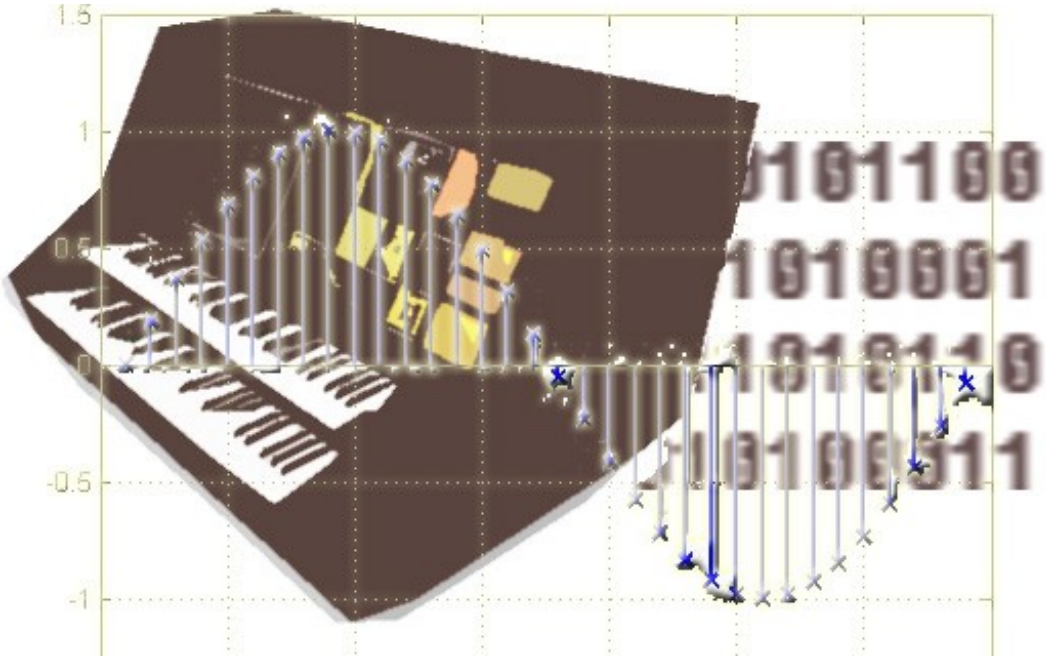
Α.Τ.Ε.Ι. ΚΡΗΤΗΣ – ΠΑΡΑΡΤΗΜΑ ΡΕΘΥΜΝΟΥ
ΤΜΗΜΑ ΜΟΥΣΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ & ΑΚΟΥΣΤΙΚΗΣ

Πτυχιακή Εργασία

Προγραμματιστικό περιβάλλον διάδρασης μέσω μηχανικών αισθητήρων και δεδομένων MIDI



Ζαβιτσάνος Φώτης-Δημήτρης (Α.Μ. 375)

Επίβλεψη: Αλεξανδράκη Χρυσούλα

- Ρέθυμνο 2009 -

Περίληψη

Η παρούσα εργασία αφορά την ανάπτυξη μιας προγραμματιστικής βιβλιοθήκης (API) και μιας συνοδευτικής εφαρμογής. Η προγραμματιστική αυτή βιβλιοθήκη στοχεύει στην υλοποίηση εφαρμογών που αξιοποιούν δεδομένα πρόσκτησης χειρονομιών τα οποία παράγονται σε πραγματικό χρόνο μέσω μηχανικών αισθητήρων. Η δε συνοδευτική εφαρμογή αποτελεί παράδειγμα χρήσης της προγραμματιστικής βιβλιοθήκης το οποίο αξιοποιεί τα δεδομένα πρόσκτησης χειρονομιών για τη δημιουργία και την αλληλεπίδραση με αλγορίθμους σύνθεσης ήχου. Στα πλαίσια του γραπτού κειμένου της εργασίας, παρουσιάζονται αρχικά οι θεμελιώδης αρχές καθώς και η σχετιζόμενη έρευνα στη συναφή ερευνητική περιοχή των "Ψηφιακών Μουσικών Οργάνων" (Digital Music Instruments), η οποία και ενδιαφέρει στην εργασία αυτή. Στη συνέχεια, περιγράφεται η υλοποίηση και ο τρόπος χρήσης του API και της ενδεικτικής εφαρμογής (demo). Τέλος, επιχειρείται μια αξιολόγηση του τελικού αποτελέσματος σε σύγκριση με το προσδοκώμενο.

Abstract

The current work is concerned with the development of an Application Programming Interface (API) and a demo application. This API has been developed with the purpose of allowing the implementation of applications that capture and manipulate gestural data acquired through user interactions with mechanical sensors. The demo application demonstrates an examplary application in which such gestural data are used in order to create and modify sound synthesis algorithms in real time. This document initially presents the theoretical background of the related research domain, which is known as "Digital Music Instruments". Then the implementation of the API and the demo application is presented. Finally, the document cncludes by attempting some concluding remarks on the evaluation of the implemented software.

Πίνακας περιεχομένων

1 Εισαγωγή

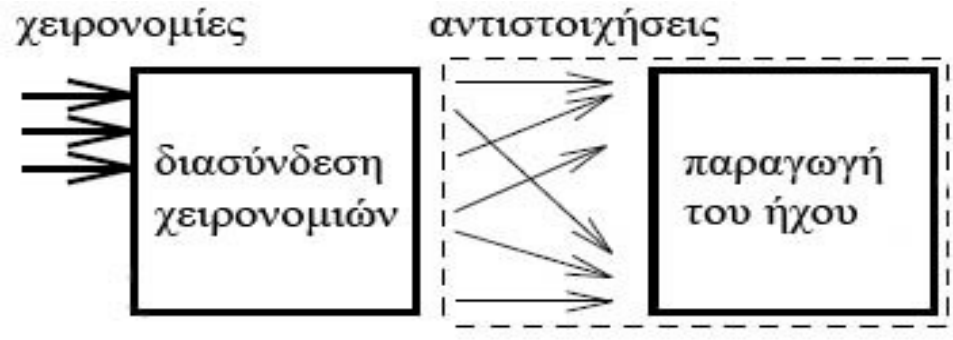
1.1 Στόχος Πτυχιακής Εργασίας

Στόχος της πτυχιακής εργασίας είναι η ανάπτυξη ενός προγραμματιστικού εργαλείου το οποίο επιτρέπει την αλληλεπίδραση χρήστη – υπολογιστή μέσω χειρονομιών (gesture interaction). Το εργαλείο αυτό απευθύνεται τόσο στο μέσο χρήστη, όσο και σε προγραμματιστές, καθώς, πέραν από μία αυτόνομη εκτελέσιμη εφαρμογή παρέχει και μία προγραμματιστική βιβλιοθήκη (Application Programming Interface – API), υλοποιημένη σε Java. Η βιβλιοθήκη αυτή έχει σαν στόχο την υλοποίηση διαδραστικών ηχητικών εγκαταστάσεων (interactive sound installations) αξιοποιώντας δεδομένα που προέρχονται από την αλληλεπίδραση του χρήστη με ένα σύνολο από αισθητήρες. Το αποτέλεσμα της αλληλεπίδρασης αυτής μπορεί να προσδιοριστεί μέσα από τη βιβλιοθήκη μέσω της δημιουργίας αντιστοιχίσεων δράσης χρήστη-ανάδρασης υπολογιστή.

Οι αντιστοιχήσεις αυτές, γνωστές ως mappings είναι το βασικότερο πεδίο έρευνας της επιστημονικής περιοχής που ασχολείται με αλληλεπίδραση χρήστη υπολογιστή με χειρονομίες (gesture-computer interaction). Η εργασία αυτή δεν προτείνει καινοτόμες ιδέες στο επίπεδο της αντιστοίχησης των εξόδων των αισθητήρων με τις εισόδους της παραγωγής του ήχου. Ωστόσο, δίνει έμφαση στη δυνατότητα προγραμματισμού των αντιστοιχιών αυτών κατά τη βούληση του χρήστη προκειμένου να αξιοποιήσει τα μηνύματα που προέρχονται από αισθητήρες σε ποικίλους αλγορίθμους σύνθεσης ήχου και να διαμορφώσει εύκολα και γρήγορα νέες προτάσεις ηχητικών εγκαταστάσεων.

1.2 Ψηφιακά Μουσικά Όργανα

Ο όρος "ψηφιακά μουσικά όργανα" αναφέρεται σε μουσικά όργανα τα οποία προσφέρουν διαφορετικούς τρόπους αλληλεπίδρασης με το μουσικό εκτελεστή σε σχέση με τα συμβατικά (ακουστικά ή ηλεκτρικά) μουσικά όργανα.



Εικόνα 1.2.1 Σύνδεση της μονάδας ελέγχου χειρονομιών με την μονάδα παραγωγής ήχου, μέσω αντιστοιχίσεων.

Στα τελευταία, η οποιαδήποτε σωματική επαφή ή χειρονομία του εκτελεστή συνδέεται άμεσα με την πρωταρχική παραγωγή του ήχου. Αντίθετα, τα ψηφιακά μουσικά όργανα προσφέρουν την δυνατότητα διαχωρισμού της διασύνδεσης εκτελεστή – οργάνου μέσω χειρονομιών με την μονάδα παραγωγής του ήχου, σε δύο εντελώς ανεξάρτητες μονάδες, που θα σχετίζονται μέσω ειδικών στρατηγικών αντιστοίχησης. Η ιδέα αυτή απεικονίζεται σαφέστερα στην Εικόνα 1.2.1, όπου τα βέλη συμβολίζουν την μονοσήμαντη αιτιακή σχέση μεταξύ των χειρονομιών του εκτελεστή και των παραμέτρων της παραγωγής του ήχου. Στο σημείο αυτό θα πρέπει να σημειωθεί ότι ορισμένοι όροι που χρησιμοποιούμε χρειάζονται διασαφήνιση, όπως ο όρος χειρονομία (gesture), που μέχρι εδώ χρησιμοποιήθηκε κατά κόρον, αλλά και ο όρος της αντιστοίχησης (mapping). Στα επόμενα κεφάλαια της εισαγωγής θα δοθούν ακριβείς ορισμοί, για όλους τους όρους που θα βοηθήσουν στην καλύτερη κατανόηση του κειμένου.

Ο όρος διασύνδεση χειρονομιών ή ελεγκτής χειρονομιών (gestural controller), μπορεί εδώ να οριστεί ως το κομμάτι εισόδου του ψηφιακού μουσικού οργάνου, όπου λαμβάνει χώρα σωματική αλληλεπίδραση με τον εκτελεστή. Αντίστοιχα, η μονάδα παραγωγής του ήχου μπορεί να ιδωθεί ως ο αλγόριθμος σύνθεσης. Ο ελεγκτής χειρονομιών αποτελείται από έναν ή περισσότερους αισθητήρες, συναρμολογημένους σαν μέρος μιας ξεχωριστής συσκευής.

Ο ελεγκτής χειρονομιών είναι το μέρος του ψηφιακού μουσικού οργάνου όπου λαμβάνει χώρα η σωματική αλληλεπίδραση. Ο όρος σωματική αλληλεπίδραση εννοεί εδώ τις ενέργειες του εκτελεστή, είτε είναι κινήσεις του σώματος, κινήσεις με τα χέρια ή χειρισμός αντικειμένων. Λόγω της ευρείας γκάμας των ανθρώπινων χειρονομιών που μπορούν να προσκτηθούν από τον ελεγκτή και ανάλογα με το περιβάλλον στο οποίο θα χρησιμοποιηθεί, ο σχεδιασμός του ελεγκτή μπορεί να διαφέρει σε κάθε περίπτωση. Προκειμένου να αναλυθούν οι διάφορες πιθανότητες, προτείνουμε μια ταξινόμηση των ήδη υπαρχόντων ελεγκτών, όπως παρακάτω:

- *Ελεγκτές σε μορφή οργάνου (Instrument-like Controllers)*: όπου ο σχεδιασμός της συσκευής εισόδου τείνει να αναπαράγει κάθε χαρακτηριστικό ενός ακουστικού οργάνου, με κάθε λεπτομέρεια. Πολλά παραδείγματα μπορούν να αναφερθούν, όπως ηλεκτρονικά keyboards, κιθάρες, σαξόφωνα, μαρίμπας και άλλα.

  Μία υποδιαίρεση αυτή της κατηγορίας ελεγκτών θα μπορούσαν να αποτελούν οι ελεγκτές που είναι εμπνευσμένοι από όργανα, οι οποίοι, παρά το γεγονός ότι ως επί το πλείστον αντλούν έμπνευση από το σχέδιο ήδη υπαρχόντων οργάνων, προορίζονται για διαφορετική χρήση. Ένα τέτοιο παράδειγμα είναι το βιολί SuperPolm που αναπτύχθηκε από τους S. Goto, A. Terrier και P. Pierrot, όπου η συσκευή εισόδου βασίζεται με κάποια αοριστία σε ένα σχήμα βιολιού, αλλά χρησιμοποιείται σαν μια γενική συσκευή για τον έλεγχο μιας μικροδομικής (granular) σύνθεσης.
- *Επαυξημένα όργανα (Augmented Instruments)*, που αποκαλούνται επίσης *Υβριδικοί Ελεγκτές (Hybrid Controllers)*, που είναι όργανα επιπλέον εξοπλισμένα με αισθητήρες. Εμπορικά επαυξημένα όργανα έχουν χρησιμοποιηθεί σε μουσικά κομμάτια από τον J.-C. Risset.
- *Εναλλασσόμενοι ελεγκτές (Alternate Controllers)*, η σχεδίαση των οποίων δεν ακολουθεί την σχεδίαση κάποιου οργάνου. Μερικά παραδείγματα αποτελούν τα Χέρια (Hands), τα πλακίδια γραφικής σχεδίασης (graphic drawing tablets) και άλλα. Έχει επίσης προταθεί ένας ελεγκτής χειρονομιών που θα χρησιμοποιεί το σχήμα της στοματικής κοιλότητας (Wanderley M., 2000)

Σύμφωνα με τα παραπάνω, εύκολα γίνεται αντιληπτό το γιατί ο παραπάνω διαχωρισμός που επιτυγχάνουν τα ψηφιακά μουσικά όργανα είναι αδύνατος στην περίπτωση των παραδοσιακών

ακουστικών οργάνων, όπου η διασύνδεση μέσω χειρονομιών αποτελεί επίσης μέρος της μονάδας παραγωγής του ήχου. Για παράδειγμα, σε ένα κλαρινέτο, η γλωττίδα, τα κλειδιά, οι οπές κτλ είναι την ίδια στιγμή και η διασύνδεση των χειρονομιών (όπου ο εκτελεστής αλληλεπιδρά με το όργανο) και τα στοιχεία που είναι υπεύθυνα για την παραγωγή του ήχου. Η ιδέα των ψηφιακών μουσικών οργάνων είναι ανάλογη με μια διενέργεια "διάσπασης" του κλαρινέτου σε δύο λειτουργίες (διασύνδεση χειρονομιών και γεννήτρια ήχων) και χρήσης αυτών ανεξάρτητα (Wanderley M., 2000).

Οι προοπτικές αυτού του διαχωρισμού είναι τεράστιες. Μπορούν να σχεδιαστούν ψηφιακά μουσικά όργανα που θα μπορούν να χρησιμοποιηθούν από μη ειδικούς εκτελεστές, ή από εκτελεστές ειδικευμένους σε κάποια άλλη μορφή τέχνης. Για παράδειγμα, με τους κατάλληλους αισθητήρες και αντιστοιχήσεις, ένας χορευτής θα μπορούσε να ελέγξει την παραγωγή του ήχου με τις κινήσεις που θα πραγματοποιεί.

1.3 Μηχανικοί Αισθητήρες και Αναγνώριση Χειρονομιών

Ο όρος χειρονομία, σαν η πλησιέστερη μετάφραση του αγγλικού όρου gesture, ερμηνεύεται στη σχετική βιβλιογραφία με δύο τρόπους: α) Σαν όρος που περιλαμβάνει τις κινήσεις του χεριού, είτε στην προσπάθεια χειρισμού κάποιου αντικειμένου (manipulation), είτε με τις παλάμες του χεριού άδειες (empty-handed), και β) Σαν όρος που πιθανό να εμπερικλείει τα προηγούμενα, αλλά και όλες τις υπόλοιπες κινήσεις του σώματος του εκτελεστή. Στην εργασία αυτή, θα υιοθετηθεί ο δεύτερος ορισμός, έχοντας γνώση ότι ο όρος "ενέργεια του εκτελεστή" θα ήταν πιο δόκιμος. Η επιλογή αυτή γίνεται προκειμένου να διατηρηθεί η ομαλότητα σε σχέση με την υπάρχουσα βιβλιογραφία.

Προκειμένου ο εκτελεστής να μπορέσει να ελέγξει την μονάδα παραγωγής του ήχου μέσω των χειρονομιών του, θα πρέπει η μονάδα διασύνδεσης να διαθέτει ένα σύστημα πρόσκτησης που θα συλλάβει τα χαρακτηριστικά των χειρονομιών για περαιτέρω χρήση στο αλληλεπιδραστικό σύστημα. Αυτή η πρόσκτηση μπορεί να γίνει με τρεις τρόπους:

- *Άμεση πρόσκτηση (Direct acquisition)*, όπου ένας ή παραπάνω αισθητήρες χρησιμοποιούνται για να ελεγχθούν οι ενέργειες του εκτελεστή. Τα σήματα από τους αισθητήρες δείχνουν μεμονωμένα βασικά χαρακτηριστικά μιας χειρονομίας: πίεση, μετατόπιση, επιτάχυνση κτλ. Συνήθως, χρησιμοποιείται διαφορετικός αισθητήρας για τη λήψη κάθε μιας μεταβλητής της χειρονομίας.
- *Έμμεση πρόσκτηση (Indirect acquisition)*, όπου οι χειρονομίες είναι απομονωμένες από τις δομικές ιδιότητες του ήχου που παράγεται από το όργανο. Τεχνικές επεξεργασίας σήματος μπορούν να χρησιμοποιηθούν στη συνέχεια προκειμένου να εξαχθούν οι ενέργειες του εκτελεστή, μέσω της ανάλυσης της θεμελιώδους συχνότητας, του φάσματος κτλ.
- *Πρόσκτηση σημάτων φυσιολογίας (Physiological signal acquisition)*, ή ανάλυση φυσιολογικών σημάτων, όπως το EMG (ηλεκτρομυογράφημα). Εμπορικά συστήματα έχουν αναπτυχθεί βασισμένα στην ανάλυση της τάσης των μυών και έχουν χρησιμοποιηθεί σε μουσικό περιβάλλον. Παρόλο που αυτή η τεχνική συλλαμβάνει την ουσία της κίνησης, είναι αρκετά δύσκολο να εφαρμοστεί αφού είναι πολύ δύσκολος ο διαχωρισμός των σημαντικών στοιχείων του σήματος που έχει αποκτηθεί από φυσιολογική κίνηση.

Η άμεση πρόσκτηση έχει το πλεονέκτημα της απλότητας σε σχέση με την έμμεση πρόσκτηση, λόγω της αμοιβαίας επιρροής διαφορετικών παραμέτρων στον τελικό ήχο. Ωστόσο, λόγω της ανεξαρτησίας των μεταβλητών που συλλαμβάνονται, κάποιες τεχνικές άμεσης πρόσκτησης είναι πιθανό να υποτιμήσουν την αλληλεξάρτηση κάποιων μεταβλητών (Wanderley M., 2000).

Παρόλα αυτά, λόγω της απλότητάς της, η άμεση πρόσκτηση είναι η πιο διαδεδομένη μέθοδος σε αυτό το πεδίο εφαρμογών. Όπως ήδη αναφέρθηκε, στην άμεση πρόσκτηση χρησιμοποιούνται αισθητήρες για να γίνει έλεγχος των ενεργειών του εκτελεστή, οι οποίοι θα πρέπει να επιλεχθούν με βάση κάποια κριτήρια.

Σε κάποια γενικότερη εφαρμογή, μια προσέγγιση στην επιλογή του κατάλληλου αισθητήρα θα ήταν ο σχεδιασμός της αίσθησης του υπολογιστή ώστε να αποκρίνεται στις πέντε ανθρώπινες αισθήσεις: γεύση, οσμή, αφή, ακοή και όραση. Η καλύτερη προσέγγιση, ωστόσο, θα ήταν να αποφασιστεί ποιες θεληματικές (ή ακόμα και μη θεληματικές) ενέργειες του χρήστη θα είναι σημαντικές για την συγκεκριμένη εφαρμογή. Με άλλα λόγια, είναι σημαντικό να αποφασιστεί το ποιες χειρονομίες ή ενέργειες του ανθρώπου είναι κατάλληλες για την εφαρμογή και να καθοριστεί ποιος αισθητήρας είναι καταλληλότερος για την μέτρηση αυτής της χειρονομίας ή ενέργειας (Putman W. & Knapp R.B., 1996).

Οι αισθητήρες μπορούν να κατηγοριοποιηθούν με διάφορους τρόπους. Μπορούν να κατηγοριοποιηθούν με βάση τους φυσικούς νόμους στους οποίους βασίζεται η λειτουργικότητά τους. Ωστόσο, μια αρχή της φυσικής μπορεί να χρησιμοποιηθεί για τη μέτρηση πολλών διαφορετικών φαινομένων. Για παράδειγμα, το πιεζοηλεκτρικό αποτέλεσμα ενός αισθητήρα μπορεί να χρησιμοποιηθεί για την μέτρηση της δύναμης, της καμπυλότητας, της επιτάχυνσης, της θερμότητας και των ακουστικών δονήσεων. Οι αισθητήρες μπορούν να κατηγοριοποιηθούν με βάση τα ειδικά φαινόμενα που μετρούν. Ωστόσο, ένα φαινόμενο μπορεί να μετρηθεί με βάση πολλές φυσικές αρχές. Για παράδειγμα, τα ηχητικά κύματα μπορούν να μετρηθούν με το πιεζοηλεκτρικό αποτέλεσμα, με τα αποτελέσματα του ηλεκτρομαγνητικού πεδίου και με τις μεταβολές στην αντίσταση. Οι αισθητήρες μπορούν επίσης να ομαδοποιηθούν με βάση μια ιδιαίτερη εφαρμογή. Για παράδειγμα, θα μπορούσε κανείς να ομαδοποιήσει όλους τους αισθητήρες που μπορούν να χρησιμοποιηθούν για την μέτρηση απόστασης. Ωστόσο, αφού υπάρχει ένας έξυπνος τρόπος να μετρηθεί η απόσταση με την χρήση οποιουδήποτε σχεδόν αισθητήρα, ούτε αυτός είναι ένας ιδιαίτερα καλός τρόπος ταξινόμησης των αισθητήρων (Putman W. & Knapp R.B., 1996).

Αρκετοί συγγραφείς θεωρούν ότι τα πιο σημαντικά χαρακτηριστικά των αισθητήρων είναι η ευαισθησία, η σταθερότητα και η επαναληψιμότητα. Άλλα σημαντικά χαρακτηριστικά σχετίζονται με τη γραμμικότητα και την επιλεκτικότητα στην έξοδο των αισθητήρων.

Σε γενικές συνθήκες, οι αισθητήρες πρέπει να είναι ακριβείς και αξιόπιστοι, με επαρκή ανάλυση. Στο χώρο της μουσικής, συχνά θεωρείται ότι η επιλογή της τεχνολογίας των μεταλλακτών των αισθητήρων για ένα συγκεκριμένο μουσικό χαρακτηριστικό σχετίζεται με την ανθρώπινη αντίληψη. Για παράδειγμα, η αντιστοίχηση της εξόδου ενός αισθητήρα που είναι αξιόπιστος αλλά όχι ιδιαίτερα ακριβής σε μια μεταβλητή που ελέγχει την ηχηρότητα μπορεί να είναι ικανοποιητική, αλλά εάν πρόκειται για τον έλεγχο του τονικού ύψους, η ανακρίβειά του πιθανό να γίνεται πιο εύκολα αντιληπτή. Οι επιλογές αυτές, όμως, έχουν να κάνουν και με την γενικότερη στρατηγική που ακολουθείται στις αντιστοιχήσεις για τον σχεδιασμό ενός ψηφιακού μουσικού οργάνου ή μιας ηχητικής εγκατάστασης.

