιβάλλον και το μικρόφωνο καθώς και η μεταβλητότητα λύνονται με την χρησιμοποίηση ενός συνόλου χαρακτηριστικών γνωρισμάτων που αντιστοιχίζονται σε περισσότερα από ένα φωνήματα και τη χρησιμοποίηση στατιστικών μεγεθών για να υπολογιστεί ποιο φώνημα μιλιέται. Αυτό μπορεί να γίνει επειδή ένα φώνημα διαρκεί για έναν σχετικά μεγάλο χρόνο, δηλαδή κατά την διάρκεια του εμφανίζονται περίπου 50 με 100 γνωρίσματα στο πεδίο της συχνότητας που μπορούν να ξεχωριστούν. Έτσι λοιπόν αντιστοιχίζουμε σε κάθε φώνημα ένα πλήθος χαρακτηριστικών γνωρισμάτων συχνότητας, που όμως αρκετά από αυτά τα γνωρίσματα θα ανήκουν και σε άλλα φωνήματα και μπορούμε με στατιστικές μεθόδους όπως οι πιθανότητες να οδηγηθούμε από το σύνολο των χαρακτηριστικών που έχουμε αναγνωρίσει στο αντίστοιχο φώνημα. Ως εκ τούτου, είναι δυνατό να προβλεφθεί ποιο φώνημα μιλήθηκε με αρκετά καλή πιθανότητα αλλά και αρκετούς υπολογισμούς.

Φυσικά ο παραπάνω τρόπος αναγνώρισης των φωνημάτων είναι κάπως γενικευμένος και θα πρέπει στο σημείο αυτό να δώσουμε μερικές πληροφορίες για το πώς λειτουργεί ένα ολοκληρωμένο σύστημα αναγνώρισης και να μπορέσουμε να καταλάβουμε σε ποιο πρακτικό επίπεδο πως γίνεται η αναγνώριση των φωνημάτων που είναι και η βάση ενός συστήματος αναγνώρισης.

Το σύστημα λεκτικής αναγνώρισης για να μάθει πώς ένα φώνημα ηχεί, πρέπει να χρησιμοποιηθεί κάποιο εργαλείο "εκπαίδευσης" (training tool). Στο εργαλείο αυτό περνάνε εκατοντάδες καταγραφές ήχου (recordings) που αφορούν κάποιο

συγκεκριμένο φώνημα. Το εργαλείο αυτό αναλύει κάθε καταγραφή και φτιάχνει ένα χαρακτηριστικό γνώρισμα για την συχνότητα κάθε 1/100 του δευτερολέπτου. Δηλαδή αν για παράδειγμα έχουμε 100 καταγραφές ήχου όπου η κάθε μια διαρκεί 1 δευτερόλεπτο και όλες αφορούν το ίδιο φώνημα, θα έχουμε σαν αποτέλεσμα από το εργαλείο εκπαίδευσης 10000 χαρακτηριστικά συχνότητας που αφορούν το φώνημα αυτό. Βέβαια εκτός από την δημιουργία των χαρακτηριστικών γνωρισμάτων συχνότητας για το φώνημα το εργαλείο εκπαίδευσης κάνει μια πολύ σημαντικότερη δουλειά φτιάχνει στατιστικούς πίνακες με το πόσο πιθανό είναι να εμφανιστεί κάποιο χαρακτηριστικό συχνότητας σε ένα φώνημα από όλες τις καταγραφές. Έτσι λοιπόν με τη χρήση του εργαλείου αυτού έχουμε κάποια στατιστικά μεγέθη που μπορούν να μας οδηγήσουν σε αρκετά καλά αποτελέσματα αναγνώρισης. Βέβαια εδώ παίζει πολύ σημαντικό ρόλο ο αριθμός των καταγραφών, η ποιότητα τους –αν για παράδειγμα χρησιμοποιούνται διάφορες προφορές ή ιδιαιτερότητες του λόγου-, η αποτελεσματικότητα του ίδιου του εργαλείου εκπαίδευσης και οι μέθοδοι που χρησιμοποιούνται για την παραγωγή των στατιστικών μεγεθών αυτών.

Ας δούμε όμως στο σημείο αυτό ένα παράδειγμα αναγνώρισης κάποιου φωνήματος πχ. του φωνήματος “h”. Στην αρχή περνάμε κάποιες καταγραφές ήχου που αφορούν το φώνημα αυτό στο εργαλείο εκπαίδευσης. Σαν αποτέλεσμα για το φώνημα αυτό παίρνουμε για παράδειγμα ότι το γνώρισμα με τον αριθμό 52 έχει πιθανότητα να εμφανιστεί στο 1/100 το δευτερολέπτου για αυτό το φώνημα 55% , το γνώρισμα με αριθμό 189 έχει πιθανότητα 30% και το γνώρισμα 53 έχει πιθανότητα 15%. Επίσης με αντίστοιχη εκπαίδευση για το φώνημα “f” έχουμε ότι γνώρισμα με αριθμό 52 έχει 10% πιθανότητα, το γνώρισμα 189 έχει 10% και το γνώρισμα 53 80%.

Ας υποθέσουμε λοιπόν ότι αρχίζει η διαδικασία της αναγνώρισης και το σύστημα αναγνωρίζει τα παρακάτω χαρακτηριστικά γνωρίσματα:

52,52,189,53,52,52

Το σύστημα αναγνώρισης υπολογίζει την πιθανότητα του ήχου να είναι ένα φώνημα "h" και την πιθανότητα να είναι το "f”.

Η πιθανότητα του "h" είναι:

80% * 80% * 30% * 15% * 80% * 80% = 1,84%

Η πιθανότητα του ήχου που είναι ένα "f" είναι:
10% * 10% * 10% * 80% * 10% * 10% = 0,0008%

Έτσι σύμφωνα με τα παραπάνω το πιθανότερο φώνημα που αναγνωρίστηκε είναι το “h”.

Τι γίνεται όμως σε μια περίπτωση που δεν έχουμε ένα φώνημα αλλά μια σειρά από φωνήματα που το ένα ακολουθεί το άλλο όπως γίνεται όταν μιλάμε. Σε αυτήν περίπτωση θα πρέπει να υπάρχει ένας μηχανισμός ο οποίος μπορεί να ξεχωρίσει τα φωνήματα από τα αναγνωρισμένα χαρακτηριστικά της συχνότητας. Ο μηχανισμός αυτός είναι μια σειρά από μαθηματικές τεχνικές που λέγονται Hidden Marcov Models. Αυτό που λοιπόν κάνουν οι τεχνικές αυτές είναι να διαχωρίζουν τα φωνήματα ώστε να μπορούν να υπολογιστούν οι πιθανότητες και να προκύψουν οι κατάλληλες αναγνωρισμένες λέξεις που αποτελούν τα φωνήματα αυτά.
Επίσης το σύστημα αναγνώρισης υπολογίζει πότε υπάρχουν λεκτικές ενάρξεις και στάσεις επειδή έχει ένα φώνημα "silent", και κάθε χαρακτηριστικό συχνότητας έχει μια πιθανότητα εμφάνισης στη σιωπή, ακριβώς όπως σε οποιοδήποτε άλλο φώνημα.
Έτσι λοιπόν το σύστημα αναγνώρισής μας μπορεί να αναγνωρίσει ποιο φώνημα μιλήθηκε ακόμα και αν ο θόρυβος ή η φωνή του χρήστη είχε κάποια παραλλαγή ή μικρή αλλοίωση. Εντούτοις, υπάρχει όπως έχουμε αναφέρει και παραπάνω και ένα άλλο πρόβλημα. Ο ήχος των φωνημάτων αλλάζει ανάλογα με ποιο φώνημα είναι πριν από αυτό που μας ενδιαφέρει και από αυτό που είναι μετά. Δηλαδή για παράδειγμα είναι διαφορετικός ο τρόπος που ακούγεται και κατ’ επέκταση ο ήχος του φωνήματος “h” στη λέξη “he” και αλλιώς στην λέξη “how”. Τα συστήματα αναγνώρισης φωνής λύνουν το πρόβλημα με τη δημιουργία των " triphones" ή τρι-φωνημάτων. Τα τρι-φωνήματα είναι ουσιαστικά ένα σύνολο από τρία φωνήματα στα οποία ουσιαστικά είναι στη μέση το φώνημα που θα αναζητούσαμε σε κάποια αναζήτηση αλλά με την παρουσία τώρα των φωνημάτων που βρίσκονται δεξιά και αριστερά από αυτό. Για παράδειγμα στη λέξη “hospital” έχουμε αρκετά τρι-φωνήματα όπως είναι το “hos”, “osp”, ”spi”, “pit”, “ita” κλπ. Φυσικά για τα τρι-φωνήματα γίνεται η αντίστοιχη διαδικασία και με τα απλά φωνήματα. Δηλαδή κατά την χρησιμοποίηση του εργαλείου εκπαίδευσης δημιουργούνται οι αντίστοιχες πιθανότητες για την παρουσία των χαρακτηριστικών συχνότητας μέσα στα τρι-φωνήματα. Δεδομένου όμως ότι

υπάρχουν κατά προσέγγιση 50 φωνήματα στην αγγλική γλώσσα , μπορούμε να υπολογίσουμε ότι πρέπει να υπάρχουν 50*50*50 = 125.000 τρι-φωνήματα. Έτσι λοιπόν ο αριθμός των συσχετισμών των πιθανοτήτων είναι πάρα πολύ μεγάλος ακόμα και για γρήγορους υπολογιστές και για αυτό θα πρέπει να περιοριστούν με κάποιο τρόπο οι πράξεις ώστε να έχουμε σχετικά γρήγορα αποτελέσματα χωρίς όμως να ρισκάρουμε την ποιότητα της αναγνώρισης. Λύση για τον περιορισμό των πράξεων που απαιτούνται από ένα σύστημα αναγνώρισης φωνής θα περιγράψουμε στην συνέχεια.

Και τώρα για το τελευταίο ζήτημά μας. Ο ήχος ενός φωνήματος δεν είναι σταθερός. Ένας ήχος όπως για παράδειγμα ο ήχος του φωνήματος "t" είναι σιωπηλός πρώτα, κατόπιν παράγει μια ξαφνική υψηλή συχνότητα, η οποία ασθενεί έπειτα στη σιωπή. Τα συστήματα αναγνώρισης φωνής λύνουν αυτό με το διαχωρισμό κάθε φωνήματος σε διάφορα τμήματα και την παραγωγή ενός διαφορετικού στιγμιότυπου για κάθε τμήμα. Το σύστημα αναγνώρισης υπολογίζει που κάθε τμήμα αρχίζει και τελειώνει, με τον ίδιο τρόπο που υπολογίζει πού ένα φώνημα αρχίζει και τελειώνει. Έτσι λοιπόν μπορεί να γίνει αναγνώριση τμημάτων των φωνημάτων με βάση κάποια χαρακτηριστικά γνωρίσματα που είναι πιο έντονα από κάποια άλλα κατά την διάρκεια προφοράς του συγκεκριμένου φωνήματος.

Με βάση όλα αυτά, το λεκτικό σύστημα αναγνώρισης έχει όλους τους απαραίτητους μηχανισμούς να αναγνωρίσουν ένα συγκεκριμένο φώνημα που μιλήθηκε. Μια σημαντική ερώτηση χρειάζεται ακόμα την απάντηση: Πώς καθορίζει ποιο φώνημα να ψάξει;

Ένα λεκτικό σύστημα αναγνώρισης λειτουργεί με το να υποθέτει ότι κάθε φορά βρίσκεται σε μια θέση (state). Κάθε φάση περιέχει ένα φώνημα με ένα ιστορικό των προηγούμενων φωνημάτων που περάσανε. Η φάση με το υψηλότερο αποτέλεσμα χρησιμοποιείται ως τελικό αποτέλεσμα αναγνώρισης. Ας δούμε ένα παράδειγμα για το πώς λειτουργεί το σύστημα αναγνώρισης.

Το σύστημα αναγνώρισης αρχίζει να ακούει και έχει περάσει στην πρώτη φάση-state. Υποθέτει ότι ο χρήστης δεν μιλά και ότι το σύστημα αναγνώρισης ακούει το φώνημα "silent". Κάθε 1/100 του δευτερολέπτου υποθέτει ότι ο χρήστης έχει αρχίσει την ομιλία, και προσθέτει μια νέα φάση για κάθε φώνημα, έτσι λοιπόν και δημιουργεί 50 νέες φάσεις όσες και τα φωνήματα της αγγλικής γλώσσας, Έτσι λοιπόν στο πρώτο 1/100 του δευτερολέπτου έχουμε 50 συν μια την αρχική φάση, 51 φάσεις με το

αποτέλεσμα των πιθανοτήτων της η καθεμία. Στο δεύτερο 1/100 του δευτερολέπτου δημιουργούνται και άλλες φάσεις αυτή την φορά οι 51 παλιές συνδυάζονται με τα 50 φωνήματα της αγγλικής γλώσσας και μας δίνουν 50*51=2550 νέες φάσεις με τις πιθανότητές της η καθεμία.

Αυτή η ίδια διαδικασία επαναλαμβάνεται για κάθε 1/100 ενός δευτερολέπτου και σαν αποτέλεσμα της αναγνώρισης είναι η φάση με το καλύτερο αποτέλεσμα πιθανότητας όπως ακριβώς έχουμε περιγράψει παραπάνω και στην αναγνώριση απλών φωνημάτων. Με αυτόν τον τρόπο οι πιθανότητες κάθε κατάστασης εξαρτώνται απόλυτα από τις πιθανότητες που είχαν οι προηγούμενες καταστάσεις και συνεπώς φτιάχνεται μια αλυσίδα από καταστάσεις που οδηγούν σε μια συνολική πιθανότητα (πιθανότητα αλυσίδας) που είναι μια συγκεντρωτική πιθανότητα και πιο συγκεκριμένα η πιθανότητα της τελικής κατάστασης. Με την σύγκριση λοιπόν των πιθανοτήτων όλων των τελικών καταστάσεων οδηγούμαστε στην αναγνώριση κάποιας λέξης.

3.1.4 Μειώνοντας τον υπολογισμό και αυξάνοντας την ακρίβεια

Όπως αναφέραμε και παραπάνω με την μέθοδο αναγνώρισης τόσο των απλών φωνημάτων όσο και των τρι-φωνημάτων υπάρχει μια αυξημένη ανάγκη για υπολογιστική ισχύ. Έτσι λοιπόν θα πρέπει να υπάρξει κάποιος τρόπος να ελαχιστοποιήσουμε την απαιτούμενη ισχύ αυτή όσο το δυνατόν γίνεται για να μπορέσουμε να έχουμε ένα λειτουργικό σύστημα που θα αναγνωρίζει τις λέξεις και τις φράσεις σε σύντομο χρονικό διάστημα αλλά και η πιθανότητα της σωστής αναγνώρισης θα εξακολουθεί να είναι μεγάλη.

Το σύστημα αναγνώρισης φωνής με τους μηχανισμούς που αναλύθηκαν παραπάνω μπορεί να προσδιορίσει ποια φωνήματα μιλήθηκαν. Ο υπολογισμός των λέξεων που μιλήθηκαν πρέπει να είναι ένας εύκολος και γρήγορος στόχος. Εάν ο χρήστης μίλησε τα φωνήματα, "h eh l oe", το ανθρώπινο αυτί θα μπορέσει να αναγνωρίσει αμέσως τη λέξη "hello". Το σύστημα αναγνώρισης μπορεί μόνο να κάνει μια σύγκριση όλων των φωνημάτων ενάντια σε ένα λεξικό προφορών που έχουμε φτιάξει και να αποφανθεί για το ότι η λέξη είναι το "hello".

Αυτό όμως δεν είναι όσο απλό ακούγεται. Πιο συγκεκριμένα υπάρχουν κάποια θέματα όσον αφορά την αναγνώριση που θα πρέπει να ληφθούν υπόψιν:

1. Ο χρήστης μπορεί να έχει προφέρει "hello " ως "h eh l oe", το οποίο όμως μπορεί να μην βρίσκεται στο λεξικό.
2. Το σύστημα αναγνώρισης μπορεί να έχει κάνει ένα λάθος και να είχε αναγνωρίσει "hello ", ως " h uh l oe". Παρατηρούμε εδώ ότι το δεύτερο φώνημα δεν έχει αναγνωριστεί σωστά.
3. Πού η μια λέξη τελειώνει και που αρχίζει η άλλη;
4. Χρειαζόμαστε πάρα πολύ μεγάλη υπολογιστική ισχύ και αρκετό χρονικό διάστημα για να μπορέσουν να γίνουν οι απαραίτητες συγκρίσεις με όλα τα φωνήματα, τα τρι-φωνήματα και τις καταγραφές του λεξικού που έχουμε στη διάθεση μας.

Μια καλή λύση για να μειωθεί ο υπολογισμός και να αυξηθεί η ακρίβεια, το σύστημα αναγνώρισης να περιορίζει τις αποδεκτές εισαγωγές από το χρήστη. Γενικά, αυτό δεν είναι μια κακή παραδοχή για τους εξής λόγους:

- Είναι απίθανο ότι ο χρήστης θα εκφέρει μια λέξη όπως "zwickangagang" δεδομένου ότι δεν είναι μια έγκυρη λέξη.
- Ο χρήστης μπορεί να περιορίζεται σε μια σχετικά μικρή γραμματική. Υπάρχουν εκατομμύρια λέξεις, αλλά οι περισσότεροι άνθρωποι χρησιμοποιούν μόνο μερικές χιλιάδες από αυτές μέσα σε μια ημέρα και μπορεί να χρειαστούν ακόμη και λιγότερες λέξεις για να επικοινωνήσουν με έναν υπολογιστή.
- Όταν οι άνθρωποι μιλούν έχουν μια συγκεκριμένη γραμματική που χρησιμοποιούν. Οι χρήστες λένε, " Open the window" και όχι "window the open".
- Ορισμένες ακολουθίες από λέξεις είναι πιο κοινές από άλλες. Το " New York City " είναι πιο κοινή από το "New York Aardvak"

Έτσι λοιπόν μπορούμε καταρχήν να φτιάξουμε ένα πολύ μικρότερο λεξικό που θα περιέχει λέξεις που θα αφορούν κάποια συγκεκριμένη εφαρμογή και όχι για παράδειγμα την αναγνώριση όλων των λέξεων της αγγλικής γλώσσας. Επίσης με βάση τις λέξεις που χρησιμοποιούμε στο λεξικό θα είναι αντίστοιχος και ο αριθμός

φωνημάτων που περιέχονται σε αυτές. Έτσι θα έχουμε μικρότερο αριθμό πράξεων και συσχετίσεων που θα πρέπει να κάνουμε και σαφώς το αποτέλεσμα θα έχει μεγαλύτερη ακρίβεια καθώς τα πιθανά αποτελέσματα αναγνώρισης θα είναι πιο σαφή αφού οι υποψήφιες φράσεις θα είναι λιγότερες.
Ας δούμε όμως μερικές τεχνικές πως γίνεται αυτός ο περιορισμός των μη αποδεκτών εισαγωγών του χρήστη στη πράξη.

3.1.4.1 Ελεύθερη από περιεχόμενο γραμματική (Context Free Grammar)

Μια από τις τεχνικές για να μειωθεί η ακρίβεια υπολογισμού και να αυξηθεί η ακρίβεια της αναγνώρισης καλείται ελεύθερη από περιεχόμενο γραμματική (Context Free Grammar-CFG). Το CFG λειτουργεί με το να περιορίζει τη δομή του λεξιλογίου και της σύνταξης της αναγνώρισης μόνο σε εκείνες τις λέξεις και προτάσεις που ισχύουν στην επικρατούσα κατάσταση της εφαρμογής.
Η εφαρμογή διευκρινίζει τη δομή λεξιλογίου και σύνταξης σε ένα αρχείο κειμένου. Το αρχείο κειμένου μοιάζει με αυτό:

<Start> = ((send mail to) | call) (Fred | John | Bill) | (Exit application)

Αυτή η γραμματική μεταφράζεται σε επτά πιθανές προτάσεις που μπορούν να αναγνωριστούν από το σύστημα αναγνώρισης:

Send mail to Fred
Send mail to John
Send mail to Bill
Call Fred
Call John
Call Bill
Exit application

Φυσικά, οι γραμματικές μπορούν να είναι πιο σύνθετες από το παραπάνω παράδειγμα. Το σημαντικό χαρακτηριστικό γνώρισμα για το CFG είναι ότι περιορίζει τι το σύστημα αναγνώρισης αναμένει να ακούσει σε ένα μικρό λεξιλόγιο και μια καλά

δομημένη και περιορισμένη σύνταξη. Όταν ο χρήστης πει την λέξη "Send" το σύστημα αναγνώρισης θα ακούσει μόνο το "mail" σαν επόμενη λέξη γιατί έτσι είναι στην γραμματική (“send mail to”). Αυτό μειώνει σημαντικά τον αριθμό της υπόθεσης για το ποια είναι η σωστή αναγνώριση.

Η αναγνώριση βρίσκει τα φωνήματα για κάθε λέξη με το να ανατρέχει στο λεξικό όπου και υπάρχουν οι εγγραφές όλων των δυνατών λέξεων σε αντιστοίχηση με τα φωνήματα που τις αποτελούν. Ένα ενδεικτικό λεξικό για το παραπάνω παράδειγμα θα ήταν το εξής:

APPLICATION	**AE P L AH K EY SH AH N**
BILL	**B IH L**
CALL	**K AO L**
EXIT	**EH G Z IH T**
FRED	**F R EH D**
JOHN	**JH AA N**
MAIL	**M EY L**
SEND	**S EH N D**
TO	**T AH**
TO(2)	**T IH**
TO(3)	**T UW**

Μερικές λέξεις έχουν περισσότερες από μια προφορές, όπως το “to” στο παραπάνω λεξικό βλέπουμε ότι έχει 3 προφορές και μπορεί έτσι να αναγνωρίσει το σύστημα ακόμα και ιδιωματισμούς της γλώσσας ανάλογα βέβαια με το πόσο καλά είναι φτιαγμένο το λεξικό. Το σύστημα αναγνώρισης μεταχειρίζεται βασικά μια λέξη με πολλαπλές προφορές σαν ξεχωριστές λέξεις. Δηλαδή το “to” στο συγκεκριμένο λεξικό πρακτικά μπορεί να είναι η ίδια λέξη με τρεις προφορές αλλά για το σύστημα είναι τρεις διαφορετικές λέξεις.

Το CFG αλλάζει ελαφρώς τη διαβάθμιση της υπόθεσης της αναγνώρισης. Παρά την υπόθεση της μετάβασης σε όλα τα φωνήματα, το σύστημα αναγνώρισης υποθέτει μόνο τη μετάβαση στα επόμενα αποδεκτά φωνήματα. Δηλαδή ξέρει ότι μετά το silent φώνημα μπορεί να δεχτεί μόνο το φώνημα “k” από το οποίο αρχίζει η λέξη “Call” ή

το φώνημα “s” που αρχίζει η φράση “send mail to” ή τέλος το φώνημα “eh” από το οποίο αρχίζει η φράση “exit application”. Αυτό γίνεται και στην συνέχεια αν για παράδειγμα αναγνωριστεί το φώνημα “K” θα πρέπει το επόμενο φώνημα να είναι “ao” που είναι το επόμενο φώνημα της λέξης “call” όπως φαίνεται και από το λεξικό. Έτσι λοιπόν βλέπουμε πόσο πολύ μπορούμε να περιορίσουμε τον αριθμό πράξεων και συγκρίσεων φωνημάτων με τις λέξεις και τις πιθανότητες που έχουν αυτά ώστε να μειώσουμε την ανάγκη για υπολογιστική ισχύ αλλά και να αυξήσουμε την ακρίβεια της αναγνώρισης και να επιτύχουμε μια πολύ γρήγορη και επιτυχημένη αναγνώριση.
Όταν ο χρήστης τελειώσει την ομιλία, το σύστημα αναγνώρισης επιστρέφει την υπόθεση με το υψηλότερο αποτέλεσμα, και οι λέξεις που ο χρήστης πρόφερε και αναγνωρίστηκαν επιστρέφονται στην εφαρμογή.
Η λεκτική αναγνώριση που χρησιμοποιεί ένα CFG απαιτεί στην πράξη έναν σχετικά μικρό υπολογιστή και μικρές δυνατότητες σε μνήμη ώστε να μπορέσει να πραγματοποιήσει αναγνώριση φωνής σε πραγματικό χρόνο.
Σε αυτό το σημείο πρέπει να αναφέρουμε ότι η τεχνική του CFG είναι κάπως κλειστή και δεν επιτρέπει την χρήση ελευθερίας από τον χρήστη να πει ότι θέλει και τον περιορίζει σε συγκεκριμένες λέξεις και με συγκεκριμένη σύνταξη. Αυτό βέβαια σε εφαρμογές όπως είναι η πλοήγηση κάποιου μενού ή κάποιας εφαρμογής με αυστηρά προκαθορισμένες εντολές είναι ιδανικό γιατί επιτυγχάνεται πολύ γρήγορη και αρκετά ακριβής αναγνώριση.

3.1.4.2 Διακριτή Υπαγόρευση (Discrete Dictation)

Όπως αναφέραμε και παραπάνω η χρησιμοποίηση του CFG δεν επιτρέπει στους χρήστες να χρησιμοποιήσουν αυθαίρετο κείμενο προς αναγνώριση λόγω του περιορισμένου λεξικού και της αυστηρής σύνταξης που χρησιμοποιείται στην περίπτωση αυτή. Εάν ένας χρήστης επιθυμεί να υπαγορεύει κάποιο κείμενο και το σύστημα αναγνώρισης να το αναγνωρίζει και να το μετατρέπει για παράδειγμα σε κάποιο αρχείο κειμένου (δηλαδή σαν εφαρμογή λογογράφου) θα πρέπει να ακολουθηθεί διαφορετική τεχνική και σίγουρα θα πρέπει να υπάρχουν περισσότερες ανοχές κατανάλωσης πόρων συστήματος, όπως επεξεργαστική ισχύς και μνήμη.
Η τεχνική που μπορεί να χρησιμοποιηθεί σε αυτήν την περίπτωση ονομάζεται Discrete Dictation και κάνει ακριβώς αυτό.

Με την Discrete Dictation, ένας χρήστης μπορεί να μιλήσει οποιαδήποτε λέξη επιθυμεί από ένα λεξικό, που συνήθως αποτελείται από 60.000 λέξεις και περικλείει όλες εκείνες τις λέξεις κάποιας γλώσσας που χρησιμοποιούνται ευρέως καθώς και πιο ειδικές αν αυτό είναι απαραίτητο. Για παράδειγμα σε μια ιατρική εφαρμογή όπου θα χρειαστούν κάποιοι ιατρικοί όροι υπάρχει η δυνατότητα να προστίθενται λέξεις. Εντούτοις, για να μπορεί να λειτουργήσει το σύστημα αναγνώρισης σωστά στην περίπτωση του ελευθέρου κειμένου θα πρέπει ο χρήστης να αφήνει μικρές διακοπές της τάξεως του ¼ του δευτερολέπτου μεταξύ των λέξεων, ώστε να γίνεται αντιληπτό στο πεδίο της συχνότητας, που αρχίζουν και που τελειώνουν οι λέξεις. Εάν ο χρήστης δεν αφήνει τις μικρές αυτές διακοπές η τεχνική καλείται "συνεχής υπαγόρευση" και η οποία θα καλυφτεί στην συνέχεια.

Μια απλή προσέγγιση στην Discrete Dictation είναι να δημιουργηθεί ένα CFG που θα είναι ένας κατάλογος όλων των χρησιμοποιούμενων λέξεων. Αυτό θα λειτουργήσει, αλλά θα έχει κάποια προβλήματα όπως :

- Θα είναι πολύ αργό. Το να γίνει η εύρεση σε ένα αρχείο 60.000 λέξεων που θα είναι το λεξικό και η σύγκριση με ένα αρχείο 60.000 εγγραφών που θα είναι η γραμματική θα καθυστερεί πάρα πολύ. Όπως είπαμε και παραπάνω το CFG είναι ιδανικό για εφαρμογές με σχετικά μικρό και προκαθορισμένο αριθμό λέξεων και συγκεκριμένης σύνταξης. Στην περίπτωση της προφοράς ελεύθερου κειμένου οι πιθανές λέξεις είναι πάρα πολλές και η σύνταξη έχει και αυτή πάρα πολλές κατηγορίες.
- Θα είναι ανακριβές, ειδικά για τις λέξεις που ηχούν παρόμοια η μια με την άλλη.
- Θα πάρει συνήθως τα ομόγραφα, όπως "to", "too" και "two" λανθασμένα.

Σύμφωνα με μετρήσεις κάποιο σύστημα αναγνώρισης ελεύθερου κειμένου που χρησιμοποιεί μόνο CFG έχει ακρίβεια αναγνώρισης λέξης λιγότερο από 80%.

Η ταχύτητα και η ακρίβεια μπορούν να βελτιωθούν με τη γνώση της γραμματικής της γλώσσας που χρησιμοποιείται, και ποιοι είναι οι πιθανοί συνδυασμοί λέξεων.

Για να μάθει όμως το σύστημα αναγνώρισης πώς οι λέξεις γειτονεύουν η μια με την άλλη και ποιοι είναι οι συνδυασμοί λέξεων σύμφωνα με την γραμματική , προστίθεται άλλο ένα βήμα στην παραγωγή της μηχανής αναγνώρισης από την βάση δεδομένων που φτιάχνουμε από τα ηχητικά αρχεία όπως περιγράψαμε παραπάνω.

Εκτός από τη χρησιμοποίηση των καταγραφών των φωνημάτων, τα εργαλεία εκπαίδευσης αναλύουν ένα τεράστιο αριθμό κειμένων, ιδανικά 1.000.000.000 λέξεις, περίπου 8 gigabytes κειμένου δηλαδή. Το κείμενο είναι χωρισμένο σε λέξεις.

Παραδείγματος χάριν, η πρόταση:
"At 8:00 he will walk down the street and then."

Μετατρέπεται σε:
“at eight o’clock he will walk down the street period”

Στην συνέχεια δημιουργούνται ακολουθίες τριών λέξεων, στην συνέχεια μετρούνται, παράγοντας ένα αρχείο που δείχνει πόσο κοινή είναι κάποια ακολουθία τριών λέξεων. Για παράδειγμα από την παραπάνω πρόταση έχουμε :

at eight 1
at eight o’clock 1
down the street 1
he will walk 1
eight o’clock he 1
o’clock he will 1
street period 1
the street period 1
walk down the 1
will walk down 1

Η πραγματική βάση δεδομένων θα παραχθεί από πέρα από ένα δισεκατομμύριο λέξεις, έτσι οι ακολουθίες τριών λέξεων θα είναι πάρα πολλές και σίγουρα ο δείκτης που θα έχουν δίπλα τους θα είναι πολύ μεγάλος αριθμός. Ένα πιο αντιπροσωπευτικό δείγμα της πραγματικής βάσης δεδομένων που θα δημιουργηθεί θα είναι σαν το παρακάτω :

New York City 553
New York Sid 1

New York State 483

… άλλες εγγραφές

Ο όρος " New York City " εμφανίστηκε 553 φορές, ενώ ο όρος "New York Sid" εμφανίστηκε μόνο μία φορά. Από αυτά τα δεδομένα το σύστημα αναγνώρισης ξέρει ότι εάν ο χρήστης έχει προφέρει τον όρο "New York " ότι είναι πιθανότερο να συνεχίσει με τον όρο " City " από ότι με τον όρο "Sid".

Έτσι λοιπόν έχουμε κάτι αντίστοιχο με αυτό που είχαμε με τις πιθανότητες των φωνημάτων και των τρι-φωνημάτων αλλά σε αυτήν την περίπτωση με ολόκληρες λέξεις και συγκεκριμένα με ακολουθίες λέξεων που βασίζονται όμως σε πραγματικά κείμενα της γλώσσας που χρησιμοποιούμε και μπορούν να μας φτιάξουν κάποιους πίνακες τους οποίους μπορούμε να βρούμε ποια είναι η πιο πιθανή λέξη που έπεται από κάποια που ήδη το σύστημα έχει αναγνωρίσει και με στατιστικούς και μαθηματικούς μηχανισμούς να αναγνωρίσουμε σχετικά γρήγορα και με ακρίβεια τη λέξη αυτή.

Οι στατιστικές αυτές μπορούν να μετατραπούν σε πιθανότητες, και αυτές οι πιθανότητες μπορούν να ενσωματωθούν σε κάθε μια από τις υποθέσεις που παράγονται κατά τη διάρκεια της αναγνώρισης για κάποια λέξη. Το αποτέλεσμα μιας υπόθεσης είναι ίσο με το ακουστικό αποτέλεσμα όπως το είδαμε από τις πιθανότητες των φωνημάτων και των τρι-φωνημάτων που αναγνωρίζονται επί την πιθανότητα να εμφανιστεί η λέξη σε μια ακολουθία όπως περιγράψαμε στο κομμάτι αυτό. Έτσι έχουμε έναν συνδυασμό των πιθανοτήτων των φωνημάτων να ανήκουν σε μια λέξη και των πιθανοτήτων που έχουν οι λέξεις να βρίσκονται σε μια ακολουθία λέξεων. Με αυτόν τον τρόπο καταλαβαίνουμε ότι μπορούμε να φτιάξουμε ένα μοντέλο τέτοιο που να επιτρέπει αναγνώριση ελευθέρου κειμένου με στόχο την ελαχιστοποίησης της απαιτούμενης επεξεργαστικής ισχύς αφού μειώνουμε το πλήθος των πράξεων, αλλά και την ακρίβεια καθώς οι πιθανότητες που έχει φτιάξει το εργαλείο εκπαίδευσης βασίζονται σε πραγματικά δεδομένα που αφορούν την γλώσσα που χρησιμοποιεί το σύστημα αναγνώρισης. Αυτό καλείται γλωσσικό μοντέλο ή language model.

