ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

HOU-CS-UGP-2012-02

«Ανάπτυξη Συστήματος Ηχομέτρησης σε Φορητή Πλατφόρμα Κινητής Τηλεφωνίας»

ΑΝΤΩΝΙΟΣ ΣΑΡΡΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΑΝΔΡΕΑΣ ΦΛΩΡΟΣ

HELLENIC OPEN UNIVERSITY
SCHOOL OF SCIENCES AND TECHNOLOGY

WWW.EAP.GR

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
HELLENIC OPEN UNIVERSITY
SCHOOL OF SCIENCES AND TECHNOLOGY

Πτυχιακή Εργασία *HOU-CS-UGP-2012-02*

Ανάπτυξη Συστήματος Ηχομέτρησης σε Φορητή Πλατφόρμα Κινητής Τηλεφωνίας

Αντώνιος Σαρρής

Ανάπτυξη Συστήματος Ηχομέτρησης σε Φορητή Πλατφόρμα Κινητής Τηλεφωνίας

Αντώνιος Σαρρής

ΑΝΔΡΕΑΣ ΦΛΩΡΟΣ **ΑΘΑΝΑΣΙΟΣ ΣΚΟΔΡΑΣ** **ΒΑΣΙΛΕΙΟΣ ΚΟΥΤΚΙΑΣ**

Επίκουρος Καθηγητής
Τμήμα Τεχνών Ήχου και Εικόνας
Ιόνιο Πανεπιστήμιο

Καθηγητής
Σχολή Θετικών Επιστημών και Τεχνολογίας
Ελληνικό Ανοικτό Πανεπιστήμιο

Δρ. Ιατρικής Πληροφορικής
Εθνικό Κέντρο Έρευνας & Τεχνολογικής Ανάπτυξης

Περίληψη: Η παρούσα εργασία παρουσιάζει την ανάπτυξη ενός συστήματος λογισμικού μετρήσεων ηχοστάθμης σε πλατφόρμα κινητής τηλεφωνίας. Αρχικά παρατίθεται μελέτη των βασικών μεθοδολογιών επεξεργασίας σημάτων, καθώς και του ακουστικού θορύβου, και αξιολογούνται τα χαρακτηριστικά των διαθέσιμων λειτουργικών συστημάτων. Η εξέταση των ως άνω χαρακτηριστικών αποτελεί τη βάση για την επιλογή της πλατφόρμας κινητής τηλεφωνίας για την υλοποίηση της εφαρμογής, η οποία στη συγκεκριμένη περίπτωση είναι η πλατφόρμα iOS. Η επιλογή της πλατφόρμας καθορίζει και το αναπτυξιακό περιβάλλον που πρέπει να χρησιμοποιηθεί, το οποίο για την συγκεκριμένη εφαρμογή είναι το περιβάλλον Xcode και η γλώσσα προγραμματισμού Objective C/C++. Η ανάπτυξη του συστήματος λογισμικού πραγματοποιήθηκε σύμφωνα με τη μεθοδολογία ICONIX, η οποία βασίζεται στην αξιοποίηση ενός υποσυνόλου της UML για τη δημιουργία διαγραμμάτων ως ενδιάμεσα προϊόντα κατά την εξέλιξη τόσο του δυναμικού, όσο και του στατικού μοντέλου του συστήματος. Η εφαρμογή αναπτύχθηκε σύμφωνα με τις επιλογές και τις δυνατότητες ενός τυπικού ηχομέτρου, ώστε να παρέχει στον χρήστη εκτός από τις βασικές παραμέτρους μέτρησης (στιγμιαία και μέση ηχοστάθμη, μέγιστο και ελάχιστο) και τη δυνατότητα επιλογών ανάμεσα στις σταθμίσεις A, B, C, καθώς και μεταξύ συχνότητας δειγματοληψίας fast / slow ανάλογα με την επιλογή του time resolution. Επίσης υπάρχει δυνατότητα επιλογής του τρόπου εμφάνισης των αποτελεσμάτων ανάμεσα σε τρείς γραφικές παραστάσεις ή τη κλασική βελόνα. Τέλος, η εργασία παρουσιάζει τα αποτελέσματα της σύγκρισης των μετρήσεων του συστήματος που αναπτύχθηκε, με αυτά μιας εξειδικευμένης μετρητικής συσκευής τύπου 1 και μιας εξειδικευμένης εφαρμογής για κινητά τηλέφωνα, για τυπικές μορφές θορύβου.