1.4 Στρατηγική Αντιστοιχίσεων

Ο όρος αντιστοίχηση αναφέρεται στη σχέση μεταξύ των παραμέτρων ελέγχου, που απορρέουν από τις ενέργειες του εκτελεστή (χειρονομίες) και των παραμέτρων της ηχητικής σύνθεσης.

Τη στιγμή που οι μεταβλητές των χειρονομιών είναι διαθέσιμες, είτε από ανεξάρτητους αισθητήρες, είτε σαν αποτέλεσμα τεχνικών επεξεργασίας σήματος, στην περίπτωση έμμεσης πρόσκτησης, πρέπει να γίνει η συσχέτιση αυτών των μεταβλητών εξόδου στις διαθέσιμες μεταβλητές εισόδου της σύνθεσης του ήχου.

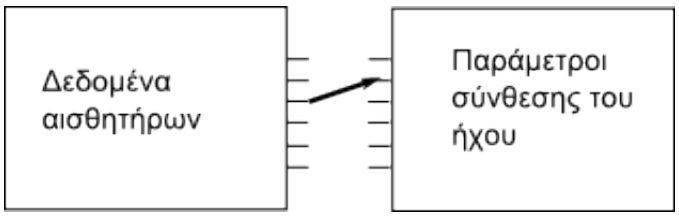
Ανάλογα με την μέθοδο ηχητικής σύνθεσης που χρησιμοποιείται, ο αριθμός και τα χαρακτηριστικά των μεταβλητών εισόδου μπορεί να διαφέρουν. Για μεθόδους σημάτων, μπορεί να έχουμε: α) πλάτη, συχνότητες και φάσεις από ημιτονοειδή μέρη για προσθετική σύνθεση, β) συντελεστές φορέα και διαμορφωτή (carrier & modulator) για διαμόρφωση συχνότητας (FM), κτλ.

Είναι ολοφάνερο ότι η σχέση μεταξύ των μεταβλητών χειρονομιών και των διαθέσιμων μεταβλητών εισόδου της σύνθεσης δεν είναι καθόλου ξεκάθαρη. Πως μπορεί κανείς, για παράδειγμα, να συσχετίσει μια χειρονομία με έναν λόγο φορέα/διαμορφωτή; (Wanderley M., 2000).

Η συστηματική μελέτη της αντιστοίχησης (mapping) είναι μια περιοχή που δεν είναι ακόμα ανεπτυγμένη. Μόνο κάποιες μελέτες έχουν προταθεί που αναλύουν την επιρροή της αντιστοίχησης σε μια εκτέλεση ψηφιακού μουσικού οργάνου ή που προτείνουν τρόπους ορισμού των αντιστοιχίσεων των μεταβλητών του ελέγχου με μεταβλητές εισόδου της σύνθεσης του ήχου.

Όσον αφορά τους τύπους αντιστοίχησης, διακρίνουμε δύο κύριες κατευθύνσεις:

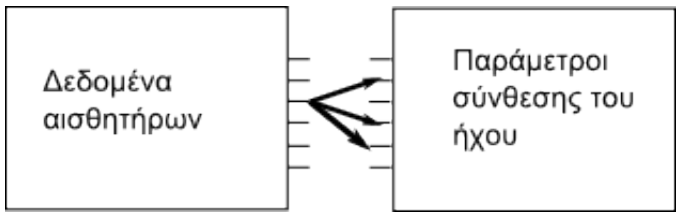
- Τη χρήση παραγωγικών μηχανισμών (πχ. νευρικά δίκτυα) για την πραγματοποίηση της αντιστοίχησης
- Τη χρήση ρητών στρατηγικών αντιστοίχησης (Hunt A., Wanderley M. & Kirk R.)



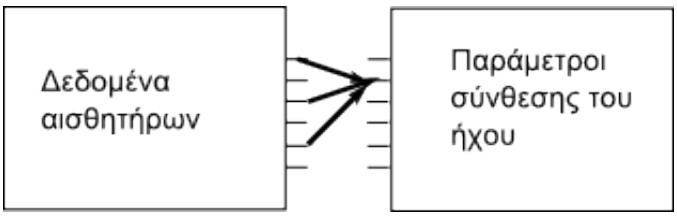
Εικόνα 1.4.1 Αντιστοίχιση ένα-προς-ένα

Στην εργασία αυτή θα μας απασχολήσει η κατεύθυνση των ρητών στρατηγικών αντιστοίχησης.

Μπορούμε να διακρίνουμε τρεις βασικές κατηγορίες στρατηγικών ρητής αντιστοίχησης: ένα-προς-ένα (Εικόνα 1.4.1), ένα-προς-πολλά (Εικόνα 1.4.2), πολλά-προς-ένα (Εικόνα 1.4.3) (Wanderley M., 2000). Είναι, φυσικά, δυνατός και ένας προφανής συνδυασμός των παραπάνω, η κατηγορία πολλά-σε-πολλά.



Εικόνα 1.4.2 Αντιστοίχιση ένα-προς-πολλά



Εικόνα 1.4.3 Αντιστοίχιση πολλά-προς-ένα

Για τον σχεδιασμό της στρατηγικής των αντιστοιχίσεων που θα ακολουθηθεί εν τέλει, πρέπει να ληφθεί υπόψη και ο τελικός σκοπός που επιχειρείται να επιτευχθεί. Για

παράδειγμα, οι αντιστοιχήσεις που θα επιλεχθούν σε περίπτωση που στόχος είναι η εξομοίωση ενός παραδοσιακού ακουστικού μουσικού οργάνου, θα διαφέρουν πολύ από τις αντίστοιχες στην περίπτωση που ο στόχος είναι ο σχεδιασμός ενός καινοτόμου ελεγκτή σύνθεσης ήχου.

Ο ρόλος που παίζουν οι αντιστοιχήσεις στο τελικό αποτέλεσμα και χαρακτήρα του ψηφιακού οργάνου, είναι πολύ καθοριστικός. Ωστόσο, οι αντιστοιχήσεις που θα επιλεχθούν μπορεί να αποδειχτούν καθοριστικές για το βαθμό δυσκολίας χρήσης του τελικού οργάνου. Για παράδειγμα, η χρήση της στρατηγικής πολλά-προς-ένα μπορεί να σημαίνει ότι πολλαπλές χειρονομίες του εκτελεστή χρησιμοποιούνται για τον έλεγχο μιας μόνο παραμέτρου παραγωγής του ήχου.

2 Υλοποίηση

Στις επόμενες ενότητες περιγράφεται η υλοποίηση της ιδέας που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Περιγράφεται το υλικό που χρησιμοποιήθηκε και η υλοποίηση του λογισμικού που αξιοποιεί το υλικό αυτό.

2.1 Υλικό (Hardware)

Από πλευράς υλικού χρησιμοποιήθηκε ένας CV-to-MIDI μετατροπέας (ενότητα 2.1.1) και ένα σύνολο από αισθητήρες (ενότητα 2.1.2), για την πρόσκτηση των χειρονομιών του εκτελεστή.

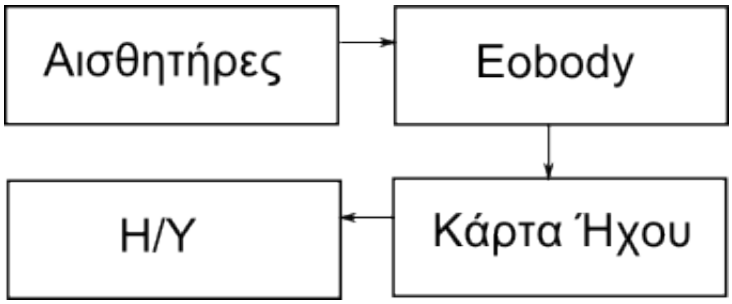
2.1.1 Μετατροπέας αναλογικής τάσης σε MIDI

Προκειμένου οι πληροφορίες που συλλέγονται από τους αισθητήρες να αξιοποιηθούν στον αλγόριθμο σύνθεσης, θα πρέπει να κωδικοποιηθούν με κάποιο τρόπο που, ανάλογα με την υλοποίηση, κρίνεται αποτελεσματικός. Ένα πρωτόκολλο επικοινωνίας που χρησιμοποιείται συχνά στις ηχητικές εγκαταστάσεις, καθώς και αυτό που χρησιμοποιήθηκε στην υλοποίηση της εργασίας,

Εικόνα 2.1.1 Ο μετατροπέας CV-to-MIDI eobody της εταιρίας eo-wave

είναι το πρωτόκολλο *MIDI (Musical Instrument Digital Interface)*. Το πρωτόκολλο MIDI αναπτύχθηκε στις αρχές της δεκαετίας του 1980 μετά από μια συνεργασία κάποιων κατασκευαστών μουσικών synthesizers και, όπως μαρτυράει και το όνομά του αποτελεί στην ουσία ένα interface επικοινωνίας μεταξύ διάφορων μονάδων παραγωγής του ήχου, όπως keyboards, synthesizers, ειδικό

λογισμικό κτλ. Στην υλοποίηση της εργασίας αυτής, το πρωτόκολλο MIDI χρησιμοποιήθηκε σαν ενδιάμεσο επικοινωνίας, δηλαδή σαν ένας τρόπος οργάνωσης των δεδομένων που λαμβάνονται από τους αισθητήρες.



Εικόνα 2.1.2 Σχηματική αναπαράσταση της συνδεσμολογίας που χρησιμοποιήθηκε

Οι αισθητήρες, ανάλογα με το μηχανικό ερέθισμα που συλλαμβάνουν, παράγουν ηλεκτρική τάση. Για την μετατροπή της τάσης των αισθητήρων σε MIDI πληροφορία χρησιμοποιήθηκε ο μετατροπέας *CV-to-MIDI (Current Voltage to MIDI)* eobody της εταιρίας eo-wave (Εικόνα 2.1.1). Ο μετατροπέας eobody διαθέτει 16 εισόδους αισθητήρων, μία *MIDI IN* και μία *MIDI OUT* θύρα, για την επικοινωνία με την κάρτα ήχου του υπολογιστή. Είναι απαραίτητο η κάρτα ήχου να διαθέτει MIDI interface, δηλαδή να μπορεί να επικοινωνήσει χρησιμοποιώντας το πρωτόκολλο MIDI και να διαθέτει τις απαραίτητες θύρες επικοινωνίας. Χρησιμοποιήθηκε η εξωτερική κάρτα ήχου *μAudio MIDISPORT 2x2* που συνδέθηκε με τον υπολογιστή μέσω θύρας USB. Στην Εικόνα 2.1.2 αναπαρίσταται η συνδεσμολογία που χρησιμοποιήθηκε.

Εικόνα 2.1.3 Μερικοί αισθητήρες της εταιρίας eo-wave, που διαθέτει το εργαστήριο.

μετράει οποιαδήποτε φυσικό φαινόμενο.

2.1.2 Αισθητήρες

Για τη δοκιμή και τη δημιουργία πειραματικών εγκαταστάσεων, χρησιμοποιήθηκαν μια σειρά από αισθητήρες που διαθέτει το εργαστήριο Μουσικής Διάδρασης και Πολυφωνίας (Εικόνα 2.1.3). Θα ακολουθήσει μια σύντομη αναφορά σε κάποιους από τους αισθητήρες που χρησιμοποιήθηκαν για τη δοκιμή της εφαρμογής. Θα πρέπει, ωστόσο, να σημειωθεί ότι θεωρητικά θα μπορούσε να γίνει χρήση οποιουδήποτε είδους αισθητήρα, που θα

2.1.2.1 Αισθητήρες πίεσης

Εικόνα 2.1.4 Αισθητήρας πίεσης

Οι αισθητήρες πίεσης της eowave είναι αντιστάτες εντοπισμού πίεσης (force sensing resistors), που είναι πολύ ανθεκτικοί. Ο τρόπος λειτουργίας τους είναι ο εξής: Όταν ασκείται πίεση στην επιφάνεια του αισθητήρα, μειώνεται η αντίσταση του κυκλώματος και με αυτόν τον τρόπο παράγεται μεγαλύτερη τάση στην έξοδο του αισθητήρα. Οι κυριότερες εφαρμογές στις οποίες γίνεται χρήση των αισθητήρων πίεσης είναι στη βιομηχανία, σε σερβοκινητήρες, σε εξοπλισμό διασκέδασης και στον ζωντανό έλεγχο μουσικής. Στην Εικόνα 2.1.4 βλέπουμε έναν αισθητήρα πίεσης.

2.1.2.2 Αισθητήρες κάμψης (flexion)

Αισθητήρες που το ηλεκτρικό αποτέλεσμα που παράγουν εξαρτάται από τον βαθμό καμπύλωσης του ελάσματος που διαθέτουν. Η πιο διαδεδομένη χρήση τους είναι στη δημιουργία γαντιών, με ένα αισθητήρα κάμψης σε κάθε δάχτυλο. Στην Εικόνα 2.1.5 βλέπουμε έναν αισθητήρα κάμψης.

Εικόνα 2.1.5 Αισθητήρας κάμψης

2.1.2.3 Αισθητήρες επιτάχυνσης

Οι αισθητήρες επιτάχυνσης ή αλλιώς επιταχυνσιόμετρα (acellerometers) έχουν πολλές εφαρμογές, όπως χειριστήρια παιχνιδιών, συστήματα ασφαλείας συσκευών, συσκευές εισόδου, πρακτικές ηλεκτρονικές συσκευές τσέπης, προϊόντα διασκέδασης, αίσθηση χορού και ζωντανής εκτέλεσης. Οι αισθητήρες αυτοί μπορούν να μετρήσουν δυναμική επιτάχυνση (πχ. δονήσεις) και στατική επιτάχυνση (πχ. βαρύτητα). Είναι κατάλληλοι για τον εντοπισμό των αργών και των απότομων χειρονομιών. Στην Εικόνα 2.1.6 βλέπουμε έναν αισθητήρα επιτάχυνσης.

Εικόνα 2.1.6 Αισθητήρας επιτάχυνσης

2.1.2.4 Αισθητήρες δονήσεων (shock)

Οι αισθητήρες δονήσεων διαθέτουν μια επιφάνεια ευαίσθητη στις δονήσεις, που τους καθιστούν ιδανικούς για χρήση σε εξομοίωση κρουστών, ή για την πυροδότηση ενεργειών μικρής διάρκειας.

Εικόνα 2.1.7 Αισθητήρας εντοπισμού

2.1.2.5 Αισθητήρες εντοπισμού

Οι αισθητήρες εντοπισμού δεν δίνουν συνεχείς τιμές, αλλά λειτουργούν σε κατάσταση toggle. Για αυτό το λόγο, στην εν λόγω εφαρμογή προτείνεται η χρήση τους σε αντιστοίχηση με την έναρξη ή παύση μιας ηχητικής μονάδας. Οι αισθητήρες αυτοί μπορούν να εντοπίσουν οποιαδήποτε κίνηση σε ακτίνα 20 μέτρων και υπό γωνία 180°. Η πιο διαδεδομένη γενική χρήση τους είναι σε συστήματα ασφαλείας. Στην Εικόνα 2.1.7 βλέπουμε έναν αισθητήρα εντοπισμού.

2.2 Υλοποίηση Λογισμικού

Η υλοποίηση της εφαρμογής έγινε με τη χρήση της γλώσσας προγραμματισμού Java. Η Java είναι μια *αντικειμενοστραφής (object oriented)*, *υψηλού επιπέδου (high level)* γλώσσα προγραμματισμού. Μια αντικειμενοστραφής και υψηλού επιπέδου γλώσσα προγραμματισμού παρέχει αυξημένη δυνατότητα στον προγραμματιστή να αναπαραστήσει τη λύση ενός συγκεκριμένου προβλήματος με όρους του ίδιου του προβλήματος -αντί με όρους που αφορούν το hardware του υπολογιστή, όπως συμβαίνει σε γλώσσες χαμηλότερου επιπέδου-, με την χρήση των *αντικειμένων (objects)*. Κάθε αντικείμενο ανήκει σε μια συγκεκριμένη *κλάση (class)* που ορίζει τη *συμπεριφορά* και το είδος των *ιδιοτήτων* κάθε αντικειμένου της συγκεκριμένης κλάσης. Για παράδειγμα, αν για την επίλυση κάποιου προβλήματος, χρειαζόταν να κατασκευάσουμε μια κλάση "Άνθρωπος" στη Java, θα ορίζαμε μια μεταβλητή - ιδιότητα "Χρώμα δέρματος" που δεν θα είχε συγκεκριμένη τιμή, αλλά κάθε αντικείμενο της κλάσης αυτής (ένας συγκεκριμένος άνθρωπος) θα είχε μια διαφορετική τιμή για την ιδιότητα αυτή. Με άλλα λόγια, κάθε άνθρωπος πρέπει να έχει ένα συγκεκριμένο χρώμα δέρματος, αλλά δεν έχουν όλοι το ίδιο. Επίσης, θα ορίζαμε μια μέθοδο, που θα την ονομάζαμε "Φάγε", επειδή κάθε άνθρωπος μπορεί να φάει -με αυτόν τον τρόπο ορίζουμε τη συμπεριφορά μιας κλάσης.

Το μεγάλο πλεονέκτημα του αντικειμενοστραφούς προγραμματισμού είναι ότι με τη χρήση των αντικειμένων μπορούμε να εξομοιώσουμε καλύτερα το πρόβλημα που θέλουμε να επιλύσουμε. Για παράδειγμα, αν θέλαμε να προγραμματίσουμε ένα παιχνίδι μπάσκετ με τη χρήση μιας αντικειμενοστραφούς γλώσσας προγραμματισμού, η συγγραφή μιας κλάσης "Παίκτης" σχεδόν θα επιβαλλόταν. Αυτή η κλάση θα είχε ιδιότητες όπως "Όνομα Παίκτη", "Ομάδα", αλλά και "Ευστοχία στα τρίποντα" που θα ήταν ένας ακέραιος με μέγιστη τιμή το 20 (άριστα). Με την κατασκευή ενός νέου αντικειμένου αυτής της κλάσης, κάθε ιδιότητα θα αποκτούσε συγκεκριμένη τιμή. Για κάποιον άσσο του NBA, η "Ευστοχία στα τρίποντα" θα έπρεπε να πάρει την τιμή 20, ενώ για τον συγγραφέα της εργασίας αυτής, η ίδια μεταβλητή-ιδιότητα θα έπρεπε, κατά πάσα πιθανότητα, να πάρει την τιμή 1.

Παρά τα παραπάνω, πολλές φορές απαιτείται η συγγραφή κλάσεων που δεν αντιπροσωπεύουν άμεσα κάποια συνθήκη του πραγματικού κόσμου, αλλά ωστόσο διευκολύνουν στην επίλυση του προβλήματος. Μια τέτοια περίπτωση θα ήταν, για παράδειγμα, μια κλάση "Γεννήτρια Τυχαίων Αριθμών" που θα διέθετε μεθόδους για παραγωγή τυχαίων ακεραίων όποτε χρειαζόταν. Στο προαναφερθέν παράδειγμα, όταν ένας παίκτης προσπαθεί να πετύχει τρίποντο, η ιδιότητά του "Ευστοχία στα τρίποντα" θα πρέπει να συνδυαστεί με έναν μαθηματικό τρόπο με το αποτέλεσμα που θα μας δώσει η "Γεννήτρια Τυχαίων Αριθμών" για να αποφασιστεί από το πρόγραμμα αν ο παίκτης θα ευστοχήσει ή θα αστοχήσει στο συγκεκριμένο τρίποντο.

Τα προαναφερθέντα χαρακτηριστικά της Java, καθώς και η δυνατότητα εύκολης και γρήγορης χρήσης εξωτερικών βιβλιοθηκών (*APIs – Application Programming Interfaces*), ήταν αυτά που συνετέλεσαν στην επιλογή της για την ανάπτυξη της εργασίας αυτής. Για την ανάπτυξη των αλγόριθμων των μονάδων παραγωγής του ήχου (synthesizers), χρησιμοποιήθηκε η εξωτερική βιβλιοθήκη Jsyn. Το Jsyn παρέχει έτοιμες κλάσεις που διευκολύνουν στη δημιουργία και διαχείριση του ήχου. Για παράδειγμα, διαθέτει την κλάση "SineOscillator", με μεθόδους για τη ρύθμιση του πλάτους και της συχνότητας της ημιτονοειδούς κυματομορφής.

Επιπλέον, η Java είναι μια γλώσσα μεταφερτή. Ο πηγαίος κώδικας υφίσταται μεταγλώττιση στον λεγόμενο *byte code*, ο οποίος διερμηνεύεται σε χαμηλότερο επίπεδο τη στιγμή της εκτέλεσης του προγράμματος. Ο byte code, ο οποίος είναι μια ενδιάμεση κατάσταση μεταξύ του πηγαίου κώδικα της Java και του μεταγλωττισμένου κώδικα, είναι κοινός για κάθε πλατφόρμα. Για αυτό το λόγο, τα προγράμματα που γράφονται στη Java έχουν την δυνατότητα να είναι μεταφερτά, δηλαδή

να μπορούν να εκτελεστούν σε οποιαδήποτε πλατφόρμα, αρκεί η πλατφόρμα να διαθέτει την αντίστοιχη *Java Virtual Machine,* που αναλαμβάνει την διερμηνεία του byte code για την συγκεκριμένη πλατφόρμα.

Ωστόσο, εξαιτίας της χρήσης του Jsyn, που ένα μέρος του είναι γραμμένο στη γλώσσα C και προορίζεται για το λειτουργικό σύστημα των Windows, η τελική εφαρμογή χάνει το στοιχείο της μεταφερτότητας που διαθέτει μια εφαρμογή γραμμένη σε καθαρή Java. Στο μέλλον δρομολογείται μια νέα έκδοση του Jsyn, που θα είναι γραμμένη εξολοκλήρου σε Java, έτσι θα είναι μεταφερτή και, κατά συνέπεια, θα είναι πλέον μεταφερτά και τα προγράμματα που κάνουν χρήση του νέου αυτού API.

Για τη συγγραφή της εφαρμογής και της προγραμματιστικής βιβλιοθήκης χρησιμοποιήθηκε επίσης ο ειδικός διορθωτής Eclipse. Το Eclipse είναι ένα πρόγραμμα *IDE (Integrated Development Environment)*, ανοιχτού κώδικα, που διαθέτει ένα πολύ βοηθητικό περιβάλλον για τον προγραμματισμό σε Java, καθώς και για άλλες γλώσσες προγραμματισμού.

Τέλος, για την ρύθμιση του Eobody, χρησιμοποιήθηκε ένα ειδικό patch σε περιβάλλον MaxMSP που διανέμεται μαζί με την συσκευασία του προϊόντος.

2.2.1 Προγραμματιστική βιβλιοθήκη (API)

Στη γλώσσα προγραμματισμού Java, συνιστάται η οργάνωση των καινούργιων κλάσεων που δημιουργούμε σε *πακέτα (packages)* προκειμένου να αποφευχθεί η πιθανότητα σύγκρουσης δύο ίδιων ονομάτων κλάσεων. Για παράδειγμα, εάν σε κάποιο project ορίσουμε μια κλάση με το όνομα StringBuffer, η Java δεν θα μπορούσε να ξεχωρίσει, κάθε φορά που θα κάναμε χρήση αυτής της κλάσης, αν θα αναφερόμασταν στην κλάση που έχουμε ορίσει ή στην κλάση με το ίδιο όνομα που διαθέτει το API της Java.

Για αυτόν τον λόγο, χρησιμοποιούνται ευρύτατα τα πακέτα, όπου ουσιαστικά πρόκειται για φακέλους στους οποίους αποθηκεύονται οι κλάσεις μας. Όταν έχουμε δημιουργήσει ένα πακέτο, το οποίο θα περιέχει τις κλάσεις που θα έχουμε συγγράψει, μπορούμε στη συνέχεια να χρησιμοποιήσουμε αυτές τις κλάσεις σε κάποιο άλλο πακέτο, εάν *εισάγουμε (import)* το πακέτο στον κώδικα του καινούργιου πακέτου.