Μερικές φορές ένας συνδυασμός λέξης δεν υπάρχει στην αρχική βάση δεδομένων. Ίσως κανένα από το κείμενα τα οποία μπαίνουν σαν είσοδος στο εργαλείο εκπαίδευσης δεν αναφέρει για παράδειγμα τον όρο " New York Aardvark," αλλά ο χρήστης ακόμα και αν πει αυτόν τον όρο το σύστημα θα μπορέσει να το αναγνωρίσει.

Το σύστημα αναγνώρισης αφήνει πάντα μια μικρή πιθανότητα ότι οποιαδήποτε λέξη μπορεί να ακολουθήσει οποιαδήποτε άλλη λέξη. Πρέπει ακριβώς να είναι πολύ βέβαιο ότι άκουσε την προφορά της λέξης σωστά και να έχει αποκλείσει όλες τις πιθανότερες ακολουθίες λέξεων. Με το τελευταίο οδηγούμαστε στο συμπέρασμα ότι όσο καλύτερα και πιο χαρακτηριστικά της γλώσσας, είναι τα κείμενα που χρησιμοποιούμε σαν είσοδο για το εργαλείο εκπαίδευσης τόσο καλυτέρα αποτελέσματα θα έχουμε κατά την δημιουργία του language model.
Το language model θα μειώσει το ποσοστό λάθους μετατρέποντας την ακρίβεια από 80% σε 95% σύμφωνα με μετρήσεις και αυτό οφείλεται στα εξής:

- Εάν δύο ή περισσότερες λέξεις ηχούν παρόμοιες για το σύστημα αναγνώρισης, επιλέγει εκείνη τη λέξη που έχει την μεγαλύτερη πιθανότητα με βάση τις γειτονικές λέξεις από την βάση δεδομένων που έχει δημιουργηθεί με τις ακολουθίες των τριών λέξεων. Έτσι έχουμε καλύτερη ακρίβεια.
- Αυτό συμβαίνει και στα ομόφωνα και έτσι μπορούν να αναγνωριστούν οι σωστές λέξεις ανάλογα με το ποιες είναι οι προηγούμενες δυο λέξεις τους. Για παράδειγμα μπορούν να αναγνωριστούν με πολύ μεγάλη επιτυχία λέξεις όπως "to", "too" και "two".

Επίσης η αναγνώριση επιταχύνεται αρκετά πιο γρήγορα επειδή οι απίθανοι συνδυασμοί λέξεων απορρίπτονται γρήγορα και έτσι το σύστημα δεν ψάχνει για τις πιθανές λέξεις ανάμεσα σε 60.000 λέξεις αλλά μόνο σε κάποιες λέξεις που βρίσκονται σε ακολουθίες που έχουν μεγάλη πιθανότητα.

3.1.4.3 Συνεχής υπαγόρευση (Continuous Dictation)

Η συνεχής υπαγόρευση επιτρέπει στο χρήστη για να προφέρει οποιαδήποτε λέξη θέλει από ένα μεγάλο λεξιλόγιο. Η αναγνώριση εδώ είναι πιο δύσκολη από ότι στην διακριτή υπαγόρευση επειδή η μηχανή αναγνώρισης δεν ξέρει εύκολα που η μια λέξη τελειώνει και που αρχίζει η επόμενη. Η συνεχής υπαγόρευση λειτουργεί παρόμοια με την ιδιαίτερη υπαγόρευση εκτός από το ότι το τέλος μιας λέξης δεν ανιχνεύεται από το “silent” φώνημα. Για αυτό και εδώ έχουμε πάρα πολλές υποθέσεις για το ποια

πρέπει να είναι μια λέξη καθώς δεν είναι εμφανές που σταματάει και που ξεκινάει κάποια λέξη οπότε και οι συνδυασμοί λέξεων αρχίζουν και αυξάνονται σε σημείο που αν αφήσουμε το σύστημα ως έχει να μην μπορούμε να αναγνωρίσουμε τίποτα. Τα συστήματα αναγνώρισης χρησιμοποιούν πολύ περισσότερες βελτιστοποιήσεις για να ελαχιστοποιήσουν την επεξεργασία και τη μνήμη στην συνεχή υπαγόρευση. Οι βελτιστοποιήσεις είναι πέρα από τις ανάγκες αυτής της εργασίας για να αναλυθούν και ο μόνος σκοπός της συγκεκριμένης αναφοράς στην συνεχή υπαγόρευση είναι για να δείξει ότι το σύστημα αναγνώρισης φωνής μπορεί να πάει και ένα βήμα παρά πέρα με κατάλληλες επεμβάσεις φυσικά.

3.1.5 Συμπερασματικά

Σύμφωνα με όλα τα παραπάνω λοιπόν ένα σύστημα αναγνώρισης φωνής μπορεί να φτιαχτεί για να εξυπηρετήσει διάφορες ανάγκες εφαρμογών, θα πρέπει όμως σε κάθε περίπτωση να εκπληρώνει τις προϋποθέσεις που περιγράφηκαν.

Θα πρέπει να υπάρχει ένα εργαλείο εκπαίδευσης το οποίο θα φτιάχνει τις αντιστοιχίες των φωνημάτων και των τρι-φωνημάτων με τα χαρακτηριστικά γνωρίσματα των συχνοτήτων ώστε να μπορεί η μηχανή του συστήματος αναγνώρισης μόλις βρει κάποιο χαρακτηριστικό του ηχητικού σήματος της φωνής του χρήστη να κάνει την αντιστοίχηση με το σωστό φώνημα και να συνεχίσει το επόμενο. Αυτή είναι και η διαδικασία του ακουστικού μοντέλου που είναι πολύ σημαντική και θα πρέπει να επιλεχθούν σωστά τα ηχητικά αρχεία που θα περάσουν σαν είσοδος στο εργαλείο εκπαίδευσης με σκοπό να δημιουργηθούν όσο το δυνατόν καλύτερα οι στατιστικί πίνακες των φωνημάτων και των χαρακτηριστικών της συχνότητας. Επίσης θα πρέπει να χρησιμοποιηθούν μαθηματικά μοντέλα όπως είναι τα Hidden Markov Models για να μπορέσει το σύστημα να αναγνωρίσει πότε μια λέξη τελειώνει και πότε αρχίζει μια άλλη. Η ύπαρξη ενός σωστού λεξικού που θα έχει τις αντιστοιχίες των λέξεων της εφαρμογής με τα φωνήματα που αποτελούν κάθε λέξη είναι πάρα πολύ σημαντική και θα πρέπει στην κατασκευή του λεξικού αυτού να είμαστε πάρα πολύ προσεχτικοί γιατί έστω και ένα φώνημα λάθος μπορεί να οδηγήσει σε αναγνώριση λάθους λέξης. Θα πρέπει να υπάρχει ανάλογα με την εφαρμογή κάποιο μοντέλο γραμματικής που να εξασφαλίζει την σωστή σύνταξη που επιβάλει η εφαρμογή. Για παράδειγμα αν έχουμε κάποιες προκαθορισμένες εντολές που θέλουμε να

υποστηρίξουμε σε μια εφαρμογή ο καλύτερος τρόπος είναι να χρησιμοποιήσουμε μια CFG γραμματική. Αν θέλουμε από την άλλη ελεύθερο κείμενο θα πρέπει να φτιάξουμε ένα γλωσσικό μοντέλο με το εργαλείο εκπαίδευσης όπως περιγράψαμε παραπάνω.

Όλα αυτά τα τμήματα είναι ανεξάρτητα μεταξύ τους αλλά όλα δουλεύουν μαζί για να γίνει εφικτή η αναγνώριση της φωνής. Έτσι λοιπόν θα πρέπει σε κάθε κομμάτι να δίνεται ιδιαίτερη σημασία τόσο κατά την φάση του σχεδιασμού όσο και κατά την φάση της υλοποίησης.

Αφού περιγράψαμε πως δουλεύει ένα σύστημα αναγνώρισης φωνής θα δούμε στην συνέχεια πως το sphinx-4, που χρησιμοποιούμε στην δική μας εφαρμογή, είναι δομημένο και πως λειτουργεί και θα διαπιστώσουμε στην πράξη ότι ακολουθεί ακριβώς τις ίδιες αρχές.

3.2 Sphinx-4

- **Επισκόπηση**

  Το sphinx-4 είναι ένα σύστημα λεκτικής αναγνώρισης που έχει γραφτεί εξ ολοκλήρου στη γλώσσα προγραμματισμού της Java. Δημιουργήθηκε μέσω μιας κοινής συνεργασίας μεταξύ της ομάδας Sphinx στο πανεπιστήμιο Carnegie Mellon, των Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), και της Hewlett Packard (HP), με τις συνεισφορές των University of California at Santa Cruz (UCSC) και του Massachusetts Institute of Technology (MIT).

  Το sphinx-4 άρχισε ως συνέχεια του sphinx-3, αλλά εξελίχθηκε σε ένα σύστημα αναγνώρισης με σκοπό να είναι πολύ πιο ευέλικτο από το sphinx-3, αποτελώντας κατά συνέπεια μια άριστη πλατφόρμα για την έρευνα λεκτικής αναγνώρισης.

- **Ικανότητες**

  1) Πραγματικού χρόνου λεκτική αναγνώριση , ικανό να αναγνωρίσει την ιδιαίτερη και συνεχή ομιλία. Όταν λέμε ιδιαίτερη εννοούμε την περίπτωση ιδιωματισμών στην ομιλία κάποιου χρήστη που χρησιμοποιεί κάποια εφαρμογή που έχει την δυνατότητα της αναγνώρισης φωνής μέσω του sphinx και όταν λέμε την συνεχή ομιλία μπορεί να γίνει συνεχόμενη αναγνώριση φωνής από κάποιο μέσο με ολόκληρες προτάσεις που βέβαια σε αυτή την περίπτωση θα πρέπει να υπάρχουν και διάφοροι περιορισμοί σε επίπεδο εφαρμογής.

  2) Γενικευμένη αρχιτεκτονική που επιτρέπει τη δημιουργία ανοιχτού σχεδιασμού εφαρμογών. Περιλαμβάνει υλοποιήσεις του παραθύρου Hamming, FFT, discrete cosine transform και διάφορες άλλες τεχνικές που χρησιμοποιούνται αυτόνομα ή σε συνδυασμό για να μπορέσουν να επιτευχθούν καλύτερα αποτελέσματα στην αναγνώριση φωνής σε σχέση πάντα με τις απαιτήσεις της εκάστοτε εφαρμογής που χρησιμοποιεί το sphinx.

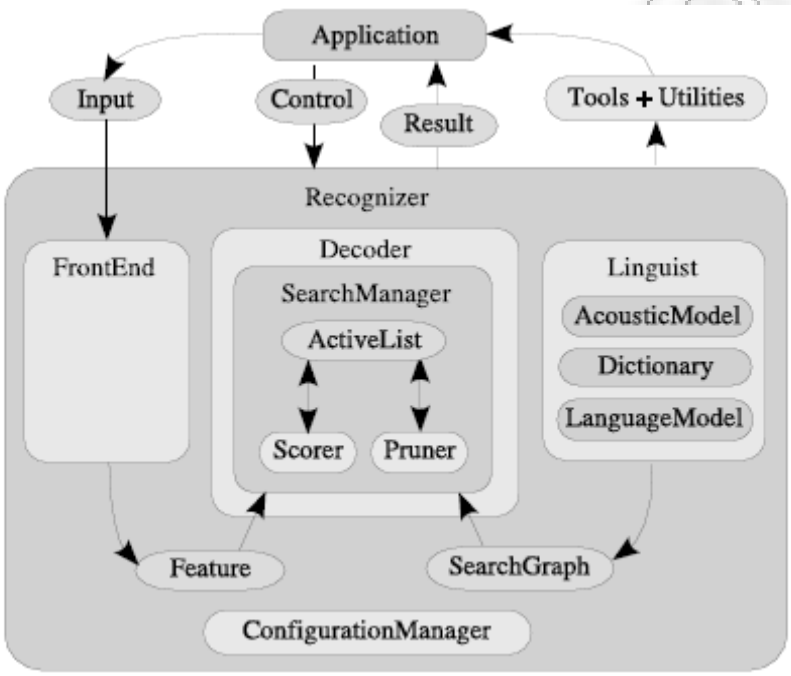
  3) Γενικευμένη αρχιτεκτονική δημιουργίας γλωσσικού μοντέλου (language model). Υποστηρίζεται και ASCII και binary μορφή unigram, bigram, trigram, τα οποία είναι μονά γράμματα, διπλά γράμματα (εδώ συμπεριλαμβάνονται και οι δίφθογγοι) και τα τρία γράμματα σε σειρά αντίστοιχα. Έτσι λοιπόν μπορούμε να περιγράψουμε πολύ καλά κάθε λέξη αφού μπορούμε να την απλοποιήσουμε στα παραπάνω δομικά στοιχεία και να πιάσουμε έτσι τυχόν ιδιωματισμούς της γλώσσας που χρησιμοποιούμε.

  4) Γενικευμένη αρχιτεκτονική δημιουργίας του ακουστικού μοντέλου. Χρησιμοποιούνται τα ακουστικά πρότυπα όλων των προηγούμενων εκδόσεων του sphinx. Μπορούμε να φτιάξουμε με κατάλληλη μεθοδολογία που θα περιγραφεί στη συνέχεια, δικό μας ακουστικό μοντέλο.

  5)Γενικευμένη διαχείριση αναζήτησης. Χρησιμοποιούνται εξελιγμένοι τρόποι αναζήτησης των ακουστικών δεδομένων με στόχο να είναι όσο το δυνατόν

καλύτερο το αποτέλεσμα της αναγνώρισης της φωνής και φυσικά να πάρουμε την σωστή λέξη, φράση ή πρόταση που θέλουμε σε μια εφαρμογή.

Ας δούμε παρακάτω πως είναι δομημένο το sphinx-4 και ποια είναι τα δομικά του στοιχεία.



Σχήμα 3.2.1 Το framework του sphinx-4

Υπάρχουν τρία πρωτεύοντα δομικά στοιχεία στο sphinx-4 : το FrontEnd, ο Decoder (αποκωδικοποιητής), και το Linguistic (γλωσσολογικό) κομμάτι .

Το FrontEnd παίρνει ένα ή περισσότερα ηχητικά σήματα εισόδου και τα παραμετροποιεί σε μια ακολουθία χαρακτηριστικών γνωρισμάτων στο πεδίο της συχνότητας (Features). Το Linguist μεταφράζει οποιοδήποτε τύπο γλωσσικού προτύπου (language model), μαζί με τις πληροφορίες προφοράς από το λεξικό και τις πληροφορίες δομής από το Acoustic Model και δημιουργεί ένα SearchGraph, ένα μοντέλο αναζήτησης δηλαδή.

Το στοιχείο SearchManager τουDecoder όπως βλέπουμε και στο σχήμα 3.2.1 χρησιμοποιεί τα χαρακτηριστικά γνωρίσματα (Features) από το FrontEnd και το SearchGraph από το Linguist για να εκτελέσει την πραγματική αποκωδικοποίηση, παράγοντας τα αποτελέσματα (Results). Αυτό που γίνεται λοιπόν με περισσότερη λεπτομέρεια είναι αυτό που εξηγήσαμε και πιο πάνω στην ανάλυση των συστημάτων αναγνώρισης φωνής. Δηλαδή με βάση τα ηχητικά σήματα της φωνής του χρήστη γίνεται έλεγχος για τα χαρακτηριστικά συχνότητας που αναγνωρίζονται τα οποία αντιπαραβάλλονται με στοιχεία όπως το λεξικό και το ακουστικό μοντέλο και συντάσσονται με αυτόν τον τρόπο οι λέξεις που αποτελούν τα αποτελέσματα του συστήματος. Οποιαδήποτε στιγμή πριν ή ακόμα και κατά τη διάρκεια τη διαδικασία αναγνώρισης, το σύστημα μπορεί να ενεργοποιήσει κάποιους ελέγχους- Controls (σχήμα 3.2.1) σε κάθε μια από τις επιμέρους ενέργειες της όλης διαδικασίας με στόχο τα καλύτερα αποτελέσματα αναγνώρισης.

Το sphinx-4 όπως τα περισσότερα συστήματα αναγνώρισης φωνής, έχει έναν μεγάλο αριθμό διαμορφώσιμων παραμέτρων, όπως για παράδειγμα είναι το μέγεθος της ακτίνας αναζήτησης. Αυτό έχει σαν στόχο το συντονισμό της απόδοσης του συστήματος ανάλογα με τις απαιτήσεις της εφαρμογής που το χρησιμοποιεί. Το Configuration Manager χρησιμοποιείται για να διαμορφώσει τέτοιες παραμέτρους. Αντίθετα με άλλα συστήματα, όμως, το Configuration Manager δίνει επίσης στο sphinx-4 τη δυνατότητα να επηρεάσει τις παραμέτρους των διαφόρων στοιχείων του συστήματος σε πραγματικό χρόνο, την ώρα δηλαδή που το σύστημα λειτουργεί και αυτό έχει σαν αποτέλεσμα το sphinx-4 να είναι ένα εύκαμπτο και ευέλικτο σύστημα και να δίνει την δυνατότητα παραμετροποίησης ακόμα και την ώρα που λειτουργεί και είναι στην διαδικασία της αναγνώρισης.

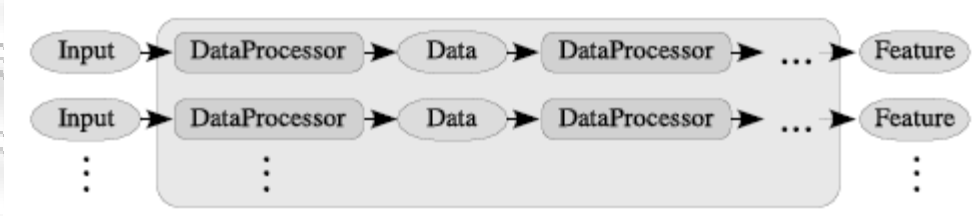
Παραδείγματος χάριν, το sphinx-4 μέσω του FrontEnd παράγει τους συντελεστές Cepstral Mel-Frequency (MFCCs) που ουσιαστικά είναι μια μορφή απεικόνισης των χαρακτηριστικών της συχνότητας. Χρησιμοποιώντας το Configuration Manager, όμως είναι δυνατό να μετατραπούν τα χαρακτηριστικά της συχνότητας σε διαφορετική μορφή με το να κατασκευαστεί ένα διαφορετικό FrontEnd που παράγει τους Perceptual Linear Prediction coefficients (PLP) χωρίς να πρέπει να τροποποιηθεί οποιοσδήποτε πηγαίος κώδικας ή να γίνει recompile στο σύστημα. Αυτό δείχνει και την μεγάλη ελαστικότητα του συστήματος που μπορεί να αυτοπροσαρμόζεται κατά

την ώρα της λειτουργίας του στις εκάστοτε ανάγκες της εφαρμογής που το χρησιμοποιεί.

Εκτός όμως από τα δομικά στοιχεία του συστήματος που αναφέρθηκαν εδώ πρέπει να τονίσουμε ότι έχουν φτιαχτεί αρκετά παράπλευρα εργαλεία που προσφέρουν στους προγραμματιστές και στους δημιουργούς των εφαρμογών δυνατότητες στατιστικών αναλύσεων όπως είναι το ποσοστό λάθους λέξης (word error rate), ο χρόνος τρεξίματος του συστήματος και η χρήση μνήμης ανά πάσα στιγμή. Όπως με το υπόλοιπο του συστήματος, τα εργαλεία είναι ιδιαίτερα διαμορφώσιμα, επιτρέποντας στους χρήστες να εκτελέσουν ένα ευρύ φάσμα ανάλυσης στο σύστημα. Επιπλέον, τα εργαλεία παρέχουν ένα περιβάλλον αλληλεπίδρασης με το σύστημα σε πραγματικό χρόνο που επιτρέπει στους χρήστες για να τροποποιούν τις παραμέτρους του συστήματος ενώ το σύστημα τρέχει, επιτρέποντας το γρήγορο πειραματισμό με τις διάφορες παραμέτρους.

Ας δούμε τώρα σε μεγαλύτερο βαθμό ανάλυσης τα δομικά στοιχεία του sphinx-4 :

- **FRONTEND :** ο σκοπός του FrontEnd είναι να μετατρέπει και να παραμετροποιεί ένα σήμα εισαγωγής (π.χ., ακουστικό σήμα) σε μια ακολουθία χαρακτηριστικών γνωρισμάτων στο πεδίο της συχνότητας.



Σχήμα 3.2.2 Ο τρόπος λειτουργία του FrontEnd

Όπως διευκρινίζεται στο σχήμα 3.2.2, το FrontEnd περιλαμβάνει μια ή περισσότερες παράλληλες αλυσίδες από μονάδες επεξεργασίας σημάτων επικοινωνίας αποκαλούμενων ως DataProcessors. Η υποστήριξη των

πολλαπλών αλυσίδων επιτρέπει τον ταυτόχρονο υπολογισμό των διαφορετικών τύπων παραμέτρων από τα ίδια ή διαφορετικά σήματα εισαγωγής. Αυτό επιτρέπει την ταυτόχρονη παραμετροποίηση σημάτων με την χρήση διαφορετικών τύπων παραμέτρου, όπως είναι για παράδειγμα τα MFCC και τα PLP, ακόμη όμως και διαφορετικών σημάτων όπως για παράδειγμα την παραμετροποίηση ακουστικών σημάτων αλλά και σημάτων video με ταυτόχρονη διαδικασία. Κάθε DataProcessor στο FrontEnd έχει ένα input και ένα output που μπορεί όμως αυτό το output να αποτελεί το input κάποιου άλλου DataProcessor, επιτρέποντας τις ακολουθίες αλυσίδων επεξεργασίας. Τα inputs και τα outputs κάθε DataProcessor είναι γενικά αντικείμενα δεδομένων και αυτά μπορεί να είναι είτε πραγματικά δεδομένα φωνής (πχ μια λέξη) είτε κάποια στοιχεία ένδειξης (markers) που δείχνουν για παράδειγμα πότε τελειώνει μια λέξη (end-point marker).

Το τελευταίο DataProcessor από κάθε αλυσίδα είναι αρμόδιο για την παραγωγή ενός αντικειμένου δεδομένων που αποτελείται από τα παραμετροποιημένα σήματα, τα οποία καλούνται χαρακτηριστικά γνωρίσματα (Features) και ουσιαστικά όπως είδαμε και στην γενική ανάλυση των συστημάτων αναγνώρισης φωνής είναι η αντιστοίχηση των ηχητικών μορφών (φωνημάτων) σε χαρακτηριστικά γνωρίσματα στο πεδίο της συχνότητας. Τα Features αυτά χρησιμοποιούνται στην συνέχεια από τον αποκωδικοποιητή (Decoder).

Το sphinx-4 παρέχει τη δυνατότητα να παραχθούν παράλληλες ακολουθίες χαρακτηριστικών γνωρισμάτων. Το σημαντικό όμως είναι ότι επιτρέπει έναν αυθαίρετο αριθμό παράλληλων ρευμάτων επεξεργασίας και παραγωγής χαρακτηριστικών γνωρισμάτων-Features. Η επικοινωνία μεταξύ των DataProcessors ακολουθεί την μορφή pull. Με την μορφή pull, ένας DataProcessor ζητά την εισαγωγή από έναν προηγούμενο DataProcessor μόνο όταν απαιτείται, σε αντιδιαστολή με την μορφή push που χρησιμοποιείται σε άλλα συστήματα επεξεργασίας, όπου ένας DataProcessor μεταδίδει το output του στον επόμενο DataProcessor της αλυσίδας μόλις παράγεται. Αυτή η μορφή pull επιτρέπει στους επεξεργαστές να κάνουν buffering (δηλαδή τοπική αποθήκευση της πληροφορίας που επεξεργάζονται), επιτρέποντας στους DataProcessors να μπορούν να κοιτάνε προς τα εμπρός ή προς τα πίσω

στην αλυσίδα και να παίρνουν σαν είσοδο τα δεδομένα που πρέπει όταν πρέπει. Η δυνατότητα αυτή προσφέρει στους DataProcessors να πραγματοποιούν πολλαπλούς συνδυασμούς παραγωγής χαρακτηριστικών γνωρισμάτων αρκετά γρήγορα με αποτέλεσμα να πραγματοποιούνται καλύτερες αντιστοιχίσεις σε φωνήματα και χαρακτηριστικά συχνότητας όπως έχει περιγραφεί και στην γενική ανάλυση των συστημάτων αναγνώρισης φωνής παραπάνω .

Με τις αλυσίδες από DataProcessors που έχει το FontEnd μπορεί και πραγματοποιεί πολλές τεχνικές επεξεργασίας σήματος. Έτσι λοιπόν στο εισερχόμενο σήμα από κάποια συσκευή ήχου (πχ μικρόφωνο που συνδέεται με κάποια κάρτα ήχου) μπορούν να γίνουν οι εξής τεχνικές:

1. preemphasis
2. windowing με raised cosine transform (π.χ., με παράθυρα Hamming και Hanning),
3. Discrete Fourier transform (μέσω FFT),
4. Mel frequency filtering,
5. bark frequency warping,
6. discrete cosine transform (DCT),
7. linear predictive encoding (LPC),
8. end pointing
9. ceptral mean normalization (CMN),
10. Mel-ceptra frequency coefficient extraction (MFCC),
11. perceptual linear prediction coefficient extraction (PLP).

Χρησιμοποιώντας το ConfigurationManager που αναφέραμε παραπάνω, οι χρήστες μπορούν να φτιάξουν τις δικές τους αλυσίδες από DataProcessors με οποιοδήποτε τρόπο και συνδυασμό τεχνικών καθώς επίσης και να ενσωματώσουν στις εφαρμογές και κάποιον DataProcessor που έχουν δημιουργήσει μόνοι τους. Έτσι είναι εμφανές ότι μπορεί κάποιος να πειράξει κατά τις δικές του απαιτήσεις και ανάγκες το FrontEnd για να πάρει καλύτερα αποτελέσματα και αυτό δίνει ένα ιδιαίτερο νόημα στην ευελιξία του sphinx-4. Φυσικά η ευελιξία αυτή μπορεί να αποδειχτεί και σε μεγάλο πρόβλημα καθώς

για να πειράξει κανείς τους DataProcessors και να φτιάξει τις δικές του αλυσίδες πρέπει να υπάρχει πολύ μεγάλη προσοχή και γνώση, γιατί με τις αλλαγές αυτές δεν οδηγούμαστε πάντα σε καλύτερα αποτελέσματα. Με διάφορες αλλαγές λοιπόν που μπορεί να πραγματοποιηθούν στους DataProcessors μπορεί αντί να βελτιώσουμε μια κατάσταση να την χειροτερέψουμε.

Συμπερασματικά λοιπόν το FrontEnd είναι το κομμάτι εκείνο που παίρνει το ακουστικό σήμα της φωνής σαν είσοδο και το περνάει μέσα από αλυσίδες από DataProcessors που είναι πολύ έξυπνα δομημένες με στόχο να δημιουργηθούν τα Features –χαρακτηριστικά γνωρίσματα του ήχου στο πεδίο της συχνότητας. Η έξυπνη δομή του FrontEnd επιτρέπει στους χρήστες να χρησιμοποιήσουν πολλές παραμέτρους ή ακόμα και να φτιάξουν δικές τους για να επιτύχουν όσο το δυνατόν καλύτερα αποτελέσματα.

- **LINGUIST:** Το κομμάτι του sphinx-4 που ονομάζεται linguist παράγει όπως είπαμε το SearchGraph που χρησιμοποιείται από τον αποκωδικοποιητή κατά τη διάρκεια της αναζήτησης, αλλά ταυτόχρονα κρύβει και την πολυπλοκότητα της δουλειάς αυτής από τους δημιουργούς εφαρμογών ή χρήστες του συστήματος. Όπως συμβαίνει σε όλα τα κομμάτια του sphinx-4, το linguist είναι μια ευέλικτη και ανοιχτή ενότητα του συστήματος, που επιτρέπει στους χρήστες για να διαμορφώνουν δυναμικά το σύστημα με τα διάφορα γλωσσολογικά στοιχεία που επιθυμούν ή ταιριάζουν στις απαιτήσεις των εφαρμογών τους.

  Μια χαρακτηριστική μορφή linguist κατασκευάζει το SearchGraph χρησιμοποιώντας τη γλωσσική δομή όπως αντιπροσωπεύεται από ένα δεδομένο LanguageModel και την τοπολογική δομή ενός δεδομένου AcousticModel. Έτσι λοιπόν η τυπική διαδικασία ενός linguist είναι να φτιάχνει το SearchGraph χρησιμοποιώντας τα δεδομένα που έχουν παραχθεί από κάποιο εργαλείο εκπαίδευσης και που αυτά είναι τα δεδομένα φωνής (φωνήματα και τρι-φωνήματα σε αντιστοίχηση με τα Features που τα αποτελούν) και τα γλωσσολογικά δεδομένα όπως λεξικά, γραμματική και σύνταξη όταν υπάρχει. Το Linguist μπορεί να χρησιμοποιήσει ένα λεξικό

(δηλαδή ένα λεξικό προφοράς) για να μπορέσει να αντιστοιχήσει τις λέξεις από το LanguageModel στις ακολουθίες στοιχείων του AcousticModel. Αυτό δηλαδή που γίνεται εδώ είναι η αντιστοίχηση των φωνημάτων σε ακολουθίες με τέτοιο τρόπο ώστε να ταιριάζουν σε λέξεις που περιέχονται σε κάποιο λεξικό προφοράς και το οποίο περιλαμβάνει τις λέξεις και τα φωνήματα ή τριφωνήματα τα οποία τις αποτελούν.

Με την ανεκτικότητα του sphinx-4 να μπορούν οι χρήστες να χρησιμοποιούν διαφορετικά γλωσσολογικά χαρακτηριστικά κατά την διάρκεια τρεξίματος του συστήματος, το sphinx-4 δίνει την δυνατότητα να μπορεί να προσαρμοστεί η αναγνώριση των λέξεων σε κάθε ανάγκη κάποιας εφαρμογής ακόμα και να αλλάζει δυναμικά ο τρόπος της αναγνώρισης κατά την διάρκεια που το σύστημα τρέχει. Παραδείγματος χάριν, μια απλή αριθμητική εφαρμογή αναγνώρισης ψηφίων μπορεί να χρησιμοποιήσει ένα απλό linguist που κρατά το διάστημα αναζήτησης εξ ολοκλήρου στη μνήμη. Αφ' ετέρου, μια εφαρμογή υπαγόρευσης με ένα λεξιλόγιο 100.000 λέξεων μπορεί να χρησιμοποιήσει ένα περίπλοκο linguist που κρατά μόνο μια μικρή μερίδα του πιθανού διαστήματος αναζήτησης στη μνήμη σε κάθε χρονική στιγμή. Το linguist αποτελείται από τρία δομικά στοιχεία το Language model, το Dictionary και το Acoustic Model και τα οποία είναι και τα τρία ανοιχτά σε σχεδιασμό και υλοποίηση και αναλύονται διεξοδικά στην συνέχεια.

A. LanguageModel : το LanguageModel του Linguist αποτελεί την δομή των λέξεων. Ουσιαστικά υπάρχουν δυο κατηγορίες δομής του LanguageModel : η graph-driven grammar και το stochastic N-Gram model. Η graph-driven grammar αντιπροσωπεύει μια κατευθυνόμενη γραφική παράσταση λέξης όπου κάθε κόμβος αντιπροσωπεύει μια μεμονωμένη λέξη και κάθε τόξο αντιπροσωπεύει την πιθανότητα να γίνει μια μετάβαση σε μια λέξη. Το stochastic N-gram model παρέχει την πιθανότητα για την νιοστή λέξη δεδομένου ότι έχουν παρατηρηθεί οι προηγούμενες ν-1 λέξεις.

Το sphinx-4 υποστηρίζει κάποιους τρόπους δομής του LanguageModel και αυτοί είναι οι παρακάτω με μια σύντομη περιγραφή ο καθένας :

SimpleWordListGrammar: καθορίζει μια γραμματική που βασίζεται σε έναν κατάλογο λέξεων. Μια προαιρετική παράμετρος καθορίζει αν και κατά πόσο η γραμματική αυτή επαναλαμβάνεται στον χρόνο. Εάν η γραμματική δεν επαναλαμβάνεται, τότε θα χρησιμοποιηθεί για την απομονωμένη αναγνώριση μιας λέξης από αυτές που υπάρχουν στον κατάλογο. Εάν υπάρχει επανάληψη τότε θα γίνει ξανά αναγνώριση των λέξεων μετά την αναγνώριση της πρώτης αλλά όλες οι λέξεις έχουν την ίδια πιθανότητα να αναγνωριστούν. Αυτό σημαίνει ότι τυχόν θόρυβος ή κακή ποιότητα μικροφώνου που αλλοιώνουν τα σήματα μπορούν πολύ εύκολα να οδηγήσουν σε λάθος αναγνώριση καθώς δεν υπάρχει κάποια πιθανότητα των λέξεων που πέρασαν για να μπορέσουν με βάση αυτή να διορθωθούν κάποια λάθη.

JSGFGrammar: υποστηρίζει το Java Speech API (JSGF). Είναι μια γραμματική η οποία έχει αυστηρό τρόπο γραφής όπου μπορεί κάποιος να φτιάξει ένα μοντέλο για να μπορεί να κάνει αναγνωρίσεις εντολών ή δεδομένων προτάσεων πεπερασμένου αριθμού. Συνοδεύεται από κάποιο λεξικό με τις λέξεις και ένα αρχείο που έχει την σύνταξη με την οποία θα αναγνωρισθούν οι λέξεις. Δεν είναι κατάλληλη για εφαρμογές αναγνώρισης ελεύθερου κειμένου αλλά είναι ιδανική για εφαρμογές που έχουν περιορισμένο αριθμό λέξεων και προκαθορισμένη σύνταξη. Η γραμματική μορφή αυτή επειδή έχει χρησιμοποιηθεί και στην εφαρμογή μας θα αναλυθεί μέσω παραδείγματος και στην ανάλυση της εφαρμογής.