Λέξεις-κλειδιά: Ηχόμετρο, Ηχοστάθμη, Θόρυβος, Καμπύλες Στάθμισης, FFT, iOS.

Περιεχόμενο: Κείμενο, πρόγραμμα σε γλώσσα Objective C/C++, διαγράμματα.

Abstract: This thesis examines the development of sound pressure measurement software on a mobile telephone platform. Firstly, we study the main methodologies of signal processing, as well as acoustic noise and evaluate the characteristics of available operating systems. These characteristics are the basis for the selection of a mobile telephone platform, which in this case is the iOS platform. The selection of a platform dictates the development environment that has to be used, which for iOS is the Xcode environment with Objective C/C++ programming language. The software development follows the ICONIX methodology, which is based on the creation of diagrams as intermediate products during the development of the static, as well as the dynamic system model, using a subset of UML. The application was developed according to the features and capabilities of a typical sound meter: in addition to the basic measurement parameters (instantaneous and average sound level, maximum and minimum) the user can choose between A, B, C weighting (or no weighting) and fast / slow sampling frequency. The application also features alternative visualizations of the measurement output, comprising three graph styles and the classic needle. Finally, the thesis presents a comparative analysis of the measurements acquired by the software, with those obtained through the use of a specialized sound measuring instrument and specialized sound measurement software for iPhone mobile phones, for typical noise levels.

Design and development of a software application that measures sound levels on a mobile telephone platform

Antonios Sarris

Andreas Floros	**Athanasios Skodras**	**Vasileios Koutkias**
Assistant Professor, Dept. of Audiovisual Arts, Ionian University	Professor, School of Sciences and Technology, Hellenic Open University	Dr. Medical Informatics Center for Research & Technology Hellas

The subject of this thesis is the design and development of a software application that measures sound levels on a mobile telephone platform. The constant evolution of mobile systems development has permitted the development and support of new types of applications. Such applications escape the narrow limits of simple communication and are based on development environments (SDKs), which allow the full utilization of the operational sub-units of devices. The software developed in the thesis takes advantage of the microphone available on mobile devices, in order to record and manage sound signals.

The selection of a platform and, consequently, of an operating system was based on a thorough search and evaluation of the characteristics of available operating systems, with respect to their suitability to the requirements of the application. This evaluation limited the number of available options between the two most popular operating systems, whose combined market share makes up most of the market, namely iOS and Android OS. Finally, iOS was selected as the platform of choice, therefore the software development environment of the object-oriented programming language Objective C/C++ was used for development.

iOS is the operating system used by Apple in all of its mobile platforms (iPod Touch, iPhone and iPad). It is a closed-code operating system designed by Apple exclusively for its own devices. Xcode Developer Tools is the programming environment employed for the development of the application. The environment consists of the Builder, LLVM compiler, iOS simulator and Instruments (observation and testing) tools.

The technical specifications of the application include the characteristics of a typical sound level measuring instrument. Specifically, the application supports the following features:

- fast / slow recording.
- Time Resolution
- Unweighted measurement Lin.
- A-curve measurement.
- B-curve measurement
- C-curve measurement.
- Instantaneous sound level measurement.
- Average sound level measurement.
- Minimum sound level recording.
- Maximum sound level recording.
- Ability to change the calibration constant.
- Measurement results sending via email.

Furthermore, the user has the ability to choose among three alternative results visualizations, including three different graphs and the classic needle.

In order to implement the aforementioned technical specifications, a theoretical study of sound pressure level (SPL) measurement methodologies, as well as relevant topics in digital sound processing (sampling, fast Fourier transform, frequency domain representation) was undertaken.

Sound is the result of pressure changes in a transmission medium, such as air, which are caused by vibrations and disturbances. The quantities used to describe sound are the range of the sound pressure changes and the number of pressure changes per second. Sound pressure

level (which is the result of air pressure change measurement) is measured in decibels (dB). This quantity is directly related to sound volume. Sound frequency is measured in Hz. A dB is not an absolute measurement unit. It is the ratio of a measured quantity to some predetermined reference level. The scale is logarithmic and uses 20μPa as the sound pressure of reference. The hearing threshold is set to 0 dB, whereas 120 dB is the approximate threshold for pain. The human ear is able to detect frequencies ranging between 20 Hz and 20,000 Hz. The quantity used for measurements is the one that corresponds to the average noise level, which is determined as the value for SPL during one time period.