Η χρήση των πακέτων μπορεί να διευκολύνει πολύ τη διαδικασία ανάπτυξης εφαρμογών, καθώς η τάση που φαίνεται να επικρατεί στον προγραμματισμό τα τελευταία χρόνια, είτε αυτός αναφέρεται σε εφαρμογές οποιουδήποτε ενδιαφέροντος, είτε σε εφαρμογές δικτύου, είναι η τάση του διαχωρισμού του περιεχομένου της εφαρμογής από τον τρόπο παρουσίασής της. Αυτό σημαίνει ότι οποιοδήποτε κομμάτι του κώδικα αφορά το ίδιο το περιεχόμενο (για παράδειγμα, σε μια εφαρμογή εμπορικής διαχείρισης, η διαχείριση της βάσης δεδομένων των πελατών), θα πρέπει να είναι σαφώς διαχωρισμένο από το κομμάτι του κώδικα που αφορά τον τρόπο παρουσίασής του, δηλαδή την *Γραφική Διεπαφή Χρήστη (Graphical User Interface - GUI).* Με αυτόν τον τρόπο, σε μια μελλοντική έκδοση του προγράμματος, που θα υιοθετούσε νέες τεχνολογίες στον τομέα του GUI, αλλά δεν θα είχε τίποτε διαφορετικό να παρουσιάσει όσον αφορά την διαχείριση των δεδομένων, οι όποιες μεταβολές του κώδικα θα γίνονταν μόνο στο κομμάτι του κώδικα που αφορά το GUI. Κάτι παρόμοιο παρατηρούμε στα πιο σύγχρονα web sites, όπου το περιεχόμενο των σελίδων αποδίδεται μέσω ελάχιστων και ξεκάθαρων HTML tags, ενώ ο τρόπος παρουσίασης πλέον ελέγχεται μέσω των CSS properties.

Προκειμένου να υλοποιηθεί η παραπάνω φιλοσοφία στη Java, θα πρέπει:

1. Να διασπαστεί ο κώδικας σε διαφορετικές κλάσεις, όπου κάθε κλάση θα έχει ένα όνομα που

θα βοηθάει στην κατανόηση της λειτουργίας της (για παράδειγμα, η κλάση FMSynthesizer είναι μια κλάση που αντιπροσωπεύει αυτό ακριβώς που δηλώνει το όνομά της – έναν FM synthesizer).

2. Να μοιραστούν οι κλάσεις αυτές σε διάφορα πακέτα, όπου το καθένα θα περιέχει τις κλάσεις που θα υποστηρίζουν την λειτουργία που το πακέτο αυτό υπόσχεται (για παράδειγμα, το πακέτο με το όνομα synthesizer, θα πρέπει να περιέχει την κλάση FMSynthesizer που αναφέρθηκε παραπάνω, καθώς και μια κλάση AMSynthesizer).

Θα πρέπει στο σημείο αυτό να αναφερθεί ότι η χρήση των πακέτων έχει μια ακόμα ιδιαίτερη σημασία στην ανάπτυξη *προγραμματιστικών βιβλιοθηκών (APIs)*. Μια προγραμματιστική βιβλιοθήκη στη Java παρέχει κάποιες έτοιμες κλάσεις, που μπορεί να χρησιμοποιήσει κάποιος άλλος προγραμματιστής για την υλοποίηση του δικού του στόχου. Σε αυτήν την περίπτωση γίνεται πιο αναγκαία η χρήση των πακέτων, καθώς ο προγραμματιστής θα μπορεί να χρησιμοποιήσει όποια κλάση χρειάζεται χωρίς να υπάρχει κίνδυνος το όνομά της να είναι το ίδιο με κάποια άλλη κλάση, με την προϋπόθεση τα ονόματα των πακέτων να είναι μοναδικά.

Η εταιρία Sun Microsystems η οποία ανέπτυξε την γλώσσα προγραμματισμού Java, προτείνει μια σύμβαση για την ονομασία των πακέτων από τους προγραμματιστές, με σκοπό την μοναδικότητα του ονόματος κάθε πακέτου. Αυτή η σύμβαση ορίζει ότι κάθε όνομα πακέτου θα πρέπει να αρχίζει από το domain name του προγραμματιστή, της εταιρίας στην οποία εργάζεται, ή του ιδρύματος στο οποίο βρίσκεται, σε αντίστροφη σειρά. Για την περίπτωση αυτής της πτυχιακής εργασίας, κάθε πακέτο αρχίζει με τον εξής τρόπο: gr.teicrete.mta, με την λογική ότι η διεύθυνση στο internet του τμήματος είναι www.teicrete.gr/mta. Η χρήση της τελείας διαχωρίζει τους φακέλους του συστήματος αρχείων στο λειτουργικό σύστημα. Στα Windows, η τελεία αντικαθίσταται από την ανάστροφη διαγώνιο (backslah) (\), ενώ στο Linux από την διαγώνιο (slash) (/). Κάθε λέξη ανάμεσα στις τελείες αντιπροσωπεύει ένα φάκελο τους file system. Έτσι, στη συνέχεια προσθέτουμε ονόματα φακέλων με βάση τη λειτουργία κάθε πακέτου. Στον πίνακα που ακολουθεί παρουσιάζονται συνοπτικά όλα τα πακέτα της πτυχιακής εργασίας και μια επεξήγηση του σκοπού του κάθε πακέτου.

Πακέτο	**Λειτουργία**
gr.teicrete.mta.midi.device	Περιέχει μια κλάση για την ανάκτηση πληροφοριών για τις τρέχουσες συσκευές midi του Η/Υ.
gr.teicrete.mta.midi.eobody.event	Περιέχει κλάσεις που αναπαριστούν τους τύπους των midi μηνυμάτων (NoteOnEvent κα).
gr.teicrete.mta.midi.eobody.parse	Περιέχει την κλάση EobodyEventFactory που είναι υπεύθυνη για τη λήψη και μετάδοση των midi μηνυμάτων (receiver → listener).
gr.teicrete.mta.midi.eobody.process	Περιέχει το interface[1] EobodyEventListener που ορίζει τις μεθόδους που πρέπει να υλοποιηθούν από μια κλάση, ώστε να ερμηνεύει τα midi μηνύματα που μεταδίδει η κλάση EobodyEventFactory.

1 Ένα *interface* είναι μια ειδική κλάση που περιέχει δηλώσεις μεθόδων, χωρίς σώμα κώδικα. Ένα interface ορίζει μια δυνατότητα που μπορεί να έχει μια άλλη κλάση και μπορεί να *υλοποιηθεί (implement)* από αυτήν την κλάση, προκειμένου αυτή να ορίσει τον τρόπο με τον οποίο υλοποιεί αυτή τη δυνατότητα. Τα interfaces χρησιμοποιούνται πολύ συχνά στη Java σε event handling routines (ρουτίνες χειρισμού γεγονότων). Σε αυτές, οι κλάσεις που θέλουν να είναι event handlers (δηλαδή που θέλουν να αποκρίνονται σε ένα συμβάν – γεγονός) πρέπει να υλοποιήσουν τις μεθόδους του αντίστοιχου interface.

Πακέτο	Λειτουργία
gr.teicrete.mta.midi.eobody.setup	Περιέχει την κλάση EobodyConnector, που παρέχει μια μέθοδο για την επικοινωνία με το Eobody, με την προϋπόθεση ότι το τελευταίο έχει συνδεθεί με τον Η/Υ μέσω της κάρτας μAudio MIDISPORT. Η κλάση αυτή δεν χρησιμοποιείται στην παρούσα εφαρμογή, διατίθεται όμως στο API, για τη μελλοντική της χρήση.
gr.teicrete.mta.midi.gui	Περιέχει κλάσεις για τον σχεδιασμό και τη λειτουργικότητα του γραφικού περιβάλλοντος της εφαρμογής.
gr.teicrete.mta.midi.mappings	Περιέχει κλάσεις που αφορούν τις αντιστοιχήσεις των midi μηνυμάτων με τις παραμέτρους της σύνθεσης του ήχου.
gr.teicrete.mta.synthesizer	Το πακέτο αυτό συνιστά το κομμάτι παραγωγής του ήχου. Περιέχει κλάσεις που υλοποιούν ένα τύπο synthesizer (AM, FM), μια κλάση που αντιπροσωπεύει μια τυπική παράμετρο αυτών των synthesizer, και μια κλάση για τον έλεγχο όλων των μονάδων παραγωγής του ήχου (SynthEngine).
gr.teicrete.mta.synthesizer.event	Περιέχει κλάσεις που αντιπροσωπεύουν τα πιθανά συμβάντα που γεννιούνται από τις μονάδες παραγωγής του ήχου. Επίσης, περιέχει ένα interface που μπορεί να υλοποιηθεί προκειμένου να οριστεί συμπεριφορά *ανατροφοδότησης (feedback)*. Στην εφαρμογή αυτή έχει υλοποιηθεί στο πακέτο που αφορά τη γραφική διεπαφή, για την real-time απεικόνιση των μεταβολών στις παραμέτρους των synthesizer.

2.2.1.1 JavaDoc

Ένα ακόμη σημαντικό πλεονέκτημα της Java έναντι άλλων γλωσσών είναι η δυνατότητα εύκολης δημιουργίας κάποιου είδους εγχειριδίου για κάποια προγραμματιστική βιβλιοθήκη (API docs), με επεξήγηση των κλάσεων και των μεθόδων που διαθέτουν. Το μόνο προαπαιτούμενο είναι η συγγραφή των κατάλληλων σχολίων στον κώδικα, έτσι ώστε το εργαλείο της Java που δημιουργεί αυτόματα αυτό το εγχειρίδιο (javadoc) να μπορέσει να αναγνωρίσει. Το αποτέλεσμα είναι αρχεία σε μορφή HTML, που μπορεί ο προγραμματιστής – χρήστης ενός API να ανοίξει μέσα από τον internet browser που διαθέτει στο σύστημά του.

Το ίδιο το JDK (Java Developers Kit), που είναι η βασική – στοιχειώδης προγραμματιστική βιβλιοθήκη της Java, διαθέτει το δικό του javadoc, που είναι διαθέσιμο στην διεύθυνση της Sun για την Java στο internet (http://java.sun.com). Αν ο αναγνώστης έχει προηγούμενη εμπειρία σε κάποια άλλη γλώσσα προγραμματισμού, χαμηλού επιπέδου, πιθανό να μην μπορεί να εκτιμήσει τη χρησιμότητα του javadoc. Για μια γλώσσα τόσο μεγάλη σε όγκο, όμως, όπως η Java, ο προγραμματιστής είναι αδύνατον να απομνημονεύσει όλες τις κλάσεις και τις μεθόδους. Είναι σχεδόν σίγουρο ότι κάποια στιγμή θα καταφύγει στο javadoc.

Η εν λόγω προγραμματιστική βιβλιοθήκη διαθέτει και αυτή το δικό της javadoc, που μπορεί να βρεθεί στο αρχείο index.html, που βρίσκεται στο φάκελο doc. Στην Εικόνα 2.2.1 μπορούμε να δούμε μια τυπική σελίδα από το documentation του API. Μπορεί να παρατηρηθεί ότι στο πάνω – δεξιά πλαίσιο (frame) φαίνονται τα πακέτα του API, ενώ κάτω δεξιά φαίνονται οι κλάσεις που περιέχει το πακέτο που έχει επιλεχθεί. Στο κεντρικό frame εμφανίζεται η κλάση που έχει επιλεχθεί

από το κάτω δεξιά frame, με διάφορες λεπτομέρειες, όπως το σύνολο των μεθόδων που αυτή διαθέτει.

Εικόνα 2.2.1 Μια σελίδα από το documentation του API, όπως φαίνεται από το παράθυρο ενός internet browser

2.2.1.2 Κληρονομικότητα (Inheritance)

Ένα ακόμα στοιχείο της Java που χρησιμοποιήθηκε στην εργασία αυτή, είναι το στοιχείο της κληρονομικότητας. Ο όρος αυτός σημαίνει την δυνατότητα μιας κλάσης να κληρονομήσει ιδιότητες και συμπεριφορά από μια άλλη – προϋπάρχουσα – κλάση, με τον ίδιο τρόπο που το παιδί μπορεί να κληρονομήσει χαρακτηριστικά από τους γονείς του. Σε αυτήν την περίπτωση, η πρώτη κλάση (αυτή από την οποία η άλλη κληρονομεί) ονομάζεται *superclass*, ενώ η δεύτερη (αυτή η οποία κληρονομεί) ονομάζεται *subclass*.

Μία subclass κληρονομεί όλες τις μεθόδους και τις μεταβλητές από την superclass, έτσι, προκειμένου η δυνατότητα αυτή να αποκτήσει κάποιο νόημα, θα πρέπει στη subclass να ορίζονται επιπλέον μέθοδοι και μεταβλητές, έτσι ώστε αυτή να αποκτήσει περισσότερες και πιο συγκεκριμένες δυνατότητες από την superclass. Στη συνέχεια, κάποια άλλη κλάση θα μπορούσε να κληρονομήσει από την προηγούμενη subclass και ούτω καθεξής. Σταδιακά, δημιουργείται μια ιεραρχία ανάμεσα σε όλες τις κλάσεις, που μοιάζει με γενεαλογικό δέντρο, όπου στην κορυφή του βρίσκεται η κλάση Object, που βρίσκεται στο πακέτο java.lang, και είναι η κλάση από την οποία όλες οι κλάσεις στη Java κληρονομούν – υποχρεωτικά.

Όπως ήδη αναφέρθηκε, συχνά μια κλάση κληρονομεί από κάποια προϋπάρχουσα, προκειμένου να βασιστεί στις δυνατότητές της και να προσθέσει και κάποιες καινούργιες. Για παράδειγμα, η κλάση "Γυναίκα" θα μπορούσε να κληρονομεί από την κλάση "Άνθρωπος" που αναφέρθηκε σε προηγούμενο κεφάλαιο, επειδή και η γυναίκα είναι άνθρωπος, που μπορεί να κάνει ότι κάνει και ένας άνθρωπος (να φάει, για παράδειγμα), αλλά μπορεί επιπλέον και να θηλάσει.

Συνεπώς, στην κλάση “Γυναίκα” θα ορίζαμε μια νέα μέθοδο με το όνομα “Θήλασε”, για να ορίσουμε αυτήν την επιπλέον δυνατότητα της γυναίκας.

Εύκολα γίνεται αντιληπτό ότι συχνά μια subclass αναφέρεται σε κάτι πιο ειδικό, πιο συγκεκριμένο από ότι μια superclass, η οποία είναι πιο γενική, πιο αφαιρετική εννοιολογικά, σε πολλές περιπτώσεις τόσο γενική, ώστε να μην μπορεί να σταθεί από μόνη της σαν οντότητα. Ένα τέτοιο παράδειγμα είναι και η κλάση “Άνθρωπος”, αφού εάν είχαμε δύο subclass “Γυναίκα” και “Άντρας”, δεν θα δημιουργούσαμε αντικείμενα της κλάσης “Άνθρωπος”, αλλά μόνο άντρες και γυναίκες, εφόσον κάθε άνθρωπος πρέπει να έχει προσδιορισμένο φύλο. Ακόμα, κάποιες συμπεριφορές που έχει κάθε άνθρωπος, εκδηλώνονται με διαφορετικό τρόπο σε έναν άντρα από ότι σε μια γυναίκα. Κάθε άνθρωπος, πριν ετοιμαστεί να βγει από το σπίτι, θα πρέπει να ντυθεί, αλλά η γυναίκα θα ακολουθήσει κάπως διαφορετική διαδικασία από έναν άντρα. Σε αυτήν την περίπτωση, στην κλάση “Άνθρωπος” θα πρέπει να οριστεί αυτή η μέθοδος – συμπεριφορά σαν *abstract (αφαιρετική)*. Μια abstract μέθοδος δεν είναι υλοποιημένη, αλλά περιμένει να υλοποιηθεί από τις sub κλάσεις. Μια κλάση που περιέχει έστω και μια abstract μέθοδο, γίνεται και αυτή abstract, που σημαίνει ότι δεν μπορεί πλέον να δημιουργηθεί ένα νέο αντικείμενο της κλάσης αυτής (δεν υπάρχουν άνθρωποι γενικώς, παρά μόνο άντρες και γυναίκες).

Παρατηρώντας την Εικόνα 2.2.1, ο αναγνώστης μπορεί να διαπιστώσει ότι στο μεγάλο frame περιγράφεται η κλάση *AbstractSensorSynthesizer,* που βρίσκεται στο πακέτο gr.teicrete.mta.synthesizer. Στην περιγραφή της κλάσης αναφέρεται ότι η κλάση αυτή είναι abstract και ότι υπάρχουν δύο subclasses: η AMSynthesizer και η FMSynthesizer. Εξετάζοντας την λίστα με τις μεθόδους της κλάσης AbstractSensorSynthesizer, ο αναγνώστης μπορεί να εντοπίσει κάποιες abstract μεθόδους, όπως η μέθοδος start. Αυτή υλοποιείται διαφορετικά στην υποκλάση AMSynthesizer και διαφορετικά στην υποκλάση FMSynthesizer, αφού άλλη διαδικασία ακολουθείται για να ξεκινήσει ένας AM synthesizer από ότι ένας FM. Συνεπώς, ένας AbstractSensorSynthesizer δεν γνωρίζει από μόνος του πως να ξεκινήσει – γνωρίζουν μόνο οι υποκλάσεις του. Σε επόμενο κεφάλαιο, θα διερευνηθεί η δυνατότητα που δίνει το API σε έναν προγραμματιστή με γνώσεις Java να επεκτείνει τις δυνατότητές του, δημιουργώντας έναν καινούργιο τύπο synthesizer από αυτούς που το API ήδη προσφέρει – έναν additive synthesizer, για παράδειγμα – κληρονομώντας από την κλάση AbstractSensorSynthesizer.

2.2.2 Δημιουργία πρωτοτύπων και αντιστοιχίσεων από αρχεία κειμένου

Η προγραμματιστική βιβλιοθήκη και η τελική εφαρμογή παρέχουν την δυνατότητα δημιουργίας κάποιων *πρωτοτύπων (presets)* και *αντιστοιχίσεων (mappings)* μέσω της συγγραφής απλών σε δομή ASCII κειμένων. Με την λέξη presets εννοούμε ουσιαστικά κάποιες έτοιμες ρυθμίσεις για το κομμάτι της παραγωγής του ήχου, δηλαδή κάποιους synthesizer με προκαθορισμένες παραμέτρους. Η έννοια των αντιστοιχίσεων περιγράφεται στο κεφάλαιο 1.4.

Λεπτομερείς οδηγίες για την συγγραφή των preset και mapping αρχείων μπορούν να βρεθούν στο κεφάλαιο 3.2.3.

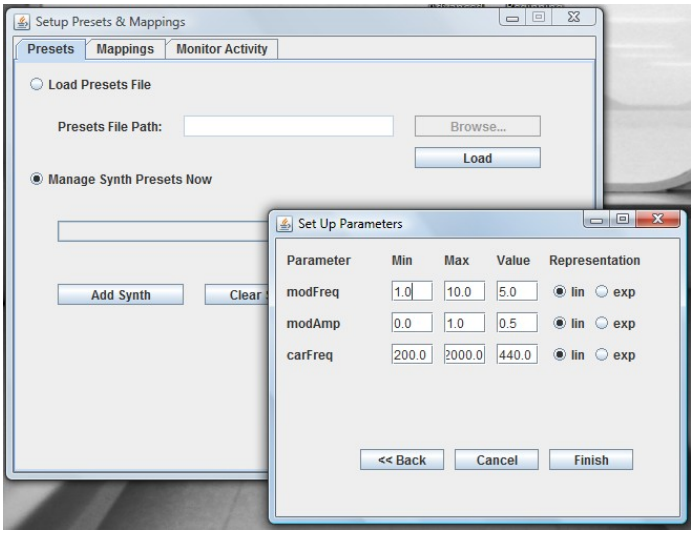
Η κλάση SynthesizerFactory του πακέτου gr.teicrete.mta.synthesizer δίνει την δυνατότητα στον προγραμματιστή να δημιουργήσει ένα ηχητικό πρωτότυπο από ένα αρχείο κειμένου, ενώ η κλάση Mapper του πακέτου gr.teicrete.mta.midi.mappings διαθέτει την μέθοδο loadMappings με την οποία μπορεί κανείς να φορτώσει της αντιστοιχήσεις που ορίζει σε ένα άλλο αρχείο κειμένου.

Ένας λιγότερο πεπειραμένος χρήστης, μπορεί να χρησιμοποιήσει το γραφικό περιβάλλον της εφαρμογής, που παρουσιάζεται στην ενότητα 2.2.3, προκειμένου να επιτύχει το παραπάνω

αποτέλεσμα. Αξίζει να σημειωθεί ότι η συγγραφή σωστών αρχείων για τον καθορισμό των presets και των mappings δεν είναι δύσκολη διαδικασία για τον χρήστη (αρκεί να μπορεί να δημιουργήσει ένα αρχείο κειμένου), και δεν απαιτεί γνώσεις προγραμματισμού σε Java.

2.2.3 Γραφική Διεπαφή Χρήστη

Η τελική εφαρμογή διαθέτει μια γραφική διεπαφή, μέσω της οποίας ο χρήστης μπορεί να διαχειριστεί της μονάδες παραγωγής του ήχου και τις αντιστοιχήσεις των παραμέτρων τους με τα δεδομένα που προσλαμβάνονται από τους αισθητήρες. Πιο συγκεκριμένα, ο χρήστης μπορεί είτε να φορτώσει ένα εξωτερικό preset αρχείο (βλ κεφάλαιο 2.2.2), είτε να δημιουργήσει το δικό του σύνολο από synthesizers. Στην Εικόνα 2.2.2 φαίνεται μία άποψη του γραφικού περιβάλλοντος, κατά τη διάρκεια δημιουργίας ενός νέου synthesizer διαμόρφωσης πλάτους (AM).



Εικόνα 2.2.2 Μία άποψη του γραφικού περιβάλλοντος της εφαρμογής

Επίσης, όσον αφορά τον έλεγχο των αντιστοιχίσεων, ο χρήστης μπορεί επίσης να φορτώσει ένα αρχείο mappings (βλ κεφάλαιο 2.2.2), είτε να δημιουργήσει τις δικές του αντιστοιχήσεις, τη στιγμή της εκτέλεσης. Τέλος, μέσω της γραφικής διεπαφής της εφαρμογής, ο χρήστης μπορεί να επισκοπήσει τη δραστηριότητα της ηχητικής εγκατάστασης που έχει δημιουργήσει, και να επέμβει στις τρέχουσες τιμές των παραμέτρων σύνθεσης, σε πραγματικό χρόνο.

Το γραφικό περιβάλλον είναι πολύ απλό και φιλικό καθώς στοχεύει στον μη πεπειραμένο χρήστη. Ωστόσο, λεπτομερείς οδηγίες χρήσης μπορούν να βρεθούν στο κεφάλαιο 3.2.2.

3 Εγχειρίδιο Χρήσης

Στις επόμενες ενότητες δίνονται λεπτομερείς οδηγίες για τον τρόπο χρήσης της τελικής εφαρμογής (Κεφάλαιο 3.2) και της προγραμματιστικής βιβλιοθήκης (Κεφάλαιο 3.3).