LMGrammar: καθορίζει μια γραμματική που βασίζεται σε ένα στατιστικό LanguageModel. Το LMGrammar παράγει έναν κόμβο γραμματικής ανά λέξη και λειτουργεί καλά μέχρι περίπου 1000 λέξεις ενώ για μεγαλύτερους αριθμούς λέξεων αντιμετωπίζει δυσκολίες τόσο υπολογιστικές όσο και λαθών κατά την αναγνώριση.

LargeTrigramModel: παρέχει την υποστήριξη για τα αληθινά στοιχεία N-Gram και μπορεί να υποστηρίξει τεράστιους αριθμούς λέξεων και προτάσεων καθώς χρησιμοποιεί αρκετούς στατιστικούς και μαθηματικούς υπολογισμούς για να μπορεί να επεξεργαστεί καλύτερα και γρηγορότερα τεράστιους αριθμούς δεδομένων. Είναι ότι καλύτερο για την αναγνώριση ελεύθερου κειμένου αλλά λόγω της κατανάλωσης υπολογιστικών πόρων δεν συνιστάται για εφαρμογές μικρών απαιτήσεων σε αναγνώριση λέξεων.

B. Dictionary: το λεξικό παρέχει τις προφορές για τις λέξεις που βρίσκονται στο LanguageModel. Οι προφορές σπάνε τις λέξεις σε ακολουθίες από υπολέξεις που βρίσκονται στο AcousticModel. Αυτές οι ακολουθίες υπο-λέξεων δεν είναι τίποτα άλλο από τα δομικά στοιχεία του AcousticModel που είναι τα φωνήματα και τα τρι-φωνήματα. Δηλαδή μια λέξη στο λεξικό έχει μια αντιστοίχηση από ηχητικά σήματα που αντιλαμβάνεται ο υπολογιστής όταν προφέρουμε την λέξη αυτή. Στο sphinx-4 υπάρχει η δυνατότητα κατά την δημιουργία του λεξικού κάθε λέξη να υπάρχει όσες φορές χρειάζεται με διαφορετική προφορά για να μπορέσουμε να πιάσουμε όσον το δυνατόν περισσότερες ιδιομορφίες της φωνής ή ιδιωματισμούς της γλώσσας κατά την διαδικασία της αναγνώρισης. Πρέπει να τονίσουμε στο σημείο αυτό ότι όσο καλά προσπαθούμε να ρυθμίσουμε το σύστημα μας να αναγνωρίζει τα φωνήματα από την φωνή μας άλλο τόσο καλά θα πρέπει να φτιάξουμε το λεξικό που θα χρησιμοποιήσουμε γιατί μπορεί τα φωνήματα να αναγνωρίζονται σωστά από το σύστημα αλλά να μην μπορεί να γίνει η σωστή ταύτιση με μια λέξη επειδή δεν έχει φτιαχτεί σωστά η προφορά της στο λεξικό.

Γ. AcousticModel : Το AcousticModel παρέχει μια αντιστοίχηση μεταξύ μιας μονάδας ομιλίας με τα εισερχόμενα χαρακτηριστικά γνωρίσματα που βγαίνουν σαν έξοδος από το FrontEnd. Η μονάδα ομιλίας δεν είναι κατά βάση λέξη μπορεί να είναι κάποιος φθόγγος ή ακόμα και το silent φώνημα που υπάρχει στην ομιλία μας και διαχωρίζει την μια λέξη από την άλλη καθώς και το ένα γράμμα από το άλλο καθώς μιλάμε. Εδώ λοιπόν υπάρχει και η διαδικασία των μοντέλων Hidden Markov (HMM) που βρίσκουν αυτά τα silent φωνήματα και μπορούν να κάνουν διακριτό στο σύστημα που τελειώνει το ένα και που αρχίζει το επόμενο φώνημα.

Όπως και σε άλλα συστήματα, η αντιστοίχηση μπορεί να γίνει εκτός από τις πληροφορίες για το ποια χαρακτηριστικά γνωρίσματα αναγνωρίζονται από το FrontEnd και από την θέση των χαρακτηριστικών αυτών τόσο σε επίπεδο λέξης όσο και σε επίπεδο σύνταξης. Παραδείγματος χάριν, στην περίπτωση των τρι-φωνημάτων , το επίπεδο σύνταξης αντιπροσωπεύει τα φωνήματα που βρίσκονται δεξιά και αριστερά από το φώνημα που αναγνωρίζουμε από το FrontEnd και το επίπεδο λέξης αντιπροσωπεύει αν το τρι-φώνημα βρίσκεται

στην αρχή ή στη μέση ή στο τέλος κάποιας λέξης ή ακόμα αν το τρι-φώνημα είναι το ίδιο μια λέξη.

Χαρακτηριστικά, το Linguist σπάει κάθε λέξη από το ενεργό λεξιλόγιο σε μια ακολουθία από υπόλέξεις (φωνήματα και τρι-φωνήματα) τα οποία και έχουν συνάφεια ως προς το περιεχόμενο. Περνά έπειτα τις υπόλέξεις αυτές και τα περιεχόμενα τους στο AcousticModel, που ανακτά τις γραφικές παραστάσεις HMM που συνδέονται με εκείνες τις υπόλέξεις. Χρησιμοποιεί έπειτα αυτές τις γραφικές παραστάσεις HMM από κοινού με το LanguageModel για να κατασκευάσει το SearchGraph.

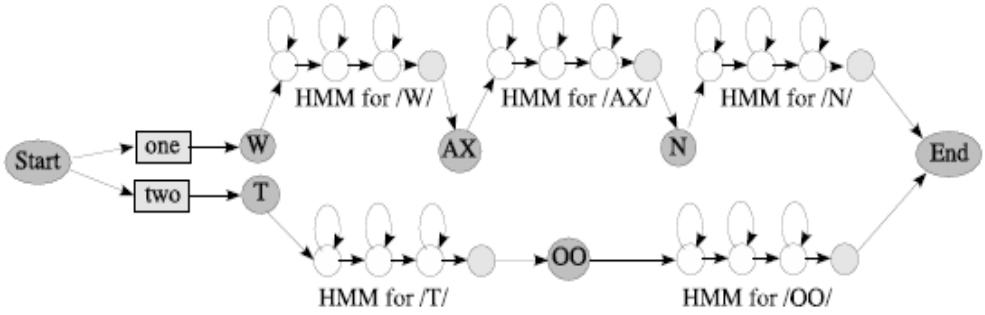
Αντίθετα από τα περισσότερα συστήματα αναγνώρισης φωνής , που αντιπροσωπεύουν τις γραφικές παραστάσεις HMM ως σταθερή δομή στη μνήμη, το sphinx-4 οι γραφικές παραστάσεις HMM δεν είναι στατικές στην μνήμη. Εδώ κάθε κόμβος αντιστοιχεί σε μια κατάσταση HMM και κάθε τόξο αντιπροσωπεύει την πιθανότητα της μετάβασης από μια κατάσταση HMM σε μια άλλη. Με αυτόν τον τρόπο μπορούμε εύκολα να περάσουμε από μια κατάσταση HMM σε μία άλλη προς όλες τις κατευθύνσεις με κριτήριο πάντα την πιθανότητα της μετάβασης από την μια κατάσταση στην άλλη. Για παράδειγμα ας υποθέσουμε ότι έχουμε αναγνωρίσει το κομμάτι “New Y” και στην συνέχεια δεν αναγνωρίζουμε το επόμενο φώνημα γιατί υπάρχει μεγάλη ποσότητα θορύβου από το περιβάλλον. Η πιθανότητα το φώνημα να είναι το “o” είναι πολύ μεγάλη και αντίστοιχα η πιθανότητα να οδηγηθούμε στην κατάσταση “New Yo” είναι πολύ μεγαλύτερη από ότι να οδηγηθούμε σε μια κατάσταση όπως η “New Ya” και αυτό επειδή κατά την χρήση του εργαλείου εκπαίδευσης έχει συναντηθεί πολλές φορές η φράση “New York” και οι ενδιάμεσες καταστάσεις έχουν μεγάλη πιθανότητα μετάβασης.

Φυσικά η διαδικασία μετάβασης μπορεί να γίνει προς όλες τις κατευθύνσεις και προς τα πίσω αν για παράδειγμα το επόμενο φώνημα στο παράδειγμα που αναφέρθηκε είναι το “a”τότε η πιθανότητα να προχωρήσουμε στην κατάσταση “New Yoa” είναι πολύ μικρή και η πιθανότητα να έγινε λάθος αναγνώριση και να έχουμε “New ya” μεγαλώνει. Αν λοιπόν το επόμενο φώνημα που θα αναγνωριστεί είναι το “r”τότε η κατάσταση “New yar” έχει πολύ μεγάλη πιθανότητα και μπορεί να καταλήξει στην φράση “New yard”. Βλέπουμε λοιπόν ότι με το να έχουμε κίνηση των καταστάσεων προς όλες τις

κατευθύνσεις μπορούμε να επιτύχουμε πολύ καλύτερα αποτελέσματα αναγνώρισης.

Δ. SearchGraph : Ακόμα κι αν οι Linguists μπορούν να υλοποιηθούν με πολύ διαφορετικούς τρόπους και οι τοπολογίες των διαστημάτων αναζήτησης που χρησιμοποιούνται από αυτούς μπορούν να ποικίλουν πολύ. Όλα τα διαστήματα αναζήτησης χαρακτηρίζονται ως SearchGraphs. Το SearchGraph είναι η δομή δεδομένων που χρησιμοποιείται κατά τη διάρκεια της διαδικασίας αποκωδικοποίησης.

Ένα παράδειγμα SearchGraph φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.2.3 SearchGraph

Όπως βλέπουμε και από το παραπάνω σχήμα το SearchGraph αποτελείται από κόμβους και μεταβάσεις. Κάθε κόμβος-SearchState αποτελείται από την χρησιμοποίηση μιας γραφικής παράστασης Hidden Markov Model. Έτσι λοιπόν με τα αποτελέσματα που παίρνει το Acoustic Model από το FrondEnd για το ποια χαρακτηριστικά στο πεδίο της συχνότητας (Features) αναγνωρίζονται καταλήγει σε κάποια φωνήματα με βάση τις βάσεις δεδομένων που έχουν δημιουργηθεί κατά την φάση της εκπαίδευσης του συστήματος. Κάθε τέτοιο φώνημα αποτελεί μια κατάσταση στο SearchGraph. Σε κάθε τέτοια κατάσταση γίνεται υλοποίηση των μοντέλων Hidden Markov (ουσιαστικά χρησιμοποιούνται οι γραφικές παραστάσεις για το κάθε φώνημα) για να υπολογιστούν οι πιθανότητες του ποια είναι η επόμενη κατάσταση, ποια η πιθανότητα μετάβασης στην τωρινή κατάσταση δεδομένης της πιθανότητας της προηγούμενης κατάστασης, ποια η πιθανότητα να έχουμε ένα silent φώνημα κλπ. Με βάση λοιπόν τις υπολογισμένες πιθανότητες έχουμε και την μετάβαση από κατάσταση σε κατάσταση. Στο παραπάνω γράφημα

βλέπουμε ότι έχει δημιουργηθεί ένα SearchGraph για την αναγνώριση 2 λέξεων των "one" και "two" και παρατηρούμε ότι υπάρχει ένας κόμβος-κατάσταση για κάθε φώνημα που αναγνωρίζεται από το AcousticModel από τα δεδομένα που του στέλνει το ForntEnd και τα οποία με την σειρά τους προέρχονται από την φωνή του χρήστη.

Είναι λογικό λοιπόν ότι το SearchGraph είναι μια διαδικασία παραγωγής ενός αποτελέσματος από τα δεδομένα των προηγούμενων δομικών στοιχείων του Linguist. Το αποτέλεσμα αυτό φτιάχνεται με τέτοιον τρόπο ώστε να είναι κατανοητός από τον Decoder που ακολουθεί και ο οποίος είναι υπεύθυνος για την τελική απόφαση της αναγνώρισης της κάθε λέξης.

Αυτό που πρέπει να γίνει αντιληπτό στο σημείο αυτό είναι ότι το SearchGraph δεν πραγματοποιεί κάποια αναγνώριση λέξης αλλά αποτελεί ένα σύνολο από ενδεχόμενες λέξεις που δημιουργούνται με βάση τις πιθανότητες που έχουν φτιαχτεί από τα ηχητικά δεδομένα καθώς αυτές περνάνε από τα επιμέρους στοιχεία του συστήματος.

Η απόφαση για το ποια λέξη είναι αυτή που θα επιστραφεί σαν αποτέλεσμα αναγνώρισης παίρνεται από τον Decoder ο οποίος όμως στηρίζεται στο SearchGraph με τρόπο που θα εξηγήσουμε στην συνέχεια.

Ένα κύριο χαρακτηριστικό του SearchGraph είναι ότι η υλοποίηση του SearchState δεν χρειάζεται να καθοριστεί. Αυτό σημαίνει ότι για κάθε εφαρμογή ο τρόπος υλοποίησης του SearchState μπορεί να ποικίλει, βασισμένος στα χαρακτηριστικά του Linguist που όπως έχουμε πει μπορεί να διαφέρει από εφαρμογή σε εφαρμογή για να ικανοποιεί διαφορετικές ανάγκες αναγνώρισης. Παραδείγματος χάριν, ένας απλός Linguist ο οποίος χρησιμοποιείται για μια εφαρμογή με μικρό λεξιλόγιο μπορεί να παρέχει μια στατική καταχώριση για το SearchGraph στην μνήμη όπου κάθε SearchState είναι απλά μια προς μια αντιστοίχηση για τις δυνατές καταστάσεις που μπορούν να πάρουν όλες οι λέξεις του λεξικού. Ένας Linguist που αντιπροσωπεύει ένα πολύ μεγάλο και σύνθετο λεξιλόγιο, εντούτοις, μπορεί να δημιουργήσει μια πιο δυναμική και καλύτερα εξελισσόμενη αντιπροσώπευση του SearchGraph. Σε αυτήν την περίπτωση, ο Linguist θα παρήγαγε το σύνολο από τα διαδοχικά SearchStates με δυναμικό τρόπο όταν αυτό ήταν αναγκαίο. Αυτό είναι πολύ σημαντικό χαρακτηριστικό καθώς έτσι υπάρχει η δυνατότητα

να υποστηριχθούν εφαρμογές που έχουν πολύ μεγάλα λεξιλόγια και λίγους συντακτικούς περιορισμούς όπως για παράδειγμα είναι η αναγνώριση ελεύθερου κειμένου.

Είναι λοιπόν λογικό ότι η απόφαση για το ποια θα είναι η υλοποίηση του SearchGraph είναι πολύ σημαντική γιατί μπορεί από την μια να οδηγήσει σε λάθος αποτελέσματα αναγνώρισης ή από την άλλη σε λάθος κατανομή των πόρων του συστήματος στο οποίο τρέχει η εφαρμογή. Ακόμα δηλαδή και αν έχουμε πολύ καλά ποσοστά αναγνώρισης αλλά καταπονούμε το σύστημα χρησιμοποιώντας πόρους που δεν χρειάζεται με βάση τις ανάγκες της εφαρμογής μας η στρατηγική υλοποίησης του SearchGraph είναι λάθος.

- **Ο Αποκωδικοποιητής-Decoder** ο βασικός ρόλος του Decoder στο sphinx-4 είναι να χρησιμοποιεί τα χαρακτηριστικά γνωρίσματα-Features από το FrontEnd από κοινού με το SearchGraph από το Linguist για να παραγάγει τα αποτελέσματα. Ο Decoder περιλαμβάνει ένα στοιχείο που ονομάζεται SearchManager και ουσιαστική αποτελεί την μονάδα εκείνη που κάνει την αποκωδικοποίηση για μια εφαρμογή.

  Ο αποκωδικοποιητής λέει μόνο στον SearchManager να αναγνωρίσει ένα σύνολο χαρακτηριστικών γνωρισμάτων-Features δομημένα σε πλαίσια-frames. Σε κάθε βήμα της διαδικασίας, ο SearchManager δημιουργεί ένα αποτέλεσμα που περιέχει όλες τις πορείες που έχουν φθάσει σε ένα τελικό στάδιο από το SearchGraph. Για να επεξεργαστεί τα αποτελέσματα αυτά που προκύπτουν , το sphinx-4 παρέχει κάποιες εφαρμογές ενσωματωμένες μέσα στον SearchManager που προσδίδουν με βάση τα στοιχεία της εκπαίδευσης του συστήματος, κάποια scores για κάθε αποτέλεσμα με σκοπό να γίνει εμφανές στο σύστημα ποιο αποτέλεσμα είναι καλύτερο.

  Έτσι λοιπόν αυτό που κάνει ο SearchManager είναι να συγκεντρώνει τα στοιχεία από τις υπόλοιπες μονάδες του sphinx-4 και να παράγει τα αποτελέσματα τις αναγνώρισης. Δηλαδή παίρνει την έξοδο του FrontEnd που είναι τα Features από την πηγή ήχου από την μια μεριά και από την άλλη το SearchGraph που παράγεται από τον Linguist. Με βάση τα στοιχεία αυτά βρίσκει πιο είναι το πιο πιθανό αποτέλεσμα που συμφωνεί με τα πραγματικά ηχητικά δεδομένα που φτάνουν στο σύστημα (μέσω του FrontEnd) αλλά και

με τις συντακτικές και γλωσσολογικές απαιτήσεις που υπάρχουν στο Linguist. Ο συνδυασμός των δεδομένων από αυτές τις δυο πηγές (FrontEnd-Linguist) βοηθούν το σύστημα να εξασφαλίσει καλή πιθανότητα αναγνώρισης ακόμα και σε περιπτώσεις που ο θόρυβος είναι αρκετός και άρα τα ηχητικά δεδομένα δεν είναι επαρκούν αλλά και σε περιπτώσεις που δεν έχει δομηθεί καλά το γλωσσολογικό μοντέλο.

3.3 Jadex

3.3.1 Εισαγωγή

Στις μέρες υπάρχουν πολλές διαφορετικές πλατφόρμες που υποστηρίζουν την χρήση agents, και υπάρχει ολοένα και μεγαλύτερο ενδιαφέρον από τους developers να προσαρμοστούν οι πλατφόρμες αυτές σε multi-agent περιβάλλον. Δυστυχώς οι περισσότερες από αυτές τις πλατφόρμες επικεντρώνονται σε συγκεκριμένους στόχους και έχουν φτιαχτεί για να εξυπηρετούν πολύ συγκεκριμένες ανάγκες και δεν μπορούν να υποστηρίξουν όλες τις σημαντικές πτυχές της τεχνολογίας των agents καλά. Μια γενική διάκριση στον τομέα των συστημάτων που υποστηρίζουν την τεχνολογία των agents είναι τα middleware συστήματα και τα reasoning-oriented συστήματα.
Η πρώτη κατηγορία στηρίζεται και υλοποιεί τα πρότυπα της FIPA. Η FIPA είναι μια κοινότητα πληροφορικής της IEEE η οποία προωθεί την τεχνολογία των agents και την διαλειτουργικότητα της τεχνολογίας αυτής με άλλες τεχνολογίες. Ως εκ τούτου τα middleware συστήματα μπορούν να υποστηρίξουν πολύ καλά την τεχνολογία των agents. Οι περισσότερες πλατφόρμες middleware αφήνουν ανοικτό σκόπιμα το ζήτημα της εσωτερικής αρχιτεκτονικής των agents και υιοθετούν μια προσανατολισμένη προς απλούς στόχους προσέγγιση. Αυτή η προσέγγιση επιτρέπει να γίνει μια αποσύνθεση της συνολικής συμπεριφοράς των agents σε μικρότερα κομμάτια και απλοποιεί την όλη διαδικασία της δημιουργίας κάποιας εφαρμογής με την χρήση των agents. Επιπλέον οι στόχοι οι ίδιοι μπορούν να γραφτούν σε μια αντικειμενοστρεφή γλώσσα προγραμματισμού όπως είναι η Java που επιτρέπει στον υπεύθυνο για την ανάπτυξη λογισμικού για να προσαρμόσει εύκολα τους στόχους που θέλει να υλοποιήσει με την τεχνολογία των agents. Αυτό που συμβαίνει εδώ λοιπόν είναι ότι κάποιος developer δεν χρειάζεται να έχει απόλυτη γνώση με το τι

γίνεται στο κομμάτι των agents αλλά μπορεί να φτιάχνει εφαρμογές που χρησιμοποιούν την τεχνολογία αυτή χρησιμοποιώντας κάποια components της τεχνολογίας και γράφοντας την όλη δομή της εφαρμογής σε μια εκ των προτέρων γνωστή γλώσσα προγραμματισμού όπως είναι η Java. Αυτό είναι αρκετά καλό για ανθρώπους που θέλουν να χρησιμοποιήσουν την τεχνολογία των agents χωρίς όμως να πρέπει να ασχοληθούν με την δομή και τις αρχές της τεχνολογίας αλλά χρησιμοποιώντας κάποια κομμάτια αυτής σαν διεπαφές επικοινωνίας. Φυσικά εδώ υπάρχουν αρκετοί περιορισμοί σε σχέση με τις δυνατότητες που έχει η τεχνολογία των agents και υποστηρίζονται πολύ βασικές δυνατότητες που όμως είναι αρκετές για την δημιουργία διαφόρων ειδών εφαρμογών.

Αντίθετα, οι reasoning-oriented πλατφόρμες εστιάζουν στο πρότυπο συμπεριφοράς ενός απλού πράκτορα και προσπαθούν να επιτύχουν καθοδήγηση στον στόχο μέσω δυνατοτήτων του agent και χαρακτηριστικά της δομής του. Τα περισσότερα επιτυχή πρότυπα συμπεριφοράς είναι βασισμένα σε προσαρμοσμένες θεωρίες που προέρχονται από επιστήμες όπως η φιλοσοφία, η ψυχολογία και η βιολογία. Ανάλογα με το επίπεδο λεπτομέρειας της θεωρίας που χρησιμοποιείται σε κάποια περίπτωση τα πρότυπα συμπεριφοράς μπορούν να γίνουν περίπλοκα και μπορούν να οδηγήσουν σε αρχιτεκτονικές που είναι δύσκολο να υλοποιηθούν και χρησιμοποιηθούν κατά την δημιουργία κάποιας εφαρμογής. Ειδικά όταν πρέπει να χρησιμοποιηθούν η τεχνητή νοημοσύνη και θεωρητικές τεχνικές όπως οι λογικές αφαίρεσης (deduction logics) είναι πολύ δύσκολη από θέμα υλοποίησης η κατασκευή των agents και μπορεί να αποδειχτεί στην πράξη μια ακατόρθωτη δουλειά. Φυσικά εκτός από τις δυσκολίες υπάρχει και το πολύ θετικό κομμάτι που είναι ότι μπορούμε με την χρήση τέτοιων πλατφορμών να φτιάξουμε εφαρμογές που πλησιάζουν πολύ κοντά σε ανθρώπινα χαρακτηριστικά όπως είναι η λογική, το συναίσθημα και η εξέλιξη, φυσικά όσο αυτό μπορεί να γίνει υπερνικώντας τους περιορισμούς που βάζουν τα ίδια τα συστήματα που χρησιμοποιούμε ή οι ανάγκες των εφαρμογών μας.

Το Jadex που παρουσιάζεται εδώ είναι μια πλατφόρμα που υποστηρίζει το reasoning και χρησιμοποιεί τεχνολογίες που είναι ανεξάρτητες από το υλικό και το λογισμικό που χρησιμοποιούν τα συστήματα μας, όπως είναι η XML και η Java. Έτσι λοιπόν με την χρήση της πλατφόρμας αυτής μπορούμε από την μια να φτιάξουμε εφαρμογές σε ανώτερο επίπεδο χρησιμοποιώντας χαρακτηριστικά της ανθρώπινης συμπεριφοράς με τέτοιον όμως τρόπο που μπορεί το αποτέλεσμα της δουλειάς μας να παίζει σε

οποιοδήποτε υλικό και σε συνεργασία με οποιοδήποτε λογισμικό. Έτσι έχουμε μια πολύ καλή πλατφόρμα για να φτιάξουμε εφαρμογές που μπορούν να υποστηρίξουν ιδιαιτέρες και δύσκολες ανάγκες, δημιουργώντας ένα έξυπνο περιβάλλον λειτουργίας.

3.3.2 Reasoning

Για να δημιουργηθούν agents με γνωστικές δυνατότητες μπορούν να χρησιμοποιηθούν διάφορες αρχιτεκτονικές από τα επιστημονικά πεδία όπως την ψυχολογία, τη φιλοσοφία και τη βιολογία. Οι περισσότερες γνωστικές αρχιτεκτονικές είναι βασισμένες σε θεωρίες για την περιγραφή της ανθρώπινης συμπεριφοράς. Οι πιο σημαντικές θεωρίες σε αυτόν τον χώρο είναι: το Believe-Desire-Intension (BDI) model, η θεωρία του Agent Oriented Programming (AOP), οι Unified Theories of Cognition (UTC που είναι μια θεωρία στην οποία στηρίζεται και η τεχνολογία SOAP)και η Subsumption Theory. Κάθε μια από αυτές τις θεωρίες αυτές έχει τα προτερήματα και τα μειονεκτήματα της και υποστηρίζει ορισμένα είδη εφαρμογών αρκετά καλά η κάθε μια. Η μηχανή του Jadex είναι βασισμένη στο πρότυπο BDI λόγω της απλότητας που προσφέρει αλλά και λόγω του ότι μπορεί να υποστηρίξει πολύ καλά ένα μεγάλο εύρος εφαρμογών χρησιμοποιώντας θεωρίες της ανθρώπινης συμπεριφοράς. Στην συνέχεια γίνεται μια μικρή ανάλυση του Believe-Desire-Intension (BDI) model.

3.3.3 BDI

Το BDI model ανακλύφτηκε σαν ιδέα από τον M. Bratman ως θεωρία του ανθρώπινου πρακτικού συλλογισμού. Η επιτυχία του είναι βασισμένη στην απλότητα του που μειώνει την εξήγηση της σύνθετης ανθρώπινης συμπεριφοράς και ασχολείται μόνο με τα κίνητρα του ανθρώπου. Αυτό σημαίνει ότι οι αιτίες για τις ενέργειες που πραγματοποιεί κάποιος άνθρωπος συσχετίζονται πάντα με τις ανθρώπινες επιθυμίες και μπορούν έτσι να παραληφθούν άλλες ανθρώπινες καταστάσεις όπως είναι τα συναισθήματα που είναι πάρα πολύ δύσκολο να προσεγγιστούν στα πλαίσια ενός συστήματος. Ένα άλλο θετικό του BDI model είναι η χρήση κάποιων ψυχολογικών εννοιών που μοιάζουν πάρα πολύ με τον τρόπο που ο άνθρωπος μιλάει για ζητήματα

συμπεριφοράς. Δηλαδή μπορούν να υλοποιηθούν ανάγκες εφαρμογών που προσπαθούν να προσεγγίσουν τις ανθρώπινες συμπεριφορές και σε ψυχολογικό επίπεδο.

Ας δούμε όμως κάποια δομικά στοιχεία του μοντέλου αυτού.

Believes

Τα believes(πεποιθήσεις) είναι χαρακτηριστικά που περιέχουν πληροφορίες σχετικά με τους agents, δηλαδή οι πεποιθήσεις αντιπροσωπεύουν τις πληροφορίες που έχει ένας agent για τον κόσμο-περιβάλλον στο οποίο βρίσκεται, αλλά και τις πληροφορίες για την δική του θέση στο περιβάλλον αυτό. Οι πεποιθήσεις δεν αντιπροσωπεύουν μόνο τις οντότητες ενός περιβάλλοντος λειτουργίας σαν ένα είδος μια προς μια αντιστοίχηση αλλά σαν μια αφαιρετική αναπαράσταση οντοτήτων με το να δίνουν έμφαση στις σημαντικές ιδιότητες των οντοτήτων αυτών και να παραλείπουν λεπτομέρειες που δεν είναι σημαντικές για το περιβάλλον λειτουργίας. Έτσι λοιπόν δημιουργείται μια "άποψη" μέσα στον agent, δηλαδή ένας μηχανισμός αξιολόγησης για να μπορεί να κρίνει τις οντότητες που υπάρχουν στο περιβάλλον του αλλά και να αξιολογεί τις συμπεριφορές τους και έτσι μπορούμε να πούμε ότι με αυτόν τον τρόπο ο κάθε agent έχει μια πεποίθηση για το περιβάλλον του και μπορεί με την πεποίθηση του αυτή να αξιολογεί τόσο τις υπόλοιπες οντότητες όσο και να μπορεί να σκεφτεί για τυχόν ενέργειες που θα πρέπει να γίνουν κατά την επικοινωνία του με αυτές τις οντότητες.

Goals

Οι διαδικασίες που περιλαμβάνουν κάποιο κίνητρο συμπεριφοράς για κάποιον agent ονομάζονται Goals (στόχοι). Αντιπροσωπεύουν τις επιθυμίες του agent και ορίζουν την πορεία των ενεργειών του. Οι στόχοι δεν χρειάζεται να εκτελούνται απαραιτήτως και επομένως ίσως δεν μπορούν να επιτευχθούν ταυτόχρονα.

Σε μια αρχιτεκτονική προσανατολισμένη στους στόχους-goals, μπορούν να χρησιμοποιηθούν διαφορετικού τύπου στόχοι, όπως για παράδειγμα είναι στόχοι που προσπαθούν να φτάσουν τον agent σε μια κατάσταση ή στόχοι που προσπαθούν να διατηρήσουν την υπάρχουσα κατάσταση του agent. Φυσικά υπάρχουν και διάφορες ενέργειες που γίνονται για την επίτευξη των στόχων αυτών. Κάθε ενέργεια παρακολουθείται και σε περίπτωση που υπάρχει αποτυχία σε κάποια ενέργεια

υπάρχουν μηχανισμοί ασφαλείας που επιβάλλουν την επανάληψη της συγκεκριμένης ενέργειας ή την ενεργοποίηση άλλων παράπλευρων ενεργειών με στόχο την επίτευξη των στόχων.

Plans

Τα σχέδια-plans είναι τα μέσα με τα οποία οι agents επιτυγχάνουν τους στόχους τους και αντιδρούν στα εμφανιζόμενα γεγονότα. Με αυτόν τον τρόπο ένα σχέδιο είναι όχι μόνο μια ακολουθία βασικών ενεργειών, αλλά μπορεί επίσης να εμπεριέχει και πιο αφηρημένα στοιχεία όπως τα subgoals-υποστόχοι. Υπάρχει περίπτωση να εκτελούνται κάποια σχέδια για να επιτύχουν τα subgoals ενός άλλου σχεδίου και με αυτόν τον τρόπο δημιουργείται μια ιεραρχία σχεδίων. Όταν ένας agent αποφασίζει να υλοποιήσει έναν στόχο κάποιου σχεδίου, δεσμεύεται (προς στιγμήν) σε αυτό το είδος ολοκλήρωσης στόχου και ως εκ τούτου έχει δηλώσει σε όλο το περιβάλλον λειτουργίας του την πρόθεση ότι θα εκτελέσει συγκεκριμένες ενέργειες προς την υλοποίηση του στόχου αυτού. Με αυτόν τον τρόπο ενημερώνεται όλο το περιβάλλον του jadex ότι ο συγκεκριμένος agent προσπαθεί να υλοποιήσει κάποιον συγκεκριμένο στόχο-goal που περικλείεται σε κάποιο συγκεκριμένο σχέδιο-plan. Η ευελιξία στα σχέδια BDI επιτυγχάνεται από το συνδυασμό δύο δυνατοτήτων. Η πρώτη δυνατότητα αφορά τη δυναμική επιλογή των κατάλληλων σχεδίων για έναν ορισμένο στόχο που εκτελείται με μια διαδικασία αποκαλούμενη "meta-level reasoning". Αυτή η διαδικασία αποφασίζει με βάση την πραγματική κατάσταση του όλου συστήματος για το ποιο σχέδιο έχει την καλύτερη πιθανότητα αν εκτελεστεί να υλοποιηθεί ο στόχος. Εάν ένα σχέδιο δεν είναι επιτυχές, η "meta-level reasoning" διεργασία μπορεί να εκτελέσει κατάλληλες ενέργειες για την αποκατάσταση του συστήματος σε περιπτώσεις λαθών κατά την υλοποίηση των σχεδίων. Η δεύτερη δυνατότητα αφορά τον καθορισμό των σχεδίων, ο οποίος μπορεί να διευκρινιστεί σαν μια ακολουθία από πιο αφηρημένα σχέδια που αποτελούνται μόνο από subgoals-υποστόχους στα πιο συγκεκριμένα σχέδια τα οποία αποτελούνται μόνο από βασικές ενέργειες και είναι πολύ σαφή στην δομή τους. Με αυτές τις δυο δυνατότητες δίνεται η δυνατότητα στο σύστημα να μπορεί να αποφασίζει ποιος agent θα κάνει τι, με ποιον τρόπο θα το κάνει, τι θα γίνει αν αυτό που πρέπει να κάνει ο agent αποτύχει και ποιες πρέπει να είναι οι ενέργειες που θα ακολουθηθούν, αλλά και ακριβώς ποια είναι η δομή του τι ακριβώς θα πρέπει να κάνει κάποιος agent.