A, B and C weighting were developed in order to interpret sound changes with respect to frequency in a way similar to the human ear. A-weighting corresponds better to high, B to medium and C to low frequencies. Notwithstanding, A weighting is the one that has been established for noise estimation, as it is highly correlated to the subjective response of the human hearing system to various sound stimuli.

Software development was based on the ICONIX methodology. This methodology utilizes a subset of UML for the creation of diagrams as intermediate products during the development of the dynamic, as well as the static model of the system. Specifically, out of the UML diagram set, the methodology implements use case diagrams, class diagrams, robustness diagrams and sequence diagrams.

The calculation of SPL is based on the following procedure. Let x(t): $0 \leq t \leq T$ be a recorded sound signal, where T is the total signal recording time in seconds. The specific sound signal is assumed to be sampled at a sampling frequency Fs (Samples/sec), so that it is represented as a set of $Fs \cdot T$ samples.

The calculation of SPL in dB, dBA, dBB and dBC for the aforementioned sound signal is as follows:

The set of sound signals is partitioned to non-overlapping windows of Δt duration, depending on the operation mode (fast/slow) and the selected time resolution value. Subsequently, the signal FFT (X(k), $0 \leq k \leq N-1$, where N is the number of samples) for the specific time window is calculated. The calculation of the selected weighting filter for the same time window is given by the following equations:

a. A weighting, $X_A(k) = \alpha_A(f_k) \cdot X(k)$ for $f_k = k \cdot \Delta f$ where $\Delta f = F_s / N$ and $\alpha_A(f) = \frac{12200^2 \cdot f^4}{(f^2+20{,}6^2)\cdot(f^2+12200^2)\cdot((f^2+107{,}7^2)^{0{,}5})\cdot((f^2+737.9^2)^{0{,}5})}$.

b. B weighting, $X_B(k) = \alpha_B(f_k) \cdot X(k)$ for $f_k = k \cdot \Delta f$ where $\Delta f = F_s / N$ and $\alpha_B(f) = \frac{12200^2 \cdot f^3}{(f^2+20{,}6^2)\cdot(f^2+12200^2)\cdot((f^2+158{,}5^2)^{0{,}5})}$.

c. C weighting, $X_C(k) = \alpha_C(f_k) \cdot X(k)$ for $f_k = k \cdot \Delta f$ where $\Delta f = F_s / N$ and

$$\alpha_C(f) = \frac{12200^2 \cdot f^2}{(f^2+20{,}6^2)\cdot(f^2+12200^2)}.$$

The signal energy is:

a. No weighting L_{in} $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X(k)|^2$.

b. A weighting $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_A(k)|^2$.

c. B weighting $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_B(k)|^2$.

d. C weighting $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_C(k)|^2$.

Therefore, the average signal energy for every possible weighting is given by:

a. No weighting L_{in} $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X(k)|^2$.

b. A weighting $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_A(k)|^2$.

c. B weighting $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_B(k)|^2$.

d. C weighting $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_C(k)|^2$.

Finally, SPL is given by:

$$SPL = 10log_{10}\left(\frac{\bar{\varepsilon}_x}{\bar{\varepsilon}_{ref}}\right) = 10log_{10}(\bar{\varepsilon}_x) - 10log_{10}(\bar{\varepsilon}_{ref})$$

where the units, depending on the type of weighting will be in dB, dBA, dBB, dBC.

The quantity $\bar{\varepsilon}_{ref}$ is the energy of the reference level, which is a constant, a fact that implies that the term $10log_{10}(\bar{\varepsilon}_{ref})$ can be replaced by a constant C, which includes the deviation of sound pressure level from real values. We therefore have

$SPL = 10log_{10}\left(\frac{\bar{\varepsilon}_x}{\bar{\varepsilon}_{ref}}\right) = 10log_{10}(\bar{\varepsilon}_x) + C$, where C is the scaling constant.