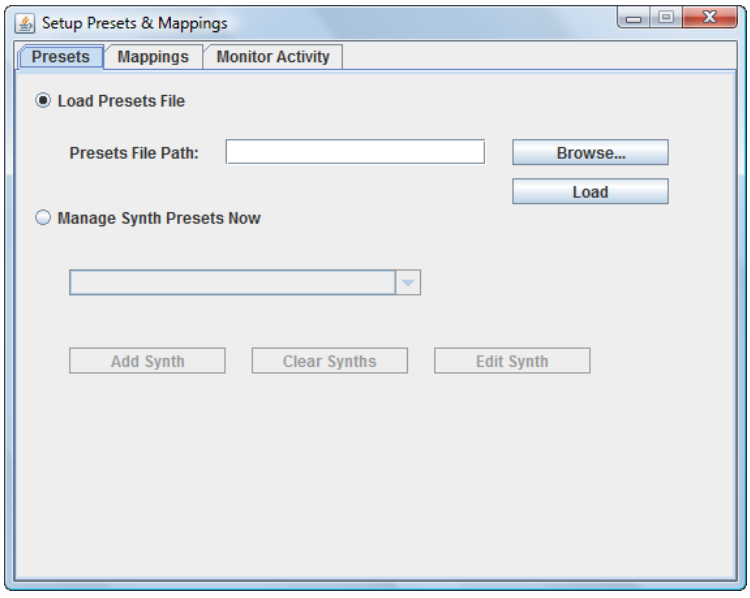
3.1 Διάθεση Λογισμικού

Η προγραμματιστική βιβλιοθήκη διατίθεται στο CD της πτυχιακής εργασίας, στο αρχείο sensorsyn.jar του φακέλου JARs. Ο προγραμματιστής πρέπει να εισάγει αυτό το αρχείο στο δικό του project, προκειμένου να χρησιμοποιήσει την προγραμματιστική βιβλιοθήκη. Στο περιβάλλον του Eclipse, αυτό γίνεται μέσα από το Properties dialog box του project. Από την επιλογή Java Build Path, κάνουμε κλικ στο button "Add external JARs" και μέσω του πλαισίου διαλόγου εντοπίζουμε το jar στο σύστημα αρχείων και επιλέγουμε Open.

Η τελική εφαρμογή διατίθεται σε εκτελέσιμο αρχείο jar . Η εκτέλεση της εφαρμογής γίνεται με διπλό κλικ στο αρχείο demo.jar, που βρίσκεται στο φάκελο JARs.

Επίσης, στη διαδρομή Source_Code/SensorSyn/src βρίσκεται ολόκληρος ο πηγαίος κώδικας της εργασίας. Υπενθυμίζεται ο τρόπος οργάνωσης των πακέτων σε διαφορετικά directories (βλ κεφάλαιο 2.2.1). Εάν αναζητούμε, για παράδειγμα, τον πηγαίο κώδικα της κλάσης FMSynthesizer θα πρέπει να ακολουθήσουμε την παρακάτω διαδρομή στον file explorer (από το root του CD-Rom): Source_Code/SensorSyn/src/gr/teicrete/mta/synthesizer/FMSynthesizer.java.

Τέλος, στη διαδρομή Source_Code/SensorSyn/doc/index.html βρίσκεται το javadoc του project. Το αρχείο ανοίγει με οποιονδήποτε internet browser.



Εικόνα 3.1.1 Η αρχική οθόνη της εφαρμογής.

3.2 Οδηγίες για απλούς χρήστες

Στο κεφάλαιο αυτό θα δοθούν οδηγίες για τη σύνδεση του hardware (Κεφάλαιο 3.2.1), για τη χρήση της γραφικής εφαρμογής (Κεφάλαιο 3.2.2) και οδηγίες συγγραφής των αρχείων πρωτοτύπων και αντιστοιχήσεων (Κεφάλαιο 3.2.3).

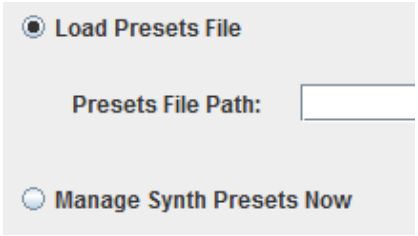
3.2.1 Σύνδεση υλικού

Στην Εικόνα 2.1.2 φαίνεται η λογική σύνδεσης του υλικού, προκειμένου να γίνει χρήση της γραφικής εφαρμογής. Ο ηλεκτρονικός υπολογιστής θα πρέπει να διαθέτει την εξωτερική κάρτα ήχου μAudio MIDISPORT 2x2, που διαθέτει θύρες MIDI IN/OUT. Μέσω αυτών των θυρών θα πρέπει να συνδεθεί η συσκευή Eobody της εταιρίας Eowave. Τέλος, στα 16 κανάλια του Eobody, ο χρήστης μπορεί να συνδέσει τους αισθητήρες της αρεσκείας του.

3.2.2 Γραφική Εφαρμογή

Με την εκκίνηση της εφαρμογής, εμφανίζεται ένα παράθυρο, όπως αυτό που φαίνεται στην Εικόνα 3.1.1. Τα βασικά στοιχεία που διακρίνονται είναι τα τρία θέματα (tabs) του παραθύρου διαλόγου (Presets, Mappings και Monitor Activity). Οποιοσδήποτε χρήστης έχει κάποια εμπειρία από τα παράθυρα διαλόγου των Windows, δεν θα έχει πρόβλημα να μετακινηθεί μεταξύ αυτών των θεμάτων, κάνοντας κλικ στο θέμα που τον ενδιαφέρει.

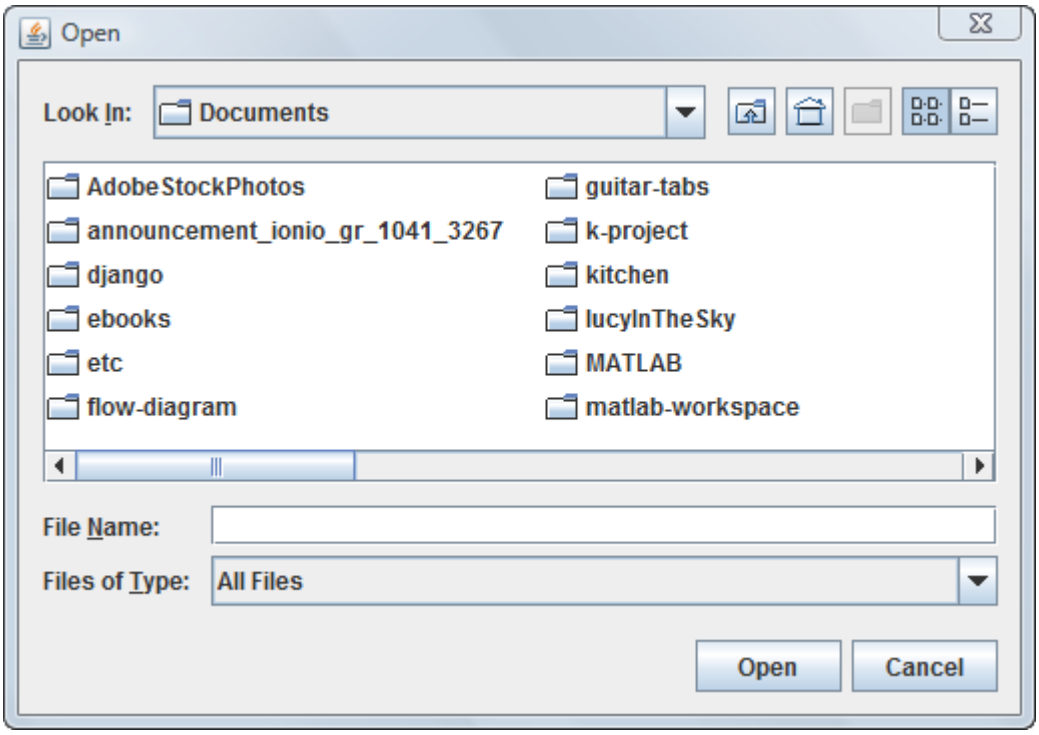
Το αρχικά επιλεγμένο θέμα είναι το Presets, έτσι στο υπόλοιπο του παραθύρου μπορούμε να δούμε όποιες επιλογές μας είναι διαθέσιμες και αφορούν τον έλεγχο των presets της παραγωγής του ήχου. Η πρώτη βασική απόφαση που πρέπει να λάβει ο χρήστης τη στιγμή που ξεκινάει τη ρύθμιση των γεννητριών του ήχου είναι το αν θα φορτώσει ένα έτοιμο αρχείο πρωτοτύπων (βλ Κεφάλαιο 2.2.2) ή αν προτιμάει να δημιουργήσει εξαρχής κάποιες γεννήτριες ήχου μέσα από το γραφικό περιβάλλον. Η επιλογή αυτή γίνεται μέσω των δύο radio buttons που βλέπουμε στην Εικόνα 3.2.1. Εάν ο χρήστης θέλει να φορτώσει ένα αρχείο πρωτοτύπων, τότε πρέπει να επιλέξει το πρώτο radio button: “Load Presets File”, διαφορετικά πρέπει να επιλέξει το “Manage Synth Presets Now”.



Εικόνα 3.2.1 Οι δύο βασικές επιλογές για τον τρόπο ελέγχου των presets.

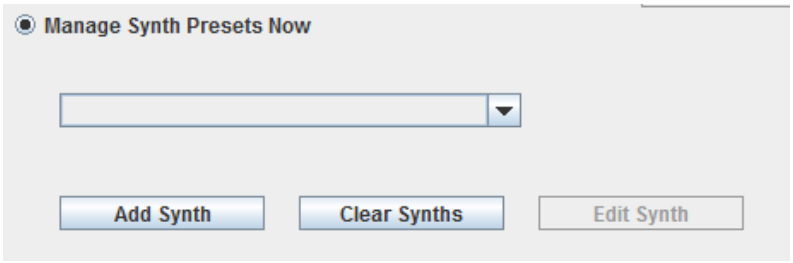
Αρχικά, κάνοντας την υπόθεση ότι ο χρήστης διαθέτει ένα έτοιμο αρχείο presets (οδηγίες για τη συγγραφή των αρχείων preset στη συνέχεια του κεφαλαίου), θα δοθούν οδηγίες για τη φόρτωση του αρχείου αυτού. Αφού επιλεχθεί το “Load Presets File”, θα πρέπει να δώσει τη διαδρομή του αρχείου στο πλαίσιο κειμένου που βρίσκεται δίπλα στην ετικέτα “Presets File Path:” (Εικόνα 3.2.1). Εάν το αρχείο βρίσκεται στον ίδιο φάκελο με την εφαρμογή, τότε αρκεί το όνομα του αρχείου. Εναλλακτικά (και πιο εύκολα, ίσως) ο χρήστης μπορεί να κάνει κλικ στο κουμπί “Browse...”. Ένα πλαίσιο διαλόγου όπως αυτό στην Εικόνα 3.2.2 εμφανίζεται για να εντοπιστεί το αρχείο από το χρήστη. Το πλαίσιο διαλόγου είναι ένα κοινό πλαίσιο διαλόγου των Windows, συνεπώς ο συγγραφέας θεωρεί δεδομένη την εξοικείωση του χρήστη με αυτό. Μόλις το αρχείο εντοπιστεί και ο χρήστης κάνει κλικ στο “Open”, το path του

αρχείου εμφανίζεται αυτόματα στο πλαίσιο κειμένου. Προκειμένου ο χρήστης να ακούσει τις ρυθμίσεις που ορίζει μέσω του preset file που μόλις υπέδειξε, θα πρέπει να κάνει κλικ στο "Load", έτσι ώστε το πρόγραμμα να φορτώσει το αρχείο και να το διερμηνεύσει, προκειμένου να



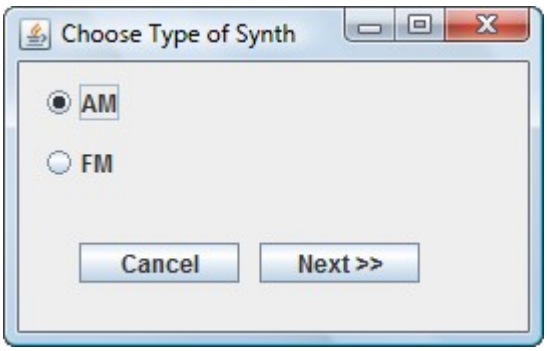
Εικόνα 3.2.2 Το πλαίσιο διαλόγου για την επιλογή του preset file

δημιουργήσει τους κατάλληλους synthesizers. Στο CD που δίνεται μαζί με την πτυχιακή εργασία, στη διαδρομή JARs/presets_mappings/ βρίσκεται το αρχείο am_fm_presets.txt. Στο αρχείο αυτό ορίζεται ένας τυπικός AM και ένας τυπικός FM synthesizer. Για να ελέγξει τη λειτουργία της εφαρμογής, ο χρήστης μπορεί να δοκιμάσει να φορτώσει αρχικά αυτό το αρχείο. Αφού κάνει κλικ στο button "Load" θα πρέπει να ακούσει το αποτέλεσμα μιας AM και μιας FM από τα ηχεία του υπολογιστή του.



Εικόνα 3.2.3 Οι επιλογές που αφορούν τη διαχείριση των Synth Presets είναι τώρα ενεργοποιημένες.

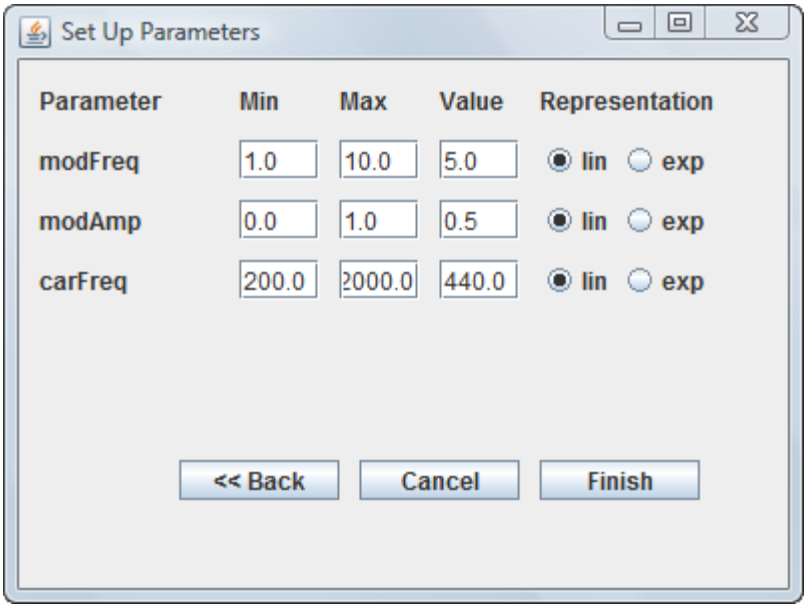
Στην περίπτωση που ο χρήστης προτιμάει να δημιουργήσει εκ νέου τις μονάδες παραγωγής του ήχου, θα πρέπει αρχικά να επιλέξει το δεύτερο radio button "Manage Synth Presets Now". Εάν είχε νωρίτερα ακολουθήσει τα βήματα που περιγράφονται παραπάνω και ο υπολογιστής του παρήγαγε τον ήχο από μια AM και μια FM, τη στιγμή που θα επιλέξει το "Manage Synth Presets Now" ο ήχος θα πρέπει να σταματήσει. Επίσης, με την επιλογή του δεύτερου radio button οι επιλογές που πριν ήταν απενεργοποιημένες (Εικόνα 3.2.3), θα πρέπει τώρα να είναι ενεργοποιημένες. Αντίθετα, οι επιλογές που αφορούν τη φόρτωση του αρχείου θα πρέπει τώρα να



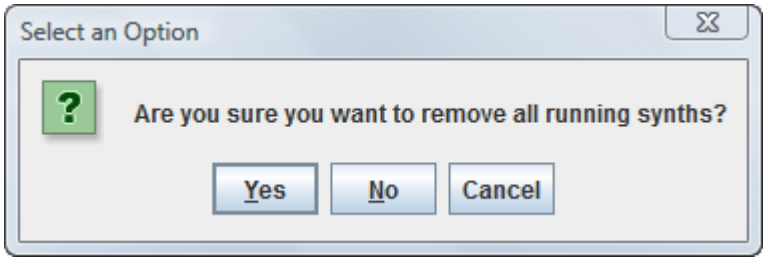
Εικόνα 3.2.4 Το πλαίσιο διαλόγου "Choose Type of Synth"

έχουν απενεργοποιηθεί. Διακρίνουμε μια αναδυόμενη λίστα (η οποία αρχικά είναι κενή – στη συνέχεια θα περιέχει τους synthesizer που θα δημιουργούνται από το χρήστη), ένα κουμπί "Add Synth", ένα κουμπί "Clear Synths" και ένα κουμπί "Edit Synth" (το τελευταίο είναι απενεργοποιημένο, αφού αφορά τον synthesizer που είναι επιλεγμένος στην αναδυόμενη λίστα). Κάνοντας κλικ στο κουμπί "Add Synth", ο χρήστης ξεκινάει τον οδηγό για τη δημιουργία ενός νέου synthesizer. Ένα καινούργιο πλαίσιο διαλόγου εμφανίζεται όπου ο χρήστης καλείται να επιλέξει εάν ο synthesizer που θέλει να δημιουργήσει θα είναι τύπου AM ή FM (βλ Εικόνα 3.2.4).

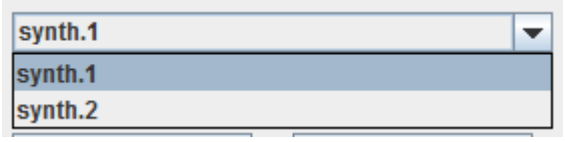
Αφού ο χρήστης επιλέξει τύπο, με τη χρήση των δύο radio buttons, θα πρέπει να κάνει κλικ στο "Next", ώστε να μεταβεί στο δεύτερο βήμα της δημιουργίας του καινούργιου synthesizer. Ένα νέο πλαίσιο διαλόγου εμφανίζεται (Εικόνα 3.2.5), στο οποίο ο χρήστης καλείται να δώσει τιμές σε όλες τις παραμέτρους που απαιτούνται για τη δημιουργία του τύπου του synthesizer που επιλέχθηκε στο προηγούμενο βήμα. Στην Εικόνα 3.2.5, φαίνονται οι παράμετροι ενός AM synthesizer. Η παράμετρος "modFreq" είναι η συχνότητα του διαμορφωτή (modulator frequency), η "modAmp" είναι το πλάτος ταλάντωσης του διαμορφωτή (modulator amplitude). Η παράμετρος "carFreq" είναι η συχνότητα ταλάντωσης του φορέα (carrier frequency). Οι παραπάνω παράμετροι είναι απαραίτητες για τη δημιουργία ενός synthesizer τύπου AM. Η στήλη "Min" αφορά τις ελάχιστες τιμές που μπορεί να πάρουν οι εκάστοτε παράμετροι, η στήλη "Max" τις μέγιστες. Η στήλη "Value" αφορά τις αρχικές τιμές που θα λάβουν οι παράμετροι με τη δημιουργία του synthesizer. Η αρχική τιμή μιας παραμέτρου ενός synthesizer θα είναι και η μόνιμη, σταθερή τιμή της παραμέτρου αυτής, εφόσον η τελευταία δεν αντιστοιχηθεί με κάποια παράμετρο των χειρονομιών του εκτελεστή (βλ Κεφάλαιο 1.4). Τέλος, η στήλη "Representation" που για κάθε παράμετρο διαθέτει δύο επιλογές -lin (linear) και exp (exponential)-, αφορά την κλίμακα στην οποία το εύρος των τιμών της παραμέτρου διατάσσονται. Η επιλογή αυτή βασίζεται στον τρόπο με τον οποίο η ανθρώπινη ακοή αντιλαμβάνεται κάποια μεγέθη. Πιο συγκεκριμένα, θα είχε νόημα μια παράμετρος που αφορά το τονικό ύψος (όπως η carFreq), να έχει εκθετική (exponential) αναπαράσταση, καθώς προσεγγίζει καλύτερα την ανθρώπινη αντίληψη του τονικού ύψους. Με αυτόν τον τρόπο, εάν αυτή η παράμετρος αντιστοιχηθεί με έναν αισθητήρα πίεσης, για παράδειγμα,



Εικόνα 3.2.5 Το πλαίσιο διαλόγου "Set Up Parameters"



Εικόνα 3.2.6 Προειδοποιητικό μήνυμα για τη διαγραφή των synthesizer



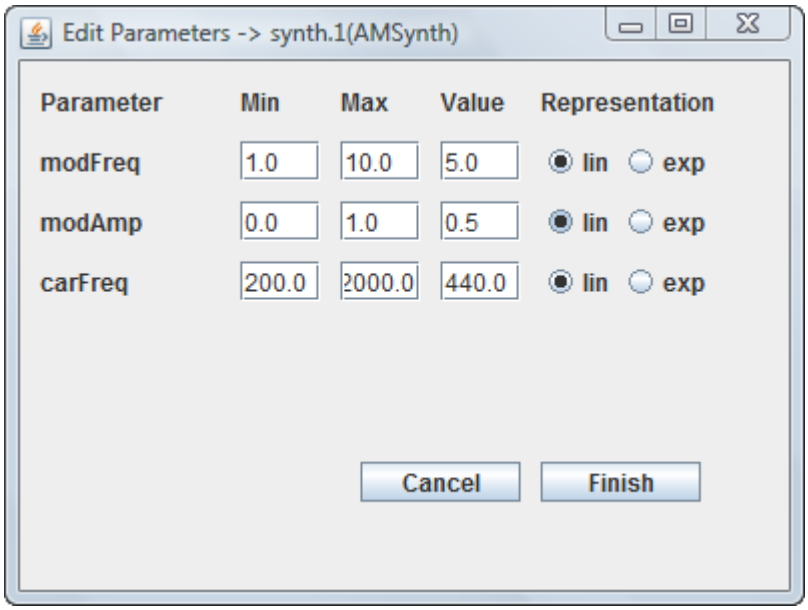
Εικόνα 3.2.7 Η λίστα με τα synthesizer

διπλασιάζοντας ο εκτελεστής την πίεση σε αυτόν τον αισθητήρα, θα αντιλαμβανόταν και διπλασιασμό στο τονικό ύψος.

Τα πλαίσια κειμένου είναι εξαρχής γεμισμένα με κάποιες τυπικές τιμές για έναν AM synthesizer, έτσι ο χρήστης αν το επιθυμεί μπορεί να κάνει κατευθείαν κλικ στο "Finish" για να ακούσει το ηχητικό αποτέλεσμα του synthesizer που μόλις δημιούργησε (με λίγη βοήθεια, ομολογουμένως). Αν οι εξορισμού τιμές δεν τον ικανοποιούν, μπορεί να δώσει τις δικές του στα σωστά πλαίσια πριν κάνει κλικ στο "Finish". Ένα σημείο που θα πρέπει να προσεχθεί ιδιαίτερα σε έναν AM synthesizer είναι το πλάτος του διαμορφωτή. Εάν ο χρήστης έχει σκοπό να δημιουργήσει πάνω από ένα synthesizer (κάτι που είναι πολύ πιθανό), τότε θα πρέπει να προσέξει το άθροισμα των modAmp να μην ξεπερνάει τη μονάδα, γιατί ο παραγόμενος ήχος θα έχει clip, ένα αποτέλεσμα πολύ δυσάρεστο. Το clip συμβαίνει όταν μια τιμή ξεπερνάει το εύρος στο οποίο μπορεί να κινηθεί, ένα μέγεθος που καθορίζεται από τον αριθμό των bits που χρησιμοποιεί κάθε υπολογιστής για την αποθήκευση της τιμής αυτής.

Κάνοντας κλικ στο "Finish" το synthesizer αρχίζει να παράγει ήχο. Επίσης, η αναδυόμενη λίστα έχει πλέον μια επιλογή: "synth.1", καθώς είναι το πρώτο synthesizer που έχει δημιουργηθεί. Στην Εικόνα 3.2.7 βλέπουμε την αναδυόμενη λίστα ύστερα από τη δημιουργία δύο synthesizer. Με κλικ στο κουμπί "Clear Synths", εμφανίζεται το προειδοποιητικό μήνυμα που φαίνεται στην Εικόνα 3.2.6. Κάνοντας κλικ στο "Yes", διαγράφονται οι synthesizer που έχουν δημιουργηθεί.

Επιλέγοντας ένα synthesizer από την αναδυόμενη λίστα και κάνοντας κλικ στο κουμπί "Edit Synth" (που είναι πλέον ενεργοποιημένο), μπορεί ο χρήστης να μεταβάλλει τις παραμέτρους του synthesizer, μέσω του πλαισίου διαλόγου που εμφανίζεται (Εικόνα 3.2.8).