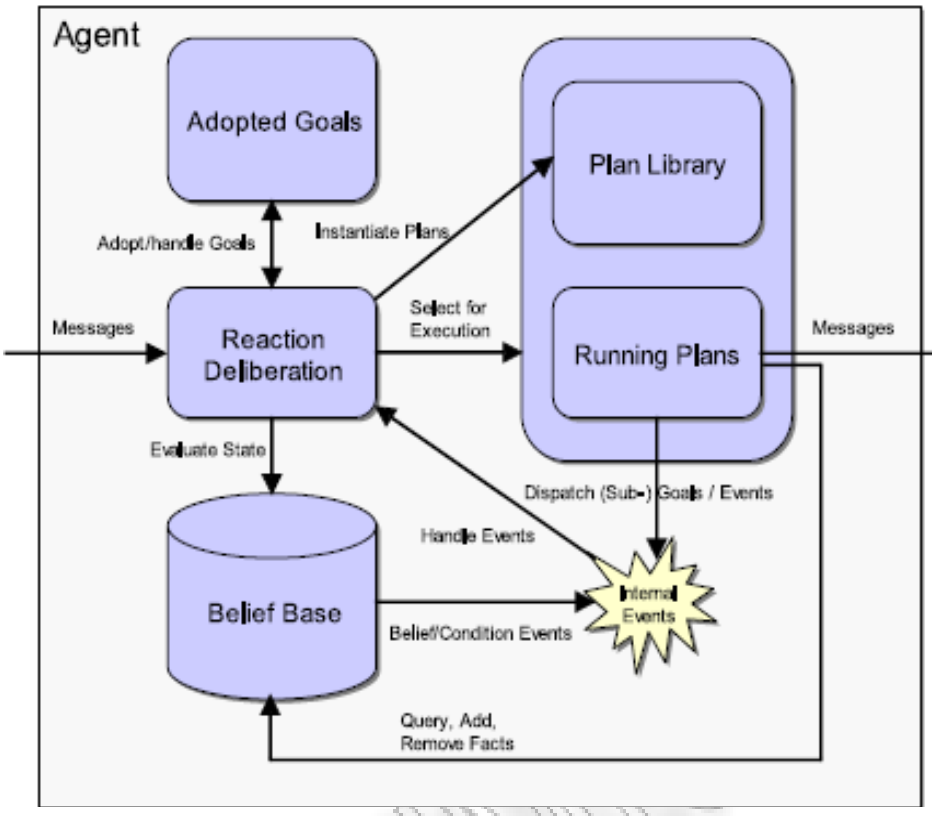
3.3.4 Η αρχιτεκτονική Jadex και υλοποίηση

Η ανάγκη για την δημιουργία μιας πλατφόρμας agents που να μπορεί να υποστηρίξει το middleware και το reasoning οδήγησε στην δημιουργία του Jadex το οποίο στηρίχθηκε σε μία υπάρχουσα και ήδη υλοποιημένη πλατφόρμα middleware η οποία χρησιμοποιείται ευρέως. . Η πλατφόρμα αυτή είναι το Jade. Το Jade είναι μια πλατφόρμα που έχει υλοποιηθεί στα πρότυπα FIPA, προσφέροντας τους απαραίτητους μηχανισμούς επικοινωνίας και υπηρεσιών όπως για παράδειγμα διαχείριση των agents αλλά και ένα πλήθος εργαλείων που βοηθούν την υλοποίηση και την αποσφαλμάτωση εφαρμογών που δημιουργούνται με αυτήν την πλατφόρμα. Στο Jade υπάρχουν κάποια θέματα ανοικτά σκόπιμα όπως είναι για παράδειγμα η κατασκευή των agents και αυτό που προσφέρεται είναι ένα απλό μοντέλο δομής που ο δημιουργός μιας εφαρμογής μπορεί να υλοποιήσει μια συμπεριφορά κάποιου agent. Λόγω λοιπόν της ανοικτής δομής αλλά και της απλότητας αυτής το Jade ήταν μια πολύ καλή πλατφόρμα για την δημιουργία του reasoning πάνω της. Επίσης εφόσον η πλατφόρμα του Jade ήταν δοκιμασμένη από την δημιουργία πολλών εφαρμογών με ιδανικά αποτελέσματα όσον αφορά το middleware και κυριότερα όσον αφορά την επικοινωνία των agents ήταν μια ιδανική πλατφόρμα για να εξελιχθεί στο Jadex με την προσθήκη του reasoning. Θα μπορούσαμε να πούμε λοιπόν ότι το Jadex είναι η αναβάθμιση του Jade και αυτό που προσφέρει τώρα είναι ένα καλύτερο μοντέλο reasoning που μπορεί να υποστηρίξει τις συμπεριφορές που περιγράψαμε παραπάνω και προέρχονται από διαφόρους χώρους της σύγχρονης επιστήμης. Με αυτόν τον τρόπο επιτυγχάνονται δύο πράγματα. Από την μία το middleware που είναι αυτό που χρησιμοποιείται και στο Jade κατά ένα μεγάλο ποσοστό επιτυγχάνει την επικοινωνία και την διαχείριση των agents και από την άλλη το reasoning επιτυγχάνει την εσωτερική δομή των agents. Φυσικά αφού όπως αναφέραμε ότι το Jadex είναι μια αναβάθμιση του Jade αποκτά ταυτόχρονα και όλα τα θετικά που είχε και η πλατφόρμα αυτή. Το πιο σημαντικό βέβαια είναι ότι είναι μια πλατφόρμα ανεξάρτητη από την αρχιτεκτονική των συστημάτων στα οποία τρέχει και αυτός είναι πολύ σημαντικός λόγος για την δική μας εφαρμογή που θα πρέπει να είναι ανεξάρτητη από συστήματα, δίκτυα και γενικότερες αρχιτεκτονικές συστημάτων. Στην συνέχεια αναφέρονται οι τρόποι που έχει ενσωματωθεί η μηχανή του reasoning στην middleware πλατφόρμα.

3.3.5 Επισκόπηση αρχιτεκτονικής

Αν δούμε απέξω την αρχιτεκτονική του Jadex παρατηρούμε ότι ο κάθε agent είναι σαν ένα μαύρο κουτί που λαμβάνει και αποστέλλει μηνύματα. Τα εισερχόμενα μηνύματα καθώς επίσης και τα εσωτερικά γεγονότα του περιβάλλοντος λειτουργίας του κάθε agent αποτελούν τις εισόδους του μηχανισμού αντίδρασης του κάθε agent ή καλύτερα της μονάδας λειτουργίας του. Έτσι λοιπόν τα εισερχόμενα μηνύματα και τα γεγονότα περιβάλλοντος περιέχουν πληροφορίες για το τι πρέπει να κάνει ο κάθε agent. Η ενέργειες αυτές βρίσκονται δομημένες στο σώμα του κάθε agent όπου και ανάλογα με το τι μήνυμα υπάρχει σαν είσοδος εκτελείται η κατάλληλη ενέργεια. Οι ενέργειες αυτές ποικίλουν. Μπορεί να είναι προώθηση μηνυμάτων σε plans που ήδη τρέχουν για να τροποποιηθεί η λειτουργία τους, έναρξη λειτουργίας νέων plans, αποστολή μηνυμάτων ή γεγονότων περιβάλλοντος σε άλλους agents καθώς και συνδυασμοί των παραπάνω. Μπορούμε όμως να αρχίσουμε να βλέπουμε τις μεγάλες δυνατότητες της πλατφόρμας του Jadex καθώς παρατηρούμε ότι υπάρχει συνεχής αλληλεπίδραση του κάθε agent με τους υπόλοιπους agents αλλά και με το περιβάλλον λειτουργίας τους και μπορούν ανά πάσα στιγμή να προσαρμόζουν την λειτουργία τους ανάλογα με τις απαιτήσεις του περιβάλλοντος ή των ίδιων των agents. Θα μπορούσαμε να πούμε λοιπόν ότι επιτυγχάνεται μια προσομοίωση της ανθρώπινης συμπεριφοράς γιατί και ο άνθρωπος ανάλογα με τα ερεθίσματα που παίρνει τόσο από τους ανθρώπους γύρω του όσο και από το περιβάλλον του προσαρμόζει την συμπεριφορά του για να ανταποκριθεί καλύτερα στις νέες απαιτήσεις που δημιουργούνται κάθε στιγμή.

Τα παραπάνω φαίνονται πιο λεπτομερώς στο παρακάτω σχήμα που παρουσιάζει την αρχιτεκτονική του κάθε agent αλλά και τις εισόδους και εξόδους του που αποτελούν τις διεπαφές με τους υπόλοιπους agents και έτσι έχουμε μια μορφή της αρχιτεκτονικής της πλατφόρμας Jadex.



Σχήμα 3.3.5.1 Αρχιτεκτονική ενός agent

Η δομή των ενεργειών (reaction-deliberation mechanism) όλων των agents είναι φτιαγμένη με τον ίδιο τρόπο. Η συμπεριφορά κάθε agent εξαρτάται από τα goals, believes και plans που έχει ή τα οποία ενεργοποιεί και απενεργοποιεί μέσω μηνυμάτων. Τα τρία δομικά αυτά στοιχεία (goals, believes και plans) που περιγράφθηκαν και παραπάνω σε πιο θεωρητικό επίπεδο θα αναλυθούν εδώ σε επίπεδο λειτουργίας των agents και πιο συγκεκριμένα πως επηρεάζουν την συμπεριφορά και τον τρόπο λειτουργίας του κάθε agent.

Believes

Ένας βασικός στόχος της πλατφόρμας του Jadex είναι ευκολία της χρήσης του. Επομένως το Jadex δεν επιβάλλει μια βασισμένη στη λογική αντιπροσώπευση των believes, κάτι που θα ήταν πολύ δύσκολο να υλοποιηθεί και να διαχειριστεί σε πρακτικό επίπεδο. Δηλαδή δεν δημιουργείται από την αρχή κάποιο believe για κάποια συγκεκριμένη λειτουργία που θέλουμε να υποστηρίξουμε. Αντί αυτού, τα συνηθισμένα αντικείμενα της Java οποιουδήποτε είδους μπορούν να περιληφθούν στο Belief Base το οποίο όπως φαίνεται και από το παραπάνω σχεδιάγραμμα είναι κάτι σαν βάση δεδομένων όπου και υπάρχουν όλα τα believes. Με αυτόν τον τρόπο

έχουμε απόλυτη επαναχρησιμοποίηση των believes καθώς και μια κεντρική δομή που λειτουργεί ακριβώς όπως μια βάση δεδομένων με όλα τα προτερήματα που αυτή μπορεί να προσφέρει. Δηλαδή μπορούμε να προσθέτουμε, να αφαιρούμε και να χρησιμοποιούμε οποιοδήποτε believe θέλουμε από το Belief Base ανάλογα με τις απαιτήσεις των εφαρμογών μας και πιο συγκεκριμένα ανάλογα με τις απαιτήσεις κάποιου agent της εφαρμογής μας. Αυτές οι ενέργειες της προσθήκης, της αφαίρεσης και της χρησιμοποιήσης γίνονται με ερωτήματα όπως είναι τα ερωτήματα μιας SQL βάσης δεδομένων. Επίσης τα believes ονομάζονται facts και μπορούν να οργανωθούν σε γενικότερα σύνολα που ονομάζονται fact sets και ομαδοποιούνται εκεί διάφορα facts που έχουν κοινά χαρακτηριστικά. Έτσι διευκολύνονται διάφορα διαχειριστικά θέματα όπως είναι η γρήγορη αναζήτηση, η σωστή και καλύτερη αποθήκευση και η οργάνωση των believes-facts μας.

Τα believes τώρα χρησιμοποιούνται σαν είσοδος στην μηχανή του reasoning και χρησιμοποιούνται για να συγκεκριμενοποιούν τις καταστάσεις των στοιχείων του Jadex που είναι οι agents και το περιβάλλον λειτουργίας τους. Ένα believe-fact για παράδειγμα μπορεί να είναι ένα σύνολο από κάποιες ενέργειες που θα πρέπει να έχουν γίνει πριν προχωρήσει η ενεργοποίηση ή τροποποίηση κάποιου plan, ή ένα σύνολο από διαδικασίες που θα πρέπει να έχουν γίνει για να έχει υλοποιηθεί ένας στόχος-goal. Δηλαδή αυτό που είναι τα believes ουσιαστικά είναι ένας τρόπος αναπαράστασης του περιβάλλοντος και ένα σύνολο από όρους και ενέργειες που θα πρέπει να γίνονται όταν αλλάζει το περιβάλλον αυτό. Ποιο συγκεκριμένα ας υποθέσουμε ότι έχουμε δυο agents, τον agent A και τον agent B. Ας υποθέσουμε ότι ο Agent A ενεργοποιείται και περιμένει σε κατάστασης αναμονής. Επίσης έχει κάποιο believe που λέει ότι όταν ενεργοποιηθεί ο agent B θα πρέπει να του στείλει ένα μήνυμα. Αν ενεργοποιήσουμε λοιπόν τον agent B ο agent A θα ενημερωθεί μέσω του belief base του ότι πρέπει να εκτελέσει κάποιο plan και να στείλει ένα μήνυμα στον agent B όπως αναφέρει το believe του αφού ο όρος της ενεργοποίησης του agent B έχει εκπληρωθεί.

Καταλαβαίνουμε λοιπόν ότι τα believes είναι η βάση της μηχανής reasoning του Jadex καθώς εδώ περιγράφονται οι διαδικασίες που θα πρέπει να πραγματοποιούνται όταν αλλάζουν οι καταστάσεις των agents ή του περιβάλλοντος λειτουργίας τους.

Goals

Το Jadex προσπαθεί να υλοποιήσει την γενική ιδέα ότι οι στόχοι-goals είναι στιγμιαίες και διακριτές επιθυμίες των agents. Κάθε agent πρέπει να προβεί σε κάποιες ενέργειες για να μπορέσει να καταλήξει σε ένα συμπέρασμα σχετικά με κάποιον στόχο. Ένα τέτοιο συμπέρασμα είναι ότι ένας στόχος έχει εκπληρωθεί, δεν μπορεί να εκπληρωθεί ή δεν είναι επιθυμητό πλέον να εκπληρωθεί λόγω κάποιων καταστάσεων που έχουν μεσολαβήσει και ουσιαστικά έχουν επηρεάσει τις επιθυμίες του agent. Αντίθετα με άλλα συστήματα agents εδώ δεν είναι υποχρεωτικό ότι κάποιος agent θα πρέπει να εμμένει στην προσπάθεια εκπλήρωσης κάποιου στόχου-goal. Μπορεί λόγω της λειτουργίας κάποιας εφαρμογής πάνω στο Jadex, και των ευμετάβλητων καταστάσεων που εισέρχονται οι agents να χρειαστεί κάποιοι στόχοι να μην εκπληρωθούν ή και να τροποποιηθούν. Αυτό το στοιχείο είναι πολύ σημαντικό γιατί πλέον μπορούμε να φτιάξουμε εφαρμογές που δεν ξεκινάνε από ένα σημείο Α και τελειώνουν σε ένα σημείο Β με μια σειρά προκαθορισμένων βημάτων αλλά μπορούμε να φτάσουμε στο σημείο Β με τέτοιον τρόπο που να προσαρμόζεται κάθε φορά στις ειδικές καταστάσεις που δημιουργούνται από όλο το περιβάλλον της εφαρμογής μας. Για να μπορέσουμε όμως να διακρίνουμε πότε ένας στόχος είναι ενεργός δηλαδή εκτελούμε ενέργειες για την εκπλήρωση του και πότε ένας στόχος δεν πρέπει να εκπληρωθεί πλέον πρέπει ο κύκλος ζωής του κάθε στόχου να έχει κάποιες παραμέτρους που να τον χαρακτηρίζουν κατάλληλα. Οι παράμετροι αυτές είναι εάν ένας στόχος έχει επιλεχθεί (option), εάν ένας στόχος είναι ενεργός (active) και εάν ένας στόχος έχει αποφασιστεί ότι δεν πρέπει να συνεχιστεί ως προς την εκπλήρωση του (suspended). Έτσι λοιπόν όταν έχει επιλεχθεί ένας στόχος για να εκπληρωθεί έχει την ένδειξη option και έτσι υπάρχει για τον στόχο αυτόν ένα σύνολο ενεργειών που θα πρέπει να γίνουν για την εκπλήρωση του. Το σύνολο των ενεργειών αυτών υπάρχει στο κομμάτι reaction-deliberation mechanism του agent όπως βλέπουμε και από το παραπάνω σχήμα. όταν οι ενέργειες αυτές αρχίζουν να πραγματοποιούνται τότε ο στόχος έχει ένδειξη ότι είναι active. Αν τώρα κατά την διάρκεια που ένας στόχος είναι active ή option μπορεί να περάσει σε κατάσταση suspended και να σταματήσουν οι ενέργειες για την εκπλήρωση του. Με αυτόν τον τρόπο μπορούμε να μεταλλάσουμε την λειτουργία κάποιου agent ανάλογα με τις συνθήκες τόσο του ίδιου του agent όσο και της καταστάσεως που είναι το περιβάλλον λειτουργίας του.

Επίσης υπάρχουν και κάποιες κατηγορίες στόχων ανάλογα με τον τρόπο που εκπληρώνονται αλλά και με βάση τα αποτελέσματα που παράγουν.

- Perform goal: είναι ένας στόχος που θεωρείται ότι έχει εκπληρωθεί όταν ένα σύνολο από ενέργειες έχουν πραγματοποιηθεί ανεξαρτήτως του αποτελέσματος που παράγεται από την εκπλήρωση του.
- Achieve goal: είναι ένας στόχος ο οποίος για να θεωρηθεί ότι έχει επιτευχθεί πρέπει να δώσει ένα συγκεκριμένο αποτέλεσμα ανεξαρτήτως του αριθμού και του περιεχομένου των ενεργειών που χρησιμοποιηθήκαν για το αποτέλεσμα αυτό. Για να επιτευχθεί ένας τέτοιος στόχος μπορεί να εκτελεστούν από ένα έως πάρα πολλά plans.
- Query goal: ο στόχος αυτός είναι παρόμοιος με τον achieve goal αλλά το αποτέλεσμα που πρέπει να υπάρξει εδώ δεν είναι ένα αποτέλεσμα που αφορά το περιβάλλον λειτουργίας του συστήματος αλλά κάποια πληροφορία που χρειάζεται ένας agent. Ένα παράδειγμα τέτοιου στόχου μπορεί να είναι η πληροφορία ότι κάποιος agent ενεργοποιήθηκε ή ότι κάποιο plan τελείωσε την δουλειά του.
- Maintain goal: ο στόχος αυτός έχει σαν δουλεία να διατηρεί μία επιθυμητή κατάσταση. Εκτελούνται δηλαδή κάποια plans για να μπορεί για παράδειγμα ένας agent να διατηρεί μια συγκεκριμένη κατάσταση ανεξαρτήτως των αλλαγών στην κατάσταση λειτουργίας του περιβάλλοντος που μπορεί να υπάρχουν.

Παρατηρούμε από όλα αυτά ότι οι στόχοι είναι μια μονάδα της πλατφόρμας του Jadex που θυμίζει αρκετά το κλασσικό από άλλα συστήματα outcome των υπό-μονάδων λειτουργίας τους αλλά εδώ τα πράγματα είναι αρκετά πιο δυναμικά και προσαρμοζόμενα στις αλλαγές που βοηθούν πάρα πολύ στην επίτευξη του reasoning.

Plans

Όπως έχουμε αναφέρει και παραπάνω για να μπορέσουν να επιτευχθούν κάποιοι στόχοι πρέπει ένα σύνολο από ενέργειες να πραγματοποιηθούν. Οι ενέργειες αυτές είναι τα plans. Κάθε plan αποτελείται από ένα head και από ένα body. Το head του plan περιγράφει τις συνθήκες κάτω από τις οποίες εκτελείται το body κάποιου plan. Το body αντίστοιχα του plan περιλαμβάνει τις εντολές που πρέπει να εκτελεστούν με στόχο την επιτυχή ολοκλήρωση του plan. Το body κάποιου plan μπορεί να

περιλαμβάνει οποιεσδήποτε Java classes και αυτό είναι πολύ σημαντικό γιατί μπορούν εύκολα να ενσωματωθούν αλλά και να δημιουργηθούν από την αρχή εφαρμογές με βάση το Jadex και να προσφέρουν το reasoning σε συνδυασμό με την επικοινωνία των agents σε middleware επίπεδο.

Αυτό σημαίνει ότι αν κάποιος μάθει την δομή των plans με το head και το body δηλαδή, μπορεί να ενσωματώσει οποιεσδήποτε εντολές που είναι γραμμένες σε Java. Το γεγονός αυτό δίνει μια ευκολία στους προγραμματιστές ώστε να μπορέσουν να μεταφέρουν τις εφαρμογές τους στην πλατφόρμα του Jadex. Φυσικά θα πρέπει να υπάρξουν και κάποιες άλλες μετατροπές που αφορούν τα believes και τα goals ώστε να είναι λειτουργική κάποια τέτοια εφαρμογή.

Agent Definition

Αφού περιγράψαμε από τι αποτελείται ένας agent και πως αλληλεπιδρά με το περιβάλλον λειτουργίας του θα πρέπει να δούμε πως φτιάχνεται ένας agent και πιο συγκεκριμένα πως ενσωματώνει τα goals, believes και plans στην δομή του. Για να δημιουργήσουμε και να ενεργοποιήσουμε κάποιον agent θα πρέπει το σύστημα να ξέρει ποιες παραμέτρους πρέπει να αρχικοποιήσει για τον agent αυτόν. Η αρχική κατάσταση του κάθε agent αποτελείται από τα believes του, τα goals του και τα plans που πρέπει να εκτελέσει. Όλα αυτά αποτελούν δομικά στοιχεία του agent. Το Jadex έχει μια προσέγγιση που βασίζεται στην δήλωση των δομικών στοιχείων αυτών και των μεθόδων και διεργασιών που αυτά εμπερικλείουν. Πιο συγκεκριμένα τα plan-bodies πρέπει να είναι φτιαγμένα όπως οι κοινές κλάσεις της Java, αλλά όλα τα υπόλοιπα στοιχεία όπως τα believes,goals και τα plan-heads δηλώνονται μέσω της γλώσσας XML που επιτρέπει να δημιουργούμε και να δηλώνουμε Jadex αντικείμενα. Για λόγους επαναχρησιμοποίησης μπορούν να ενσωματωθούν σε πακέτα οι οντότητες που σχετίζονται σε επίπεδο λειτουργίας (believes, goals και plans). Όλες οι οντότητες του αρχείο XML που ονομάζεται agent definition file γράφονται μέσα σε tags με συγκεκριμένη δομή που περιλαμβάνει όλα τα απαραίτητα στοιχεία για το κάθε agent. Έτσι λοιπόν για να φτιάξουμε έναν νέο agent θα πρέπει να φτιάξουμε το definition file του και να ενσωματώσουμε εκεί όλα τα στοιχεία που θέλουμε να έχει. Τα στοιχεία αυτά θα είναι τα believes, τα goals και τα plan-heads. Τα plan-bodies καλούνται από το agent definition file μέσω των plan-heads αλλά το περιεχόμενο τους φτιάχνεται σε ξεχωριστά αρχεία που έχουν την ονομασία του κάθε plan και δεν

είναι τίποτα άλλο από κλασσικές Java κλάσεις οι οποίες βέβαια έχουν και κάποιες επιπλέον δυνατότητες που προσφέρει η Jadex πλατφόρμα, όπως είναι για παράδειγμα η αποστολή μνημάτων σε άλλους agents.
Το πώς ακριβώς φτιάχνεται ένα definition file για έναν agent καθώς και πως φτιάχνεται το plan-body θα το δείξουμε αναλυτικά στην ανάλυση της εφαρμογής μας στην συνέχεια της εργασίας.

3.3.6 Συμπερασματικά για το Jadex

Το Jadex λοιπόν με βάση όλα τα παραπάνω αποτελεί μια αναβάθμιση της middleware πλατφόρμας του Jade που επιτρέπει την επικοινωνία ανάμεσα σε agents και που τώρα προστίθεται η δυνατότητα για το reasoning. Το reasoning είναι μια προσομοίωση της ανθρώπινης συμπεριφοράς, δηλαδή της δυνατότητας που έχει ο άνθρωπος να προσαρμόζεται με βάση τις συνθήκες που επικρατούν στο περιβάλλον του. Ένας agent μπορεί να έχει κάποια believes τα οποία είναι μια μορφή που αναπαριστά σε πραγματικό χρόνο την κατάσταση γειτονικών agents καθώς και την κατάσταση του περιβάλλοντος λειτουργίας. Επίσης έχει κάποιους στόχους οι οποίοι μπορεί να είναι ενεργοί ή ακόμα και απενεργοποιημένοι ανάλογα με την κατάσταση που βρίσκεται ο agent ή το περιβάλλον λειτουργίας του. Κάθε στόχος για να επιτευχθεί χρειάζεται να πραγματοποιηθούν κάποιες ενέργειες. Οι ενέργειες αυτές είναι τα plans και τα οποία αποτελούνται από ένα head και ένα body. Όλα τα χαρακτηριστικά και οι παράμετροι του agent ενσωματώνονται σε ένα definition file που είναι ένα αρχείο xml και το οποίο συνοδεύεται από τα αρχεία των bodies των plans που πρέπει να εκτελέσει ο agent για να εκπληρώσει τους στόχους του.
Όλη η πλατφόρμα καθότι είναι γραμμένη σε Java είναι ανεξάρτητη αρχιτεκτονικής συστήματος, λειτουργικού συστήματος και προσφέρει όλα τα θετικά που έχει η αντικειμενοστραφής γλώσσα προγραμματισμού Java όπως είναι για παράδειγμα η επαναχρησιμοποίηση.
Η δημιουργία κάποιας εφαρμογής στην πλατφόρμα του Jadex δίνει την δυνατότητα να έχουμε πολύ καλή επικοινωνία εκμεταλλευόμενοι το γεγονός της επικοινωνίας σε επίπεδο middleware αλλά και την δυνατότητα να προσεγγίσουμε ανθρώπινες συμπεριφορές όπως είναι η σκέψη και το συναίσθημα. Αυτά έχουν σαν αποτέλεσμα

να μπορούμε να φτιάξουμε πολύ ωραίες εφαρμογές που μας προσφέρουν πολύ αναβαθμισμένες δυνατότητες.
Περισσότερη ανάλυση στο Jadex θα γίνει με πραγματικά δεδομένα κατά την ανάλυση της εφαρμογής μας στην συνέχεια της εργασίας αυτής.

4. Μεθοδολογία Υλοποίησης

Στο κομμάτι αυτό της εργασίας παρουσιάζεται η εφαρμογή που έχει φτιαχτεί. Θα παρουσιαστούν εδώ λεπτομερώς οι απαιτήσεις υλοποίησης, η αρχιτεκτονική, τα επιμέρους τμήματα που αποτελούν τη εφαρμογή καθώς και εξηγήσεις για τον τρόπο που έχει δημιουργηθεί και τρέχει. Τέλος θα δοθεί ένα πρακτικό παράδειγμα χρήσης της εφαρμογής που θα συνοδεύεται από τις απαραίτητες φωτογραφίες που θα αναπαριστούν την εξέλιξη της εφαρμογής την ώρα που τρέχει καθώς και συμβουλές για μελλοντικές χρήσεις και παραμετροποιήσεις.

4.1 Απαιτήσεις Εφαρμογής

Στο σημείο αυτό αναφέρονται οι απαιτήσεις που θα πρέπει να εκπληρώνει η εφαρμογή όπως αυτές διαμορφώθηκαν από το γενικότερο project στο οποίο μετέχει αυτή η εφαρμογή. Το project αυτό είναι όπως έχουμε αναφέρει μια ιατρική εφαρμογή που επιτρέπει το face και το speech recognition και που αυτές οι δύο λειτουργίες συνυπάρχουν σε μια κοινή πλατφόρμα middleware που υποστηρίζει την τεχνολογία των agents καθώς και εξελιγμένες δυνατότητες διαχείρισης δεδομένων όπως είναι το reasoning που προσομοιάζει τις ανθρώπινες συμπεριφορές. Οι απαιτήσεις λοιπόν της εφαρμογής μας είναι :

- Η πρώτη σημαντική απαίτηση που είναι κοινή για όλες τις επιμέρους εφαρμογές είναι η ύπαρξη κάποιας πλατφόρμας middleware που να υποστηρίζει την τεχνολογία των agents και το reasoning που θα επιτρέψει την κατάλληλη διαχείριση των δεδομένων. Το reasoning όπως αναφέραμε και παραπάνω μοιάζει πολύ με την ανθρώπινη συμπεριφορά και περιέχει στοιχεία όπως γνώση του περιβάλλοντος λειτουργίας των agents, συναισθήματα που οδηγούν τους agents να πάρουν αποφάσεις ανάλογα με τις εξελίξεις στο

περιβάλλον τους και συνεχείς ενημερώσεις για την κατάσταση τους. Η πλατφόρμα λοιπόν που έχει επιλεχθεί είναι το Jadex και υποστηρίζει πολύ καλά όλα τα παραπάνω αυτά στοιχεία της πρώτης απαίτησης. Με άλλα λόγια λοιπόν θα πρέπει όλες οι επιμέρους εφαρμογές του project να είναι χτισμένες πάνω στο Jadex.

- Η δεύτερη απαίτηση που αφορά μόνο την συγκεκριμένη εφαρμογή είναι να μπορεί να γίνει speech recognition (αναγνώριση ομιλίας) και πιο συγκεκριμένα αναγνώριση φωνητικών εντολών. Θα πρέπει λοιπόν ο χειριστής της εφαρμογής να μπορεί να την χειρίζεται μόνο με την προφορά των εντολών που θέλει και χωρίς καμία χρησιμοποίηση οποιασδήποτε άλλης συσκευής εισόδου όπως είναι για παράδειγμα πληκτρολόγια και ποντίκια.
- Η τρίτη απαίτηση είναι να υπάρχει γραφικό περιβάλλον πλοήγησης στην εφαρμογή. Το γραφικό αυτό περιβάλλον θα πρέπει να επιτρέπει την πλοήγηση σε διάφορα δεδομένα που αφορούν ασθενείς. Θα πρέπει επίσης το όλο γραφικό περιβάλλον να είναι δυναμικό, αυτό σημαίνει ότι θα πρέπει να προσαρμόζεται ανάλογα με τα δεδομένα που δέχεται από ένα άλλο σύστημα που είναι ο reasoning server και ο οποίος έχει συλλέξει τα στοιχεία για κάποιον ασθενή και τα στέλνει στην εφαρμογή μας ώστε να ενημερωθεί το γραφικό περιβάλλον. Τα στοιχεία αυτά από ασθενή σε ασθενή διαφέρουν οπότε το γραφικό περιβάλλον θα πρέπει να έχει φτιαχτεί με τέτοιο τρόπο ώστε να μπορεί κάθε φορά να προσαρμόζεται σε αυτά.
- Τέλος όλες οι παραπάνω απαιτήσεις θα πρέπει να υλοποιηθούν με τέτοιον τρόπο ώστε η τελική εφαρμογή να είναι διαφανής και ο χρήστης να μην καταλαβαίνει τα επιμέρους συστατικά αυτής. Αυτό σημαίνει ότι τόσο η πλατφόρμα jadex όσο και η μηχανή αναγνώρισης φωνητικών εντολών δεν θα πρέπει να εμφανίζονται κατά την λειτουργία της εφαρμογής και το μόνο που πρέπει να γίνεται είναι να αλληλεπιδρούν με το γραφικό περιβάλλον και το οποίο θα πρέπει να προσαρμόζεται κατάλληλα με σκοπό να εμφανίζει τα κατάλληλα γραφικά αποτελέσματα.

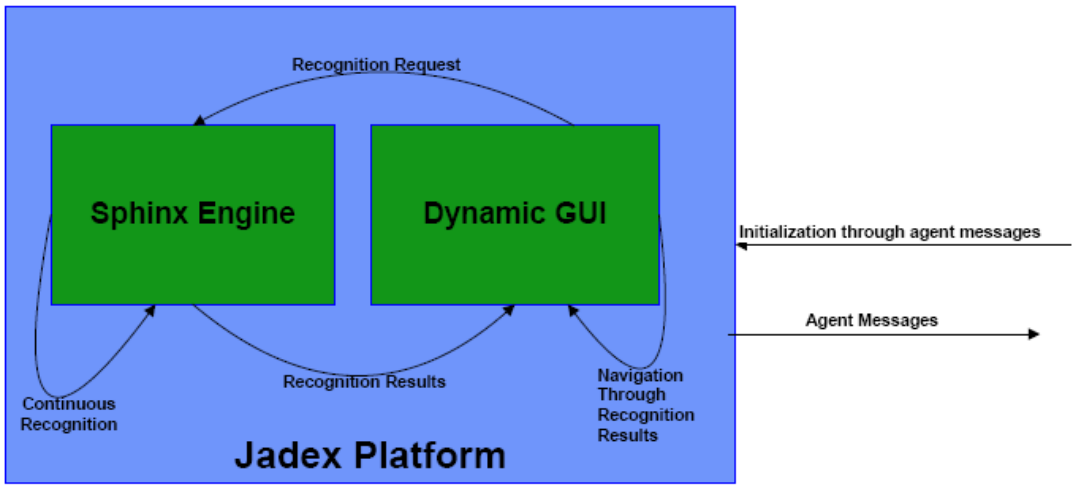
Έκτος όμως από τις απαιτήσεις σε επίπεδο υλοποίησης και δομής υπάρχουν και οι απαιτήσεις σε επίπεδο λειτουργίας. Δηλαδή τι θα πρέπει να κάνει η εφαρμογή την

ώρα που λειτουργεί. Αυτές οι απαιτήσεις επέδρασαν στον τρόπο υλοποίησης της εφαρμογής και είναι σκόπιμο για τον λόγο αυτό να αναφερθούν στο σημείο αυτό.