In short, the operation of the system is as follows. The system receives 44100 values per second (ranging between [-32768,+32767] Sint16) in total from the microphone. These values are received in blocks, the number of which depends on the selection of the user between slow or fast operation mode of the measuring system, as well as between high, normal or low time resolution values. The received values per block are normalized to the [-1, 1] interval and the FFT and frequency response of the filter (A, B, C or no filter) are calculated. The instantaneous SPL value per sound sample block is based on the average energy value, which is in turn based on the total energy of the weighted signal. The value for the current SPL is calculated as the average of the instantaneous SPL values.

Finally, the thesis concludes with the use of the developed measuring system for the measurement of typical noise sources. The measurements were taken in the national road, a quiet room, a mild circulation road and in a store. In order to determine the original microphone gain, as well as scaling purposes, the results were compared to those obtained through the use of a type 1 sound meter, during stable sound emission. The same sound meter was used to compare measurements in a laboratory. The rest of the measurements were compared to those obtained using specialized sound measurement software for iPhone mobile phones. The results of the comparison led to the conclusion that the technical characteristics of the microphone vary among the various models, which yields that the calibration process should be conducted on a model specific basis. Finally, our experimentation revealed the significance of the microphone's response in measuring the current value of the sound pressure level.

Περιεχόμενα

1 Εισαγωγή 15
2 Φορητές πλατφόρμες – Λειτουργικά συστήματα 19
2.1 Φορητές Πλατφόρμες 19
2.1.1 Κινητά Τηλέφωνα 19
2.1.2 Προσωπικοί Ψηφιακοί Βοηθοί (PDAs) 20
2.1.3 Έξυπνα Κινητά Τηλέφωνα (Smartphones) 20
2.1.4 Ταμπλέτες (Tablet PCs) 22
2.2 Λειτουργικά Συστήματα 23
2.2.1 Android OS 24
2.2.2 Bada OS 25
2.2.3 BlackBerry OS 26
2.2.4 iOS 27
2.2.5 Symbian OS 29
2.2.6 WebOS 30
2.2.7 Windows Phone OS 31
2.3 Διαλειτουργικότητα 33
3 Βασικές Έννοιες Σημάτων 35
3.1 Η Έννοια του Σήματος 35
3.2 Συνάρτηση Δέλτα Συνεχούς Χρόνου 35
3.3 Ο Μετασχηματισμός Fourier 36
3.3.1 Ο Διακριτός Μετασχηματισμός Fourier 37
3.3.2 Ταχείς Αλγόριθμοι Διακριτού Μετασχηματισμού Fourier 38
3.3.3 Ταχύς Μετασχηματισμός Fourier με Αποδεκάτιση Χρόνου 39
3.3.4 Χρήσιμες Ιδιότητες Μετασχηματισμού Fourier 40
3.4 Δειγματοληψία Σημάτων Συνεχούς Χρόνου 40
3.4.1 Περιοδική Δειγματοληψία 41
3.4.2 Το Θεώρημα Δειγματοληψίας 41
3.5 Κβάντωση Σημάτων 41
3.6 Γραμμικά Χρονικά Αναλλοίωτα Συστήματα 42
3.6.1 Η Ιδιότητα της Γραμμικότητας 42
3.6.2 Κρουστική Απόκριση και Συνέλιξη 42
3.6.3 Αρμονική Απόκριση και Συνάρτηση Μεταφοράς 44
4 Ακουστικός Θόρυβος 46
4.1 Φυσικές Ιδιότητες Θορύβου 46
4.1.1 Δονήσεις και Κύματα 46