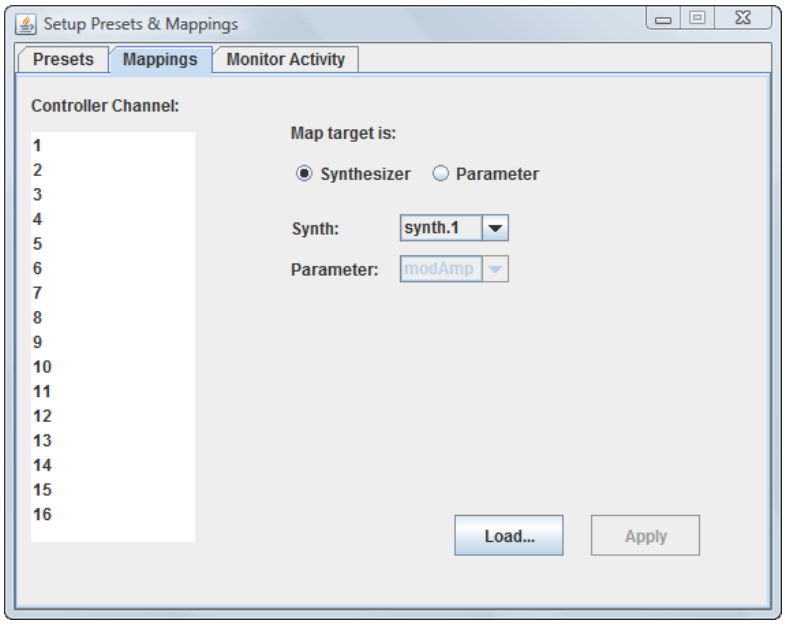


Εικόνα 3.2.8 Το πλαίσιο διαλόγου "Edit Parameters"

Από τη στιγμή που ο χρήστης έχει ρυθμίσει όλα τα synthesizer που θα λειτουργούν, είτε μέσω ενός preset file, είτε ρυθμίζοντάς τα μέσω του γραφικού περιβάλλοντος, θα πρέπει να δημιουργήσει τις κατάλληλες αντιστοιχήσεις προκειμένου να υλοποιήσει την ιδέα που είχε θέσει εξαρχής ως στόχο. Για να γίνει αυτό πρέπει να κάνει κλικ στο δεύτερο θέμα του βασικού παραθύρου, ονόματι "Mappings".

Κάνοντας κλικ στο δεύτερο θέμα, τα περιεχόμενα του παραθύρου αλλάζουν και εμφανίζονται οι ρυθμίσεις που αφορούν τις αντιστοιχήσεις. Όπως φαίνεται στην Εικόνα 3.2.9, στα αριστερά του πλαισίου υπάρχει μια λίστα με όλα τα κανάλια του Eobody (16). Επιλέγοντας ένα από

αυτά, σημαίνει ότι όποιες αλλαγές πραγματοποιηθούν στο δεξί μέρος του πλαισίου θα αφορούν το επιλεγμένο κανάλι (αφού ο χρήστης κάνει κλικ στο "Apply"). Όταν ο χρήστης επιλέξει κάποιο κανάλι, το κουμπί "Apply" γίνεται ενεργό. Ο χρήστης πρώτα θα πρέπει να επιλέξει εάν ο στόχος της αντιστοίχησης θα είναι κάποιο synthesizer ή κάποια παράμετρος ενός synthesizer. Στην πρώτη περίπτωση, τα MIDI μηνύματα που θα στέλνονται μέσω του συγκεκριμένου καναλιού θα λειτουργούν ως toggle (on/off) για τον επιλεγμένο synthesizer. Στη δεύτερη περίπτωση, κάποιο data byte του MIDI μηνύματος θα ελέγχει αριθμητικά την παράμετρο που έχει επιλεχθεί. Η επιλογή αυτή γίνεται με τη χρήση των radio buttons που βρίσκονται κάτω από την ετικέτα "Map target is:". Αρχικά είναι επιλεγμένο το "synthesizer".



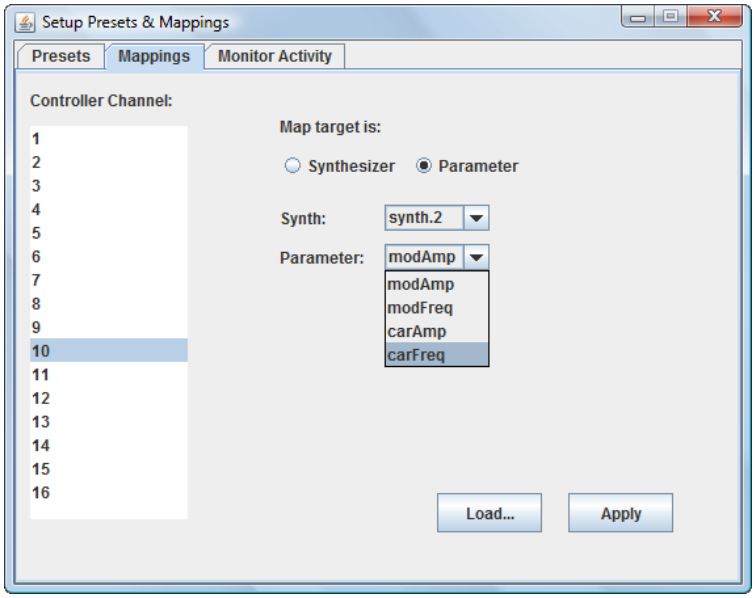
Εικόνα 3.2.9 Το θέμα "Mappings", που αφορά τον έλεγχο των αντιστοιχίσεων

Ακριβώς από κάτω, βρίσκονται δύο αναδυόμενες λίστες (Εικόνα 3.2.10), που χρησιμεύουν για την επιλογή του στόχου της αντιστοίχησης. Εάν ο χρήστης έχει επιλέξει σαν τύπο map target το synthesizer, τότε η δεύτερη λίστα που αφορά τις παραμέτρους των synthesizer θα είναι απενεργοποιημένη.

Εικόνα 3.2.10 Οι δύο λίστες για την επιλογή του map target

Προκειμένου να γίνει καλύτερα κατανοητή η διαδικασία ρύθμισης των αντιστοιχίσεων, κάνουμε μια υπόθεση: Ο χρήστης έχει δημιουργήσει (είτε μέσω γραφικού περιβάλλοντος, είτε μέσω ενός preset file) δύο synthesizers: το synth.1, τύπου AM και το synth.2, τύπου FM. Ύστερα, κάνει κλικ στο δεύτερο θέμα, "Mappings". Έστω ότι έχει συνδέσει έναν αισθητήρα κίνησης στο κανάλι 1 του Eobody και θέλει όποτε ο αισθητήρας εντοπίζει κίνηση, να ενεργοποιείται (αν πριν ήταν ανενεργός) ή να απενεργοποιείται (αν πριν ήταν ενεργός) ο AM synthesizer. Θα πρέπει καταρχάς να επιλέξει το Controller Channel 1. Εφόσον εξορισμού είναι επιλεγμένο ως map target το synthesizer, και στην αναδυόμενη λίστα του Synth είναι εξορισμού επιλεγμένο το synth.1, το μόνο που μένει είναι να κάνει κλικ στο "Apply". Έστω τώρα ότι με ένα δεύτερο αισθητήρα, που έχει συνδέσει στο κανάλι 2, θέλει να ελέγχει με τον ίδιο τρόπο τον FM synthesizer. Θα πρέπει να επιλέξει το Controller Channel 2. Ύστερα, από την αναδυόμενη λίστα "Synth" θα πρέπει να επιλέξει το synth.2, εφόσον ο synth.2 είναι ο synthesizer που θέλει να ορίσει ως map target για το controller channel 2. Τέλος, πρέπει να κάνει κλικ στο "Apply" για να ισχύσουν οι ρυθμίσεις του. Τέλος, έστω ότι στο κανάλι 10 έχει συνδέσει έναν αισθητήρα πίεσης και θέλει μέσω αυτού να ελέγχει τη συχνότητα του φορέα του FM synthesizer. Ακολουθώντας την ίδια λογική, θα πρέπει να επιλέξει το controller channel 10 από τη λίστα στα αριστερά. Ύστερα, θα πρέπει να επιλέξει σαν map target την επιλογή "Parameter", αφού τώρα θέλει να αντιστοιχήσει το κανάλι 10 με μια παράμετρο ενός synthesizer και όχι με το ίδιο το synthesizer. Ύστερα, θα πρέπει να επιλέξει το synth.2 από την αναδυόμενη λίστα του Synth, για να δηλώσει ότι η παράμετρος που θέλει να αντιστοιχήσει ανήκει στον FM synthesizer. Αφού γίνει αυτό, από την αναδυόμενη λίστα του

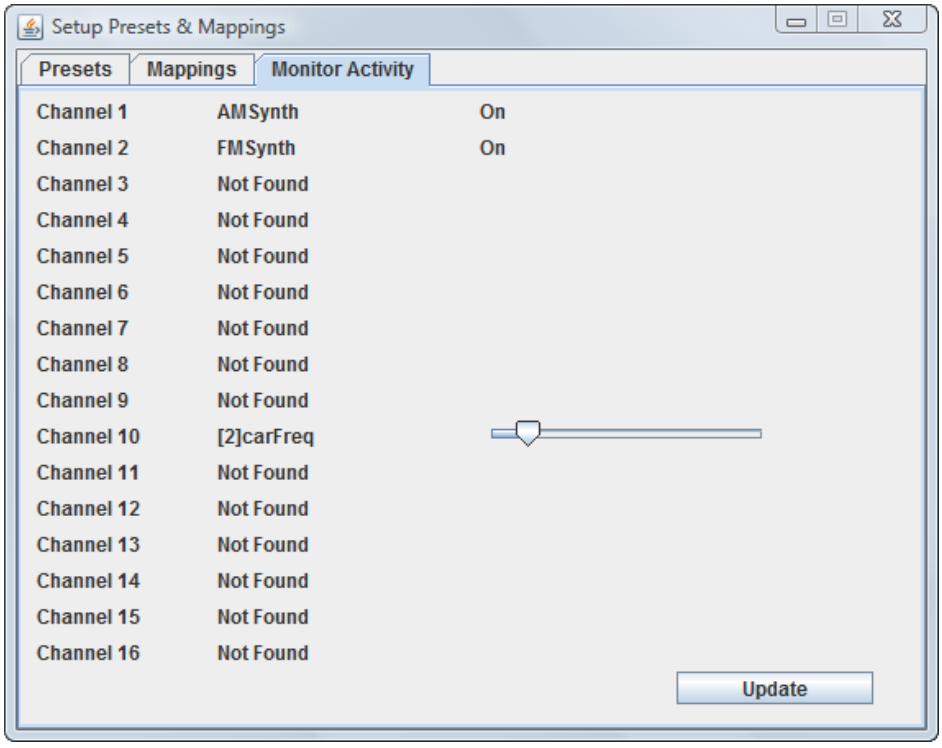
"Parameter" (που τώρα είναι ενεργοποιημένη) θα πρέπει να επιλέξει την παράμετρο "carFreq" που αντιπροσωπεύει τη συχνότητα του φορέα (Εικόνα 3.2.11). Τέλος, θα πρέπει για μια ακόμα φορά να κάνει κλικ στο "Apply" προκειμένου να λάβει ισχύ και αυτή η αντιστοίχηση.



Εικόνα 3.2.11 Αντιστοιχίζοντας την παράμετρο carFreq του synth.2 στο controller channel 10.

Όπως και για τη ρύθμιση των presets, έτσι και στην περίπτωση των mappings, υπάρχει η δυνατότητα φόρτωσης ενός έτοιμου αρχείου mappings (οδηγίες για τη συγγραφή τους στη συνέχεια του κεφαλαίου). Για να φορτώσει ένα αρχείο mappings, ο χρήστης θα πρέπει να κάνει κλικ στο κουμπί "Load...", που βρίσκεται δίπλα στο "Apply", για να εμφανιστεί ένα πλαίσιο διαλόγου όπως αυτό στην Εικόνα 3.2.2, μέσω του οποίου ο μπορεί να εντοπίσει το αρχείο των αντιστοιχήσεων στο σύστημα αρχείων.

Από τη στιγμή που ο χρήστης έχει ορίσει το preset και όλες τις αντιστοιχήσεις, μπορεί πλέον να δοκιμάσει την ηχητική εγκατάσταση που μόλις προγραμμάτισε, αλληλεπιδρώντας με τους αισθητήρες. Επιπλέον, μπορεί να του είναι χρήσιμη η οπτική ανατροφοδότηση που του παρέχει το τρίτο θέμα, "Monitor Activity". Κάνοντας κλικ σε αυτό το θέμα, ο χρήστης μπορεί να δει μια καρτέλα που λειτουργεί σαν monitor για την επίβλεψη της δραστηριότητας της διαδραστικής



Εικόνα 3.2.12 Μέσω αυτής της καρτέλας ο χρήστης μπορεί να παρακολουθεί τη δραστηριότητα της διαδραστικής εγκατάστασης σε πραγματικό χρόνο.

ηχητικής εγκατάστασης σε πραγματικό χρόνο. Προκειμένου να γίνει καλύτερα κατανοητό, μπορούμε να επιστρέψουμε στην περίπτωση του χρήστη που αναφέρθηκε παραπάνω: Ο χρήστης έχει δημιουργήσει δύο synthesizer, το synth.1 που είναι τύπου AM και το synth.2 που είναι τύπου FM. Έχει ρυθμίσει τις αντιστοιχήσεις έτσι ώστε ο αισθητήρας που είναι συνδεδεμένος στο κανάλι 1 του Eobody να λειτουργεί σαν toggle για τον synth.1, ο αισθητήρας που είναι συνδεδεμένος στο κανάλι 2 να λειτουργεί σαν toggle για τον synth.2, ενώ ο αισθητήρας που είναι συνδεδεμένος στο κανάλι 10 ρυθμίζει τη συχνότητα του φορέα του synth.2. Εάν κάνει κλικ στο θέμα "Monitor Activity", θα δει μια εικόνα παρόμοια με την Εικόνα 3.2.12. Από αυτήν ο χρήστης μπορεί να πληροφορηθεί ότι ο AM synthesizer είναι αντιστοιχισμένος με το κανάλι 1, ενώ ο FM synthesizer με το κανάλι 2. Το κανάλι 10 ελέγχει την παράμετρο carFreq. Εάν ο χρήστης, για παράδειγμα, ασκήσει πίεση στον αισθητήρα πίεσης που είναι συνδεδεμένος στο κανάλι 10 θα δει τον slider της παραμέτρου carFreq να μετατοπίζεται προς τα δεξιά (ενώ παράλληλα θα πρέπει να αντιλαμβάνεται ακουστικά μια μετατόπιση του τονικού ύψους σε ψηλότερες συχνότητες), υποδεικνύοντας την μεταβολή που συμβαίνει real time στην παράμετρο αυτή.

Μια επιπλέον δυνατότητα που παρέχει η καρτέλα αυτή είναι η δυνατότητα χειροκίνητης ρύθμισης της τρέχουσας τιμής κάποιας παραμέτρου (κάτι που μπορεί επίσης να γίνει επιλέγοντας "Edit Synth" από το θέμα "Presets"), μετατοπίζοντας τον κατάλληλο slider και κάνοντας κλικ στο κουμπί "Update" που βρίσκεται κάτω δεξιά του παραθύρου.

3.2.3 Συγγραφή αρχείων πρωτοτύπων και αντιστοιχήσεων

Ο αναγνώστης που στάθηκε συνεπής στην ανάγνωση του προηγούμενου κεφαλαίου θα γνωρίζει ήδη πως να φορτώσει ένα αρχείο με δικά του presets κατευθείαν στη γραφική εφαρμογή. Προκειμένου να εξοικειωθεί με τη συγγραφή των αρχείων αυτών παραθέτεται το αρχείο am_fm_presetes.txt που βρίσκεται στο CD της πτυχιακής εργασίας και επεξήγησή του:

```
1.  synth.1.name=AMSynthesizer
2.  synth.1.param.1.name=modAmp
3.  synth.1.param.1.min=0
4.  synth.1.param.1.max=1
5.  synth.1.param.1.representation=linear
6.  synth.1.param.1.defaultvalue=0.3
7.  synth.1.param.2.name=modFreq
8.  synth.1.param.2.min=1
9.  synth.1.param.2.max=10
10. synth.1.param.2.representation=exponential
11. synth.1.param.2.defaultvalue=8
12. synth.1.param.3.name=carFreq
13. synth.1.param.3.min=200
14. synth.1.param.3.max=2000
15. synth.1.param.3.representation=exponential
```

```
16.  synth.1.param.3.defaultvalue=500
17.
18.  synth.2.name=FMSynthesizer
19.  synth.2.param.1.name=modAmp
20.  synth.2.param.1.min=0
21.  synth.2.param.1.max=60
22.  synth.2.param.1.representation=linear
23.  synth.2.param.1.defaultvalue=50
24.  synth.2.param.2.name=modFreq
25.  synth.2.param.2.min=10
26.  synth.2.param.2.max=100
27.  synth.2.param.2.representation=linear
28.  synth.2.param.2.defaultvalue=50
29.  synth.2.param.3.name=carAmp
30.  synth.2.param.3.min=0
31.  synth.2.param.3.max=1
32.  synth.2.param.3.representation=linear
33.  synth.2.param.3.defaultvalue=0.3
34.  synth.2.param.4.name=carFreq
35.  synth.2.param.4.min=200
36.  synth.2.param.4.max=2000
37.  synth.2.param.4.representation=linear
38.  synth.2.param.4.defaultvalue=500
```

Στις γραμμές 1-16 ορίζεται ένας AM Synthesizer. Στη γραμμή 1 δηλώνεται ότι πρόκειται για synthesizer τύπου AM. Στη γραμμή 2 δηλώνεται ότι η πρώτη παράμετρος είναι η παράμετρος modAmp. Στις γραμμές 3-6 ορίζονται οι ιδιότητες της παραμέτρου modAmp του synthesizer. Στη γραμμή 3 ορίζεται η ελάχιστη τιμή, στη γραμμή 4 η μέγιστη. Στη γραμμή 5 δηλώνεται ότι η αναπαράσταση της παραμέτρου θα είναι γραμμική και στη γραμμή 6 δίνεται η εξ' ορισμού τιμή της παραμέτρου. Με τον ίδιο ακριβώς τρόπο, στις γραμμές 7-11 ορίζεται η παράμετρος modFreq, σαν η δεύτερη παράμετρος του AM synthesizer, και στις γραμμές 12-16 ορίζεται η παράμετρος carFreq σαν η τρίτη παράμετρος του AM synthesizer. Στις γραμμές 18-38 ορίζεται ένας FM Synthesizer, ακολουθώντας την ίδια λογική, με τη μόνη διαφορά ότι χρειάζεται να οριστούν τέσσερις παράμετροι για έναν FM synthesizer.

Γενικεύοντας, θα πρέπει να συνοψιστούν οι εξής οδηγίες:

- Ο χρήστης μπορεί να δηλώνει ένα καινούργιο synthesizer με τον εξής τρόπο:

```
synth.(αριθμός).name=(τύπος)
```

Ο αριθμός θα πρέπει να είναι ακέραιος και να έχει αύξουσα λογική, δηλαδή δεν γίνεται να οριστεί ο synthesizer νούμερο 4 αν δεν έχει οριστεί ο synthesizer νούμερο 3. Η δήλωση του synthesizer νούμερο 3 δεν είναι απαραίτητο να προηγείται στο αρχείο κειμένου. Θα μπορούσε να βρίσκεται χαμηλότερα. Ωστόσο, για λόγους σαφήνειας προτείνεται ο ορισμός των synthesizer με κάποια σειρά. Δεν υπάρχει περιορισμός όσον αφορά τον αριθμό των synthesizer που μπορεί να ορίσει ο χρήστης.

Ο τύπος θα πρέπει να είναι κάποιος έγκυρος τύπος synthesizer (AMSynthesizer ή FMSynthesizer). Δεν θα πρέπει να παρεμβάλλεται κενό στη λέξη του ονόματος.

- Οι παράμετροι ενός synthesizer ορίζονται με τον εξής τρόπο:

synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου).name=(όνομα παραμέτρου)

Ο αριθμός synthesizer θα πρέπει να ανταποκρίνεται σε κάποιον synthesizer που ορίζεται από το χρήστη στο αρχείο κειμένου, διαφορετικά η δήλωση της παραμέτρου θα αγνοηθεί. Δεν είναι απαραίτητο οι δηλώσεις των παραμέτρων να έπονται των δηλώσεων των synthesizer, ωστόσο και πάλι προτείνεται να ακολουθηθεί αυτή η πρακτική για λόγους σαφήνειας.

Ο αριθμός παραμέτρου έχει τους ίδιους κανόνες με τον αριθμό synthesizer. Πρέπει να είναι ακέραιος και να έχει αύξουσα λογική, συνεπώς δεν γίνεται να οριστεί η παράμετρος 3 αν δεν έχει οριστεί η παράμετρος 2.

Το όνομα παραμέτρου θα πρέπει να ανταποκρίνεται σε κάποια από τα ονόματα παραμέτρων που αντιστοιχούν για τον τύπο του synthesizer: modFreq, modAmp και carFreq για ένα synthesizer τύπου AM, modFreq, modAmp, carFreq και modFreq για ένα synthesizer τύπου FM.

- Η ελάχιστη τιμή που μπορεί να λάβει μια παράμετρος ορίζεται με τον εξής τρόπο:

synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου).min=(τιμή)

Για τον αριθμό synthesizer και τον αριθμό παραμέτρου ισχύει ότι προαναφέρθηκε.

Η τιμή μπορεί να είναι και δεκαδική.

- Με παρόμοιο τρόπο ορίζεται και η μέγιστη τιμή μιας παραμέτρου:

synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου).max=(τιμή)

- Καθώς και η εξ' ορισμού τιμή:

synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου).defaultvalue=(τιμή)

- Τέλος, ο τρόπος αναπαράστασης ορίζεται με τον ακόλουθο τρόπο:

synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου).representation=(linear ή exponential)

Εάν ο αναγνώστης διάβασε με υπομονή τις παραπάνω διευκρινήσεις δεν θα έχει κανένα πρόβλημα στην κατανόηση των γραμμών 18-38, όπου ορίζεται ένας FM Synthesizer, ούτε στη συγγραφή ενός δικού του αρχείου preset.

Μετά τη συγγραφή ενός αρχείου preset, είναι αρκετά λογικό ο χρήστης να επιθυμεί να συγγράψει και ένα αρχείο mappings, έτσι ώστε να ολοκληρώσει την διαδραστική εγκατάσταση που έχει οραματιστεί. Παρακάτω παραθέτεται το αρχείο mappings.txt που συμπεριλαμβάνεται στο CD της πτυχιακής εργασίας:

```
1.  // channel 1 toggle for synth 1 (AMSynthesizer)
2.  eobody.channel.1=synth.1
```

```
3.
4.  // channel 10 controls the the param 3 (modulator frequency) of the AMSynth
5.  eobody.channel.10=synth.1.param.3
```

Η γραμμή 1 και η γραμμή 4 είναι *σχόλια (comments)*. Ο χρήστης μπορεί να συμπεριλάβει σχόλια στα αρχεία preset και mappings, έτσι ώστε να τον βοηθήσουν να θυμηθεί πιο εύκολα στο μέλλον τι έχει κάνει. Τα σχόλια σηματοδοτούνται από δύο ανάποδες κάθετες (//). Εάν σε μια γραμμή ο χρήστης πληκτρολογήσει δύο ανάποδες κάθετες, η συνέχεια της γραμμής αποτελεί σχόλιο και δεν έχει καμία σημασία για τον ορισμό των presets ή των mappings. Στα σχόλια μπορεί κανείς να γράψει σε ελεύθερη γλώσσα μια επεξήγηση της λειτουργικότητας ενός σημείου του αρχείου. Για παράδειγμα, στο σχόλιο της γραμμής 1 εξηγείται η λειτουργία της παρακάτω γραμμής (γραμμή 2). Συγκεκριμένα, στη γραμμή 2 δηλώνεται η αντιστοίχηση του καναλιού 1 με τον synthesizer 1. Υπενθυμίζεται ότι όταν αντιστοιχίζεται ένα κανάλι με έναν synthesizer, πρόκειται για αντιστοίχηση τύπου toggle.