- Το γραφικό περιβάλλον θα πρέπει να σηκώνεται χωρίς να περιέχει κανένα στοιχείο που να αφορά τους ασθενείς. Θα πρέπει να λοιπόν να είναι ένα γραφικό κοινό για όλους τους ασθενείς χωρίς να περιέχει όμως τα ειδικά στοιχεία τους στην πρώτη αυτή φάση.
- Το γραφικό περιβάλλον θα πρέπει μόλις δεχτεί το μήνυμα agent με τα δεδομένα του ασθενή να προσαρμοστεί κατάλληλα με αυτά.
- Η πλοήγηση θα πρέπει να γίνεται μέσω φωνής και για αυτό θα πρέπει να βρεθούν κατάλληλες προγραμματιστικές μέθοδοι ώστε να μπορέσουν να καλυφθούν τυχόν λανθασμένες αναγνωρίσεις φωνητικών εντολών.
- Θα πρέπει η αναγνώριση φωνής να λειτουργεί συνέχεια κατά την διάρκεια της εφαρμογής και παράλληλα με τις αλλαγές του γραφικού περιβάλλοντος, ώστε να επιτευχθεί η πλοήγηση και αποστολή μηνυμάτων που πρέπει να υποστηρίζει η εφαρμογή.
- Η δυνατότητα αποστολής agent μηνυμάτων πρέπει να περιορίζεται με κάποιες συνθήκες. Για παράδειγμα η αποστολή μηνύματος θα πρέπει να είναι εφικτή μόνο όταν για παράδειγμα ο χρήστης-γιατρός έχει ενεργοποιήσει την καρτέλα Διάγνωση ή Θεραπεία του ασθενή και μόνο εφόσον έχουν ενεργοποιηθεί κάποιες εξετάσεις ή διαγνώσεις. Πρέπει να προστατευτεί λοιπόν η εφαρμογή από άσκοπες αποστολές μηνυμάτων agent με μηδενικά δεδομένα ή λανθασμένα δεδομένα που μπορεί να προκύψουν από τυχόν λανθασμένες αναγνωρίσεις φωνητικών εντολών.
- Θα πρέπει επίσης σε κάθε σημείο του γραφικού περιβάλλοντος να υπάρχει η δυνατότητα αναίρεσης κάποιας εντολής ή μεταφοράς σε άλλο σημείο του ώστε να μπορέσει να διευκολυνθεί η πλοήγηση στην εφαρμογή αλλά και να μπορέσουν να διορθωθούν ενέργειες που έχουν προκαλέσει λανθασμένες αναγνωρίσεις.

4.2 Αρχιτεκτονική

Η εφαρμογή αποτελείται από τρία δομικά στοιχεία που το καθένα έχει τον πολύτιμο ρόλο του στην σύνθεση της όλης εφαρμογής. Στην συνέχεια περιγράφουμε τα δομικά αυτά στοιχεία και τις λειτουργίες που είναι υπεύθυνα να κάνουν εν συντομία.
Το δομικά στοιχεία φαίνονται στο παρακάτω γράφημα όπου και παρουσιάζεται η αρχιτεκτονική της εφαρμογής :



Σχήμα 4.2.1 Αρχιτεκτονική της εφαρμογής

- **Jadex Platform:** Η πλατφόρμα Jadex όπως έχουμε αναφέρει και στην περιγραφή της είναι μια πλατφόρμα middleware η οποία όμως έχει κάποια πιο εξελιγμένα στοιχεία που μπορεί να προσφέρει στις εφαρμογές που χτίζονται πάνω σε αυτήν. Μπορεί να υποστηρίξει το reasoning, δηλαδή να προσομοιάσει ανθρώπινες συμπεριφορές όπως είναι η σκέψη, η αίσθηση, το συναίσθημα και η απόφαση βάση εξελίξεων στο περιβάλλον λειτουργίας. Η επιλογή της πλατφόρμας έγινε για τις εξελιγμένες αυτές δυνατότητες που

μπορεί να μας προσφέρει. Σαν πλατφόρμα middleware μας προσφέρει δυνατότητες επικοινωνίας μεταξύ των agents χωρίς να μας ενδιαφέρει το σε τι σύστημα βρίσκονται οι agents αυτοί. Έτσι λοιπόν μπορούμε να φτιάξουμε τις κατάλληλες διεπαφές της εφαρμογής για σύνδεση με άλλα συστήματα. Οι διεπαφές αυτές δεν είναι τίποτα άλλο από την δημιουργία agents που να μπορούν να στέλνουν και να δέχονται μηνύματα με συγκεκριμένες προδιαγραφές. Στην εφαρμογή μας για παράδειγμα για να αρχικοποιηθεί το γραφικό περιβάλλον θα πρέπει να γίνει λήψη ενός μηνύματος jadex που να περιέχει όλες τις πληροφορίες που θέλουμε. Το μήνυμα αυτό μπορεί να αποσταλεί από οποιονδήποτε agent που έχει δημιουργηθεί όμως με βάση τις συγκεκριμένες απαιτήσεις του μηνύματος. Δεν μας ενδιαφέρει αν ο agent αυτός βρίσκεται σε ένα σύστημα ίδιο με το δικό μας ή αν βρίσκεται σε ένα σύστημα με τελείως διαφορετική αρχιτεκτονική. Ο agent πάντως που θα στέλνει το πρώτο μήνυμα αρχικοποίησης του γραφικού περιβάλλοντος θα βρίσκεται σε κάποιον reasoning server και ο οποίος θα έχει σαν αντικείμενο να συντάξει όσο το δυνατόν καλύτερα το μήνυμα χρησιμοποιώντας όλες τις reasoning δυνατότητες του jadex. Ο reasoning server δεν αποτελεί μέρος της εφαρμογής αυτής και για αυτό έχουν φτιαχτεί οι διεπαφές λήψης και αποστολής μηνυμάτων προς αυτόν καθώς και δύο agents που στέλνουν και δέχονται μηνύματα για να μπορέσουμε να προσομοιάσουμε το περιβάλλον λειτουργίας του γενικότερου project. Τα μηνύματα λόγω απουσίας του reasoning server έχουν στατικό περιεχόμενο και δεν προηγείται κάποια μορφής επεξεργασίας δεδομένων για την σύνταξή τους όπως θα γίνεται από τον reasoning server. Σύμφωνα λοιπόν με τα παραπάνω, η επιλογή του jadex έγινε για δυο λόγους. Είναι μια πολύ καλή middleware πλατφόρμα επικοινωνίας σε επίπεδο agents και προσφέρει εξελιγμένες δυνατότητες διαχείρισης δεδομένων που θα χρειαστεί ο reasoning server με τον οποίο και θα πρέπει η παρούσα εφαρμογή να μπορεί να συνδεθεί μέσω των κατάλληλων διεπαφών που θα πρέπει να έχουν δημιουργηθεί. Οι διεπαφές που έχουν δημιουργηθεί, το περιεχόμενο των μηνυμάτων καθώς και ο τρόπος που έχει δομηθεί η εφαρμογή για να μπορέσει να εκμεταλλευτεί τις δυνατότητες του Jadex θα αναλυθούν κατά την παρουσίαση της υλοποίησης που ακολουθεί στην συνέχεια.

- **Sphinx Engine:** Το sphinx engine είναι η μηχανή αναγνώρισης φωνητικών εντολών και αποτελεί την βάση όλης της εφαρμογής. Η μηχανή αναγνώρισης φωνητικών εντολών έχει σαν στόχο την αποτελεσματικότερη αναγνώριση των εντολών που προφέρουν οι χειριστές της εφαρμογής για να μπορέσουν να πλοηγηθούν στο γραφικό περιβάλλον της εφαρμογής, να ζητήσουν ή ακόμα και να στείλουν δεδομένα από και προς τον reasoning server. Η επιλογή του sphinx και πιο συγκεκριμένα του sphinx-4 έγινε γιατί αποτελεί ένα open-source project και μπορούσε να ενσωματωθεί σε μια εφαρμογή που δεν εξαρτάται από αρχιτεκτονικές συστημάτων και τοπολογίες δικτύων που συνδέονται με το σύστημα που τρέχει η εφαρμογή. Επίσης το sphinx-4 με τις κατάλληλες ρυθμίσεις που έχουν γίνει αποτελεί την ιδανική επιλογή για αναγνώριση φωνητικών εντολών σε σχέση με το μεγάλο ποσοστό επιτυχών αναγνωρίσεων που πραγματοποιεί και φυσικά σε σχέση με ανταγωνιστικές δωρεάν μηχανές αναγνώρισης φωνής που υπάρχουν. Πιο συγκεκριμένα τώρα η μηχανή αναγνώρισης αποτελείται από ένα λεξικό φτιαγμένο ειδικά για την εφαρμογή καθώς και μια συντακτική δομή που είναι σε θέση να ξεχωρίσει αρκετές φωνητικές εντολές. Επίσης έχουν γίνει αρκετές αλλαγές στην υλοποίηση της μηχανής με εξέλιξη της δομής της καθώς και ειδικές προσθήκες που βελτιστοποιούν την διαχείριση της σε διάφορα θέματα, όπως είναι για παράδειγμα η έναρξη και το σταμάτημα της λειτουργίας της, το καθάρισμα του μικροφώνου για να έχουμε καλύτερα αποτελέσματα αναγνώρισης και πολλά άλλα που θα αναφερθούν αναλυτικά στην υλοποίηση της εφαρμογής που ακολουθεί. Το sphinx engine έχει έναν πολύ σημαντικό ρόλο στην εφαρμογή με το να προσφέρει την δυνατότητα στους χειριστές της εφαρμογής να μπορούν να διαχειρίζονται όλη την εφαρμογή χωρίς την χρήση πληκτρολογίου ή ποντικιού ή οποιασδήποτε άλλης συσκευής εισόδου κάποιου συστήματος, παρά μόνο με την ίδια τους την φωνή. Υπάρχει αλληλεπίδραση και με τα άλλα δυο δομικά στοιχεία της εφαρμογής. Με το γραφικό περιβάλλον από την μια γιατί ανάλογα με τις αναγνωρίσεις φωνητικών εντολών αυτό προσαρμόζεται κατάλληλα αλλά και με την jadex πλατφόρμα γιατί με ειδικές εντολές μπορεί να γίνει σύνταξη και αποστολή μηνυμάτων που περιέχουν δεδομένα της εφαρμογής. Για την κατασκευή της τελικής μορφής της μηχανής αναγνώρισης της εφαρμογής ακολουθήθηκαν διάφοροι

τρόποι και μηχανισμοί και τέλος επιλέχθηκε ο τρόπος αυτός που προσέφερε το καλύτερο αποτέλεσμα σύμφωνα με τις απαιτήσεις της εφαρμογής. Οι τρόποι δημιουργίας μαζί με τις δυσκολίες που προέκυψαν θα αναφερθούν στο κομμάτι της υλοποίησης που αφορά το sphinx engine στην συνέχεια της εργασίας.

- **Dynamic GUI:** Είναι ένα στοιχείο που δημιουργήθηκε από το μηδέν. Δημιουργήθηκε σε γλώσσα προγραμματισμού Java όπως είναι φτιαγμένα και τα άλλα δύο δομικά στοιχεία και έτσι και αυτό το κομμάτι μπορεί να τρέξει σε οποιοδήποτε σύστημα ανεξαρτήτως της αρχιτεκτονικής του. Ο ρόλος του είναι πολύ βασικός από πολλές απόψεις. Καταρχήν αποτελεί ένα οπτικό αποτέλεσμα της δουλειάς που κάνουν τα άλλα δυο δομικά στοιχεία. Τόσο το εάν τα jadex μηνύματα είναι σωστά όσο και το εάν οι αναγνωρίσεις των φωνητικών εντολών είναι σωστές φαίνονται με οπτικό τρόπο από το γραφικό περιβάλλον. Επίσης έχει δοθεί ιδιαίτερη σημασία στο να μπορούν να περιορίζονται τα λειτουργικά λάθη της εφαρμογής από λάθος αναγνωρίσεις. Αυτό που γίνεται δηλαδή είναι να μπλοκάρονται όσο το δυνατόν οι λάθος αναγνωρίσεις –που έχουν πάντα κάποια πιθανότητα να συμβούν- και να μην επηρεάζουν την εξέλιξη της εφαρμογής. Το γραφικό περιβάλλον επίσης έχει υλοποιηθεί με τέτοιον τρόπο ώστε να μπορεί να ανταπεξέλθει όσο καλύτερα γίνεται στις απαιτήσεις της εφαρμογής. Υπάρχει έντονη αλληλεπίδραση με τα άλλα δύο δομικά στοιχεία της εφαρμογής. Από την μία με βάση τις αναγνωρίσεις φωνητικών εντολών το γραφικό περιβάλλον προσαρμόζεται ανάλογα και από την άλλη με βάση τα μηνύματα Jadex που δέχεται από άλλα συστήματα (όπως είναι ο reasoning server) προσαρμόζει ανάλογα το περιεχόμενο του. Η ανάλυση σε βάθος του γραφικού περιβάλλοντος θα γίνει σε βάθος στο κομμάτι υλοποίηση της εφαρμογής που ακολουθεί.

Αφού περιγράψαμε με λίγα λόγια τα στοιχεία από τα οποία αποτελείται η εφαρμογή ας δούμε με κάποιες παραπάνω λεπτομέρειες, πως αυτά τα στοιχεία δομούνται ώστε να δώσουνε σαν αποτέλεσμα την εφαρμογή μας.

4.3 Υλοποίηση

Στο κομμάτι αυτό της εργασίας θα περιγράψουμε την υλοποίηση της εργασίας. Η περιγραφή αυτή ουσιαστικά αποτελείται από την αναφορά των δομικών στοιχείων που απαρτίζουν την εφαρμογή αλλά και την ανάλυση σε επίπεδο κώδικα κάποιων κομματιών που είναι χαρακτηριστικά στην εφαρμογή αυτή.

Η εφαρμογή λοιπόν όπως έχουμε πει αποτελείται από τρία δομικά στοιχεία : την μηχανή αναγνώρισης φωνητικών εντολών sphinx, την middleware πλατφόρμα agents Jadex και από το δυναμικό γραφικό περιβάλλον. Έτσι λοιπόν εδώ παρουσιάζουμε τα κομμάτια από το οποία αποτελείται το καθένα από τα παραπάνω δομικά στοιχεία καθώς και τα κομμάτια που αφορούν στην ένωση των δομικών στοιχείων αυτών αλλά και κάποια κομμάτια που έχουν δημιουργηθεί για να υποστηριχθεί ή καλύτερα να προσομοιωθεί το γενικότερο περιβάλλον λειτουργίας (ιατρικό project).

Ακολουθεί σύντομή περιγραφή όλων των κομματιών υλοποίησης της εφαρμογής :

- **Recognition κλάση:** Η κλάση αυτή ουσιαστικά μεταχειρίζεται την μηχανή αναγνώρισης φωνητικών εντολών. Μέσω αυτής της κλάσης γίνονται ενέργειες όπως είναι η δέσμευση των κατάλληλων πόρων του συστήματος που απαιτούνται για την λειτουργία της μηχανής αναγνώρισης, την έναρξη και το σταμάτημα της καθώς και την ενεργοποίηση και απενεργοποίηση της διαδικασίας αναγνώρισης. Όλα αυτά γίνονται μέσω κατά΄λληλων μεθόδων που έχουν δημιουργηθεί στην κλάση αυτή και θα αναλυθούν στην συνέχεια. Η κλάση αυτή εμπεριέχει στοιχεία και εκτελεί ενέργειες για το δομικό στοιχείο που είναι η μηχανή αναγνώρισης φωνητικών εντολών.
- **Medicalagent :** Ο agent αυτός είναι ο κυρίως agent της εφαρμογής και είναι ουσιαστικά αυτός που σηκώνει και μεταχειρίζεται όλα τα δεδομένα της εφαρμογής. Στον agent αυτόν υπάρχει ένα plan που ενεργοποιείται κατά την έναρξη του agent . Επίσης ο agent αυτός είναι φτιαγμένος με τέτοιο τρόπο ώστε να περιμένει κάποιο συγκεκριμένο μήνυμα το οποίο περιέχει το ontology με τα στοιχεία του ασθενή. Μόλις λάβει το συγκεκριμένο μήνυμα το plan του αναλαμβάνει να προσαρμόσει το γραφικό περιβάλλον λειτουργίας στα δεδομένα του εισερχόμενου μηνύματος. Ο agent αυτός είναι κομμάτι του Jadex γιατί προσφέρει την δυνατότητα να δεχτεί ένα εισερχόμενο μήνυμα και να ενεργοποιήσει κάποιο πλάνο που και οι δυο

λειτουργίες αυτές είναι λειτουργίες που αφορούν αποκλειστικά το κομμάτι του Jadex.

- **MedicalstartPlan:** Αυτό είναι το plan που είπαμε ότι ενεργοποιεί ο Medicalagent. Το plan αυτό είναι μια Java κλάση ουσιαστικά που έχει ένα πολύ σημαντικό ρόλο στην όλη εφαρμογή. Εδώ γίνεται η δημιουργία του γραφικού περιβάλλοντος και η ενημέρωση με τα στοιχεία του ασθενή όταν αυτά έρθουν μέσω κάποιου μηνύματος agent. Επίσης αφού γίνει η προσαρμογή του γραφικού περιβάλλοντος σε αυτό το plan γίνεται η έναρξη της μηχανής αναγνώρισης καθώς και η διαδικασία της συνεχής αναμονής για αναγνωρισμένα αποτελέσματα ώστε να μπορέσει να επιτευχθεί η πλοήγηση μέσω της φωνής στο γραφικό περιβάλλον. Εδώ γίνεται και η αποστολή ενός agent μηνύματος με περιεχόμενο ένα ontology που έχει φτιαχτεί από την κλάση Medicalgui με κατάλληλες ενέργειες. Το πλάνο αυτό είναι πολύ περιλαμβάνει στοιχεία από όλα τα δομικά στοιχεία και είναι το κομμάτι εκείνο της υλοποίησης που έχει φτιαχτεί κατά κύριο λόγο για να ενώσει όλες τις τεχνολογίες μεταξύ τους.
- **Medicalgui :** Η κλάση αυτή είναι ουσιαστικά η κλάση που περιλαμβάνει όλα τα στοιχεία του δυναμικού γραφικού περιβάλλοντος. Εδώ λοιπόν υπάρχει η διαδικασία του χτισίματος του γραφικού περιβάλλοντος, δηλαδή την κατασκευή όλων των στοιχείων που το αποτελούν. Εδώ γίνεται επίσης και η δήλωση όλων των στοιχείων εκείνων που αφορούν την πλοήγηση του γραφικού περιβάλλοντος. Πιο συγκεκριμένα εδώ υπάρχουν οι απαραίτητες ενέργειες σε επίπεδο γραφικού που θα πρέπει να γίνουν όταν αναγνωριστεί μια λέξη. Τέλος υπάρχει εδώ και η σύνθεση ενός ontology με πληροφορίες για τον ασθενή που θέλουμε να στείλουμε μέσω ενός μηνύματος agent. Η κλάση αυτή είναι κομμάτι του δυναμικού γραφικού περιβάλλοντος.
- **Zoomframe:** Είναι μια πολύ απλή κλάση η οποία χρησιμοποιείται για την δημιουργία ενός frame για να βάλουμε μέσα μια πιο καλής ανάλυσης φωτογραφία κάποιας εξέτασης. Είναι και αυτό κομμάτι του δυναμικού γραφικού περιβάλλοντος.
- **Starter agent :** Ο agent αυτός χρησιμοποιείται για να προσομοιάσουμε το περιβάλλον λειτουργίας του γενικότερου ιατρικού project. Ο agent αυτός έχει σαν στόχο την αποστολή ενός μηνύματος agent στον Medicalagent. Το

μήνυμα αυτό έχει σαν περιεχόμενο ένα ontology το οποίο περιλαμβάνει τα στοιχεία του ασθενή. Το περιεχόμενο φτιάχνεται από το InitialPlan το οποίο καλεί ο Starter agent. Ο agent αυτός ανήκει στο κομμάτι του Jadex.

- **InitialPlan :** Το plan αυτό είναι μία Java κλάση που καλείται όπως είπαμε από τον Starter agent. Σκοπός αυτής της κλάσης είναι να γεμίσει όλα τα απαραίτητα στοιχεία του ασθενή σε μια οντότητα και να μπορέσει ο Starter στείλει ένα μήνυμα agent με τα απαραίτητα για την εφαρμογή στοιχεία αυτά. Εδώ λοιπόν γίνεται η σύνταξη και η αποστολή του μηνύματος. Η κλάση ανήκει στο κομμάτι του Jadex καθώς έχει τον ρόλο της δημιουργίας και της αποστολής του μηνύματος agent με τα στοιχεία του ασθενή.
- **Receiver agent:** Και αυτός ο agent έχει δημιουργηθεί για να καλύψει τις ανάγκες της προσομοίωσης του γενικότερου ιατρικού project. Ο agent αυτός ενεργοποιεί το ReceiverPlan που έχει σαν ρόλο την λήψη μηνύματος με πληροφορίες που αφορούν τον ασθενή. Έτσι λοιπόν ο agent Receiver είναι και αυτός κομμάτι του Jadex.
- **ReceiverPlan :** Το plan αυτό είναι μια Java κλάση που έχει σαν ρόλο το να περιμένει εισερχόμενα μηνύματα. Τα μηνύματα αυτά προέρχονται όπως έχουμε αναφέρει παραπάνω από το MedicalstartPlan και έχουν σαν περιεχόμενο ένα ontology που περιέχει πληροφορίες για τον ασθενή. Ρόλος του πλάνου αυτού λοιπόν είναι να λάβει το εισερχόμενο μήνυμα και να αναλύσει τις εμπεριεχόμενες πληροφορίες του ασθενή αλλά και να τις εκτυπώσει ώστε να φανεί με καλύτερο τρόπο η όλη επικοινωνία. Το plan αυτό είναι κομμάτι της πλατφόρμας του Jadex καθώς είναι υπεύθυνο για την λήψη μηνυμάτων agent.

Ας δούμε τώρα λοιπόν σε μεγαλύτερη ανάλυση τα παραπάνω κομμάτια της εφαρμογής συνοδευμένα και με κάποια πολύ χαρακτηριστικά κομμάτια κώδικα :

Recognition.class

Όπως αναφέραμε και παραπάνω η κλάση αυτή περιέχει όλα τα στοιχεία εκείνα που απαιτούνται για την διαχείριση της μηχανής αναγνώρισης φωνητικών εντολών του sphinx.

Αποτελείται από μια σειρά μεθόδων που μπορούν να καλεστούν μέσα στην εφαρμογή και να πραγματοποιήσουν ενέργειες όπως είναι η αρχικοποίηση με δέσμευση πόρων , η έναρξη της αναγνώρισης και ο τερματισμός της.

Στην συνέχεια παρουσιάζονται οι μέθοδοι που απαρτίζουν την κλάση Recognition μαζί με τις πληροφορίες του ρόλου της καθεμιάς:

- **callsphinx() :** Η μέθοδος αυτή είναι υπεύθυνη για την δέσμευση των πόρων του συστήματος που χρειάζονται από το σύστημα. Εδώ γίνεται η δημιουργία των αντικειμένων του sphinx που θα πρέπει να ενεργοποιηθούν για να μπορέσει να δουλέψει η μηχανή αναγνώρισης.

  Καταρχήν λοιπόν δημιουργείται το στοιχείο του sphinx ConfigurationManager που όπως περιγράψαμε και στο θεωρητικό υπόβαθρο παραπάνω είναι το στοιχείο εκείνο που εμπεριέχει όλη την δομή του sphinx που χρησιμοποιείται στην εφαρμογή μας. Αυτό γίνεται με τις παρακάτω εντολές :

  url = Recognition.class.getResource("recognition.config.xml");
  ConfigurationManager cm = new ConfigurationManager(url);

  Παρατηρούμε ότι κατά την δημιουργία του ConfigurationManager έχουμε σαν όρισμα ένα αρχείο το recognition.config.xml. Το αρχείο αυτό είναι στον ίδιο φάκελο με τα υπόλοιπα αρχεία της εργασίας και περιέχει όλες τις πληροφορίες για την δομή του sphinx. Για παράδειγμα εδώ περιέχονται η θέση του λεξικού που χρησιμοποιείται καθώς και η θέση της γραμματικής που όπως έχουμε αναφέρει είναι 2 πολύ σημαντικά αρχεία για την μηχανή αναγνώρισης.

  Το λεξικό που χρησιμοποιούμε στην εφαρμογή αυτή περιέχει κάποιες λέξεις που ουσιαστικά είναι οι φωνητικές εντολές που θέλουμε να μπορεί η εφαρμογή μας να αναγνωρίζει. Το λεξικό φαίνεται παρακάτω:

  BLOODEXAMS B L AA D EH G Z AA M SH
  CARDIOLOGY K AA R D IY OW L OW JH IY
  DIAGNOSIS D AY AH G N OW S AH S
  EXAMS EH G Z AA M SH
  FIRSTCHOICE F EH R S T CH OY S

FIRSTCHOICE(2)	**F ER S T CH OY S**
PATIENT	**P EY SH IH EH N T**
PROFILE	**P R OW F AA IH L**
STOP	**S T AA P**
STOP(2)	**S T OW P**
SECONDCHOICE	**S EH K OW N D CH OY S**
COMMUNICATE	**K OW M Y UW N IH K EY T**
THIRDCHOICE	**TH EH IH R D CH OY S**
TREATMENT	**T R IH T M EH N T**
XRAY	**EH K S R EY IY**
ZOOMING	**Z UW M IH NG**

Όπως βλέπουμε υπάρχουν λέξεις για την πλοήγηση αλλά και την αποστολή μηνυμάτων και σε δύσκολες ως προς την αναγνώριση λέξεις έχουν προστεθεί δύο προφορές για καλύτερα αποτελέσματα αναγνώρισης. Φυσικά πάντα μπορούμε να προσθέσουμε νέες λέξεις, με προσοχή βέβαια στην προφορά.
Στην συνέχεια παρουσιάζουμε και την σύνταξη που χρησιμοποιείται :

public <command> = (DIAGNOSIS | FIRSTCHOICE | SECONDCHOICE | THIRDCHOICE | TREATMENT | EXAMS | PROFILE | BLOODEXAMS | CARDIOLOGY | XRAY | ZOOMING | STOP | COMMUNICATE);

Η γραμματική δεν είναι τίποτα άλλο από τις λέξεις του λεξικού με καθέτους ενδιάμεσα τους ώστε να πούμε στην μηχανή αναγνώρισης ότι πρέπει να επιλεγεί μια από όλες αυτές τις λέξεις κάθε φορά. Σε προσθήκη κάποιας νέας λέξης θα πρέπει να συμπληρώσουμε και την γραμματική με την αντίστοιχη λέξη.
Έτσι λοιπόν καλώντας την callsphinx() μέθοδο δημιουργούμε σε πρώτη φάση το ConfigurationManager που με βάση το αρχείο config αναζητά τα δύο αυτά αρχεία (λεξικό και γραμματική) και τα τοποθετεί στην μνήμη για να είναι διαθέσιμα σε όλη την διάρκεια της εφαρμογής με αρκετά γρήγορο τρόπο. Εκτός όμως από το λεξικό και την γραμματική από τον

ConfigurationManager δημιουργούμε δυο πολύ σημαντικά στοιχεία που είναι ο recognizer και το microphone με αυτές τις εντολές :

recognizer = (Recognizer) cm.lookup("recognizer");
microphone = (Microphone) cm.lookup("microphone");

Τα στοιχεία και του recognizer και του microphone βρίσκονται στο αρχείο recognition.config.xml. Τέλος με την χρήση της εντολής :

recognizer.allocate();

όλα τα παραπάνω ενσωματώνονται δεσμεύονται οι απαραίτητοι πόροι του συστήματος. Οι πόροι αυτοί είναι μνήμη καθώς κάποια από τα αρχεία όπως το λεξικό και η γραμματική μπαίνουν εκεί αλλά και η κάρτα ήχου που δεσμεύεται για την αναμονή ηχητικού σήματος από την είσοδο της. Επίσης υπάρχουν και κάποιες περιπτώσεις εδώ αναγνώρισης τυχόν λάθος κατά την διαδικασία της αρχικοποίησης όπου και εκτυπώνεται το αντίστοιχο λάθος ενημερώνοντας τον χρήστη για το πρόβλημα.
Όταν λοιπόν γίνει η κλήση της callsphinx() έχουμε αρχικοποιήσει την μηχανή αναγνώρισης και είναι έτοιμη να ξεκινήσει.

- **startsphinx() :** Η μέθοδος αυτή έχει την πολύ απλή διαδικασία να αρχίσει την καταγραφή ήχου από την είσοδο της κάρτας ήχου και πιο συγκεκριμένα από το μικρόφωνο. Αυτό γίνεται με την εντολή:

  microphone.startRecording();

- **recording() :** Η μέθοδος αυτή είναι ουσιαστικά η μέθοδος που κάνει την αναγνώριση. Καταρχήν καθαρίζεται το buffer που κρατάει το εισερχόμενο ηχητικό σήμα :

  microphone.clear();

αυτό γίνεται για να μπορούμε όσο το δυνατόν να μην μπερδεύουμε την αναγνώριση με ηχητικά δεδομένα προηγούμενης αναγνώρισης.
Στην συνέχεια κάνουμε την αναγνώριση με αυτές τις δύο εντολές :

result = recognizer.recognize();
resultText=result.getBestFinalResultNoFiller();
και έτσι έχουμε σαν αποτέλεσμα ένα String με το αποτέλεσμα της αναγνώρισης.

Έτσι λοιπόν έχουμε τώρα όλες τις κατάλληλες μεθόδους που μπορούμε να χρησιμοποιήσουμε για να μπορέσουμε να ενεργοποιήσουμε την μηχανή αναγνώρισης.
Επίσης σε δοκιμές που έγιναν έχει φτιαχτεί και ο τερματισμός της μηχανής αναγνώρισης με τις παρακάτω εντολές :

microphone.stopRecording();
recognizer.deallocate();

Το κακό με το σταμάτημα είναι ότι για να ξεκινήσουμε πάλι την όλη διαδικασία θα πρέπει να ξανακαλέσουμε την callspinx() η οποία όμως καθότι δημιουργεί όλα τα απαραίτητα αντικείμενα και δεσμεύει πόρους καθυστερεί περίπου 7-12 δευτερόλεπτα ανάλογα με το μηχάνημα που χρησιμοποιούμε. Για την δική μας εφαρμογή που έχουμε πλοήγηση μέσω φωνής δεν είναι αποδεκτό να έχουμε μια τέτοια καθυστέρηση. Θα μπορούσε όμως η ενέργεια του σταματήματος να υπάρξει σε μελλοντικές τροποποιήσεις της εφαρμογής.

Medicalgui.class

Η κλάση αυτή είναι υπεύθυνη για την δημιουργία του γραφικού περιβάλλοντος αλλά και για την δημιουργία όλων των ζητημάτων πλοήγησης σε αυτό κατά την ώρα που αναγνωρίζονται οι λέξεις από την μηχανή αναγνώρισης. Η Medicalgui κλάση αποτελείται από τις εξής μεθόδους:

- **initComponents() :** Η μέθοδος αυτή έχει σαν στόχο να ορίσει όλα τα στοιχεία που αποτελούν το γραφικό περιβάλλον. Να φτιάξει λοιπόν όλο το γραφικό

περιβάλλον όπως αυτό αποτελείται από tab-menus, λίστες, checkboxes και διάφορα άλλα στοιχεία αλλά και να τα ταξινομήσει σωστά σαν σύνολο με τον τρόπο που έχει σχεδιαστεί ώστε να έχουμε ένα αρκετά καλό οπτικά αποτέλεσμα. Η μέθοδος αυτή απλά ορίζει τα επιμέρους αντικείμενα και τα οριοθετεί. Με την μέθοδο αυτή δεν βάζουμε δεδομένα στα αντίστοιχα στοιχεία του γραφικού περιβάλλοντος καθώς τα δεδομένα προέρχονται από μήνυμα agent και θα πρέπει να γίνουν μέσω κάποιας άλλης μεθόδου που θα παίρνει το περιεχόμενο του μηνύματος. Η κλήση της μεθόδου αυτής γίνεται στον Constructor της κλάσης και αυτό σημαίνει ότι κάθε φορά που φτιάχνουμε ένα αντικείμενο της κλάσης Medicalgui φτιάχνουμε και την δομή του γραφικού περιβάλλοντος.

- **callMedgui() :** Η μέθοδος αυτή έχει τον απλό ρόλο να εμφανίσει το γραφικό περιβάλλον όπως αυτό έχει δομηθεί από την initComponents(), χωρίς δηλαδή τα δεδομένα του ασθενή.
- **loadguisphinx(LoadPatient datasum) :** Η μέθοδος αυτή είναι η μέθοδος που παίρνει σαν είσοδο το ontology LoadPatient με τα δεδομένα του ασθενή και γεμίζει τα αντίστοιχα στοιχεία του γραφικού περιβάλλοντος. Έτσι λοιπόν κάθε φορά που καλείται η μέθοδος αυτή έχει σαν στόχο να πάρει όλα τα στοιχεία του ασθενή από το ontology και να τα τοποθετήσει στα αντίστοιχα πεδία του γραφικού περιβάλλοντος. Η μέθοδος αυτή είναι πολύ σημαντική καθώς προσφέρει την επιθυμητή δυναμικότητα στο γραφικό περιβάλλον, αφού υπάρχει απόλυτη προσαρμογή με τα δεδομένα του ασθενή από την μια αλλά και δεν μας ενδιαφέρει ποια είναι τα δεδομένα αυτά. Δηλαδή για διαφορετικό ασθενή γεμίζει με διαφορετικό τρόπο τα εισερχόμενα δεδομένα πάντα βέβαια σε συμφωνία με την δομή του ontology. Δεν υπάρχει κάποιος λόγος να αναφερθούμε σε παραδείγματα κώδικα εδώ γιατί υπάρχουν πολλά σχόλια στην κλάση αυτή που εξηγούν το κάθε βήμα τι ακριβώς είναι με αρκετή λεπτομέρεια.
- **recognise(Recognition demo) :** Τέλος υπάρχει η μέθοδος αυτή που είναι και η ψυχή της πλοήγησης του γραφικού περιβάλλοντος. Η μέθοδος αυτή παίρνει σαν είσοδο ένα αντικείμενο της κλάσης Recognition και επιστρέφει ένα ontology με δεδομένα του ασθενή. Αφού παίρνει σαν είσοδο ένα αντικείμενο

της κλάσης Recognition μπορούμε να χρησιμοποιήσουμε εντολές και μεθόδους από την κλάση αυτή. Έτσι λοιπόν χρησιμοποιούμε την μέθοδο :

apot=demo.recording();

και θέτουμε το αποτέλεσμα της αναγνώρισης στο String apot. Με βάση τώρα το περιεχόμενο του apot μπορούμε και είμαστε σε θέση να κάνουμε την πλοήγηση στο γραφικό μας περιβάλλον. Με διαδοχικά λοιπόν if και else εξετάζουμε το περιεχόμενο της αναγνωρισμένης λέξης που έχει αποθηκευτεί στο apot και προσαρμόζουμε το γραφικό μας ανάλογα. Θα πρέπει εδώ να τονιστεί ότι έχει γίνει μεγάλη προσπάθεια σε επίπεδο κώδικα να μπορέσουμε να μπλοκάρουμε όσο το δυνατόν καλύτερα τυχόν λάθος αναγνωρίσεις που μπορεί να συμβούν κατά την εκτέλεση της εφαρμογής. Για κάθε αναγνωρισμένη λέξη λοιπόν υπάρχει μια σειρά από εντολές που πραγματοποιούνται για να δείξουμε το αποτέλεσμα της αναγνώρισης οπτικά και να προσφέρουμε πλοήγηση.