Αντίθετα, στη γραμμή 5 δηλώνεται μια αντιστοίχηση με μια παράμετρο ενός synthesizer. Πιο συγκεκριμένα, αντιστοιχίζεται το κανάλι 10 με την παράμετρο 3 του synthesizer 1. Θα πρέπει εδώ να σημειωθεί ότι τα σχόλια είναι γραμμένα με τη λογική ότι το αντίστοιχο preset file που χρησιμοποιείται είναι το am_fm_presets.txt, για αυτό και ο synthesizer 1 αναφέρεται ως AM synth. Ο χρήστης μπορεί να χρησιμοποιήσει αυτό το αρχείο mappings σε συνδυασμό με κάποιο άλλο preset file, κάτι απόλυτα θεμιτό. Σε αυτήν την περίπτωση, δεν είναι απαραίτητο ότι ο synth.1 θα είναι AM synthesizer.

Η λογική των αντιστοιχήσεων στο παραπάνω παράδειγμα θα πρέπει να είναι αρκετά προφανής, ωστόσο, για λόγους συνέπειας ακολουθούν οδηγίες:

- Οι αντιστοιχήσεις καναλιού σε synthesizer γίνονται με την παρακάτω δήλωση:

eobody.channel.(αριθμός καναλιού)=synth.(αριθμός synthesizer)

Ο αριθμός καναλιού θα πρέπει να είναι ένας ακέραιος από 1 εώς και 16 και αντιπροσωπεύει το κανάλι του Eobody στο οποίο είναι συνδεδεμένος ο αισθητήρας που θέλουμε να συμμετέχει στην αντιστοίχηση.

Ο αριθμός synthesizer θα πρέπει να είναι ακέραιος και να ανταποκρίνεται σε κάποιον synthesizer στο preset file ή στους synthesizer που έχουν οριστεί στη γραφική εφαρμογή[2].

- Οι αντιστοιχήσεις καναλιού προς κάποια παράμετρο γίνονται με την παρακάτω δήλωση:

eobody.channel.(αριθμός καναλιού)=synth.(αριθμός synthesizer).param.(αριθμός παραμέτρου)

Για τον αριθμό καναλιού και τον αριθμό synthesizer ισχύουν οι ίδιοι νόμοι που περιγράφηκαν παραπάνω. Ο αριθμός παραμέτρου θα πρέπει να είναι ένας ακέραιος που να αντιστοιχεί σε κάποια ορισμένη παράμετρο, είτε στο preset file, είτε στη γραφική εφαρμογή.

Κλείνοντας αυτό το κεφάλαιο και την επεξήγηση των preset και mappings αρχείων, ζητείται από τον αναγνώστη να υποθέσει ότι κάποιος θέλει να πετύχει το εξής αποτέλεσμα: Θέλει να δημιουργήσει έναν AM synthesizer και να συνδέσει έναν αισθητήρα ανίχνευσης κίνησης στο κανάλι 1 που θα ανοίγει και θα κλείνει τον synthesizer, και έναν αισθητήρα επιτάχυνσης στο κανάλι 2 που θα ελέγχει τη συχνότητα του διαμορφωτή (την παράμετρο modFreq), με τη χρήση των preset

2 Παρόλο που στο κείμενο δεν έχει δηλωθεί ξεκάθαρα, υπάρχει η δυνατότητα ο χρήστης να χρησιμοποιήσει ένα αρχείο αντιστοιχήσεων σε συνδυασμό με τη ρύθμιση preset μέσω της γραφικής εφαρμογής, και αντίστροφα. Δηλαδή, αν χρησιμοποιεί τη γραφική διεπαφή, μπορεί να συνδυάσει κάποιο αρχείο, είτε preset, είτε mappings με χειροκίνητες ρυθμίσεις.

και mappings files. Παρακάτω εξηγούνται τα βήματα που θα πρέπει να ακολουθήσει:

Θα πρέπει να δημιουργήσει ένα preset file με τις ακόλουθες δηλώσεις:

```
synth.1.name=AMSynthesizer
synth.1.param.1.name=modAmp
synth.1.param.1.min=0
synth.1.param.1.max=1
synth.1.param.1.representation=linear
synth.1.param.1.defaultvalue=0.3
synth.1.param.2.name=modFreq
synth.1.param.2.min=1
synth.1.param.2.max=10
synth.1.param.2.representation=exponential
synth.1.param.2.defaultvalue=8
synth.1.param.3.name=carFreq
synth.1.param.3.min=200
synth.1.param.3.max=2000
synth.1.param.3.representation=exponential
synth.1.param.3.defaultvalue=500
```

Στη συνέχεια θα πρέπει να δημιουργήσει ένα mappings file με τις ακόλουθες δηλώσεις:

```
eobody.channel.1=synth.1
eobody.channel.2=synth.1.param.2
```

Εάν ο αναγνώστης μελέτησε προσεκτικά το κεφάλαιο αυτό, δεν θα έχει πρόβλημα να κατανοήσει τη λογική που ακολουθείται στα παραπάνω βήματα.

Αφού αποθηκεύσει τα παραπάνω αρχεία (υπενθυμίζεται ότι πρόκειται για απλά αρχεία κειμένου, συνεπώς μπορεί να τα διαχειριστεί μέσω ενός στοιχειώδους editor, όπως το Notepad των Windows), εκτελεί την εφαρμογή sensorsyn. Στην πρώτη καρτέλα πρέπει να κάνει κλικ στο κουμπί “Browse...” και να εντοπίσει το αρχείο presets που μόλις αποθήκευσε. Μόλις το εντοπίσει στο σύστημα αρχείων, θα πρέπει να κάνει κλικ στο “Open”. Τέλος, θα πρέπει να κάνει κλικ στο “Load” για να ακούσει την AM διαμόρφωση. Στη συνέχεια πρέπει να μεταβεί στη δεύτερη καρτέλα (Mappings), να κάνει κλικ στο κουμπί “Load...”, να εντοπίσει το αντίστοιχο αρχείο mappings που μόλις αποθήκευσε και να πατήσει “Open”. Αυτό ήταν. Μπορεί τώρα να ελέγξει τη λειτουργία της εγκατάστασης που μόλις προγραμμάτισε αλληλεπιδρώντας με τους αισθητήρες που έχει συνδέσει στο Eobody.

3.3 Οδηγίες για προγραμματιστές

Οποιοσδήποτε προγραμματιστής με γνώσεις προγραμματισμού σε Java, μπορεί να χρησιμοποιήσει το API, να βασιστεί στη λειτουργικότητά του και να την εξελίξει για να

κατασκευάσει τη δική του εφαρμογή ή μια βελτιωμένη έκδοση αυτού του API. Θα μπορούσε για παράδειγμα να φτιάξει ένα ακόμα τύπο synthesizer, έναν additive synthesizer, ίσως. Στη συνέχεια και σε όλη την έκταση αυτού του κεφαλαίου, θα δοθούν οι απαραίτητες οδηγίες για τον προγραμματιστή που επιθυμεί να κάνει χρήση αυτού του API.

Στο κεφάλαιο 2.2.1 δόθηκε ένας πίνακας των πακέτων του API και μια σύντομη περιγραφή τους. Στη συνέχεια θα ακολουθήσει μια πιο αναλυτική συζήτηση των κλάσεων κάθε πακέτου.

3.3.1 Το πακέτο gr.teicrete.mta.midi.device

Ξεκινώντας από το πακέτο gr.teicrete.mta.midi.device, βλέπουμε την κλάση DeviceControl, όπου περιέχει static μεθόδους που μπορούν να χρησιμοποιηθούν για να ανακτηθούν πληροφορίες για τις midi συσκευές που είναι συνδεδεμένες στον υπολογιστή. Πιο συγκεκριμένα, υπάρχει η μέθοδος getInfos() όπου επιστρέφει ένα array από αντικείμενα της κλάσης MidiDevice.Info, που εμπεριέχουν πληροφορίες για τις midi συσκευές του υπολογιστή, και η μέθοδος getDevices() όπου επιστρέφει ένα array από αντικείμενα της κλάσης MidiDevice, που αντιπροσωπεύουν τις midi συσκευές του υπολογιστή. Υπενθυμίζεται ότι αυτές οι μέθοδοι είναι static, που σημαίνει ότι μπορούν να κληθούν με τη χρήση της ίδιας της κλάσης, χωρίς την ανάγκη δημιουργίας instance της κλάσης DeviceControl. Υπενθυμίζεται επίσης η χρησιμότητα του javadoc (βλ κεφάλαιο 2.2.1.1), όπου μπορεί να βρεθεί στο CD, στη διαδρομή Source_Code/SensorSyn/doc/index.html.

3.3.2 Το πακέτο gr.teicrete.mta.midi.eobody.event

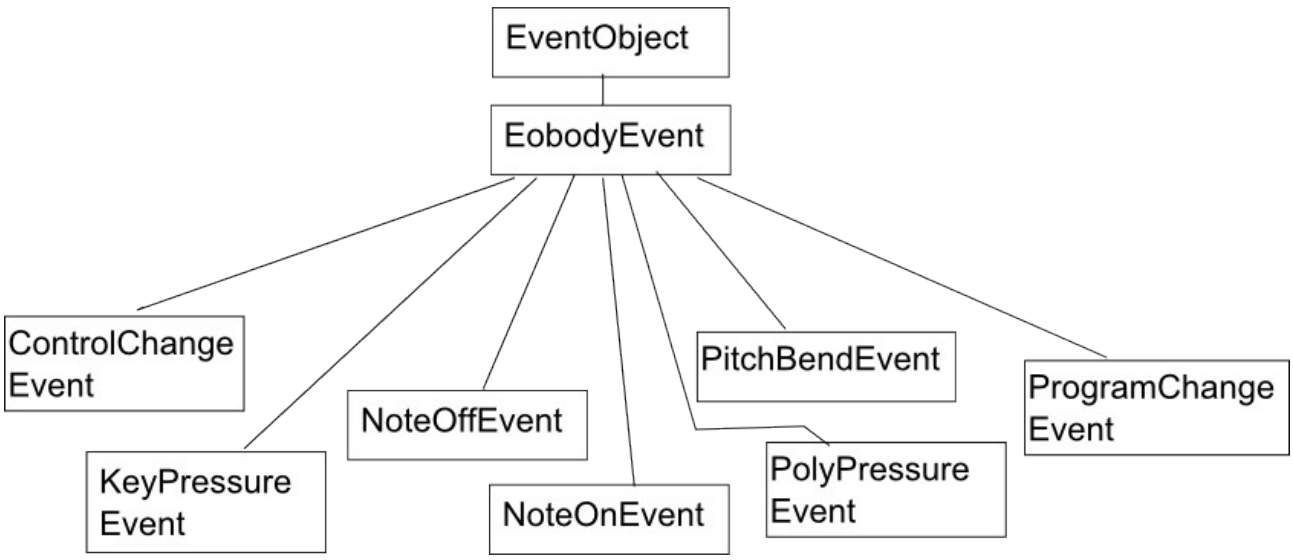
Το πακέτο gr.teicrete.mta.midi.eobody.event, όπως αναφέρθηκε και στο κεφάλαιο 2.2.1 περιέχει κλάσεις που αναπαριστούν τους τύπους των midi μηνυμάτων. Προκειμένου η λειτουργικότητα του API να σχεδιαστεί με βάση συγκεκριμένο προγραμματιστικό pattern του αντικειμενοστραφούς προγραμματισμού, στο πακέτο αυτό βρίσκεται η κλάση EobodyEvent, που κάνει extend την ήδη υπάρχουσα κλάση EventObject, που είναι η superclass όλων των event κλάσεων στη Java. Οι event κλάσεις (όπως για παράδειγμα, η κλάση ActionEvent) αντιπροσωπεύουν ένα συμβάν το οποίο γίνεται 'generate' από κάποια άλλο στοιχείο – κλάση. Για παράδειγμα, ένα instance της κλάσης JButton μπορεί να γεννήσει ένα instance της κλάσης ActionEvent. Προκειμένου να γίνει η αναπαράσταση όλων των πιθανών event (NoteOnEvent, ControlChangeEvent κα), προτιμήθηκε ο ορισμός της κλάσης EobodyEvent σαν superclass όλων των midi συμβάντων, που θα γίνονται generate από κάποια άλλη κλάση, ένα EventFactory, που θα αναφερθεί στη συνέχεια. Ο constructor της κλάσης EobodyEvent περιέχει ένα explicit superconstructor call και απαιτεί δύο ορίσματα, ένα reference σε αντικείμενο της κλάσης Object, που αντιπροσωπεύει την πηγή από την οποία προήλθε το event (για παράδειγμα το event source ενός ActionEvent, μπορεί να είναι ένα JButton), και ένα reference του MIDI μηνύματος που προκάλεσε το EobodyEvent. Η κλάση EobodyEvent διαθέτει την public μέθοδο getChannel που επιστρέφει ένα integer που αντιπροσωπεύει το midi channel από το οποίο προήλθε το midi μήνυμα. Εάν το midi μήνυμα είναι τύπου Sysex ή Meta message, τότε η μέθοδος getChannel επιστρέφει -1, όπως φαίνεται και από τον κώδικά της:

```
public int getChannel() {
        if(midiMessage instanceof ShortMessage) {
```

```
                int channel = ((ShortMessage)midiMessage).getChannel();
                return channel;
            } else {
                return -1;
            }
        }
```

Το πακέτο javax.sound.midi του JDK διαθέτει τρεις κλάσεις για την αναπαράσταση των midi μηνυμάτων: ShortMessage, SysexMessage και MetaMessage, που είναι subclasses της κλάσης MidiMessage του ίδιου πακέτου. Στη συνθήκη if ελέγχεται εάν το midi message από το οποίο γεννήθηκε το event αυτό είναι instance της κλάσης ShortMessage. Εάν ναι, τότε η μέθοδος επιστρέφει το midi channel, αλλιώς επιστρέφει -1. Γενικότερα, για τις ανάγκες των raw midi messages που καλύπτει το API, υποστηρίζονται μόνο τα Short Messages.

Οι υπόλοιπες κλάσεις του πακέτου gr.teicrete.mta.midi.eobody.event κάνουν extend την κλάση EobodyEvent. Αυτές είναι οι κλάσεις: ControlChangeEvent, KeyPressureEvent, NoteOffEvent, NoteOnEvent, PitchBendEvent, PolyPressureEvent και ProgramChangeEvent. Κάθε μια από αυτές διαθέτει ένα constructor που απαιτεί τα ίδια ορίσματα: ένα Object (το event source) και ένα midi message (το midi μήνυμα που προκάλεσε το αντίστοιχο midi event). Ο constructor κάθε κλάσης περιέχει explicit super constructor call – δηλαδή καλεί τον constructor της super κλάσης (EobodyEvent), και αυτός με τη σειρά του τον constructor της super class, EventObject. Επίσης, κάθε κλάση διαθέτει get μεθόδους για την ανάκτηση των data byte του midi message που προκάλεσε το event. Για παράδειγμα, η κλάση NoteOnEvent διαθέτει τις μεθόδους getKey() και getVelocity() όπου επιστρέφουν τα κατάλληλα data bytes σε integer μορφή. Στην Εικόνα 3.3.1 φαίνεται το hierarchy tree του πακέτου gr.teicrete.mta.midi.eobody.event.



Εικόνα 3.3.1 Η ιεραρχία των κλάσεων του πακέτου gr.teicrete.mta.midi.eobody.event

3.3.3 Το πακέτο gr.teicrete.mta.midi.eobody.parse

Το πακέτο gr.teicrete.mta.midi.eobody.parse διαθέτει την κλάση EobodyEventFactory, που

είναι το event factory, δηλαδή η κλάση που δημιουργεί τα EobodyEvents. Επίσης, η κλάση EobodyEventFactory υλοποιεί (implements) το interface Receiver του πακέτου javax.sound.midi που αντιπροσωπεύει την μονάδα μεταβίβασης μηνυμάτων της midi συσκευής του υπολογιστή. Το interface Receiver διαθέτει μια μέθοδο που πρέπει να υλοποιηθεί από την implementing class, την μέθοδο send[3]. Η μέθοδος αυτή στην κλάση EobodyEventFactory υλοποιείται ως εξής:

```
public void send(MidiMessage message, long lTimeStamp)
        {
                fireEobodyEvent(message);
        }
```

Το δεύτερο όρισμα της send αγνοείται, καθώς δεν αφορά τα raw midi μηνύματα. Το μόνο πράγμα που κάνει αυτή η μέθοδος είναι να καλεί την μέθοδο *fireEobodyEvent* με όρισμα το Midi Message που μεταδίδει ο Receiver[4]. Συνεπώς ότι σπουδαίο συμβαίνει στη μέθοδο fireEobodyEvent:

```
public void fireEobodyEvent(MidiMessage message) {

        if(message instanceof ShortMessage) {
                int msgCommand = ((ShortMessage)message).getCommand();
                switch(msgCommand)
                {
                case ShortMessage.NOTE_OFF:
                        for(int ctr = 0; ctr < listenerList.size(); ctr++) {
                                ((EobodyEventListener)listenerList.get(ctr)).noteOffReceived(
                                        new NoteOffEvent(this, message));
                        }
                        break;
                case ShortMessage.NOTE_ON:
                        for(int ctr = 0; ctr < listenerList.size(); ctr++) {
                                ((EobodyEventListener)listenerList.get(ctr)).noteOnReceived(
                                        new NoteOnEvent(this, message));
                        }
                        break;
                [...][5]
```

Και εδώ οποιοδήποτε άλλο μήνυμα εκτός από το Short Message αγνοείται. Μόνο εάν το midi message που μεταδίδει ο Receiver είναι instance της κλάσης ShortMessage ο interpreter μπαίνει στο σώμα της if, όπου γίνεται downcasting του message σε ShortMessage, εφόσον πλέον

3 Διαθέτει επίσης και την μέθοδο close(), η οποία στην υλοποίησή της σε αυτό το API μένει κενή.

4 Εκτός από τον Receiver ενός midi device, υπάρχει και ο Transmitter. Τα ονόματά τους μάλλον μπερδεύουν παρά βοηθούν, καθώς ο Transmitter είναι αυτός που μεταδίδει τα δεδομένα στον Receiver του midi device.

5 Ο κώδικας της μεθόδου συνεχίζεται. Για λόγους οικονομίας δεν συμπεριλαμβάνεται εδώ η συνέχεια του, αφού έχει παρόμοια λογική. Ο αναγνώστης μπορεί να βρει ολόκληρο τον κώδικα της εργασίας στο CD, στη διαδρομή Source_Code/SensorSyn/src

είναι σίγουρο ότι είναι instance της subclass αυτής. Ύστερα, χρησιμοποιώντας τη μέθοδο getCommand της κλάσης ShortMessage ελέγχεται (μέσω μιας δομής switch) το είδος του midi message (αν είναι note-on, note-off κτλ) και αναλόγως το είδος, καλείται η αντίστοιχη μέθοδος όλων των μελών της λίστας με τους *EobodyEventListeners*, που διατηρεί η κλάση EobodyEventFactory. Η κλάση EobodyEventListener θα αναλυθεί στη συνέχεια. Για την ώρα αυτό που χρειάζεται να γνωρίζει ο αναγνώστης είναι ότι η κλάση EobodyEventFactory, σαν event factory όλων των EobodyEvents, διατηρεί μια LinkedList σαν μέσο αποθήκευσης για αντικείμενα που υλοποιούν το interface EobodyEventListener, που είναι τα αντικείμενα που πρέπει να ειδοποιηθούν όποτε προκύπτει ένα EobodyEvent, έτσι ώστε να συμβεί κάτι στη συνέχεια. Αυτό που θα συμβεί, θα εξαρτάται από τον τρόπο που υλοποιείται η διεπαφή EobodyEventListener. Η κλάση EobodyEventFactory διαθέτει τις μεθόδους addEobodyEventListener, removeEobodyEventListener και clearListenerList για την διαχείριση της LinkedList που αποθηκεύει τους EobodyEventListeners.

3.3.4 Το πακέτο gr.teicrete.mta.midi.eobody.process

Στο πακέτο gr.teicrete.mta.midi.eobody.process βρίσκεται η διεπαφή EobodyEventListener, που αναφέρθηκε προηγουμένως. Η διεπαφή αυτή κάνει extend τη διεπαφή EventListener. Περιέχει τις δηλώσεις των μεθόδων noteOnReceived(NoteOnEvent), noteOffReceived(NoteOffEvent) και ούτω καθεξής. Αυτές τις μεθόδους πρέπει να υλοποιήσει η κλάση που θα κάνει implement τη διεπαφή αυτή, ώστε να καθορίζει το τι θα συμβαίνει όταν θα προκύπτει ένα EobodyEvent. Το EobodyEventFactory που κάνει generate τα EobodyEvents είναι επιφορτισμένο με το να "θυμάται" ποια αντικείμενα έχουν γίνει register ως EobodyEventListeners, και να καλεί τη σωστή μέθοδο αυτών, ανάλογα με τον τύπο του midi μηνύματος που μεταδίδει (καθώς εκτός από event factory είναι και receiver).

Όπως σωστά μαντεύει ο αναγνώστης, υπάρχει μια κλάση στο API που κάνει implement τη διεπαφή EobodyEventListener – είναι η κλάση Mapper, που θα παρουσιαστεί στη συνέχεια του κειμένου.

3.3.5 Το πακέτο gr.teicrete.mta.midi.eobody.setup

Στο πακέτο gr.teicrete.mta.midi.eobody.setup βρίσκεται η κλάση EobodyConnect. Όπως ήδη αναφέρθηκε, η κλάση αυτή δεν χρησιμοποιείται στην τελική εφαρμογή, ωστόσο παραμένει στο API καθώς μπορεί να αποδειχτεί χρήσιμη σε κάποιον προγραμματιστή. Διαθέτει τη μέθοδο contactEobody, που αναλαμβάνει να προετοιμάσει τις midi συνδέσεις για να λειτουργήσει η εφαρμογή. Παρακάτω φαίνεται ο κώδικας της μεθόδου:

```
public boolean contactEobody() throws MidiUnavailableException{
        MidiDevice.Info[] devs = DeviceControl.getInfos();
        for(int ctr = 0; ctr < devs.length; ctr++) {
            if(devs[ctr].toString().equals("In-A USB MIDISPORT 2x2")) {
                eobodyReceiver = new EobodyEventFactory();
                eobodyDevice = MidiSystem.getMidiDevice(devs[ctr]);
```

```
                    eobodyTransmitter = eobodyDevice.getTransmitter();
                    eobodyTransmitter.setReceiver(eobodyReceiver);
                }
            }
            if(eobodyReceiver != null) {
                return true;
            } else {
                return false;
            }
        }
```

Το πρώτο πράγμα που μπορεί να συμπεράνει ο αναγνώστης είναι ότι η κλάση χρησιμοποιεί τη static μέθοδο getInfos() της κλάσης DeviceControl του πακέτου gr.teicrete.mta.midi.device για να αποθηκεύσει ένα array με τις πληροφορίες όλων των midi συσκευών που διαθέτει ο υπολογιστής. Ένα πιο σημαντικό συμπέρασμα όμως είναι ότι ο κώδικας της μεθόδου δεν αναζητάει την περιγραφή του Eobody σε αυτό το array, αλλά της κάρτας MIDISPORT 2x2 στην οποία είναι συνδεδεμένο το Eobody. Μια ιδέα, λοιπόν, για επέκταση του API σε αυτό το σημείο θα ήταν η συγγραφή μιας μεθόδου που θα αναζητάει το Eobody μέσω της κάρτας που θα υποδεικνύεται κατά τη διάρκεια της εκτέλεσης του κώδικα.

Στη συνέχεια του κώδικα μπορεί να δει κανείς πιο ξεκάθαρα τη σχέση μεταξύ midi device, transmitter και receiver. Ο eobodyTransmitter είναι ο τυπικός transmitter του midi device, ενώ σαν receiver του transmitter θέτουμε ένα νέο αντικείμενο της κλάσης EobodyEventFactory.

Τέλος, αν η τιμή του eobodyReceiver είναι διάφορη του null (που σημαίνει ότι έχει βρεθεί το midi device που αναζητούσε ο κώδικας μέσα στο βρόγχο for), τότε η μέθοδος επιστρέφει true, διαφορετικά επιστρέφει false.

3.3.6 Το πακέτο gr.teicrete.mta.midi.gui

Το πακέτο gr.teicrete.mta.midi.gui περιέχει τρεις κλάσεις που αναλαμβάνουν τη γραφική διεπαφή της εφαρμογής (ο αναγνώστης μπορεί να θυμηθεί σε αυτό το σημείο τη λογική διαχωρισμού της λειτουργικότητας από τον τρόπο παρουσίασης που αναφέρθηκε στο Κεφάλαιο 2.2.1). Ο προγραμματιστής μπορεί να χρησιμοποιήσει αυτές τις κλάσεις για τη γραφική διεπαφή της εφαρμογής του. Οι κλάσεις που περιέχει το πακέτο gr.teicrete.mta.midi.gui είναι οι εξής: SetupWindow, SetupHandler και MonitorPanel.

Η κλάση SetupWindow είναι η κλάση του κεντρικού παραθύρου της εφαρμογής. Το κεντρικό παράθυρο που βλέπουμε στην Εικόνα 3.1.1 είναι instance της κλάσης SetupWindow. Η κλάση αυτή κάνει extend την κλάση JFrame του πακέτου javax.swing που αποτελεί ένα τυπικό παράθυρο του λειτουργικού συστήματος στο οποίο η εφαρμογή εκτελείται. Επίσης, η κλάση αυτή διαθέτει μια main μέθοδο:

```
public static void main(String[] args) {
        new SetupWindow();
    }
```

Η main μέθοδος της κλάσης αυτής αποτελεί το entry point της εφαρμογής. Συνεπώς, όταν η εφαρμογή εκτελείται, δημιουργείται ένα νέο αντικείμενο της κλάσης SetupWindow.

Το μεγαλύτερο μέρος του κώδικα αυτής της κλάσης ασχολείται με τη δημιουργία του γραφικού περιβάλλοντος, οπότε δεν αξίζει ιδιαίτερης προσοχής. Θα μπορούσαν, ωστόσο, να βγουν κάποια συμπεράσματα από το παρακάτω σημείο του κώδικα που βρίσκεται στον constructor του SetupWindow:

```
mapper = new Mapper();

setupHandler = new SetupHandler(this, mapper);

browseButton.addActionListener(setupHandler);
loadButton.addActionListener(setupHandler);
[...]
tabs.addChangeListener(setupHandler);
```

Εδώ, συμβαίνουν πολλά ενδιαφέροντα πράγματα. Πρώτα δημιουργείται ένα νέο αντικείμενο της κλάσης Mapper (όπως ήδη αναφέρθηκε, πρόκειται για την κλάση που τόλμησε να κάνει implement το interface EobodyEventListener), και αποθηκεύεται ένα reference στη μεταβλητή mapper. Στη συνέχεια, δημιουργείται ένα νέο αντικείμενο της κλάσης SetupHandler (που είναι η δεύτερη κλάση του πακέτου gr.teicrete.mta.midi.gui) και πρόκειται για την κλάση που αναλαμβάνει τη λειτουργικότητα του GUI. Όπως φαίνεται και στο απόσπασμα του κώδικα, για τη δημιουργία του αντικειμένου της κλάσης SetupHandler, καλείται ο constructor της κλάσης με δύο ορίσματα. Αν ο αναγνώστης ανατρέξει στο documentation του API, θα ανακαλύψει ότι σαν πρώτο όρισμα απαιτείται ένα αντικείμενο της κλάσης SetupWindow, που, όπως περιγράφεται στο documentation, πρόκειται για το instance της κλάσης SetupWindow από όπου προκύπτουν τα events τα οποία η κλάση SetupHandler αναλαμβάνει να χειριστεί. Συνεπώς, εξετάζοντας την κλάση SetupHandler, μπορεί να διαπιστώσει κανείς ότι η τελευταία κάνει implement πολλά interfaces: ActionListener, ListSelectionListener και ChangeListener. Πρόκειται για listeners που ακούν σε events που προκύπτουν από τα γραφικά components που χρησιμοποιούνται στην κλάση SetupWindow. Το δεύτερο όρισμα του constructor είναι ένα αντικείμενο της κλάσης Mapper, το οποίο η κλάση SetupHandler θα χρησιμοποιήσει για να υλοποιήσει κομμάτι της λειτουργικότητας.

Γενικά, στις περίπου 1100 γραμμές κώδικα της κλάση SetupHandler, δεν συμβαίνει κάτι παραπάνω από την υλοποίηση των μεθόδων που απαιτούνται για να γίνουν implement οι προαναφερθέντες διεπαφές. Στην προσπάθεια αυτή, ορίζονται πολλές βοηθητικές μέθοδοι, όπως η παρακάτω

```
private void updateSynthMappingsGUI() {
        LinkedList<AbstractSensorSynthesizer> synthList = SynthEngine.getSynthList();
        Iterator<AbstractSensorSynthesizer> i = synthList.iterator();
        int noSynth = 0;
        while(i.hasNext()) {
            noSynth++;
            setupWindow.synthTargetCombo.addItem("synth." + noSynth);
            i.next();
```

```
            }
        }
```

που αναλαμβάνει να ενημερώσει μερικά στοιχεία του GUI, όποτε συμβαίνουν κάποιες μεταβολές στη λίστα με τα ενεργά synthesizers. Στον παραπάνω κώδικα χρησιμοποιούνται κλάσεις που δεν έχουν ακόμα αναφερθεί. Θα αναλυθούν στη συνέχεια, όταν θα αναφερθεί το πακέτο gr.teicrete.mta.synthesizer.

Παρά τις πολλές γραμμές κώδικα της κλάσης SetupHandler, η λειτουργία που επιτελεί είναι ξεκάθαρη: Αποτελεί στην ουσία τον συνδετικό κρίκο μεταξύ παρουσίασης και λειτουργικότητας. Για αυτό, όπως πιθανόν να παρατήρησε ο αναγνώστης, στο απόσπασμα του κώδικα του constructor της κλάσης SetupWindow, το instance της κλάσης SetupHandler, που ονομάζεται setupHandler, γίνεται register σαν listener για όλα τα γραφικά components της κλάσης SetupWindow.

Φτάνοντας στην κλάση MonitorPanel, αυτό που αξίζει να αναφερθεί είναι ότι πρόκειται για μια κλάση που κάνει extend την κλάση JPanel, συνεπώς αποτελεί ένα JPanel. Οι τρεις καρτέλες – θέματα του γραφικού περιβάλλοντος, που περιγράφηκαν λεπτομερώς στο κεφάλαιο 3.2 είναι τρία διαφορετικά Jpanels, όπως μπορεί να διαπιστώσει κανείς εξετάζοντας ένα κομμάτι του constructor της κλάσης SetupWindow:

```
tabs = new JTabbedPane();


presetsPanel = new JPanel();
mappingsPanel = new JPanel();
monitorPanel = new MonitorPanel(this);


// tabs
tabs.insertTab("Presets", null, presetsPanel, "Control Synth Presets", 0);
tabs.insertTab("Mappings", null, mappingsPanel, "Control Mappings", 1);
tabs.insertTab("Monitor Activity", null, monitorPanel, "Visual representation of the "
                        + "mapped Synths & Parameters", 2);


getContentPane().add(tabs);
```

Η κλάση JTabbedPane αποτελεί ένα γραφικό στοιχείο όπου εμφανίζει καρτέλες (tabs). Εδώ δημιουργείται ένα instance αυτής της κλάσης και αποθηκεύεται στη μεταβλητή tabs. Ένα JTabbedPane μπορεί να εμφανίσει πολλά διαφορετικά JPanels οργανωμένα σε καρτέλες. Στη συνέχεια, λοιπόν, δημιουργούνται δύο JPanels, που αποθηκεύονται στις μεταβλητές presetsPanel και mappingsPanel, και ένα MonitorPanel (που, όπως, αναφέρθηκε κάνει extend την κλάση Jpanel, άρα είναι και αυτό ένα JPanel) που αποθηκεύεται στη μεταβλητή monitorPanel. Στη συνέχεια, με τη χρήση της μεθόδου insertTab της κλάσης JTabbedPane, προσθέτονται τα τρία panels στο instance της κλάσης JTabbedPane, με το όνομα tabs.

Πιθανόν ο αναγνώστης να αναρωτιέται γιατί το τελευταίο tab πρέπει να περιέχει ένα MonitorPanel και όχι ένα JPanel, όπως τα προηγούμενα δύο. Μπορεί να λάβει την απάντησή του εξετάζοντας τον κώδικα της κλάσης MonitorPanel. Πρώτα από όλα, θα παρατηρήσει ότι πρόκειται για ένα ειδικό JPanel, που κάνει implement τα interfaces SynthEventListener και ActionListener. Το δεύτερο έχει αναφερθεί ήδη αρκετές φορές σε αυτό το κείμενο. Το πρώτο (SynthEventListener)

είναι ένα interface του ίδιου του API, στο πακέτο gr.teicrete.mta.synthesizer.event και αντιπροσωπεύει έναν listener που ακούει σε event του συστήματος παραγωγής του ήχου του API. Με λίγα λόγια, αυτό το panel ξεχωρίζει από τα προηγούμενα, επειδή μπορεί και "ακούει" σε συμβάντα του συστήματος παραγωγής του ήχου. Αυτό, με λίγη σκέψη, αποδεικνύεται απόλυτα λογικό, αν συνυπολογιστεί η λειτουργία που επιτελεί η συγκεκριμένη καρτέλα (βλ Κεφάλαιο 3.2). Επίσης, ένας λιγότερο μυημένος σε θέματα κληρονομικότητας των αντικειμενοστραφών γλωσσών προγραμματισμού, βρίσκεται μπροστά σε ένα εύγλωττο παράδειγμα του πόση ισχύ κρύβει η κληρονομικότητα, αρκεί να θυμηθεί ότι μια subclass κληρονομεί όλες τις ιδιότητες και τις μεθόδους της superclass, με σκοπό να επεκτείνει και να εξειδικεύσει τις λειτουργίες της. Έτσι, μέσω της κληρονομικότητας, από ένα γενικό JPanel (και μόλις 200 επιπλέον γραμμές κώδικα), προμηθευόμαστε ένα πολύ πιο ειδικό MonitorPanel, που -όπου χρησιμοποιηθεί σαν γραφικό component- μπορεί να πληροφορεί τον χρήστη για τις real time μεταβολές στις παραμέτρους της σύνθεσης του ήχου.

3.3.7 Το πακέτο gr.teicrete.mta.midi.mappings

Το πακέτο gr.teicrete.mta.midi.mappings, όπως αναφέρθηκε και στο κεφάλαιο 2.2.1, περιέχει κλάσεις υπεύθυνες για τις αντιστοιχήσεις των παραμέτρων σύνθεσης του ήχου με τα midi δεδομένα που προσλαμβάνονται. Περιέχει το interface Mappable, που αντιπροσωπεύει οποιοδήποτε αντικείμενο μπορεί να αντιστοιχηθεί με ένα συγκεκριμένο κανάλι. Ένα κανάλι αντιπροσωπεύεται από την κλάση ControllerChannel του ίδιου πακέτου. Ο constructor της κλάσης αυτής είναι ως εξής:

```
public ControllerChannel(int channel) {
        this.midiChannel = channel;
}
```

Παίρνει σαν όρισμα ένα integer που αντιπροσωπεύει τον αριθμό του καναλιού του Eobody και το αποθηκεύει στη μεταβλητή midiChannel. Η κλάση αυτή διατηρεί μια ακόμα εσωτερική μεταβλητή, τύπου Mappable, την mapTarget, που αντιπροσωπεύει το αντικείμενο[6] με το οποίο έχει αντιστοιχηθεί το εν λόγω κανάλι.

Τέλος, η κλάση Mapper διατηρεί ένα array 16 θέσεων από αντικείμενα τύπου ControllerChannel, και το αρχικοποιεί στον constructor με ορίσματα από 1-16. Ωστόσο, θα ήταν χρήσιμο να εξεταστεί ολόκληρος ο κώδικας του constructor της κλάσης Mapper:

```
public Mapper() {

        for(int ctr=0; ctr < 16; ctr++) {
                mappings[ctr] = new ControllerChannel(ctr+1);
        }

        rec.addEobodyEventListener(this);
```

6 Στη συνέχεια θα δειχθεί ότι αυτά τα αντικείμενα μπορεί να είναι είτε της κλάσης AbstractSensorSynthesizer, είτε της κλάσης SynthParameter.

```
        MidiDevice.Info[] devs = DeviceControl.getInfos();
        try {
                for(int ctr = 0; ctr < devs.length; ctr++) {
                        if(devs[ctr].toString().equalsIgnoreCase("In-A USB MIDISPORT"  +
                                                                + "2x2")) {
                                dev = MidiSystem.getMidiDevice(devs[ctr]);
                                dev.open();
                                trans = dev.getTransmitter();
                                trans.setReceiver(rec);
                        }
                }
        } catch(MidiUnavailableException e) {
                e.printStackTrace();
        }

}
```

Στο βρόγχο for γίνεται η αρχικοποίηση του array. Στη συνέχεια δηλώνεται η παρούσα κλάση (Mapper) σαν EobodyEventListener του rec. Το rec είναι μεταβλητή – μέλος της κλάσης Mapper και πρόκειται για ένα instance της κλάσης EobodyEventFactory, που περιγράφηκε παραπάνω και αποτελεί υλοποίηση του Receiver interface. Η κλάση Mapper, όπως ήδη αναφέρθηκε κάνει implement τη διεπαφή EobodyEventListener, συνεπώς αυτό που συμβαίνει σε αυτό το σημείο είναι ότι ορίζεται ένας EobodyEventListener για το EobodyEventFactory rec. Άρα, η κλάση Mapper είναι υπεύθυνη για να καθορίσει το τι θα συμβαίνει όταν θα προκύπτει κάποιο EobodyEvent.

Για τη διαχείριση του ControllerChannel array, η κλάση Mapper διαθέτει δύο μεθόδους: τη μέθοδο loadMappings, που παίρνει σαν όρισμα ένα String που αντιπροσωπεύει το όνομα του αρχείου mappings (αυτή η μέθοδος καλείται από τον SetupHandler, όταν ο χρήστης φορτώνει ένα mappings αρχείο), και τη μέθοδο updateMapping που παίρνει σαν όρισμα ένα integer που αντιπροσωπεύει τον αριθμό του ControllerChannel και ένα Mappable αντικείμενο, και στην ουσία πραγματοποιεί μια αντιστοίχηση μεταξύ του δοσμένου καναλιού και του Mappable αντικειμένου. Εάν αυτή η αντιστοίχηση επικαλύπτει κάποια προηγούμενη (εάν προηγουμένως το map target του ControllerChannel αντικειμένου δεν ήταν null) η μέθοδος επιστρέφει την τιμή true, διαφορετικά επιστρέφει false.

Η κλάση Mapper πρέπει να υλοποιήσει και όλες τις μεθόδους που δηλώνονται στο interface EobodyEventListener, και τις υλοποιεί καλώντας τη μέθοδο respondToEvent, περνώντας σαν όρισμα το EobodyEvent που προκάλεσε την κλήση της εκάστοτε μεθόδου. Για παράδειγμα, εάν από κάποιο κανάλι ερχόταν ένα Note on μήνυμα, τότε -αφού η κλάση Mapper έχει καταγραφεί σαν EobodyEventListener του EobodyEventFactory, που είναι μέλος της κλάσης Mapper- θα καλούταν η μέθοδος noteOnReceived της κλάσης Mapper. Η μέθοδος αυτή έχει ως εξής:

```
public void noteOnReceived(NoteOnEvent evt) {
        respondToEvent(evt);
}
```

Όπως αναφέρθηκε και παραπάνω, καλείται η μέθοδος respondToEvent με όρισμα το ίδιο το NoteOnEvent. Η μέθοδος respondToEvent έχει την εξής μορφή:

```
1.      private void respondToEvent(EobodyEvent evt) {
2.
3.              int evtChannel = evt.getChannel();
4.              ControllerChannel ctrChannel = mappings[evtChannel];
5.              Mappable mapped = ctrChannel.getMapTarget();
6.
7.
8.              if(mapped instanceof AbstractSensorSynthesizer) {
9.
10.
11.                     AbstractSensorSynthesizer syn = (AbstractSensorSynthesizer)mapped;
12.
13.                     if(syn.synthOn()) {
14.                             syn.stop();
15.                     } else {
16.                             syn.start();
17.                     }
18.
19.
20.
21.             } else if(mapped instanceof SynthParameter) {
22.
23.                     SynthParameter param = (SynthParameter)mapped;
24.
25.
26.                     double x = 0, mx, mn, mappedValue;
27.
28.                     // must detect the subclass of the EobodyEvent because
29.                     // Pitch Bend events must be scaled differently
30.                     if(evt instanceof ControlChangeEvent) {
31.                             x = ((ControlChangeEvent)evt).getValue();
32.                     } else if(evt instanceof KeyPressureEvent) {
33.                             x = ((KeyPressureEvent)evt).getPressure();
```

```
34.                     } else if(evt instanceof NoteOffEvent) {
35.                             x = ((NoteOffEvent)evt).getKey();
36.                     } else if(evt instanceof NoteOnEvent) {
37.                             x = ((NoteOnEvent)evt).getKey();
38.                     } else if(evt instanceof PitchBendEvent) {
39.                             x = ((PitchBendEvent)evt).getValue();
40.                             x = x/16384;
41.                             x = x * 127;
42.                     } else if(evt instanceof PolyPressureEvent) {
43.                             x = ((PolyPressureEvent)evt).getPressure();
44.                     } else if(evt instanceof ProgramChangeEvent) {
45.                             x = ((ProgramChangeEvent)evt).getProgram();
46.                     }
47.
48.                     mx = param.getMax();
49.                     mn = param.getMin();
50.
51.                     // Now checking the SynthParameter's type of representation,
52.                     // so the correct equation will be used.
53.
54.                     if(param.getRepresentation()==
                                        SynthParameter.LINEAR_REPRESENTATION) {
55.                             mappedValue = (x/127)*(mx-mn) + mn;
56.                     } else {
57.                             mappedValue = mn*Math.pow(10, ((Math.log10(mx/mn)*x/127)));
58.                     }
59.
60.
61.                     try {
62.                             param.setValue(mappedValue);
63.
64.                     } catch(BadSynthValueException badValue) {
65.                             badValue.printStackTrace();
66.                     }
67.             }
```

```
68.     }
```

Η μέθοδος αυτή παίρνει σαν όρισμα ένα EobodyEvent, συνεπώς το NoteOnEvent γίνεται cast σε EobodyEvent. Στη συνέχεια, στις γραμμές 3-5 αποθηκεύεται το ControllerChannel από το οποίο προήλθε το event και το mapTarget του. Στη γραμμή 8 ελέγχεται εάν το mapTarget είναι instance της κλάσης AbstractSensorSynthesizer. Η κλάση αυτή (θα περιγραφεί στη συνέχεια) αποτελεί τη superclass όλων των synthesizer. Συνεπώς, ο έλεγχος αυτός γίνεται για να αποφασιστεί εάν υπάρχει αντιστοίχηση τύπου toggle, ή εάν υπάρχει αντιστοίχηση κλασσικού τύπου, όπου το mapTarget είναι κάποια παράμετρος ενός synthesizer (γραμμή 21). Στο σώμα της if (γραμμές 13-17) ελέγχεται εάν ο synthesizer είναι ενεργός. Εάν ναι, τότε καλείται η μέθοδος stop του synthesizer και ο synthesizer απενεργοποιείται. Εάν όχι, καλείται η μέθοδος start και το synthesizer ενεργοποιείται. Στη γραμμή 21, ελέγχεται εάν το mapTarget είναι instance της κλάσης SynthParameter (όπου είναι η κλάση που αντιπροσωπεύει μια παράμετρο σύνθεσης). Σε αυτήν την περίπτωση, τα πράγματα είναι λίγο πιο περίπλοκα. Στις γραμμές 30-46, με μια σειρά από else-if, ελέγχεται ποιας υποκλάσης είναι instance το EobodyEvent με το οποίο κλήθηκε η respondToEvent. Μόλις βρεθεί, το EobodyEvent γίνεται cast σε αυτή τη subclass. Στην περίπτωση του PitchBend (όπου χρησιμοποιούνται δύο data bytes για την αναπαράσταση του ίδιου μεγέθους), πραγματοποιείται διαφορετικό scaling σε σχέση με τις άλλες περιπτώσεις midi μηνυμάτων (βλ 38-41). Στις γραμμές 48-58 γίνεται ο υπολογισμός της νέας τιμής που πρέπει να λάβει η παράμετρος σύνθεσης, με βάση τα data του εισερχόμενου midi μηνύματος, χρησιμοποιώντας διαφορετική μαθηματική σχέση στην περίπτωση γραμμικής αναπαράστασης και διαφορετική στην περίπτωση εκθετικής αναπαράστασης. Τέλος, στη γραμμή 62 καλείται η μέθοδος setValue της κλάσης SynthParameter, ώστε η παράμετρος να αποκτήσει την καινούργια τιμή που υπολογίστηκε προηγουμένως.

3.3.8 Το πακέτο gr.teicrete.mta.synthesizer

Στο πακέτο gr.teicrete.mta.synthesizer βρίσκονται όλες οι κλάσεις που σχετίζονται με την παραγωγή του ήχου. Καταρχάς υπάρχουν οι κλάσεις BadRepresentationException, BadSynthParameterException και BadSynthValueException που κάνουν extend την κλάση Exception, και αντιπροσωπεύουν τα τρία είδη exception που μπορεί να προκύψουν σε run-time από τις μεθόδους των κλάσεων αυτού του πακέτου.

Παραδείγματα μεθόδων που εμπεριέχουν την πιθανότητα δημιουργίας exception μπορούν να βρεθούν στην κλάση SynthParameter. Η κλάση αυτή αντιπροσωπεύει μια παράμετρο ενός synthesizer και κάνει implement το Mappable interface του πακέτου gr.teicrete.mta.midi.mappings, άρα μπορεί να οριστεί σαν mapTarget για ένα ControllerChannel. Η κλάση SynthParameter ως επί το πλείστον διαθέτει μεθόδους set και get για τον χειρισμό των ιδιοτήτων μιας SynthParameter, που είναι η ελάχιστη τιμή, η μέγιστη τιμή, η εξ' ορισμού ή τρέχουσα τιμή και η αναπαράσταση (γραμμική ή εκθετική). Έτσι έχουν οριστεί οι public μέθοδοι getMin, getMax, getValue, getParameterName (για την ανάκτηση ενός String με το χαρακτηριστικό όνομα μιας παραμέτρου – για παράδειγμα modAmp) και getRepresentation. Στο σύνολο των μεθόδων set υπάρχει, κατ' αρχάς η μέθοδος setRange, όπου δέχεται δύο ορίσματα: ένα integer για την ελάχιστη τιμή και ένα για την μέγιστη τιμή. Επίσης, υπάρχει η μέθοδος setValue, που δέχεται μια double τιμή, που αντιπροσωπεύει την τιμή που θέλει η καλούσα μέθοδος να λάβει η συγκεκριμένη παράμετρος. Η μέθοδος setValue είναι μια μέθοδος όπου μπορεί να δημιουργήσει ένα BadSynthValueException. Αυτό θα συμβεί όταν η setValue προσπαθήσει να δώσει μια τιμή στην παράμετρο που βρίσκεται εκτός του εύρους τιμών που αυτή μπορεί να λάβει (Min,Max). Ίσως όμως όσα αναφέρονται να γίνουν πιο κατανοητά εάν εξεταστεί λεπτομερώς ο κώδικας της μεθόδου αυτής:

```
public void setValue(double val) throws BadSynthValueException {
        if((value < min) || (value>max)) {
                throw new BadSynthValueException();
        } else {
                this.value = val;

                try {
                        SynthEngine.getSynthesizer(synth).setParameter(parameterName, value);
                } catch (BadSynthParameterException e) {
                        e.printStackTrace();
                }
        }
        fireParameterChangedEvent();
}
```

Στη συνθήκη if ελέγχεται εάν η τιμή βρίσκεται μέσα στο εύρος min,max. Εάν όχι, η μέθοδος "πετάει" (throws) ένα νέο BadSynthValueException. Αυτό το Exception πρέπει να εντοπιστεί στο σώμα της καλούσας μεθόδου και να αντιμετωπιστεί αναλόγως. Εάν η τιμή βρίσκεται μέσα στο εύρος των τιμών που η παράμετρος μπορεί να λάβει, τότε γίνεται η αλλαγή τιμής. Αυτή η αλλαγή έχει δύο επίπεδα. Το θεωρητικό επίπεδο, όπου καλύπτεται από τη γραμμή this.value = val. Εδώ, η double μεταβλητή – μέλος της SynthParameter, value παίρνει την τιμή val. Στη συνέχεια, στο try block καλείται η static μέθοδος getSynthesizer της κλάσης SynthEngine (όπου θα εξεταστεί στη συνέχεια) για την ανάκτηση ενός reference του synthesizer νούμερο synth (integer μεταβλητή – μέλος που αντιπροσωπεύει το νούμερο του synthesizer στο οποίο ανήκει η συγκεκριμένη παράμετρος). Η μέθοδος setParameter είναι abstract μέθοδος της κλάσης AbstractSensorSynthesizer (και υλοποιείται από τις subclasses AMSynthesizer και FMSynthesizer). Όποιας κλάσης και αν είναι το AbstractSensorSynthesizer που επιστρέφει η getSynthesizer, διαθέτει τη μέθοδο setParameter. Συνεπώς, καλείται προκειμένου να μεταβληθεί η παράμετρος όσον αφορά τον αλγόριθμο σύνθεσης του ήχου, που υλοποιείται στις υποκλάσεις του AbstractSensorSynthesizer, με τη βοήθεια του JSyn API. Η μέθοδος setParameter είναι πιθανό να "πετάξει" BadSynthParameterException (εάν το String όρισμα που αφορά το όνομα της παραμέτρου δεν ανταποκρίνεται σε κάποια παράμετρο του synthesizer), που αντιμετωπίζεται από το catch block. Τέλος, καλείται η μέθοδος fireParameterChangedEvent, όπου στην ουσία δημιουργεί ένα ParameterChangedEvent (κλάση του πακέτου gr.teicrete.mta.synthesizer.event) για να ειδοποιήσει όλους SynthEventListeners (κλάση του πακέτου gr.teicrete.mta.synthesizer.event) για τη μεταβολή στη τιμή μιας παραμέτρου σύνθεσης του ήχου[7].