Όπως αναφέραμε όμως η μέθοδος αυτή επιστρέφει ένα ontology. Αυτό γίνεται για να μπορέσουμε σε κατάλληλα σημεία του γραφικού περιβάλλοντος μας να μπορέσουμε να φτιάξουμε τα δεδομένα με τα στοιχεία του ασθενή που θέλουμε να στείλουμε μέσω μηνύματος agent. Έτσι λοιπόν με την κλήση της μεθόδου αυτής μπορούμε με κατάλληλες ενέργειες (πχ με την χρησιμοποίηση κάποιου βρόγχου όπως θα δούμε και στην ανάλυση του MedicalstartPlan) να γεμίζουμε ένα ontology και με κατάλληλους μηχανισμούς από το σημείο κλήσης να το στείλουμε με ένα μήνυμα σε κάποιον άλλο agent που υπάρχει στην πλατφόρμα του Jadex. Και σε αυτή την μέθοδο υπάρχουν πολλά σχόλια στον κώδικα που αναφέρουν ακριβώς τα επιμέρους της στοιχεία.

Θα πρέπει να αναφέρουμε εδώ ότι όταν δημιουργούμε ένα αντικείμενο της κλάσης Medicalgui καλώντας τον Constructor της κάνουμε αυτόματα δυο πράγματα. Πρώτον καλούμε την initComponents() και έτσι φτιάχνουμε την δομή του γραφικού περιβάλλοντος και δεύτερον φτιάχνουμε ένα αντικείμενο της κλάσης zoomframe που θα εξηγήσουμε αμέσως μετά τι ακριβώς κάνει και γιατί χρειάζεται.

zoomframe.class

Η κλάση αυτή έχει δημιουργηθεί για να φτιάχνει ένα νέο παράθυρο και να περιέχει μια μεγενθυμένη φωτογραφία κάποιας εξέτασης του ασθενή. Η κλάση αυτή αποτελείται από τις παρακάτω μεθόδους:

- **initComponents() :** Η μέθοδος αυτή όπως και η αντίστοιχη της Medicalgui κλάσης έχει σαν στόχο την δημιουργία της δομή του παραθύρου που θέλουμε να φτιάξουμε. Η μέθοδος αυτή καλείται όταν καλείται ο Constructor της για την δημιουργία ενός αντικειμένου της zoomframe κλάσης.
- **open(String path) :** Η μέθοδος αυτή εμφανίζει το παράθυρο και μάλιστα γεμισμένο με την μεγενθυμένη φωτογραφία που θέλουμε. Αυτό γίνεται αφού η μέθοδος αυτή παίρνει σαν είσοδο ένα String με το όνομα path που είναι και η διαδρομή της φωτογραφίας που θέλουμε. Έτσι καλώντας αυτή την μέθοδο εμφανίζουμε το νέο παράθυρο με την επιθυμητή φωτογραφία.
- **discard() :** Η μέθοδος αυτή χρησιμοποιείται για να μπορέσουμε να κλείσουμε το παράθυρο. Εδώ χρησιμοποιείται η εντολή :

  dispose();

  για να μπορέσουμε να αποδεσμεύσουμε και τους πόρους που χρησιμοποιεί το νέο αυτό παράθυρο στην μνήμη, δηλαδή καταστρέφουμε το αντικείμενο καθώς πλέον δεν μας χρειάζεται όταν έχουμε αποφασίσει ότι πρέπει να καλέσουμε την συγκεκριμένη μέθοδο.

Η κλάση αυτή καλείται από την Medicalgui κλάση όταν έχουμε επιλέξει κάποια εξέταση και θέλουμε να κάνουμε zoom στην φωτογραφία της εξέτασης. Η διαδικασία γίνεται με το να προφέρουμε την λέξη zoom. Επίσης για να σταματήσουμε την προβολή αρκεί να προφέρουμε την λέξη stop και τότε εκτελείται η μέθοδος dispose() που και κλείνει το παράθυρο.

Medicalagent.agent.xml

Το αρχείο αυτό αποτελεί το agent definition file του Medical agent όπως αυτό έχει δομηθεί με βάση τις αρχές και τις δυνατότητες του Jadex. Είναι ουσιαστικά ο κυρίως agent της εφαρμογής μας και όταν των ξεκινάμε εκτελεί τις σημαντικότερες

λειτουργίες της εφαρμογής μας. Ο agent αυτός υποστηρίζει δύο πράγματα. Πρώτον ξεκινάει το MedicalstartPlan που είναι και το βασικό πλάνο της εφαρμογής και θα περιγράψουμε στην συνέχεια και δεύτερον την υποστήριξη λήψης και αποστολής μηνυμάτων agent με περιεχόμενο το ontology που εμπεριέχει τα στοιχεία λήψης και αποστολής που αφορούν τον ασθενή.

Ας δούμε στο σημείο αυτό πως μεταχειρίζεται και πως ενεργοποιεί ο agent αυτός το plan που διαθέτει.

- **Δήλωση του plan :** Η δήλωση του πλάνου γίνεται με τον παρακάτω κώδικα:

```
<plans>
    <plan name="startmedical">
        <body>new MedicalstartPlan()</body>
        <waitqueue>
            <messageevent ref="request_initial"/>
        </waitqueue>
    </plan>
</plans>
```

Αυτό που έχει δηλωθεί εδώ είναι ότι ο agent αυτός έχει ένα πλάνο που ονομάζεται startmedical και ουσιαστικά είναι η εκτέλεση της κλάσης MedicalstartPlan όπως φαίνεται από το tag <body>. Επίσης αυτό το πλάνο όταν εκτελείται περιμένει να ενεργοποιηθεί το messageevnt που έχει το όνομα request_initial.

- **Εκτέλεση κατά την ενεργοποίηση του agent :** Κατά την ενεργοποίηση του agent εκτελείται το plan που έχει το όνομα startmedical και ουσιαστικά όπως είπαμε είναι η εκτέλεση της κλάσης MedicalstartPlan. Αυτό γίνεται με τον παρακάτω κώδικα:

```
<initialstates>
    <initialstate name="default">
        <plans>
            <initialplan ref="startmedical"/>
        </plans>
    </initialstate>
```

</initialstates>

- **Περιγραφή των υποστηριζόμενων μηνυμάτων:** Ο agent αυτός έχει την δυνατότητα και λήψης και αποστολής μηνύματος. Για να μπορέσει να τις υποστηρίζει αυτές τις δύο δυνατότητες θα πρέπει όμως οι τύποι μηνυμάτων αυτών να περιγραφούν στο definition file του. Αυτό γίνεται με το εξής κομμάτι κώδικα :

//Receive

<events>

<messageevent name="request_initial" direction="receive" type="fipa">

<parameter name="performative" class="String" direction="fixed">

<value>jadex.adapter.fipa.SFipa.REQUEST</value>

</parameter>

</messageevent>

//Send

<messageevent name="request_datasend" direction="send" type="fipa">

<parameter name="performative" class="String" direction="fixed">

<value>jadex.adapter.fipa.SFipa.REQUEST</value>

</parameter>

</messageevent>

</events>

Έτσι λοιπόν έχουμε δυο τύπων μηνύματα που υποστηρίζονται από τον συγκεκριμένο agent και αυτά είναι το μήνυμα αποστολής που ονομάζεται request_initial και το μήνυμα λήψης που ονομάζεται request_datasend. Ας δούμε όμως τώρα πως χρησιμοποιούνται τα δυο μηνύματα αυτά από το MedicalstartPlan που όπως είπαμε καλείται από τον agent αυτόν κατά την ενεργοποίηση του.

MedicalstartPlan.class

Η κλάση αυτή έχει φτιαχτεί με τους κανόνες δημιουργίας ενός plan του Jadex. Έτσι λοιπόν σύμφωνα με τους κανόνες αυτούς θα πρέπει όλες οι εντολές της κλάσης αυτής να περικλείονται σε μια μέθοδο που ονομάζεται **body ()** . Ας δούμε όμως ποιες ακριβώς είναι οι εντολές που χρησιμοποιούνται εδώ και ποιες ενέργειες γίνονται για να μπορέσουμε να καταλάβουμε καλύτερα τον μοναδικό και πολύ σημαντικό ρόλο που έχει η κλάση αυτή στην εφαρμογή μας.

Καταρχήν φτιάχνουμε ένα αντικείμενο της κλάσης Medicalgui και καλούμε την μέθοδο της callsphinx() με τις παρακάτω εντολές:

Medicalgui demostart=new Medicalgui();

demostart.callMedgui();

Με αυτόν τον τρόπο φτιάχνουμε και εμφανίζουμε το γραφικό περιβάλλον λειτουργίας χωρίς όμως αυτό να περιέχει ακόμα καμία πληροφορία για τον ασθενή.

Στην συνέχεια περιμένουμε για ένα εισερχόμενο agent μήνυμα :

IMessageEvent me = waitForMessageEvent("request_initial");

Το μήνυμα αυτό χαρακτηρίζεται από το όνομα request_initial και όπως περιγράψαμε από το definition file του Medicalagent είναι ένα μήνυμα λήψης.

Όταν λοιπόν το μήνυμα έρθει τότε εκτελούμε το :

loaddata = (LoadPatient)me.getContent();

και ουσιαστικά αυτό που κάνουμε είναι να πάρουμε το περιεχόμενου του εισερχόμενου μηνύματος που είναι ένα ontology LoadPatient και περιέχει όλα τα στοιχεία του ασθενή.
Στην συνέχεια καλούμε το :

demostart.loadguisphinx(loaddata);

που είναι η μέθοδος της κλάσης Medicalgui με όρισμα το ontology που πήραμε από εισερχόμενο μήνυμα. Με αυτόν τον τρόπο κάνουμε ενημέρωση και προσαρμόζουμε κατάλληλα το γραφικό περιβάλλον με τα στοιχεία του ασθενή που βρίσκονται στο ontology και το πήραμε από το εισερχόμενο agent μήνυμα.
Αφού λοιπόν έχουμε προσαρμόσει το γραφικό περιβάλλον φτιάχνουμε ένα αντικείμενο της κλάσης Recognition και καλούμε τις κατάλληλες μεθόδους του για να αρχίσουμε την διαδικασία της αναγνώρισης. Έτσι λοιπόν εκτελούμε τα παρακάτω:

Recognition demo = new Recognition();
demo.callsphinx();
demo.startsphinx();

Είμαστε έτοιμοι λοιπόν και έχουμε αρχίσει την μηχανή αναγνώρισης φωνητικών εντολών.
Αυτό που πρέπει να γίνει στην συνέχεια είναι να εξασφαλίσουμε από την μία την πλοήγηση στο γραφικό περιβάλλον μέσω της αναγνώρισης των φωνητικών εντολών και από την άλλη να είμαστε σε θέση όταν χρειαστεί να μπορούμε να συντάξουμε και να αποστείλουμε ένα agent μήνυμα με τα κατάλληλα στοιχεία που αφορούν τον ασθενή. Αυτό γίνεται με την δημιουργία ενός ατέρμονα βρόγχου :

while(true) {
….
}

όπου μέσα υπάρχουν οι κατάλληλες εντολές για τις δύο αυτές διαδικασίες.
Μέσα στον βρόγχο αυτόν λοιπόν καλούμε κάθε φορά την μέθοδο recognise() της κλάσης Medicalgui με τον εξής τρόπο :

LoadPatient senddata =demostart.recognise(demo);

Η μέθοδος λοιπόν recognise() παίρνει σαν όρισμα το αντικείμενο της κλάσης Recognition που φτιάξαμε παραπάνω και επιστρέφει πάντα ένα ontology με την ονομασία senddata που είναι και το ontology που θα περιέχει τα δεδομένα του χρήστη που θέλουμε να αποστείλουμε. Παράλληλα μέσα στην μέθοδο ανάλογα με το αποτέλεσμα της αναγνώρισης που γίνεται έχουμε και την κατάλληλη προσαρμογή του γραφικού περιβάλλοντος με τον τρόπο που αναφέραμε στην περιγραφή της μεθόδου recognise() της κλάση Medicalgui παραπάνω.
Αφού τώρα έχουμε το ontology σαν επιστροφή από την μέθοδο recognise() μπορούμε να εξετάσουμε το περιεχόμενο του. Αν είναι μηδενικό τότε απλώς εκτυπώνουμε ένα μήνυμα ότι δεν έχει δοθεί η κατάλληλη εντολή για να αποσταλεί κάποιο μήνυμα. Αν τώρα έχει δεδομένα πρέπει να φτιάξουμε ένα μήνυμα και να το αποστείλουμε στον κατάλληλο παραλήπτη. Αυτό γίνεται με τον εξής τρόπο:

```
jadex.adapter.fipa.AgentIdentifier receiver;
receiver = new AgentIdentifier("Receiver", true);
IMessageEvent sender = createMessageEvent("request_datasend");
sender.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).addValue(recei
ver);
sender.setContent(senddata);
sendMessage(sender);
```

Παρατηρούμε ότι φτιάχνουμε το μήνυμα με το ontology που μας γύρισε η μέθοδος recognise() και το στέλνουμε στον agent που έχει το όνομα Receiver. Το μήνυμα που θα στείλουμε έχει το όνομα request_senddata και όπως έχει περιγραφεί από το definition file του Medicalagent είναι ένα μήνυμα αποστολής.

Επειδή όλη αυτή η διαδικασία βρίσκεται στον ατέρμονα βρόγχο επιτυγχάνουμε την διαρκή πλοήγηση μέσω της αναγνώρισης φωνητικών εντολών αλλά έχουμε και την δυνατότητα να αποστείλουμε κάποιο μήνυμα με δεδομένα του ασθενή όταν πρέπει (όταν δηλαδή το ontology δεν είναι κενό) ανά πάσα στιγμή.

Τέλος η κλάση αυτή αποτελεί τον συνδετικό κρίκο όλων των χρησιμοποιούμενων τεχνολογιών και είναι η σημαντικότερη ίσως κλάση που υπάρχει στην εφαρμογή. Με βάση την σημαντικότητα της αλλά και τις απαιτήσεις της προσαρμόστηκαν όλα τα υπόλοιπα τμήματα της εφαρμογής για να μπορέσουν να ικανοποιήσουν τις ανάγκες που δημιουργεί η υλοποίηση της κλάσης αυτής.

Starter.agent.xml

Ο agent αυτός έχει φτιαχτεί για να προσομοιάσει το περιβάλλον λειτουργίας του γενικότερου ιατρικού project στο οποίο μετέχει η εφαρμογή. Πιο συγκεκριμένα έχει δημιουργηθεί για να κάνει την σύνταξη και την αποστολή ενός μηνύματος με τα στοιχεία του ασθενή που χρειάζεται του MedicalstartPlan του Medicalagent για να προσαρμόσει το γραφικό περιβάλλον όπως περιγράψαμε και παραπάνω.

Ας δούμε όμως στο σημείο αυτό τι περιλαμβάνει το definition file του Starter agent.

Ο agent λοιπόν αυτός έχει σαν στόχο να εκτελέσει ένα plan. Το plan αυτό έχει δηλωθεί με τον παρακάτω τρόπο:

```
<plan name="starter">
        <body>new InitialPlan()</body>
</plan>
```

έχει λοιπόν το όνομα starter και ουσιαστικά εκτελεί την κλάση InitialPlan που θα αναλύσουμε αμέσως παρακάτω.

Για να εκτελεστεί όμως το συγκεκριμένο πλάνο κατά την εκκίνηση του agent θα πρέπει να υπάρχουν και οι εξής γραμμές κώδικα στο definition file του agent :

```
<initialstates>
        <initialstate name="default">
```

```
            <plans>
                <initialplan ref="starter"/>
            </plans>
        </initialstate>
</initialstates>
```

Με αυτόν τον τρόπο ενεργοποιείται η εκτέλεση του plan με το όνομα starter κάθε φορά που ξεκινάει ο agent Starter.

Επίσης αναφέραμε ότι ο agent αυτός έχει δημιουργηθεί για να αποστέλλει ένα μήνυμα και μάλιστα όχι οποιοδήποτε μήνυμα αλλά το μήνυμα που περιμένει να δεχτεί ο Medicalagent. Για να γίνει αυτό θα πρέπει να ορίσουμε τον τύπο του μηνύματος αυτού στο definition file του agent με τον παρακάτω τρόπο:

```
<events>
    <messageevent name="request_send" type="fipa" direction="send">
        <parameter name="performative" class="String"
        direction="fixed">
            <value>jadex.adapter.fipa.SFipa.REQUEST</value>
        </parameter>
    </messageevent>
</events>
```

Παρατηρούμε ότι το όνομα του μηνύματος είναι το ίδιο με το όνομα που περιμένει να δεχτεί ο Medicalguiagent με την μόνη διαφορά ότι το direction τώρα είναι στο send και όχι στο receive όπως ήταν στον Medicalagent. Με αυτό τον τρόπο προσφέρουμε στον agent αυτόν να έχει την δυνατότητα αποστολής του μηνύματος που περιμένει ο Medicalagent και ο οποίος όπως αναφέραμε προσαρμόζει το γραφικό περιβάλλον με

βάση τα στοιχεία αυτού του μηνύματος μέσω της κλάσης MedicalstartPlan που εκτελεί.

Ας δούμε όμως στην συνέχεια το plan που εκτελεί ο Starter agent και που είναι η κλάση InitialPlan.

InitialPlan.class

Και αυτή η κλάση έχει φτιαχτεί με τους κανόνες των πλάνων του Jadex. Όλες οι εντολές λοιπόν της κλάσης αυτής περικλείονται σε μια μέθοδο που ονομάζεται **body()**. Όπως είπαμε ο σκοπός του agent Starter και άρα και της κλάσης που αυτός καλεί είναι η σύνταξη και η αποστολή ενός μηνύματος με περιεχόμενο στοιχεία κάποιου ασθενή μορφοποιημένα σε ένα ontology. Το μήνυμα αυτό αποστέλλεται στον Medicalgui agent, που αναφέραμε πως συμπεριφέρεται μετά την λήψη του συγκεκριμένου μηνύματος παραπάνω.

Έτσι λοιπόν με αυτήν εδώ την κλάση έχουμε δύο λειτουργίες που πραγματοποιεί :

- **Σύνταξη μηνύματος:** Υπάρχουν μια σειρά από εντολές που έχουν σαν στόχο την δημιουργία του ontology και το γέμισμα αυτού με στοιχεία που αφορούν τον ασθενή. Στον κώδικα της κλάσης InitialPlan υπάρχουν αρκετά σχόλια που επεξηγούν πως ακριβώς φτιάχνεται το ontology και πως εισάγουμε σε αυτό δεδομένα. Ας δούμε όμως εδώ ένα μικρό παράδειγμα που αφορά το προφίλ του ασθενή :

  ```
  PatientProfile profiledata = new PatientProfile();
  profiledata.setAddress("Xiou 12 A");
  profiledata.setBirthday("03/11/1956");
  profiledata.setFullName("Tsarmpopoulos Kyriakos");
  profiledata.setMedicalID("1234jdad144233");
  profiledata.setPhotoPath("C:\\jadex-
  0.941\\src\\jadex\\examples\\dist\\kyriakos.jpg");

  ptalldata.setProfile(profiledata);´
  ```

Με αυτόν τον τρόπο αποθηκεύουμε στο ontology (ptalldata) το profiledata που αποτελείται από τα δεδομένα του ασθενή που έχουν σχέση με το προφίλ του. Τα δεδομένα αυτά όπως μπορούμε να δούμε και από το παραπάνω κομμάτι κώδικα είναι η διεύθυνση (Xiou 12 Α), η ημερομηνία γέννησης (03/11/1956), το πλήρες όνομά του (Tsarmopoulos Kyriakos), ο νοσοκομειακός κωδικός του (1234jdad144233) καθώς και η διαδρομή στον τοπικό δίσκο όπου βρίσκεται η φωτογραφία του ασθενή.

Αντίστοιχα φτιάχνονται τα δεδομένα που αφορούν τις εξετάσεις, τις διαγνώσεις και τις θεραπείες που αφορούν τον ασθενή από την κλάση InitialPlan.

- **Αποστολή μηνύματος :** Αφού φτιάξουμε το μήνυμα πρέπει και να το στείλουμε και μάλιστα να το στείλουμε στον σωστό παραλήπτη που είναι ο Medicalagent. Αυτό γίνεται ως εξής:

```
jadex.adapter.fipa.AgentIdentifier receiver;
receiver = new AgentIdentifier("Medicalagent", true);
IMessageEvent me = createMessageEvent("request_send");
me.getParameterSet(jadex.adapter.fipa.SFipa.RECEIVERS).addValue(receiver);
me.setContent(ptalldata);
sendMessage(me);
```

Παρατηρούμε ότι το όνομα του μηνύματος είναι ίδιο με αυτό που περιγράφεται στο definition file του Starter agent και το όνομα του παραλήπτη είναι το Medicalagent που είναι και αυτός που θέλουμε να λάβει το συγκεκριμένο μήνυμα.

Receiver.agent.xml

Ο agent αυτός έχει φτιαχτεί για να προσομοιάσει το περιβάλλον λειτουργίας του γενικότερου ιατρικού project στο οποίο μετέχει η εφαρμογή. Πιο συγκεκριμένα έχει δημιουργηθεί για να λάβει ένα μήνυμα με τα στοιχεία του ασθενή που αποστέλλει το

MedicalstartPlan του Medicalagent όπως περιγράψαμε παραπάνω.Ας δούμε όμως στο σημείο αυτό τι περιλαμβάνει το definition file του Receiver agent.
Ο agent λοιπόν αυτός έχει σαν στόχο να εκτελέσει ένα plan. Το plan αυτό έχει δηλωθεί με τον παρακάτω τρόπο:

```
<plan name=" receiver">
        <body>new ReceiverPlan()</body>
</plan>
```

έχει λοιπόν το όνομα receiver και ουσιαστικά εκτελεί την κλάση ReceiverPlan που θα αναλύσουμε αμέσως παρακάτω.
Για να εκτελεστεί όμως το συγκεκριμένο πλάνο κατά την εκκίνηση του agent θα πρέπει να υπάρχουν και οι εξής γραμμές κώδικα στο definition file του agent :

```
<initialstates>
        <initialstate name="default">
                <plans>
                        <initialplan ref=" receiver "/>
                </plans>
        </initialstate>
</initialstates>
```

Με αυτόν τον τρόπο ενεργοποιείται η εκτέλεση του plan με το όνομα receiver κάθε φορά που ξεκινάει ο agent Receiver.
Επίσης αναφέραμε ότι ο agent αυτός έχει δημιουργηθεί για να λαμβάνει ένα μήνυμα και μάλιστα όχι οποιοδήποτε μήνυμα αλλά το μήνυμα που στέλνει ο Medicalagent. Για να γίνει αυτό θα πρέπει να ορίσουμε τον τύπο του μηνύματος αυτού στο definition file του agent με τον παρακάτω τρόπο:

```
<events>
```

```
<messageevent name="request_datasend" type="fipa" direction="
receive ">

    <parameter name="performative" class="String"
    direction="fixed">

        <value>jadex.adapter.fipa.SFipa.REQUEST</value>

    </parameter>

</messageevent>

</events>
```

Παρατηρούμε ότι το όνομα του μηνύματος είναι το ίδιο με το όνομα που περιμένει να δεχτεί ο Medicalguiagent με την μόνη διαφορά ότι το direction τώρα είναι στο receive και όχι στο send όπως ήταν στον Medicalagent. Με αυτό τον τρόπο προσφέρουμε στον agent αυτόν να έχει την δυνατότητα λήψης του μηνύματος που αποστέλλει ο Medicalagent. Ας δούμε όμως στην συνέχεια το plan που εκτελεί ο Receiver agent και που είναι η κλάση ReceiverPlan.

ReceiverPlan.class

Η κλάση αυτή έχει δημιουργηθεί για να λαμβάνει ένα μήνυμα όπως αυτό έχει οριστεί στο definition file του agent Receiver που περιγράψαμε παραπάνω. Και αυτή η κλάση έχει φτιαχτεί με του κανόνες του Jadex που αφορούν τα plans. Έτσι όλες οι εντολές τις κλάσης αυτής περικλείονται σε μια μέθοδο με το όνομα **body()**. Η κλάση λοιπόν αυτή έχει έναν ρόλο. Να λάβει το μήνυμα από τον Medicalagent και να εκτυπώσει τα περιεχόμενα του μηνύματος που αφορούν τον ασθενή. Το μήνυμα έχει σαν περιεχόμενο ένα ontology το οποίο περιέχει όλα τα διαθέσιμα στοιχεία του ασθενή που επιθυμούμε να δούμε. Ας δούμε όμως πως γίνονται αυτά:

- **Λήψη μηνύματος :** Με το παρακάτω κομμάτι κώδικα η κλάση αυτή περιμένει να λάβει το μήνυμα.

IMessageEvent me = waitForMessageEvent("request_datasend");

Περιμένει λοιπόν το μήνυμα που θα έρθει και θα έχει το όνομα request_datasend, που όπως έχουμε αναφέρει και παραπάνω στέλνει ο Medicalagent.
Αφού λάβει το μήνυμα παίρνει το περιεχόμενο του που είναι ένα ontology και το αποθηκεύει σε μια μεταβλητή για να μπορεί να το μεταχειριστεί. Αυτό γίνεται ως εξής :

loaddata = (LoadPatient)me.getContent();

Έτσι λοιπόν τώρα έχουμε βάλει σε μια μεταβλητή το ontology που στάλθηκε μέσω μηνύματος από τον Medicalagent.

- **Προβολή δεδομένων :** Η προβολή δεδομένων γίνεται με την αντίστροφη διαδικασία σχηματισμού του ontology. Δηλαδή ενώ στον σχηματισμό κάνουμε set το χαρακτηριστικό που θέλουμε, εδώ το κάνουμε get. Για να το καταλάβουμε καλύτερα υπάρχει το εξής κομμάτι κώδικα:

```
PatientDiagnosis ptdiagrecdata[] = loaddata.getDiagnoses();
String [] ptdiagrecdesc= new String[ptdiagrecdata.length];
        for(int i = 0 ; i < ptdiagrecdesc.length ; ++i) {
                ptdiagrecdesc[i]=ptdiagrecdata[i].getDiagnosis();
                System.out.println("The " +(i+1)+ " diagnosis is :
                "+ptdiagrecdesc[i]+"\n");
        }
```

Το παραπάνω κομμάτι κώδικα δείχνει πως παίρνουμε τα στοιχεία των διαγνώσεων από το ληφθέν ontology. Οι διαγνώσεις είναι μια λίστα και για αυτό χρησιμοποιείται αυτός ο τρόπος. Το ίδιο συμβαίνει και με τις θεραπείες

γιατί σε έναν ασθενή μπορεί να αντιστοιχούνται πολλές διαγνώσεις και θεραπείες. Αφού γίνει η καταχώρηση των δεδομένων λήψης τα εκτυπώνουμε και στην οθόνη για να έχουμε και ένα οπτικό μέτρο ότι όλα δουλεύουν σωστά.

4.4 Προγράμματα που χρησιμοποιήθηκαν και ενέργειες εγκατάστασης

Για την υλοποίηση της παρούσας εφαρμογής χρησιμοποιήθηκαν κάποια προγράμματα και εφαρμογές για να μπορέσουν να δημιουργηθούν όλα τα κομμάτια της. Το κάθε κομμάτι είχε τις δικές του απαιτήσεις υλοποίησης και παραμετροποίησης και για αυτό χρησιμοποιήθηκαν μια σειρά από προγράμματα για το λόγο αυτό. Εδώ περιγράφονται συνοπτικά τα προγράμματα που χρησιμοποιήθηκαν συνοδευμένα από λίγα λόγια για το καθένα που αναφέρουν το σε ποιο κομμάτι της εφαρμογής χρησιμοποιήθηκαν.

- **Java jdk 1.5.0 :** Χρησιμοποιήθηκε για να μπορέσει να υποστηριχθεί η γλώσσα προγραμματισμού Java, η οποία είναι η βάση για όλα τα επιμέρους κομμάτια που αποτελούν την εφαρμογή.
- **Jadex-0.941 :** Αποτελεί την εφαρμογή Jadex που είναι η middleware πλατφόρμα agents που χρησιμοποιεί η εφαρμογή.
- **Sphinx–4.0.1alpha :** Αποτελεί την μηχανή αναγνώρισης φωνής που έχει χρησιμοποιηθεί για την αναγνώριση φωνητικών εντολών της εφαρμογής.
- **NetBeans 5.0 :** Είναι ένα IDE που χρησιμοποιείται για την δημιουργία Java προγραμμάτων και έχει πολύ μεγάλες δυνατότητες ελέγχου και διαχείρισης των προγραμμάτων κατά την φάση υλοποίησης τους. Χρησιμοποιήθηκε για την δημιουργία του γραφικού περιβάλλοντος καθώς και κατά ένα μεγάλο μέρος για την σύνδεση του γραφικού με την μηχανή αναγνώρισης φωνητικών εντολών.
- **Ant 1.6.5 :** Είναι ένα πρόγραμμα που χρησιμοποιείται για να κάνει build τις εφαρμογές χρησιμοποιώντας ένα config.xml αρχείο όπου εκεί περιγράφεται η δομή και οι παράμετροι της εφαρμογής. Χρησιμοποιήθηκε κατά την φάση παραμετροποίησης της μηχανής αναγνώρισης φωνής του sphinx.

- **SphinχTrain :** Είναι ένα εργαλείο που συνοδεύει το sphinx και προορίζεται για την δημιουργία του acoustic model του sphinx. Χρησιμοποιήθηκε κατά την προσπάθεια δημιουργίας ενός ελληνικού acoustic model στην αρχική φάση της υλοποίησης. Η δημιουργία του ελληνικού μοντέλου δεν ήταν επιτυχής και θα ακολουθήσει περιγραφή της παραπάνω προσπάθειας καθώς και ανάλυση των λόγων που οδήγησαν στην αποτυχία.
- **Protégé 3.1 :** Είναι ένα πρόγραμμα που χρησιμοποιήθηκε από όλη την ομάδα του γενικότερου ιατρικού project για την δημιουργία των ontologies που χρησιμοποιούνται από τις επιμέρους εφαρμογές. Στην συγκεκριμένη εφαρμογή χρησιμοποιήθηκε για την δημιουργία του ontology που είναι ουσιαστικά το περιεχόμενο των μηνυμάτων agents που ανταλλάσσονται από και προς την εφαρμογή.

Όλα τα παραπάνω προγράμματα που αναφέρθηκαν είναι δωρεάν και τα περισσότερα από αυτά είναι open-source και υπάρχουν στο συνοδευτικό cd μαζί με την ίδια την εφαρμογή.

Ας δούμε στο σημείο αυτό πως έγινε η εγκατάσταση των προγραμμάτων αυτών, πως χρησιμοποιήθηκαν και ποιες ενέργειες έγιναν στην φάση της υλοποίησης. Οι πληροφορίες αυτές μπορούν να βοηθήσουν στην καλύτερη κατανόηση της υλοποίησης της εφαρμογής αλλά και στην παραμετροποίηση της για να καλύψει διαφορετικές χρήσεις και απαιτήσεις που ίσως δημιουργηθούν μελλοντικά.

1. Καταρχήν πρέπει να γίνει εγκατάσταση της Java στο μηχάνημα που θέλουμε. Χρησιμοποιούμε το jdk 1.5.0 και αφού κάνουμε την εγκατάσταση βάζουμε στο Path του συστήματος τον φάκελο jadk_1_5_0\bin για να μπορούμε να χρησιμοποιήσουμε από οποιοδήποτε σημείο του μηχανήματος μας εντολέ όπως είναι το java (run) και το javac (compile).
2. Στην συνέχεια πρέπει να εγκαταστήσουμε το jadex. Αυτό που κάνουμε είναι να φτιάξουμε ένα φάκελο στον δίσκο του συστήματος μας, όπως για παράδειγμα είναι το C:\jadex-0.941. Σε αυτόν το φάκελο κάνουμε extract όλους τους φακέλους του jadex. Στην συνέχεια θα πρέπει να φτιάξουμε μια user variable στο σύστημα μας που ονομάζεται CLASSPATH και να βάλουμε

μέσα σαν τιμές τα jar αρχεία που βρίσκονται στο φάκελο C:\jadex-0.941\lib. Με αυτόν τον τρόπο έχουμε πρόσβαση στο jadex και μπορούμε να το χρησιμοποιήσουμε στα προγράμματα μας καλώντας τις κατάλληλες μεθόδους που θέλουμε.