Τέλος, η set μέθοδος setRepresentation της κλάσης SynthParameter "πετάει" και αυτή BadRepresentationException, εάν το integer όρισμα που δέχεται δεν ισούται ούτε με τη static final int LINEAR_REPRESENTATION, ούτε με τη static final int EXPONENTIAL_ REPRESENTATION.

7 Η κλάση MonitorPanel του πακέτου gr.teicrete.mta.midi.gui, όπως μπορεί να θυμηθεί ο αναγνώστης κάνει implement το SynthEventListener interface προκειμένου να ειδοποιηθεί για μεταβολές στην παραγωγή του ήχου, και να τις αναπαραστήσει γραφικά σε πραγματικό χρόνο.

Ένα σημαντικό βήμα για την κατανόηση του πυρήνα παραγωγής του ήχου της εργασίας αυτής αποτελεί η κατανόηση των κλάσεων AbstractSensorSynthesizer, AMSynthesizer και FMSynthesizer. Η κατανόηση των κλάσεων αυτών θα επιτρέψει στον προγραμματιστή να δημιουργήσει καινούργιους synthesizer, κάτι που θα εμπλουτίσει πολύ το API. Η ιεραρχία είναι απλή: Η κλάση AbstractSensorSynthesizer είναι μια abstract superclass από την οποία οι κλάσεις AMSynthesizer και FMSynthesizer κληρονομούν. Η κλάση AbstractSensorSynthesizer κάνει implement το Mappable interface, προκειμένου να μπορεί να αντιστοιχηθεί με ένα ControllerChannel. Επίσης, διαθέτει μια LinkedList με instances της κλάσης SynthParameter (οι παράμετροι ενός synthesizer). Τέλος, διαθέτει -μεταξύ άλλων- τις abstract μεθόδους start, stop, setParameter και getParameter, για το ξεκίνημα, το σταμάτημα, την μεταβολή της τιμής μιας παραμέτρου και την ανάκτηση μιας παραμέτρου, αντίστοιχα. Οι μέθοδοι αυτοί υλοποιούνται από τις subclasses.

Παρακάτω βλέπουμε την υλοποίηση της start μεθόδου στην κλάση AMSynthesizer:

```
public void start() {
        if(!(SynthEngine.isEngineOn())) {
                SynthEngine.powerOnSynthesizer();
        }
        SynthEngine.addSynthesizer(this);



        // start the sound
        mod = new SineOscillator();
        mod.amplitude.set(modAmp.getValue());
        mod.frequency.set(modFreq.getValue());
        car = new SineOscillator();
        car.frequency.set(carFreq.getValue());
        car.amplitude.connect(mod.output);
        mod.start();
        car.start();

        lineout = new LineOut();

        car.output.connect(0, lineout.input, 0);
        car.output.connect(0, lineout.input, 1);

        lineout.start();

        this.isOn = true;
```

```
        fireSynthOnEvent();
}
```

Οι πρώτες τέσσερις γραμμές αφορούν την επικοινωνία της κλάσης με την κλάση SynthEngine. Εν ολίγοις, η κλάση SynthEngine αντιπροσωπεύει το σύστημα παραγωγής του ήχου και πρέπει να γνωρίζει όλους τους synthesizers που έχουν δημιουργηθεί. Επίσης, η SynthEngine αναλαμβάνει να εκκινήσει το engine του JSyn, ή να το σταματήσει όταν πρέπει. Η static μέθοδος isEngineOn της κλάσης SynthEngine επιστρέφει true αν η engine είναι ενεργή, false αν δεν είναι. Έτσι, στη συνθήκη if, εάν η engine δεν είναι ήδη ενεργή, ενεργοποιείται μέσω της μεθόδου powerOnSynthesizer. Στη συνέχεια, με τη χρήση της μεθόδου addSynthesizer, προστίθεται ο synthesizer αυτός στη λίστα της SynthEngine, προκειμένου η τελευταία να κρατάει λογαριασμό για τους synthesizer που έχουν δημιουργηθεί[8].

Στη συνέχεια της μεθόδου γίνεται χρήση κλάσεων του JSyn. Η κλάση SineOscillator είναι μια γεννήτρια ημιτονοειδούς κυματομορφής. Χρησιμοποιούνται δύο SineOscillators, ο mod και ο car (modulator και carrier). Με τη χρήση των μεθόδων amplitude.set και frequency.set ρυθμίζονται οι τιμές πλάτους και συχνότητας των modulator και carrier με βάση τις τιμές των αντίστοιχων SynthParameter. Με τη χρήση της μεθόδου connect, όπου η έξοδος του mod συνδέεται με το amplitude του car, επιτυγχάνεται η AM διαμόρφωση. Για να ξεκινήσουν οι γεννήτριες mod και car, πρέπει να γίνει κλήση της μεθόδου start της κλάσης SineOscillator. Για να ακουστεί ο ήχος, πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης LineOut και να συνδεθεί με την έξοδο του car, όπως γίνεται με τη μέθοδο connect. Το αντικείμενο lineOut πρέπει να ξεκινήσει και αυτό, με τη μέθοδο start, όπως και γίνεται. Τέλος, καλείται η μέθοδος fireSynthOnEvent, προκειμένου να δημιουργηθεί ένα SynthOnEvent και να ειδοποιηθούν οι SynthEventListeners.

Εάν ο αναγνώστης ενδιαφέρεται να εξετάσει τον αντίστοιχο κώδικα της μεθόδου start της κλάσης FMSynthesizer, μπορεί να δει με ποιο τρόπο (με τη χρήση κλάσεων του JSyn) επιτυγχάνεται η FM διαμόρφωση. Παίρνοντας παράδειγμα από αυτά τα αποσπάσματα κώδικα, αλλά και εξετάζοντας τον τρόπο υλοποίησης όλων των abstract μεθόδων της κλάσης AbstractSensorSynthesizer (ίσως και με μια στοιχειώδη μελέτη του Jsyn), μπορεί να αναπτύξει τον δικό του synthesizer, κάνοντας extend την κλάση AbstractSensorSynthesizer.

Η κλάση SynthEngine, που αντιπροσωπεύει την "καρδιά" του συστήματος παραγωγής του ήχου, περιέχει ένα σύνολο από static μεθόδους για τη διαχείριση των synthesizer και των SynthEventListener. Έτσι, για τη διαχείριση των synthesizer, παρέχει τις παρακάτω μεθόδους: addSynthesizer, removeSynthesizer, stopAllSynthesizers, getNoSynths, getSynthList (που επιστρέφει μια LinkedList με τους synthesizer) και getSynthesizer. Για τη διαχείριση των SynthEventListener, η κλάση SynthEngine διαθέτει τις παρακάτω μεθόδους: addSynthEventListener, removerSynthEventListener και getListenerList. Επίσης, υπάρχουν οι μέθοδοι startSynthesizer και stopSynthesizer, για την εκκίνηση και διακοπή λειτουργίας του engine του JSyn, και η μέθοδος isEngineOn που ελέγχει εάν η engine του JSyn βρίσκεται σε λειτουργία.

Η τελευταία κλάση του πακέτου αυτού είναι η κλάση SynthesizerFactory. Η κλάση αυτή αναλαμβάνει να κάνει interpret ένα αρχείο preset και να δημιουργήσει τους κατάλληλους synthesizer. Εάν ο προγραμματιστής σκοπεύει να δημιουργήσει ένα δικό του synthesizer και θέλει να μπορεί να τον χρησιμοποιεί μέσω των αρχείων preset, θα πρέπει να επέμβει στον κώδικα αυτής της κλάσης, ή να δημιουργήσει το δικό του SynthesizerFactory, πιθανώς κάνοντας extend την

8 Πιθανόν η λογική σύνδεσης των κλάσεων να μπερδεύει τον αναγνώστη, ωστόσο, το βασικό αξίωμα είναι το εξής: Η SynthEngine πρέπει να ξέρει τους synthesizer της και κάθε synthesizer πρέπει να ξέρει τις παραμέτρους του. Αυτή η προσέγγιση επιτυγχάνεται με LinkedLists. Τα LinkedLists είναι πολύ ευέλικτες δομές. Στη C, κάτι αντίστοιχο θα ήταν μια μεταβλητή τιμής σε μια θέση της μνήμης και στην επόμενη θέση ένας pointer που θα δείχνει στην θέση μνήμης της επόμενης τιμής της λίστας. Έτσι, μια LinkedList μπορεί να μεγαλώνει δυναμικά, χωρίς να γνωρίζει το πρόγραμμα εκ των προτέρων τον μέγιστο αριθμό στοιχείων που μπορεί να περιέχει η λίστα.

κλάση αυτή και προσθέτοντας μεθόδους όπου θα εντοπίζουν τους ορισμούς του νέου synthesizer στο αρχείο preset. Σε οποιαδήποτε περίπτωση, εάν ο προγραμματιστής θέλει να επεκτείνει τη λειτουργικότητα των preset αρχείων, ώστε να υποστηρίζουν κάποιο νέο synthesizer, θα πρέπει να κατανοήσει τον τρόπο λειτουργίας της κλάσης SynthesizerFactory.

Η κλάση SynthesizerFactory διαθέτει ένα constructor που απαιτεί ένα όρισμα τύπου String. Το όρισμα αυτό αντιπροσωπεύει το filepath του αρχείου preset. Εξετάζοντας, όμως, τον κώδικα του constructor μπορεί να προχωρήσει καλύτερα η συζήτηση:

```
public SynthesizerFactory(String presetsFile) {

        try {
                presets.load(new FileInputStream(presetsFile));
        } catch(FileNotFoundException badFileException) {
                badFileException.printStackTrace();
        } catch(IOException ioException) {
                ioException.printStackTrace();
        }

        createSynths();
}
```

Η γραμμή του κώδικα που ενδιαφέρει πρωτίστως είναι η γραμμή εντός του try block. Το presets είναι ένα αντικείμενο της κλάσης Properties, του πακέτου java.util. Η κλάση αυτή είναι subclass της κλάσης Hashtable και αντιπροσωπεύει ένα σύνολο ιδιοτήτων. Τα αντικείμενα της κλάσης Properties μπορούν να αποθηκευτούν σε ένα αρχείο, αλλά και το αντίστροφο, με τη χρήση της μεθόδου load. Όπως φαίνεται στον κώδικα, δημιουργείται ένα νέο instance της κλάσης FileInputStream, που δέχεται σαν όρισμα το String με το όνομα του αρχείου που περιέχει τα properties (το αρχείο preset), το οποίο περνάει σαν όρισμα της μεθόδου load, του αντικειμένου presets. Έτσι, το αντικείμενο presets, περιέχει πλέον το σύνολο των properties που ορίζονται στο αρχείο πρωτοτύπων. Στη συνέχεια του constructor, μετά τα catch blocks, καλείται η μέθοδος createSynths, όπου ερμηνεύει τα δεδομένα που φορτώθηκαν στο αντικείμενο presets και δημιουργεί τους αντίστοιχους synthesizer:

```
protected void createSynths() {

        int chctr = 1;

        // if a synth is specified
        while(presets.getProperty("synth." + chctr + ".name") != null) {

                LinkedList<SynthParameter> params = createParams(chctr);

                // if it is an AMSynthesizer...
```

```
            if((presets.getProperty("synth." + chctr + ".name").equals("AMSynthesizer"))) {

                AMSynthesizer AMSynth = new AMSynthesizer(params, chctr);
            }

            // if it is an FMSynthesizer
            else if((presets.getProperty("synth." +chctr + ".name").equals("FMSynthesizer"))) {

                FMSynthesizer FMSynth = new FMSynthesizer(params, chctr);
            }

            chctr++;
        }
}
```

Στη συνθήκη του βρόγχου while ελέγχεται εάν υπάρχει ορισμένο κάποιο synth νούμερο chctr. Με λίγα λόγια, με χρήση της integer μεταβλητής chctr, η οποία αυξάνεται κατά μία μονάδα σε κάθε επανάληψη, ελέγχεται εάν υπάρχει στο αρχείο κειμένου κάποια γραμμή της μορφής "synth.(chctr).name=...". Εάν υπάρχει, με τη χρήση της μεθόδου createParams, δημιουργείται ένα LinkedList με βάση τις ιδιότητες των παραμέτρων που ορίζονται στο αρχείο preset για το συγκεκριμένο νούμερο synthesizer. Στη συνέχεια ελέγχεται εάν πρόκειται για AMSynthesizer ή για FMSynthesizer (συνθήκη if – else if). Ανάλογα με τον τύπο του synthesizer, εκτελείται το αντίστοιχο block κώδικα. Εάν πρόκειται για AMSynthesizer, δημιουργείται ένα νέο instance της κλάσης AMSynthesizer (γραμμή 47). Ο constructor της κλάσης αυτής απαιτεί δύο ορίσματα: ένα LinkedList με τις SynthParameters του AMSynthesizer και ένα integer για το νούμερο του synthesizer. Εάν πρόκειται για FMSynthesizer, δημιουργείται ένα νέο instance της κλάσης FMSynthesizer, ακριβώς με τον ίδιο τρόπο.

Ολοκληρώνοντας τη διερεύνηση της κλάσης SynthesizerFactory, θα πρέπει να σημειωθεί ότι ο προγραμματιστής μπορεί, εντός της μεθόδου createSynths, να προσθέσει μια else-if προκειμένου να καλύψει το δικό του synthesizer[9]. Το υπόλοιπο κομμάτι του κώδικα δεν χρειάζεται να υποστεί καμία μεταβολή, εκτός από τον ορισμό της αντίστοιχης κλάσης που θα κάνει extend την κλάση AbstractSensorSynthesizer.

3.3.9 Το πακέτο gr.teicrete.mta.synthesizer.event

Ύστερα από τη συζήτηση που προηγήθηκε στην παρουσίαση των προηγούμενων πακέτων, δεν έχουν μείνει πολλά να αναφερθούν για αυτό το πακέτο. Έχουν ήδη αναφερθεί οι κλάσεις SynthOnEvent, SynthOffEvent και ParameterChnagedEvent, που αντιπροσωπεύουν τα είδη των

9 Για παράδειγμα: else if((presets.getProperty("synth."+chctr+".name").equals("AdditiveSynthesizer"))) { […] }. Προς χάριν ακρίβειας, ο πιο ενδεδειγμένος τρόπος είναι η κατασκευή μιας νέας κλάσης που θα κάνει extend την κλάση SynthesizerFactory (AdditiveSynthesizerFactory). Η μέθοδος createSynths είναι protected, συνεπώς κληρονομείται στην υποκλάση.

event που μπορεί να προκύψουν από τη μονάδα παραγωγής του ήχου, που αντιπροσωπεύει το πακέτο gr.teicrete.mta.synthesizer. Αυτά τα events έχουν μια σημαντική χρησιμότητα: Αποτελούν το κομμάτι του συστήματος διάδρασης που παρέχει τη δυνατότητα υλοποίησης κάποιου είδους ανάδρασης (feedback). Ένα παράδειγμα ανάδρασης παρουσιάστηκε στο κεφάλαιο 3.2.2, όπου αναφέρθηκαν οι δυνατότητες του tab "Monitor Activity". Στη συνέχεια, στο κεφάλαιο 3.3.6, όπου παρουσιάστηκε η προγραμματιστική πλευρά του GUI, δείχτηκε ότι η κλάση MonitorPanel κάνει implement το interface SynthEventListener, που βρίσκεται στο υπό εξέταση πακέτο. Οι κλάσεις που υλοποιούν το interface αυτό, "ειδοποιούνται" όποτε δημιουργείται ένα νέο SynthOnEvent, SynthOffEvent ή ParameterChangedEvent.

Τέλος, αξίζει να αναφερθεί η κλάση SynthEventAdapter, όπου κάνει implement το interface SynthEventListener, αφήνοντας το σώμα των μεθόδων κενό. Αυτή η πρακτική (όπου εφαρμόζεται σε πολλές περιπτώσεις και στο JDK API), επιτρέπει τη γρήγορη υλοποίηση μιας μόνο μεθόδου (για παράδειγμα της parameterChangedReceived), μέσω μιας κλάσης που θα κάνει extend τον SynthEventAdapter.

4 Συμπεράσματα

Μια επαναδιατύπωση του αρχικού στόχου της πτυχιακής εργασίας θα επέτρεπε την αναφορά των βασικών σημείων που ορίστηκαν εξαρχής σαν θεμελιώδεις συνιστώσες του προσδοκώμενου αποτελέσματος και την αξιολόγηση του βαθμού στον οποίο αυτά επετεύχθησαν. Αυτά, εν συντομία μπορούν να συνοψιστούν στα εξής:

- **Πολλαπλές μορφές χρήσης της τελικής εργασίας.** Η τελική εργασία απευθύνεται τόσο στον έμπειρο προγραμματιστή (προγραμματιστική βιβλιοθήκη), όσο και στον λιγότερο έμπειρο χρήστη (γραφική εφαρμογή). Η γραφική εφαρμογή, όπως είναι αναμενόμενο, υστερεί σε δυνατότητες και είναι σαφώς πιο περιορισμένη σε σχέση με το API. Για παράδειγμα, η γραφική εφαρμογή δεν μπορεί να υποστηρίξει απευθείας ένα νέο τύπο synthesizer που μπορεί να δημιουργηθεί από έναν προγραμματιστή. Απαιτούνται πολλές αλλαγές στον ίδιο τον κώδικα των κλάσεων που αφορούν το γραφικό περιβάλλον, ώστε να εμφανίζονται σε αυτό οι νέοι τύποι synthesizer. Τέλος, θα πρέπει να τονιστεί η αξία των preset και mappings αρχείων, που η εκμάθησή τους από τους χρήστες μπορούν να επιταχύνουν πολλές χρονοβόρες επαναληπτικές διαδικασίες της γραφικής διεπαφής.
- **Ευελιξία στον προγραμματισμό των αντιστοιχιών.** Όπως αναφέρθηκε και στο κεφάλαιο 1.1 της εισαγωγής, ο στόχος της πτυχιακής εργασίας δεν αφορούσε τις αντιστοιχήσεις αυτές καθ' εαυτές, αλλά τη δυνατότητα προγραμματισμού των αντιστοιχιών, γρήγορα και με όσο το δυνατόν πιο γενικό τρόπο. Ο στόχος αυτός κρίνεται ότι επετεύχθη πλήρως, καθώς ο τρόπος υλοποίησης των κλάσεων των synthesizer, επιτρέπει στους ίδιους τους synthesizer να γνωρίζουν και να διαχειρίζονται τις παραμέτρους τους με τον κατάλληλο για αυτούς τρόπο. Ένα σημαντικό μειονέκτημα σε αυτό το σημείο είναι ότι δεν υποστηρίζεται ο τύπος αντιστοίχησης ένα-προς-πολλά (βλ Κεφάλαιο 1.4 και Εικόνα 1.4.2).
- **Επεκτασιμότητα.** Οι δυνατότητες επέκτασης του τελικού API, όσον αφορά τη μονάδα παραγωγής του ήχου είναι, πρακτικά, ανεξάντλητες. Η εκμετάλλευση των χαρακτηριστικών του αντικειμενοστραφούς προγραμματισμού με τη Java, οδήγησε σε αυτό το επιθυμητό αποτέλεσμα. Ωστόσο, η δυνατότητα επέκτασης παρέχεται μόνο στον έμπειρο προγραμματιστή, καθώς ο απλός χρήστης δεν διαθέτει την απαραίτητη εξοικείωση με την γλώσσα προγραμματισμού Java, που απαιτείται για την επέκταση της προγραμματιστικής βιβλιοθήκης.

Οπωσδήποτε, θα πρέπει να τονιστεί για μια ακόμη φορά ότι η πτυχιακή αυτή εργασία, δεν αποτελεί σε καμία περίπτωση μια καινοτόμα πρόταση στο επίπεδο των αντιστοιχήσεων. Πολύ περισσότερο, φιλοδοξεί να αποτελέσει – με την ανάλογη επέκταση στον τομέα σύνθεσης του ήχου – ένα χρήσιμο εργαλείο για την εύκολη δημιουργία και δοκιμή των αντιστοιχήσεων στα Ψηφιακά Μουσικά Όργανα.

Ιδιαίτερα χρήσιμη θα μπορούσε να αποδειχτεί η δυνατότητα που παρέχει η τελική εφαρμογή για επεξεργασία και παρακολούθηση των αντιστοιχήσεων σε πραγματικό χρόνο. Με αυτόν τον τρόπο, ο τελικός χρήστης μπορεί να δοκιμάσει γρήγορα και εύκολα διαφορετικά είδη αντιστοιχήσεων.

5 Βιβλιογραφία - Αναφορές

1. Eckel, B. (2003). *"Thinking in Java 3rd Edition"*, Prentice Hall.
2. Greanier, T. (2005). *"Java – Εισαγωγή στη Σύγχρονη Τεχνολογία"*, Αθήνα: Μ. Γκιούρδας.
3. Hunt, A., Wanderley, M.M. & Kirk, R. (2000). *"Towards a Model for Instrument Mapping in Expert Musical Interaction"*.
4. Knapp, R.B. & Cook, P.R. *"The Integral Music Controller: Introducing a Direct Emotional Interface to Gestural Control of Sound Synthesis"*.
5. Miranda, E.R. & Wanderley, M.M. (2006). *"New Digital Instruments: Control and Interaction Beyond the Keyboard",* Middleton: A-R Editions.
6. Putnam, W. & Knapp, R.B. (1996). *"Input/Data Acquisition System Design for Human Computer Interfacing"*.
7. Rovan, J.B., Wanderley, M.M., Dubnov, S. & Depalle, P. *"Instrumental Gestural Mapping Strategies as Expressivity Determinants in Computer Music Performance"*, IRCAM – France.
8. Wanderley, M.M. (2000). "*Gestural Control Of Music",* IRCAM – Centre Pompidou.
9. Διαμαντόπουλος, Τ. (2004). *"Προγραμματισμός & Σύνθεση Ήχου"*, Αθήνα: Εκδόσεις Έλλην.