3. Μετά κάνουμε εγκατάσταση το sphinx4. Η εγκατάσταση αυτή γίνεται για να μπορέσουμε να τροποποιήσουμε την μηχανή αναγνώρισης φωνής πέραν από το σημείο να αλλάξουμε απλώς το λεξικό ή την συντακτική δομή που χρησιμοποιείται. Φτιάχνουμε λοιπόν ένα φάκελο πχ C:\spinx και κάνουμε εκεί μέσα όλους τους φακέλους τους sphinx. Για να μπορέσουμε να κάνουμε build τα αρχεία πρέπει στο σημείο αυτό να χρησιμοποιήσουμε το πρόγραμμα ant. Κάνουμε λοιπόν εγκατάσταση το ant και στην συνέχεια μπορούμε με την εντολή ant –find build.xml μπορούμε να κάνουμε το build του sphinx. Το build.xml είναι ένα αρχείο του sphinx το οποίο περιέχει όλες τις παραμέτρους που πρέπει να συμπεριληφθούν στο build του sphinx. Με αυτόν τον τρόπο μπορούμε κάθε φορά που αλλάζουμε την δομή από τα java αρχεία του sphinx ή των παραμέτρων που χρησιμοποιούνται από το αρχείο build.xml να φτιάχνουμε την εφαρμογή στα μέτρα μας και α περνούμε τα εκτελέσιμα αρχεία και να τα τρέχουμε σε οποιοδήποτε μηχάνημα το οποίο φυσικά υποστηρίζει Java. Στην εφαρμογή μας και πιο συγκεκριμένα αυτό που υπάρχει είναι το αποτέλεσμα της διαδικασίας αυτής δηλαδή τα αρχεία εκείνα τα οποία έχουν γίνει build σύμφωνα με τις απαιτήσεις μας και μπορούν να χρησιμοποιηθούν τώρα από άλλες εφαρμογές. Η διαδικασία του χτισίματος της εφαρμογής μπορεί και πρέπει να γίνει σε περιπτώσεις που οι παράμετροι της μηχανής αναγνώρισης φωνητικών εντολών δεν εξυπηρετούν μελλοντικές χρήσεις της εφαρμογής. Οι παράμετροι όπως είναι το λεξικό και η σύνταξη μπορούν να αλλάξουν και από τα τελικά αρχεία (αυτά που υπάρχουν στην εφαρμογή) με κατάλληλες ενέργειες που περιγράψαμε στην ανάλυση της υλοποίησης της εφαρμογής.
4. Ακολουθεί η εγκατάσταση του NetBeans 5.0. Το Netbeans είναι ένα IDE για προγραμματισμό σε Java. Κάνουμε λοιπόν install το NeatBeans από το εκτελέσιμο αρχείο που βρίσκεται στο συνοδευτικό cd. Το Neatbeans χρησιμοποιήθηκε όπως αναφέραμε για την δημιουργία του γραφικού περιβάλλοντος. Σαν αποτέλεσμα παράγει ένα project το οποίο στην δική μας

εφαρμογή περικλείεται στον φάκελο Medical που βρίσκεται και αυτός στο συνοδευτικό cd. Αυτό που πρέπει λοιπόν να γίνει για αλλαγές στο γραφικό περιβάλλον της εφαρμογής είναι να γίνει import το project που υπάρχει στον φάκελο Medical και στην συνέχεια έχουμε την δυνατότητα να τροποποιήσουμε το γραφικό περιβάλλον είτε σε επίπεδο κώδικα είτε σε επίπεδο γραφικό. Θα πρέπει στο σημείο αυτό να αναφερθεί ότι το Medical project του NetBeans εκτός από το γραφικό περιβάλλον περιέχει και την σύνδεση με την μηχανή αναγνώρισης οπότε υπάρχει η δυνατότητα να αλλάξουν και κάποια στοιχεία από εκεί. Φυσικά το γραφικό περιβάλλον μπορεί να αλλάξει και από τα αρχεία java που υπάρχουν στην δομή της εφαρμογής με έναν απλό editor. Αυτό το επιπλέον που προσφέρει το NetBeans είναι η παρουσίαση (preview) του γραφικού περιβάλλοντος και οι αλλαγές σε αυτό το επίπεδο, που βοηθάει σε πιο καλαίσθητα αποτελέσματα.

5. Στο σημείο αυτό θα πρέπει να αναφέρουμε και την εγκατάσταση του Protégé 3.1 το οποίο χρησιμοποιήθηκε για την δημιουργία του ontology που περνάει σαν περιεχόμενο στα μηνύματα που ανταλλάσει η εφαρμογή. Στο συνοδευτικό cd υπάρχει το εκτελέσιμο που κάνει εγκατάσταση το Protégé και υπάρχει και ένας φάκελος που ονομάζεται jadex-beanynizer-0.941 και πρέπει να μπει ston φάκελο plugins που δημιουργείται στον φάκελο εγκατάστασης του Protégé για να υποστηρίζει τις λειτουργίες του jadex. Όλες οι πληροφορίες για το ontology περιέχονται στο αρχείο WinHPN.pprj που υπάρχει στο συνοδευτικό cd και είναι ουσιαστικά ένα Protégé project. Ανοίγοντας το αρχείο αυτό μπορούν να γίνουν οποιεσδήποτε αλλαγές στο project και στην συνέχεια να παραχθούν τα κατάλληλα αρχεία .java που περιέχουν τις αλλαγές αυτές. Θα πρέπει στην συνέχεια να γίνουν τα αρχεία αυτά compile και να προστεθούν στο path της εφαρμογής καθώς να αλλάξουν κατάλληλα και τα αρχεία της εφαρμογής που μεταχειρίζονται το ontology ώστε να επιτύχουμε τις αλλαγές που θέλουμε. Θα μπορούσαμε για παράδειγμα με αυτόν τον τρόπο να αλλάξουμε πεδία που αφορούν τα στοιχεία του ασθενή, είδος εξετάσεων που αφορούν τον ασθενή καθώς και το περιεχόμενο των μηνυμάτων που ανταλλάσει η εφαρμογή με άλλες εφαρμογές και συστήματα.

4.5 Αποτυχημένες ενέργειες

Θα πρέπει στο σημείο αυτό να αναφέρουμε κάποιες ενέργειες οι οποίες δεν τελεσφόρησαν κατά την υλοποίηση της εφαρμογής. Θα περιγράψουμε τις προσπάθειες αυτές καθώς και τους λόγους που οδήγησαν στην αποτυχία τους. Οι ενέργειες αυτές θα μπορούσαν να χρησιμοποιηθούν από μελλοντικές χρησιμοποιήσεις ή τροποποιήσεις της υπάρχουσας εφαρμογής και για αυτό αναφέρονται εδώ.

4.5.1 SphinxTrain

Στο σημείο αυτό της εργασίας θα περιγραφεί η προσπάθεια που έγινε για την δημιουργία ελληνικού ακουστικού μοντέλου. Για να γίνει αυτό χρησιμοποιήθηκε το εργαλείο εκπαίδευσης (training tool) του sphinx ου ονομάζεται SphinxTrain. Το SphinxTrain είναι ένα εργαλείο που χρησιμοποιείται για την δημιουργία του acoustic model του sphinx μέσα από μια σειρά ενεργειών που πρέπει να ακολουθηθούν. Στην συνέχεια περιγράφουμε τις ενέργειες που ακολουθηθηκαν περιληπτικά :

- **SphinxTrain build:** Καταρχήν έγινε το build του εργαλείου αυτού. Από τα source files λοιπόν έπρεπε να δημιουργηθούν κάποια εκτελέσιμα προγράμματα που αποτελούν το SphinxTrain. Χρησιμοποιήθηκε λοιπόν το visual studio 2005 για να μπορέσουν να δημιουργηθούν τα εκτελέσιμα αρχεία.
- **Ηχογράφηση :** Στην συνέχεια έπρεπε να δημιουργηθούν κάποια ηχογραφημένα αρχεία με περιεχόμενο τις λέξεις που θέλαμε η μηχανή του sphinx να αναγνωρίζει. Η ηχογράφηση δεν περιλάμβανε φυσικά μόνο τις λέξεις αλλά ολόκληρες προτάσεις που περιείχαν τις λέξεις αυτές μέσα τους.
- **Μετασχηματισμός ηχητικών αρχείων :** Τα εκτελέσιμα αρχεία του SphinxTrain έπρεπε να πάρουν σαν είσοδο τα ηχογραφημένα αρχεία. Για να γίνει αυτό έπρεπε τα αρχεία αυτά α μετατραπούν στο κατάλληλο format, που ήταν το RAW format.
- **Αντιστοίχηση με κείμενο :** Μαζί με τα ηχογραφημένα αρχεία που ήταν πλέον μετασχηματισμένα στο κατάλληλο format έπρεπε να υπάρχει και ένα αρχείο που είχε την αντιστοίχηση των ηχογραφημένων αρχείων με το κείμενο που αυτά περιείχαν. Για παράδειγμα:

Για ένα ηχογραφημένο αρχείο με το όνομα Εξετάσεις.raw που υπήρχε και το περιεχόμενο της ηχογράφησης ήταν "Οι εξετάσεις είναι πολύ καλές", στο αρχείο αυτό υπήρχε μια εγγραφή που ήταν η εξής :

Εξετάσεις.raw (Οι εξετάσεις είναι πολύ καλές)

Αυτό έπρεπε να γίνει για όλα τα ηχογραφημένα αρχεία.

- **Δημιουργία λεξικού :** Στην συνέχεια έπρεπε να δημιουργηθεί ένα λεξικό με όλες τις λέξεις που θα θέλαμε να κάνουμε την αναγνώριση τους με το sphinx.
- **Χρησιμοποίηση του SpinxTrain:** Αφού λοιπόν είχαμε όλα τα αρχεία εισόδου για τα προγράμματα που αποτελούν το SpinxTrain, καλούμε τα προγράμματα αυτά ένα ένα με διάφορα ορίσματα που περιγράφονται αναλυτικά στον παρακάτω οδηγό:

  http://www.speech.cs.cmu.edu/sphinxman/scriptman1.html

Σαν αποτέλεσμα της όλης διαδικασίας είχαμε την δημιουργία του ακουστικού μοντέλου και μάλιστα στην ελληνική γλώσσα.

Μετά την ενσωμάτωση του νέου ακουστικού μοντέλου στην εφαρμογή μας η αναγνώριση ήταν πολύ ανακριβής με ποσοστά της τάξης του 10% για τις αναγνωρίσεις.

Η όλη διαδικασία ξαναέγινε με περισσότερα ηχογραφημένα αρχεία αυτήν την φορά και το ποσοστό των επιτυχημένων αναγνωρίσεων ανέβηκε πάρα πολύ λίγο.

Έτσι λοιπόν το συμπέρασμα από όλη αυτήν την διαδικασία είναι ότι για την δημιουργία ενός ακουστικού μοντέλου από την αρχή θα πρέπει να συνοδεύεται από πολύ μεγάλο αριθμό ηχογραφήσεων και μάλιστα όχι ηχογραφημένων προτάσεων αλλά ολόκληρων κειμένων που φυσικά θα περιέχουν τις λέξεις που θέλουμε. Στα πλαίσια όμως αυτής της εργασίας δεν υπήρξε ούτε το κατάλληλο ανθρώπινο δυναμικό ούτε ο απαραίτητος χρόνος για την συγκομιδή των ηχογραφήσεων αυτών και για αυτό χρησιμοποιήθηκε ένα ακουστικό μοντέλο που έχει δημιουργηθεί από το Carnegie Mellon University και είναι στην αγγλική γλώσσα.

Παρά το ότι δεν δούλεψε η προσπάθεια αυτή της δημιουργία του ακουστικού μοντέλου για τον λόγο που περιγράφθηκε παραπάνω, δόθηκε η ευκαιρία να μελετήσουμε την διαδικασία της δημιουργίας του και να μπορέσουμε να δούμε σε βάθος την δομή της μηχανής αναγνώρισης του sphinx.

4.5.2 Προσπάθεια δυναμικής αλλαγής της γραμματικής

Σε αυτό το σημείο θα πρέπει να αναφέρουμε και άλλη μια προσπάθεια που έγινε για την βελτιστοποίηση της εφαρμογής, η οποία όμως δεν τελεσφόρησε.
Αυτό που επιχειρήθηκε ήταν ενώ είναι σε πλήρη λειτουργία η εφαρμογή να μπορέσουμε να αλλάξουμε το αρχείο της γραμματικής. Έτσι λοιπόν δημιουργήθηκε μια επιπλέον Java κλάση η οποία είχε τον ρόλο να αλλάζει το αρχείο της γραμματικής που χρησιμοποιεί η εφαρμογή.
Η διαδικασία που ακολουθήθηκε ήταν η εξής:

- Αρχικά φτιάχτηκαν 5 αρχεία γραμματικής. Το καθένα από αυτά είχε διαφορετικές λέξεις. Έτσι το σύστημα όταν χρησιμοποιούταν ένα από αυτά τα αρχεία μπορούσε να αναγνωρίζει μόνο αυτές τις λέξεις. Στο λεξικό της εφαρμογής υπήρχαν καταχωρημένες οι λέξεις όλων των αρχείων γραμματικής. Έτσι λοιπόν με το που άρχιζε η εφαρμογή σήκωνε το πρώτο αρχείο που περιείχε λέξει για την βασική πλοήγηση όπως ήταν για παράδειγμα το “PROFILE”, “EXAMS”, “DIAGNOSIS” και “TREATMENT”. Με αυτόν τον τρόπο είχαμε μόνο επιλογή για αυτές τις λέξεις.
- Αφού αναγνωριζόταν μια από τις παραπάνω λέξεις στον κώδικα που αφορούσε τις ενέργειες που θα έπρεπε να γίνουν κατά την αναγνώριση της λέξης κάναμε αντιγραφή και μετονομασία μέσω της κλάσης διαχείρισης αρχείων που φτιάξαμε και ουσιαστικά κάναμε αντικατάσταση της χρησιμοποιούμενης γραμματικής με την γραμματική που περιείχε λέξεις για το υπομενού της αρχικής λέξης που αναγνωρίστηκε. Έτσι για παράδειγμα αν αναγνωριζόταν η λέξη “ EXAMS ” αντικαθιστούσαμε στον κώδικα ενεργειών της αναγνώρισης της λέξης “EXAMS” την χρησιμοποιούμενη γραμματική με ένα αρχείο που είχε φτιαχτεί ειδικά για την περίπτωση αυτή και περιλάμβανε λέξεις για εντολές πλοήγησης που αφορούσαν μόνο εξετάσεις.
- Η παραπάνω διαδικασία επαναλαμβανόταν για όλες τις λέξεις βασικής πλοήγησης. Επίσης υπήρχαν δικλείδες ασφαλείας σε επίπεδο κώδικα ώστε σε κάθε σημείο να μπορούμε να έχουμε τις κατάλληλες εντολές πλοήγησης διαθέσιμες.

Η συγκεκριμένη διαδικασία αν είχε επιτυχές αποτέλεσμα θα μπορούσε κατά ένα μεγάλο μέρος να οδηγήσει σε πολύ μεγάλα ποσοστά επιτυχούς αναγνώρισης και να

βοηθήσει πολύ το σύστημα ώστε να αντιμετωπίσει προβλήματα όπως είναι ο θόρυβος του περιβάλλοντος χώρου και της ιδιομορφίας της ομιλίας του χρήστη που πολλές φορές οδηγούν σε λανθασμένες αναγνωρίσεις φωνητικών εντολών.

Ο λόγος όμως που δεν πέτυχε η χρησιμοποίηση αυτής της τεχνικής όμως είναι γιατί όταν ξεκινάμε την μηχανή αναγνώρισης του sphinx όπως εξηγήσαμε και σε επίπεδο κώδικα στην ανάλυση της κλάσης Recognition είναι ότι πρέπει να δεσμεύσουμε τους κατάλληλους πόρους του συστήματος. Για την δέσμευση αυτή λοιπόν φτιάχνουμε ένα αντικείμενο του Configuration Manager ο οποίος φορτώνει όλα τα απαραίτητα στοιχεία από το config.xml αρχείο. Στο config.xml αρχείο υπάρχουν στοιχεία όπως είναι η θέση του αρχείου του λεξικού και της γραμματικής που θα χρησιμοποιήσουμε. Για την γρηγορότερη λοιπόν αναζήτηση και γενικότερα λειτουργία της μηχανής τα 2 αυτά αρχεία φορτώνονται στην μνήμη με την εντολή allocate() που είναι η δέσμευση των πόρων. Έτσι λοιπόν για να δουλέψει όλη αυτή η τεχνική εν ώρα λειτουργίας θα πρέπει να κάνουμε deallocate() τους πόρους να ξαναφτιάξουμε το αντικείμενο του Configuration Manager με άλλη αυτήν την φορά θέση για το αρχείο της γραμματικής. Για να γίνει όμως αυτή η διαδικασία αποδέσμευσης και δέσμευσης πόρων, δηλαδή επαναρχικοποίηση της εφαρμογής απαιτείται κάποιος χρόνος. Ο χρόνος αυτός είναι της τάξης των 12 δευτερολέπτων. Σε μια εφαρμογή όμως που θέλουμε να έχουμε συνεχή πλοήγηση δεν μπορούμε σε κάποια αναγνώριση να σταματάμε την όλη διαδικασία για 12 δευτερόλεπτα και μετά να την ξαναξεκινάμε. Έτσι λοιπόν η τεχνική αυτή απορρίφθηκε για την υπάρχουσα δομή της και λειτουργίας της.

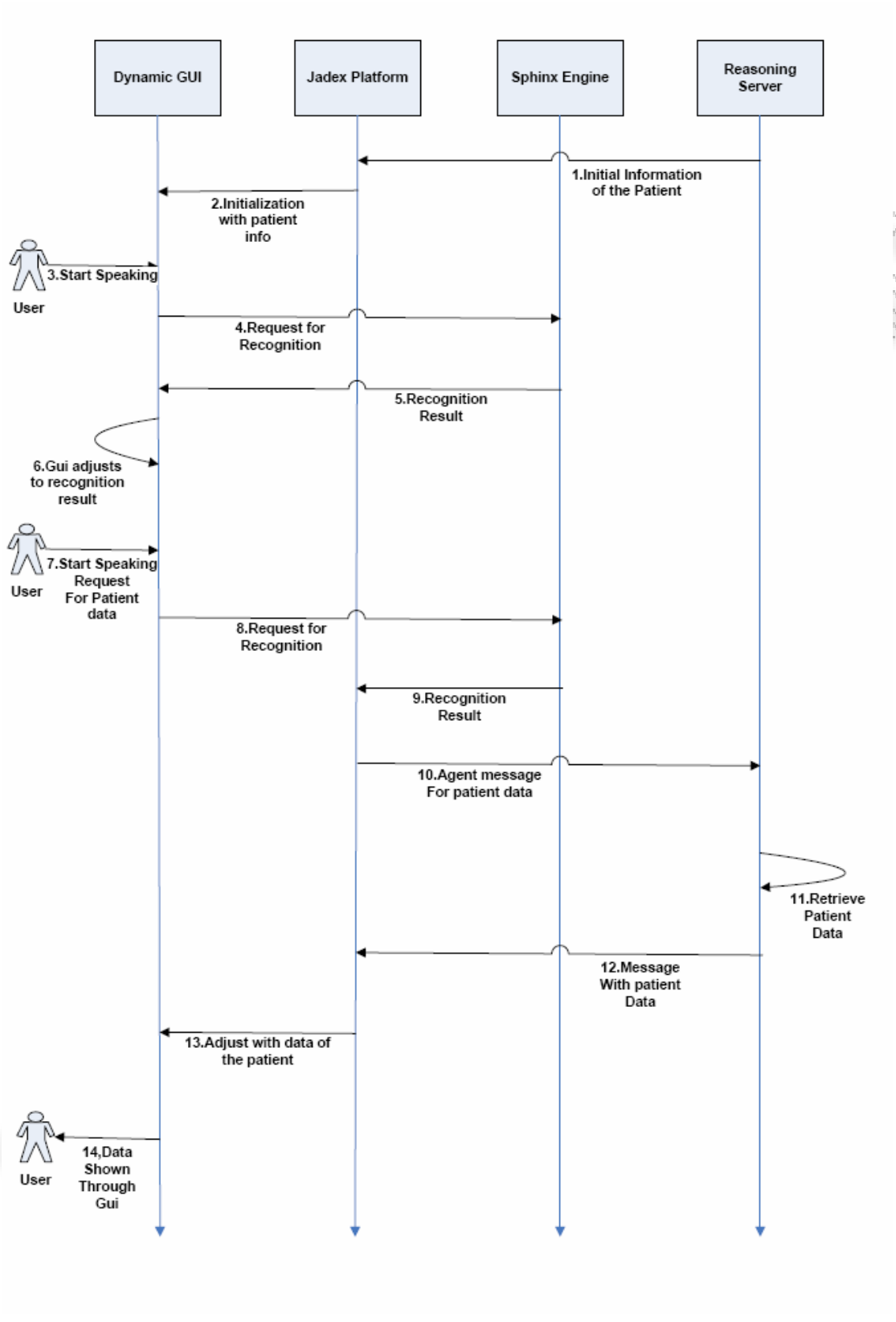
Φυσικά αυτή η τεχνική μπορεί να χρησιμοποιηθεί για κάποια μελλοντική χρήση της εφαρμογής που θα επιτρέπει την αναμονή κάποιου χρόνου ανάμεσα στις αναγνωρίσεις και την αλλαγή της χρησιμοποιούμενης σύνταξης.

5. Αποτελέσματα

5.1 Περίπτωση χρήσης της εφαρμογής

Έτσι λοιπόν αφού αναλύσαμε σε βάθος τα επιμέρους στοιχεία εκείνα που αποτελούν την εφαρμογή μας καλό θα ήταν να δούμε ένα σχεδιάγραμμα μια γενικής περίπτωσης χρήσης και να καταλάβουμε πως όλα αυτά που αναφέραμε σε επίπεδο υλοποίησης γίνονται πράξη όταν χρησιμοποιείται η εφαρμογή.

Το παρακάτω είναι μια ροή γεγονότων που δείχνει την εξέλιξη των καταστάσεων της εφαρμογής. Εδώ ο Receiver και ο Starter agents είναι κομμάτι toy Reasoning Server καθώς εδώ προσπαθούμε να προσομοιάσουμε την θέση της εφαρμογής στο γενικότερο ιατρικό project στο οποίο μετέχει.



Σχήμα 5.1.1 Περίπτωση χρήσης εφαρμογής

Η περίπτωση που περιγράφεται στην παραπάνω περίπτωση χρήσης είναι μια κοινή λειτουργία του συστήματος που περιλαμβάνει την πλοήγηση του γραφικού και την υποβολή εντολών για εμφάνιση συγκεκριμένων δεδομένων που αφορούν κάποιον συγκεκριμένο ασθενή. Υπάρχουν κάποια δομικά στοιχεία που φαίνονται στα κουτάκια πάνω πάνω στο σχήμα. Τα στοιχεία αυτά είναι :

- Dynamic Gui : Δυναμικό γραφικό περιβάλλον. Είναι δυναμικό γιατί μπορεί να αρχικοποιηθεί για κάθε ασθενή ξεχωριστά με στοιχεία που λαμβάνει από τον Reasoning Server. Τα στοιχεία αυτά μπορεί να είναι από την μία στοιχεία του προφίλ του ασθενή (για παράδειγμα ονοματεπώνυμο, αριθμός εισαγωγής στο νοσοκομείο κλπ.), από την άλλη μπορεί να είναι στοιχεία που αφορούν τον αριθμό και ποιες εξετάσεις έχει κάνει ή πρόκειται να πραγματοποιήσει ο ασθενής.
- Jadex Platform : είναι υπεύθυνη η πλατφόρμα αυτή για την επικοινωνία της εφαρμογής μας με τον Reasoning Server ο οποίος είναι ένα εξωτερικό σύστημα. Επίσης μέσω της πλατφόρμας αυτής ανταλλάσσονται όλα τα μηνύματα agents που απαιτεί η εφαρμογή για την επικοινωνία με άλλα συστήματα.
- Sphinx Engine : Είναι μηχανή αναγνώρισης φωνής. Καθώς η εφαρμογή είναι ανοιχτή γίνεται συνέχεια αναγνώριση φωνής για να μπορέσουν να εντοπιστούν οι εντολές του χρήστη της εφαρμογής. Οι εντολές αυτές μπορεί να αφορούν την πλοήγηση του γραφικού περιβάλλοντος ή την ανάκτηση δεδομένων από τον Reasoning Server.
- Reasoning Server : Το στοιχείο αυτό δεν αποτελεί μέρος της παρούσας εφαρμογής. Είναι ένας Server με μια βάση δεδομένων που έχει όλα τα στοιχεία που αφορούν τους ασθενείς. Επίσης εδώ καθορίζονται αποφάσεις σχετικά με το υλικό που πρέπει να παρουσιαστεί από το γραφικό περιβάλλον καθώς και το ποιες αναζητήσεις δεδομένων μπορούν να γίνουν από το χρήστη της εφαρμογής. Η εφαρμογή μας για το στοιχείο αυτή περιλαμβάνει όλες τις διεπαφές διασύνδεσης μαζί του.

Αφού περιγράψαμε τώρα τα στοιχεία που αποτελούν την περίπτωση χρήσης ας δούμε αναλυτικά την αλληλουχία βημάτων που ακολουθούνται όπως αυτά φαίνονται στο παραπάνω σχήμα (αριθμημένα βέλη).

1. Initial information of the patient : Ο Reasoning Server στέλνει ένα Jadex μήνυμα με τα απαραίτητα στοιχεία του ασθενή. Τα στοιχεία αυτά είναι το ονοματεπώνυμο του ασθενή, η διεύθυνση του, ο αριθμός εισόδου του στο νοσοκομείο, οι εξετάσεις οι οποίες έχει κάνει μαζί με τα αποτελέσματα τους, το ιστορικό ασθενείας του και διάφορα άλλα στοιχεία που αφορούν τον ίδιο των ασθενή. Το μήνυμα αυτό αφού είναι Jadex μήνυμα το διαχειρίζεται η πλατφόρμα του Jadex. Άρα ο παραλήπτης είναι η πλατφόρμα.
2. Initialization with patient info: Η Jadex πλατφόρμα μεταφέρει το μήνυμα στο γραφικό περιβάλλον το οποίο κάνει parsing το μήνυμα και διαχωρίζει τις πληροφορίες του ασθενή. Με βάση τις πληροφορίες αυτές χτίζεται το γραφικό προσαρμοσμένο στα στοιχεία του ασθενή.
3. Start speaking : Ο χρήστης (γιατρός) βλέπει στο τερματικό του το γραφικό περιβάλλον που έχει σχηματιστεί με τις πληροφορίες του ασθενή. Αρχίζει λοιπόν να μιλάει και να προφέρει εντολές. Στο συγκεκριμένο σημείο του use case οι εντολές αυτές αφορούν την πλοήγηση του γραφικού περιβάλλοντος.
4. Request for Recognition: Ουσιαστικά κάθε φορά που προφέρεται μια λέξη από τον χρήστη αναλαμβάνει το sphinx engine να κάνει την αναγνώριση.
5. Recognition Result : Αφού το sphinx engine πραγματοποιήσει την αναγνώριση στέλνει το αποτέλεσμα αυτής πίσω στο γραφικό περιβάλλον.
6. Gui adjusts to recognition result : Με το αποτέλεσμα της αναγνώρισης τώρα το γραφικό περιβάλλον προσαρμόζεται ανάλογα με την εντολή που ειπώθηκε από τον χρήστη και αναγνωρίστηκε από το sphinx engine. Για παράδειγμα μπορεί να έχει γίνει μια αλλαγή καρτέλας (μεταφορά δηλαδή από την καρτέλα του προφίλ στην καρτέλα των εξετάσεων).
7. Start speaking request for patients data : Ο χρήστης αφού βλέπει το προσαρμοσμένο στην προηγούμενη εντολή που πρόφερε, προφέρει μια νέα εντολή τώρα με την οποία ζητάει κάποια επιπλέον δεδομένα για τον ασθενή. Τα επιπλέον δεδομένα αυτά μπορεί να είναι για παράδειγμα μια μεγέθυνση ακτινογραφίας ή το τελευταίο καρδιογράφημα.
8. Request for Recognition: Στο σημείο αυτό πάλι αναλαμβάνει το sphinx engine όπου πραγματοποιεί την αναγνώριση της εντολής του χρήστη.

9. Recognition Result : Το αποτέλεσμα της αναγνώρισης από το sphinx engine στέλνεται στο Jadex platform.
10. Agent message for patient data : Το Jadex platform γνωρίζοντας το αποτέλεσμα της αναγνώρισης από το sphinx engine φτιάχνει ένα agent μήνυμα που περιέχει αναφέρει τις πληροφορίες που έχει ζητήσει ο χρήστης. Το μήνυμα αυτό το στέλνει στον Reasoning Server.
11. Retrieve patient data : Λαμβάνοντας το agent μήνυμα ο Reasoning Server κάνει ανάκτηση των δεδομένων που έχει ζητήσει ο χρήστης.
12. Message with patient data : Ο Reasoning Server συντάσσει ένα agent μήνυμα με δεδομένα που έχουν ζητηθεί και το στέλνει στο Jadex platform.
13. Adjust with data of the patient : Το Jadex platform μεταφέρει στο γραφικό περιβάλλον τα δεδομένα που έχουν ζητηθεί και αυτό προσαρμόζει το περιεχόμενο του ανάλογα με τα δεδομένα αυτά.
14. Data shown through gui : Ο χρήστης βλέπει τα δεδομένα που είχε ζητήσει στο γραφικό περιβάλλον το οποίο έχει προσαρμοστεί κατάλληλα με αυτά.

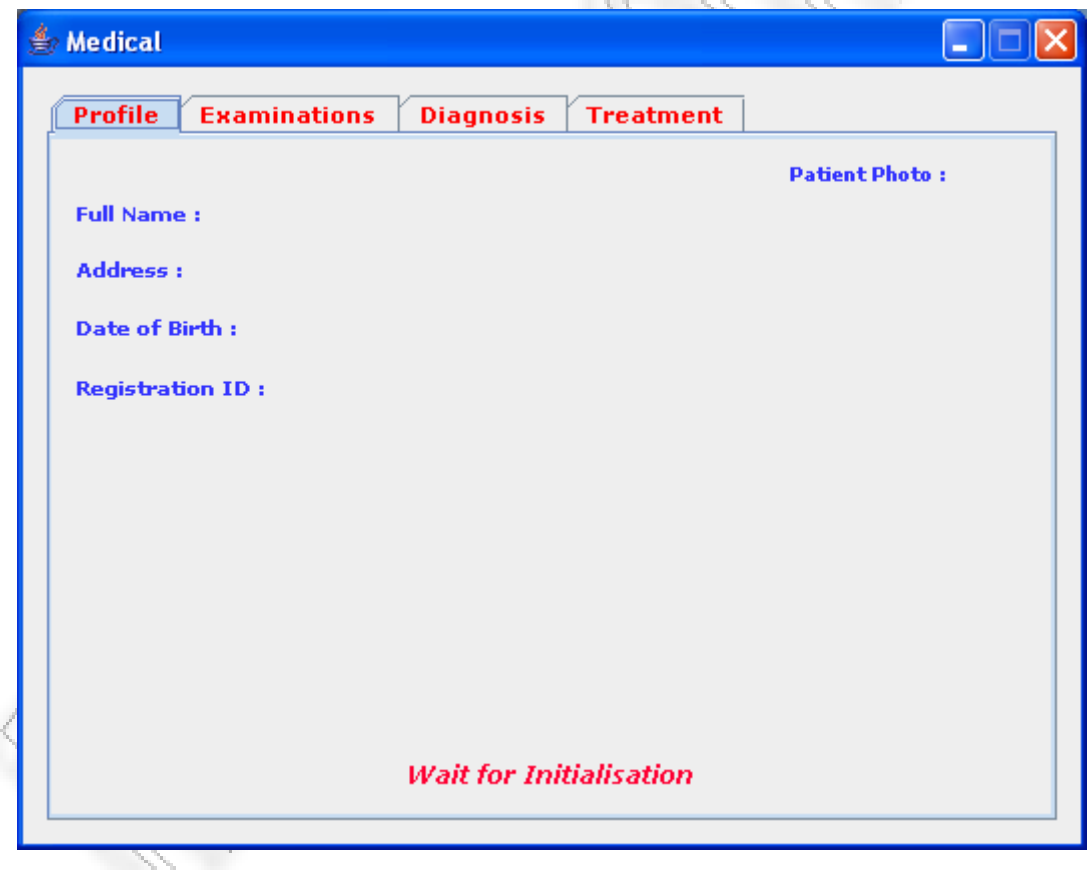
Τα βελάκια που περιγράφονται παραπάνω δεν είναι όλα μηνύματα που ανταλλάσσονται μεταξύ των διαφορετικών δομικών στοιχείων που αποτελούν την εφαρμογή. Τα μηνύματα λοιπόν τύπου agent που διαχειρίζεται (στέλνει και δέχεται) το Jadex platform είναι πραγματικά μηνύματα, ενώ όλα τα υπόλοιπα είναι εσωτερικές διαδικασίες της εφαρμογής. Οι εσωτερικές αυτές διαδικασίες είναι κλήσεις μεθόδων των διαφόρων αντικειμένων που αποτελούν τα δομικά στοιχεία. Η αναπαράσταση τόσο των μηνυμάτων όσο και των εσωτερικών διαδικασιών με βέλη στο παραπάνω σχεδιάγραμμα έγινε για την καλύτερη απεικόνιση της περίπτωσης χρήσης.
Η περίπτωση χρήσης που αναλύθηκε παραπάνω είναι γενικής μορφής και δεν περιλαμβάνει συγκεκριμένα δεδομένα. Αυτό γίνεται για να μπορέσουμε να καταλάβουμε καλύτερα την δομή της εφαρμογής και πως αυτή είναι δομημένη υπό το πρίσμα της λειτουργίας της σε γενικότερο επίπεδο. Στην συνέχεια της εργασίας θα περιγραφεί συγκεκριμένη περίπτωση χρήσης με πραγματικά δεδομένα και διάφορα screenshots του γραφικού περιβάλλοντος που εξηγούν σε μεγαλύτερο βαθμό ανάλυσης την προσαρμογή του γραφικού περιβάλλοντος καθώς και την μορφή των δεδομένων που υπάρχουν ή μπορεί να ζητηθούν από τους χρήστες.

5.2 Λειτουργία της εφαρμογής

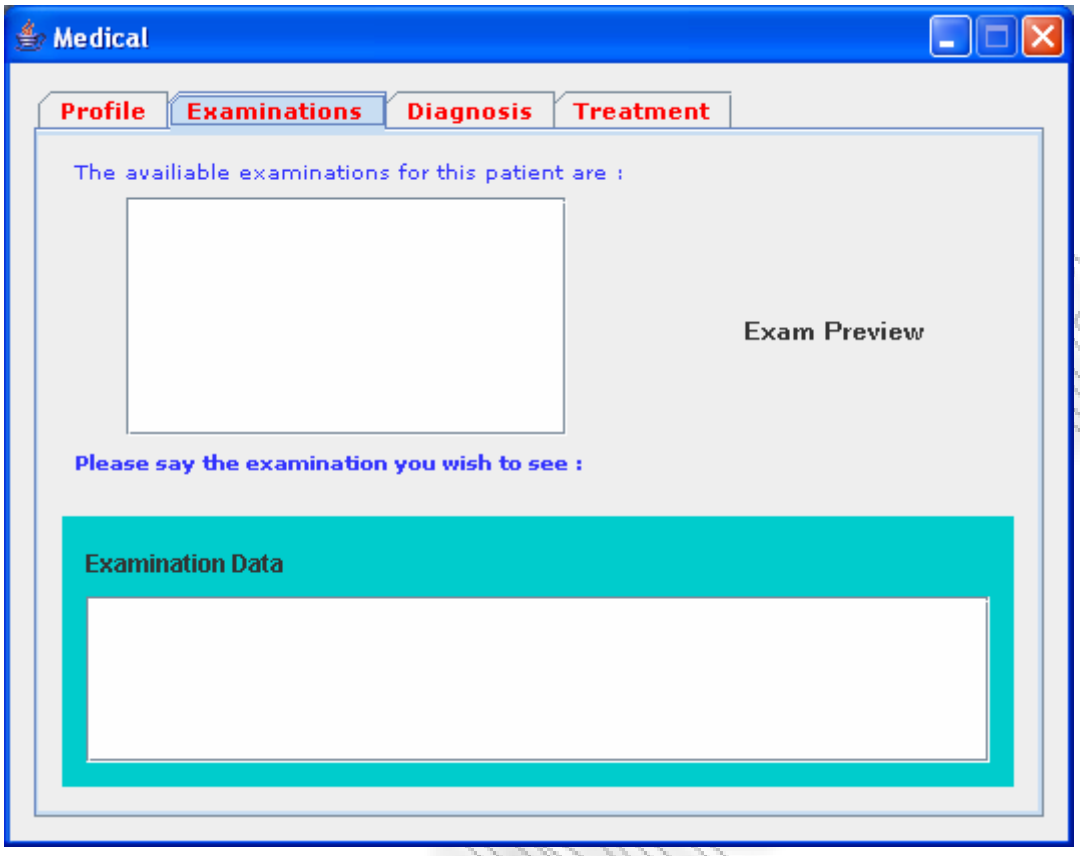
1) Ο χρήστης-γιατρός ανοίγει την εφαρμογή του. Όπως βλέπουμε και από τα παρακάτω σχήματα ανοίγει η εφαρμογή με το γραφικό να είναι φτιαγμένο αλλά να μην περιέχει κανένα στοιχείο που να αφορά τον ασθενή.

Το πρώτο σχήμα είναι η σελίδα το προφίλ και το δεύτερο είναι η σελίδα των εξετάσεων. Επίσης το ίδιο ισχύει και για το diagnosis και treatment, έχει φτιαχτεί δηλαδή το γραφικό περιβάλλον αλλά δεν περιέχουν ακόμα τα στοιχεία του ασθενή.

Πρέπει στο σημείο αυτό να αναφέρουμε ότι στην σελίδα που αφορά το προφίλ του ασθενή υπάρχει ένα label που λέει : "Wait for Initialisation" . Αυτό δηλώνει ότι η μηχανή αναγνώρισης φωνητικών εντολών δεν έχει ενεργοποιηθεί ακόμα και το γραφικό περιβάλλον περιμένει δεδομένα.



Σχήμα 5.2.1 Σελίδα προφίλ χωρίς δεδομένα ασθενή



Σχήμα 5.2.2 Σελίδα εξετάσεων χωρίς δεδομένα ασθενή

2) Εδώ βλέπουμε την εξέλιξη όταν έχουμε ξεκινήσει τον Starter agent και έχει αποσταλεί το μήνυμα. Όταν λοιπόν το μήνυμα παραληφθεί από τον Medicalagent έχουμε την ενημέρωση του γραφικού περιβάλλοντος με τα στοιχεία του ασθενή που περιέχονται στο μήνυμα.

Στο παρακάτω σχήμα βλέπουμε πως είναι πλέον η σελίδα του γραφικού περιβάλλοντος που αφορά το προφίλ του ασθενή ενημερωμένη με τα στοιχεία του ασθενή. Υπάρχουν τα στοιχεία του μηνύματος όπως είναι το πλήρες όνομα, η διεύθυνση, η ημερομηνία γέννησης, ο κωδικός εγγραφής και τέλος η διεύθυνση της φωτογραφίας του ασθενή. Όλα αυτά τα στοιχεία περιέχονται στο ontology που λήφθηκε από τον Medicalagent.

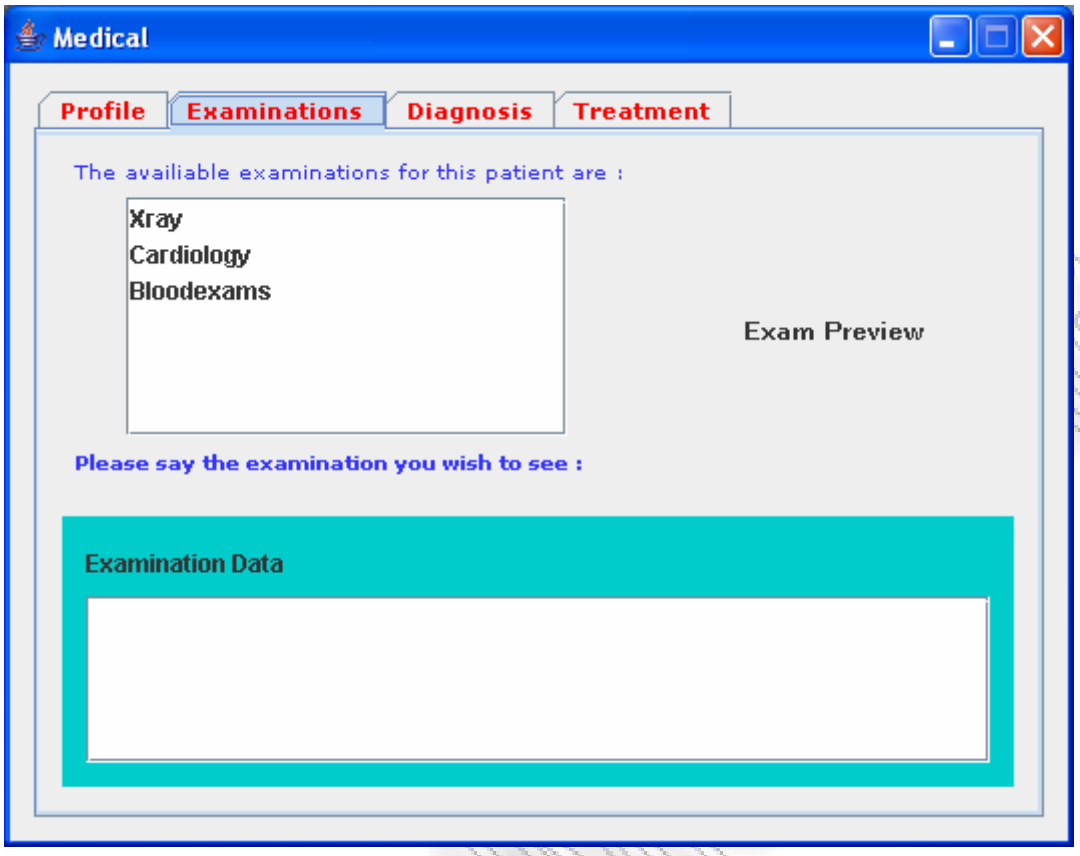
Πρέπει να σημειωθεί εδώ ότι μόλις τελειώσει το φόρτωμα των στοιχείων του ασθενή στο γραφικό περιβάλλον ενεργοποιείται η μηχανή αναγνώρισης φωνητικών εντολών, με τον τρόπο που περιγράψαμε στην υλοποίηση της εφαρμογής. Για να ξέρουμε πότε η μηχανή είναι ενεργοποιημένη και μπορεί ο χρήστης –γιατρός να προφέρει κάποια εντολή ενημερώνουμε το label που υπάρχει στην σελίδα του προφίλ και από “Wait for Initialisation” το κάνουμε “Start speaking”. Έτσι λοιπόν όλα είναι έτοιμα και

μόλις αναγνωρισθεί κάποια εντολή γίνεται η αντίστοιχη πλοήγηση στο γραφικό περιβάλλον.

Σχήμα 5.2.3 Σελίδα προφίλ με δεδομένα ασθενή

3) Αφού λοιπόν όλα είναι έτοιμα ας υποθέσουμε ότι ο χρήστης-γιατρός προφέρει την λέξη "EXAMS" γιατί θέλει να δει την σελίδα με τις εξετάσεις του ασθενή. Αυτό που πραγματοποιείται με την αναγνώριση της λέξης "EXAM" είναι να προσαρμοστεί το γραφικό περιβάλλον στην σελίδα των εξετάσεων.

Η σελίδα αυτή φαίνεται στο παρακάτω σχήμα:



Σχήμα 5.2.4 Σελίδα εξετάσεων με δεδομένα ασθενή

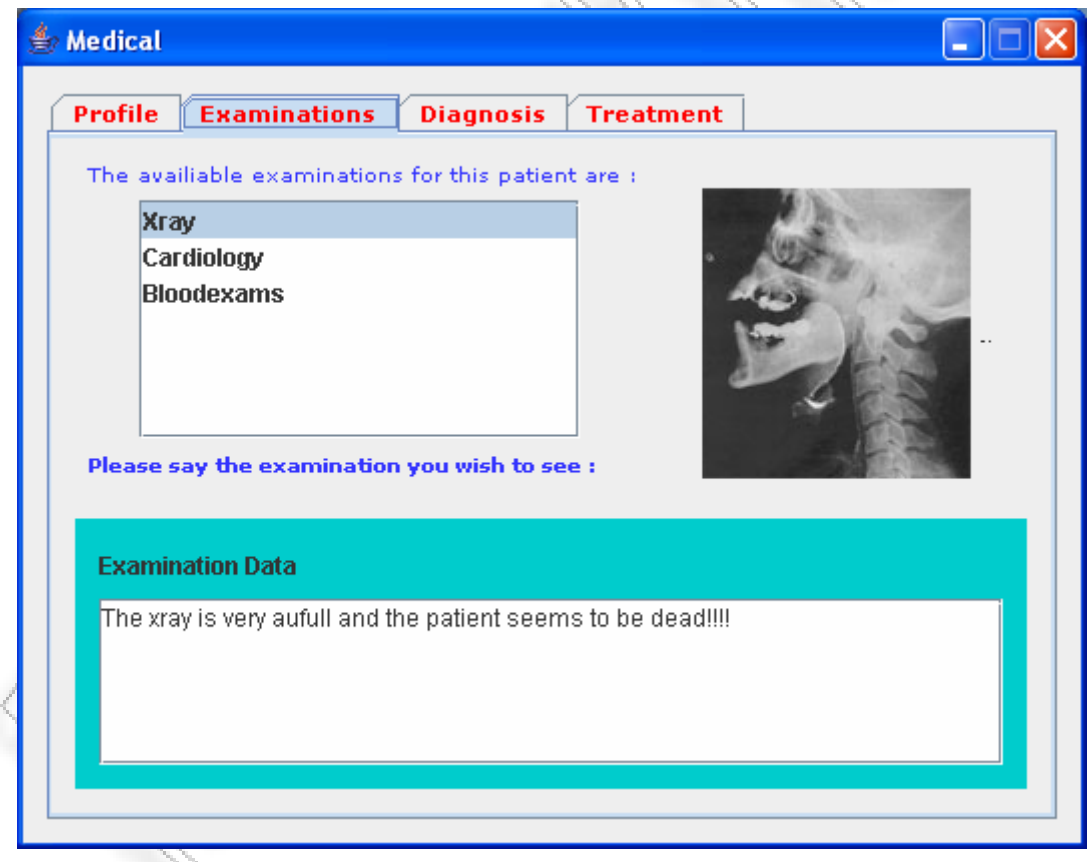
Όπως μπορούμε να δούμε υπάρχουν κάποιες εξετάσεις που αφορούν τον συγκεκριμένο ασθενή και αυτές είναι οι "XRAY", "CARDIOLOGY" και "BLOODEXAMS". Οι εξετάσεις αυτές προέρχονται από το ontology του μηνύματος που έλαβε ο Medicalagent.

Εδώ πρέπει να τονίσουμε ότι δεν μας ενδιαφέρει η σειρά που είναι δομημένες οι εξετάσεις μέσα στο ontology που λαμβάνουμε αλλά θα πρέπει να είμαστε προετοιμασμένοι για το ποιο είναι το όνομα της κάθε εξέτασης. Αυτό σημαίνει ότι αν θέλουμε μελλοντικά να προσθέσουμε μια εξέταση καινούργια θα πρέπει εκτός από το να φτιάξουμε το ontology κατάλληλα, να φτιάξουμε και μια σειρά από άλλα πράγματα όπως είναι το λεξικό, η γραμματική σύνταξη αλλά και να φτιαχτεί η αλληλουχία ενεργειών που θα γίνουν όταν προφερθεί η λέξη που αντιστοιχεί στο όνομα της εξέτασης αυτής. Όταν είμαστε όμως προετοιμασμένοι δεν μας ενδιαφέρει πόσες και με πια σειρά θα μας έρθουν οι εξετάσεις από το ontology που λαμβάνουμε καθώς η υλοποίηση έχει γίνει με πλήρως δυναμικό τρόπο.

Όπως παρατηρούμε από το παραπάνω σχήμα εκτός από το όνομα υπάρχουν και κάποια χαρακτηριστικά που αφορούν την κάθε εξέταση. Αυτά είναι το "Exam

Preview" όπου εμφανίζεται η φωτογραφία της εξέτασης εφόσον υπάρχει και ένα πεδίο που ονομάζεται "Examination Data" και αφορά κάποια δεδομένα σε μορφή κειμένου που υπάρχουν για την κάθε εξέταση.

5) Ας δούμε όμως τώρα τι γίνεται όταν επιλέγουμε μια εξέταση. Ας υποθέσουμε λοιπόν ότι ο χρήστης-γιατρός θέλει να επιλέξει την ακτινογραφία από την λίστα εξετάσεων που βλέπει από το παραπάνω σχήμα. Για να γίνει αυτό θα πρέπει να προφέρει την λέξη "XRAY". Με το που ειπωθεί λοιπόν αυτή η λέξη και αναγνωρισθεί σωστά αυτό που εμφανίζεται στο γραφικό περιβάλλον φαίνεται στο παρακάτω σχήμα:

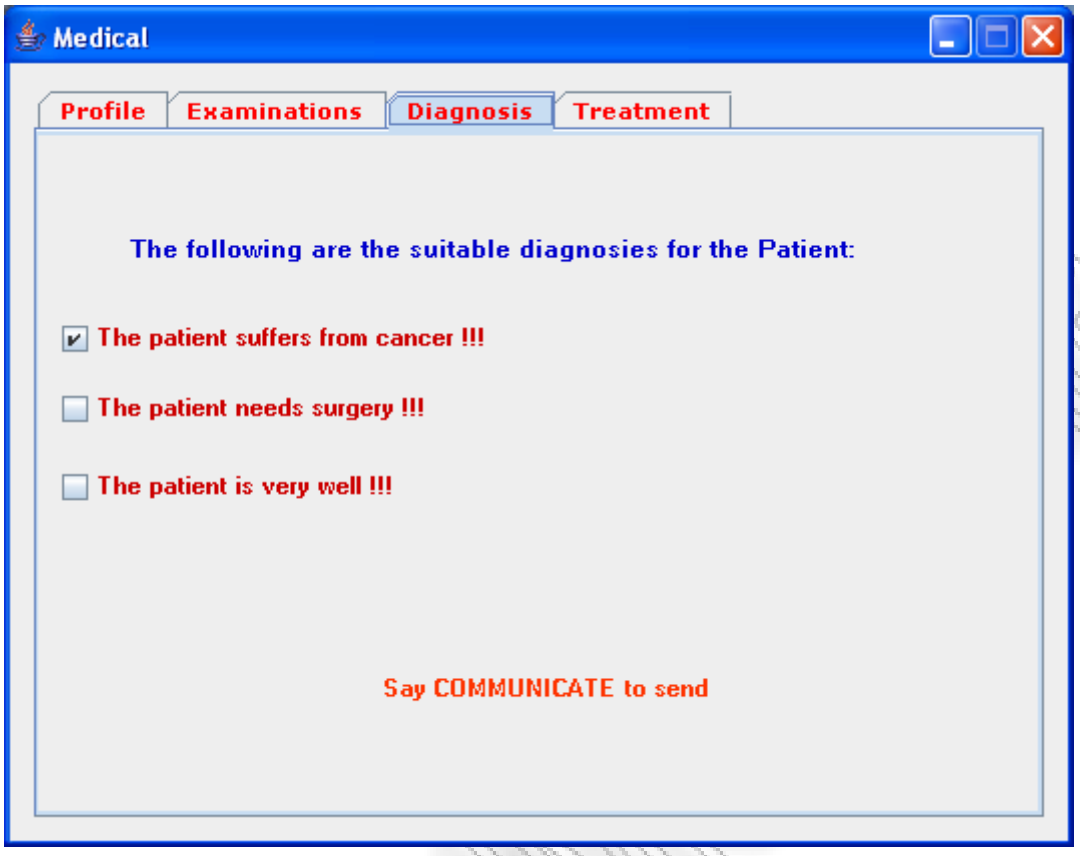


Σχήμα 5.2.5 Σελίδα εξετάσεων με ενεργοποιημένη την ακτινογραφία

Παρατηρούμε λοιπόν ότι εδώ έχουν γεμίσει όλα τα πεδία που αφορούν την ακτινογραφία. Δηλαδή υπάρχει η εικόνα της καθώς και η περιγραφή της που είναι το κείμενο που βρίσκεται στο πεδίο Examinations data.

Φυσικά ο χρήστης-γιατρός έχει ανά πάσα στιγμή την δυνατότητα να μεταβεί σε κάποια άλλη κεντρική σελίδα του γραφικού περιβάλλοντος. Οι κεντρικές σελίδες είναι οι Profile, Examinations, Diagnosis και Treatment και οι μεταβάσεις γίνονται με την προφορά των λέξεων “PROFILE”, “EXAMS”, “DIAGNOSIS” και “TREATMENT” αντίστοιχα.

6) Ας υποθέσουμε λοιπόν ότι στην συνέχεια ο χρήστης-γιατρός θέλει να δει και να επιλέξει μια διάγνωση από την σελίδα των διαγνώσεων. Για να γίνει αυτό θα πρέπει να προφέρει την λέξη “DIAGNOSIS” και όταν αυτή αναγνωρισθεί σωστά το γραφικό περιβάλλον θα προσαρμοστεί κατάλληλα και θα δείξει την σελίδα με τις διαγνώσεις. Η σελίδα με τις διαγνώσεις έχει γεμίσει με τις διαγνώσεις που υπάρχουν στο ontology που έχει ληφθεί. Έτσι και οι διαγνώσεις είναι τελείως δυναμικές και γεμίζουν με στοιχεία που προέρχονται από το εισερχόμενο μήνυμα.
Κάθε διάγνωση γεμίζει ένα checkbox το οποίο και μπορεί να τσεκαριστεί με κατάλληλη φωνητική εντολή. Στην συγκεκριμένη λοιπόν εφαρμογή έχουμε φτιάξει τρεις πιθανές διαγνώσεις που ενεργοποιούνται με τις φωνητικές εντολές “FIRSTCHOICE”, “SECONDCHOICE” και “THIRDCHOICE” αντίστοιχα. Όταν τσεκάρουμε κάποια διάγνωση με μια από αυτές τις εντολές έχουμε την δυνατότητα να την ξετσεκάρουμε προφέροντας την ίδια φωνητική εντολή που χρησιμοποιήσαμε για το τσεκάρισμα της.
Ανά πάσα στιγμή όταν βρισκόμαστε στην σελίδα των διαγνώσεων μπορούμε να προφέρουμε την λέξη “COMMUNICATE” και αμέσως δημιουργείται ένα μήνυμα agent και αποστέλλεται στον Receiver agent που περιμένει για κάποιο τέτοιο μήνυμα μετά την ενεργοποίηση του. Το μήνυμα αυτό θα περιέχει την ή τις διαγνώσεις που έχει τσεκάρει ο χρήστης-γιατρός πρωτού να προφέρει την εντολή “COMMUNICATE”.
Μερικά από αυτά που περιγράφουμε σχετικά με τις διαγνώσεις φαίνονται στο παρακάτω σχήμα όπου και έχει επιλεγεί η πρώτη διάγνωση με την χρήση της εντολής “FIRSTCHOICE” :



Σχήμα 5.2.6 Σελίδα διαγνώσεων με ενεργοποιημένη την πρώτη διάγνωση

7) Ας δούμε στο σημείο αυτό πως λειτουργεί και η σελίδα των θεραπειών. Για να μπορέσουμε να μπούμε στην σελίδα αυτή θα πρέπει να αναγνωριστεί η εντολή "TREATMENT". Την εντολή αυτήν μπορεί ο χρήστης-γιατρός να την προφέρει ανά πάσα στιγμή. Η σελίδα των θεραπειών δουλεύει με παρόμοιο τρόπο με την σελίδα των διαγνώσεων που περιγράψαμε ακριβώς παραπάνω.

Έτσι λοιπόν υπάρχουν και εδώ τρεις θεραπείες που γεμίζουν με βάση τα δεδομένα που παίρνουμε από το ontology του εισερχόμενου μηνύματος agent. Οι τρεις αυτές θεραπείες μπορούν να επιλεγούν με τις εντολές "FIRSTCHOICE", "SECONDCHOICE" και "THIRDCHOICE" όπως και συμβαίνει και στις διαγνώσεις.

Στην σελίδα των θεραπειών υπάρχει μια μικρή διαφοροποίηση από την σελίδα των διαγνώσεων. Εδώ με κάθε τσεκάρισμα κάποιας θεραπείας γεμίζει και ένα πεδίο που βρίσκεται στο κάτω μέρος της σελίδας με κάποια σχόλια που αφορούν την θεραπεία που έχει επιλεχτεί.

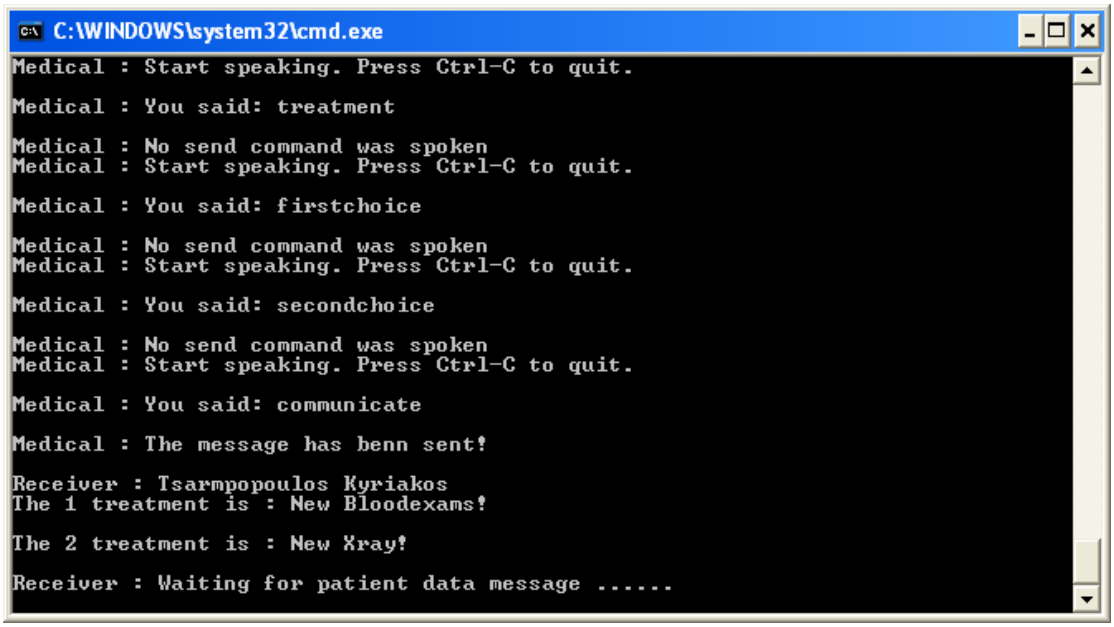
Στο παρακάτω σχήμα που ακολουθεί φαίνεται η σελίδα των διαγνώσεων με επιλεγμένες τις δύο πρώτες θεραπείες. Η επιλογή αυτή έχει γίνει με την προφορά και αναγνώριση των λέξεων “FIRSTCHOICE” και “SECONDCHOICE” αντίστοιχα :

Σχήμα 5.2.7 Σελίδα θεραπειών με ενεργοποιημένες τις 2 πρώτες θεραπείες

Στην συνέχεια αναφέρουμε τι ακριβώς γίνεται όταν με τις δύο αυτές θεραπείες τσεκαρισμένες ο χρήστης-γιατρός προφέρει τη λέξη “COMMUNICATE”.

8)Έτσι λοιπόν όταν έχουμε τσεκάρει τις δυο πρώτες θεραπείες και προφέρουμε την εντολή “COMMUNICATE” συντάσσουμε και αποστέλλουμε ένα agent μήνυμα με τον τρόπο που περιγράψαμε στην υλοποίηση της εφαρμογής. Ο Receiver agent που περιμένει να λάβει αυτό το μήνυμα, όταν το λάβει το εκτυπώνει για να επιβεβαιωθεί ότι η όλη διαδικασία δουλεύει σωστά.

Αυτά φαίνονται και στο παρακάτω σχήμα όπου και υπάρχει μια σειρά από αναγνωρισμένες φωνητικές εντολές και πως αυτές επιδρούν στην εφαρμογή. Η φωτογραφία αυτή περιγράφει την διαδικασία που περιγράψαμε παραπάνω:



Σχήμα 5.2.8 Εκτύπωση διαδικασίας αποστολής των επιλεγμένων θεραπειών

Βιβλιογραφία και άλλες πηγές

- K. F. Lee, H. W. Hon, and R. Reddy, "*An overview of the SPHINX speech recognition system*," *IEEE Transactions on Acoustics, Speech and Signal Processing, Jan. 1990.*
- X. Huang, F. Alleva, H. W. Hon, M. Y. Hwang, and R. Rosenfeld, "The SPHINX-II speech recognition system: an overview," *Computer Speech and Language* 1993.
- Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel , "*Sphinx-4: A Flexible Open Source Framework for Speech Recognition*", SMLI TR-2004-139 November 2004
- L. Braubach, A.Pokahr, W. Lamersdorf , "*Jadex: A Short Overview*", Distributed and Information Systems Group, University of Hamburg, Germany
- A. Pokahr, L. Braubach, and W. Lamersdorf. "*Jadex: Implementing a BDI Infrastructure for JADE Agents*.", 2003.
- Μέλος του Jadex BDI Forum με ενεργή συμμετοχή και ανταλλαγή απόψεων σχετικά με την υλοποίηση της εφαρμογής. [**Online**] - https://sourceforge.net/forum/?group_id=80240
- Μέλος του Cmu Forum με ενεργή συμμετοχή και ανταλλαγή απόψεων σχετικά με την υλοποίηση της εφαρμογής. [**Online**] - https://sourceforge.net/forum/?group_id=1904
- Documentation, implementation examples, installation guide for Sphinx-4 [**Online**] - http://cmusphinx.sourceforge.net/html/cmusphinx.php
- Documentation, implementation examples, installation guide for Jadex [**Online**] - http://vsis-www.informatik.uni-hamburg.de/projects/jadex
- Οδηγός χρησιμοποίησης εργαλείου εκπαίδευσης για το Sphinx-4. [**Online**] - http://www.speech.cs.cmu.edu/sphinxman/scriptman1.html
- Carnegie Mellon University. CMU pronouncing dictionary. [**Online**] - http://www.speech.cs.cmu.edu/cgi-bin/cmudict

ΠΑΡΑΡΤΗΜΑ Α

Περιεχόμενα Συνοδευτικού CD

Το συνοδευτικό cd αποτελείται από τέσσερις φακέλους. Η παρουσίαση λοιπόν των περιεχομένων αναφέρεται με βάση τους φακέλους αυτούς. Επισημαίνεται στο σημείο αυτό ότι όλα τα προγράμματα που υπάρχουν εδώ παρέχονται δωρεάν μέσω του διαδικτύου.

Application

- **Final Demo.rar :** Είναι ένα συμπιεσμένο αρχείο το οποίο αν γίνει εξαγωγή περιέχει όλους τους απαραίτητους φακέλους και αρχεία για την εκτέλεση της τελικής έκδοσης της εφαρμογής. Μέσα στον εξαγόμενο φάκελο υπάρχει ένα αρχείο **Run instructions.txt** το οποίο και αναφέρει τα βήματα που πρέπει να ακολουθηθούν για να τρέξει η εφαρμογή.

Integrated Application

- **WinHPN.rar :** Είναι ένα συμπιεσμένο αρχείο το οποίο αν γίνει εξαγωγή περιέχει όλους τους απαραίτητους φακέλους και αρχεία για την εκτέλεση του συνολικού project που μετέχει η εφαρμογή. Το γενικότερο αυτό ιατρικό project αποτελείται από τρεις εφαρμογές όπως και έχει περιγραφεί και στο κομμάτι της εργασίας για το γενικότερο περιβάλλον λειτουργίας της εφαρμογής μας. Εδώ είναι η υλοποίηση αυτή.

Programs

- **additional acoustic models for sphinx 4 :** Εδώ περιέχονται 3 εναλλακτικά ακουστικά μοντέλα που μπορούν να χρησιμοποιηθούν από το sphinx-4 σε μελλοντικές εφαρμογές.
- **ant :** Εδώ περιέχεται το πρόγραμμα ant που όπως έχουμε αναφέρει μπορεί να κάνει build κομμάτια κώδικα και χρησιμοποιείται για την δημιουργία των εκτελέσιμων αρχείων του sphinx-4
- **jadex src :** Εδώ είναι η έκδοση του jadex που χρησιμοποιείται από την εφαρμογή μας. Εμπεριέχεται ο πηγαίος κώδικας.

- **Java 1.5.0.1(src) :** Εμπεριέχεται το Java Development Kit 1.5.0.1 που χρησιμοποιείται από την εφαρμογή μας .(Έκδοση Windows).
- **net beans 5.0 :** Εδώ εμπεριέχονται δύο φάκελοι. Ο πρώτος **(net beans 5.0 (src))** περιέχει το εκτελέσιμο αρχείο για την εγκατάσταση του Net beans 5.0 που έχει χρησιμοποιηθεί για την δημιουργία του γραφικού περιβάλλοντος της εφαρμογής μας. Ο δεύτερος φάκελος (**net beans project files**) περιέχει το project του net beans που χρησιμοποιήθηκε για την δημιουργία του γραφικού περιβάλλοντος.
- **protege files :** Εδώ περιέχονται δύο φάκελοι. Ο πρώτος (**protege 3.1 (src)**) περιέχει τα αρχεία εγκατάστασης για το πρόγραμμα Protégé 3.1 που χρησιμοποιήθηκε από την εφαρμογή μας για την δημιουργία των ontologies. Επίσης υπάρχει και ένα plugin που απαιτείται για την υποστήριξη του jadex (**jadex-beanynizer-0.941.zip**). Ο δεύτερος φάκελος (**protege application project**) περιέχει το project που δημιουργήθηκε για την εφαρμογή μας με τα ontologies καθώς και τα αποτελέσματα (java κλάσεις) που παράγοντα από αυτό.
- **Sphinx-4 (src) :** Εδώ υπάρχει ο πηγαίος κώδικας της έκδοσης του sphinx-4 που χρησιμοποιήθηκε από την εφαρμογή μας.
- **SphinxTrain :** Εδώ υπάρχει το εργαλείο εκπαίδευσης και δημιουργίας ακουστικών μοντέλων για το sphinx-4. Υπάρχει ο κώδικας σε έναν φάκελο (**SpinxTrain(src)**) καθώς και ένα συμπιεσμένο αρχείο (**SphinxTrain(compiled with .net).rar**) που περιέχει και τα εκτελέσιμα αρχεία που έχουν δημιουργηθεί με compile χρησιμοποιώντας το visual studio 2005.

Documentation

- **Πτυχιακή Εργασία (Τσαρμπόπουλος Κυριάκος,2006).doc :** Είναι το παρόν έγγραφο σε windows word format.
- **Πτυχιακή Εργασία (Τσαρμπόπουλος Κυριάκος,2006).pdf :** Είναι το παρόν έγγραφο σε adobe acrobat format.
- **Παρουσίαση (Τσαρμπόπουλος Κυριάκος,2006).ppt :** Είναι το αρχείο που περιέχει την παρουσίαση της παρούσης πτυχιακής εργασίας.