4.1.2 Ακουστική Ισχύς, Ενεργειακή Πυκνότητα και Ένταση 48
4.1.3 Σημειακή Πηγή 49
4.1.4 Γραμμική Πηγή 49
4.1.5 Κλίμακες Θορύβου 51
4.1.6 Χαρακτηριστικά Ενθόρυβων Σημάτων 54
4.2 Ψυχοακουστική - Μέτρα Θορύβου 56
4.2.1 Ο Μηχανισμός της Ακοής 56
4.2.2 Επιπτώσεις Θορύβου και Επιτρεπτά Επίπεδα 58
4.2.3 Εναλλακτικά Μέτρα Θορύβου 60
4.3 Υπολογισμός Επιπέδου Ηχητικής Πίεσης 62
4.4 Ηχόμετρα 63
4.5 Μικρόφωνα 65
5 Σχεδιασμός 69
5.1 Απαιτήσεις Υψηλού Επιπέδου 69
5.2 Μοντέλο Περιπτώσεων Χρήσης (Use Case Model) 71
5.2.1 Διάγραμμα Περιπτώσεων Χρήσης (Use Case Diagram) 71
5.2.2 Ενδεικτικές Οθόνες του Συστήματος 72
5.2.3 Τεκμηρίωση Περιπτώσεων Χρήσης 74
5.3 Διάγραμμα Πεδίου Προβλήματος (Αρχικό) 77
5.4 Διαγράμματα Ευρωστίας Συστήματος 78
5.5 Διάγραμμα Ακολουθίας Συστήματος 84
5.6 Διάγραμμα Κλάσεων (Αναθεωρημένο) 90
5.7 Διάγραμμα Κλάσεων (Τελικό) 91
6 Ανάπτυξη Λογισμικού 92
6.1 Επιλογή Λειτουργικού Συστήματος 92
6.2 iOS Software Development Kit 94
6.3 Η Γλώσσα Προγραμματισμού Objective-C 95
6.4 Εργαλεία Προγραμματισμού 100
6.5 Επισκόπηση Συστήματος 101
6.6 Τεχνική Ανάλυση Λογισμικού 102
6.7 Προγραμματιστικές Διεπαφές 108
7 Η εφαρμογή iSPL PRO 114
7.1 Εγκατάσταση εφαρμογής 114
7.2 Χρήση της εφαρμογής 115
7.2.1 Αρχική οθόνη 115
7.2.2 Οθόνη επιλογών - ρυθμίσεων 118

7.2.3 Οθόνη πληροφοριών 119
8 Συγκριτικές Δοκιμές – Μετρήσεις Τυπικών Μορφών Θορύβου 120
8.1 Βαθμονόμηση – Αρχικές Μετρήσεις 121
8.2 Μέτρηση Τυπικών Πηγών Θορύβου - Συγκριτικές Μετρήσεις 125
8.2.1 Μέτρηση Κυκλοφοριακού Θορύβου Εθνικής Οδού 125
8.2.2 Μέτρηση σε Δρόμο Ήπιας Κυκλοφορίας 126
8.2.3 Μέτρηση σε Κατάστημα 127
8.2.4 Μέτρηση σε Ήσυχο Δωμάτιο 127
8.3 Συμπεράσματα Μετρήσεων 128
9 Συμπεράσματα 130
Αναφορές 133
Παράρτημα Α: Πηγαίος κώδικας 135

1 Εισαγωγή

Στόχος της παρούσας πτυχιακής εργασίας είναι η σχεδίαση και ανάπτυξη ενός συστήματος λογισμικού το οποίο θα χρησιμοποιείται για την πραγματοποίηση μετρήσεων ηχοστάθμης. Το συγκεκριμένο λογισμικό υλοποιήθηκε σε φορητή πλατφόρμα (smartphone) και έχει τη δυνατότητα να εκμεταλλεύεται το μικρόφωνο που διαθέτει η συσκευή για την καταγραφή και διαχείριση των ηχητικών σημάτων. Η ανάπτυξη του παραπάνω συστήματος λογισμικού έγινε με χρήση του αναπτυξιακού περιβάλλοντος μιας αντικειμενοστραφούς γλώσσας προγραμματισμού. Η τελική επιλογή της πλατφόρμας και κατ' επέκταση του λειτουργικού συστήματος πραγματοποιήθηκε ύστερα από διερεύνηση και αξιολόγηση των διαθέσιμων χαρακτηριστικών τους ως προς την καταλληλότητά τους για τις ανάγκες της συγκεκριμένης εφαρμογής στη φάση της ανάπτυξης του κύκλου ζωής του λογισμικού.

Οι συνέχειες εξελίξεις στον τομέα ανάπτυξης φορητών συστημάτων έχει επιτρέψει την ανάπτυξη και υποστήριξη νέων μορφών εφαρμογών. Οι εφαρμογές αυτές ξεφεύγουν από τα στενά όρια μιας απλής επικοινωνίας και στηρίζονται σε αναπτυξιακά περιβάλλοντα (SDKs), επιτρέποντας την πλήρη χρησιμοποίηση των λειτουργικών υπομονάδων των συσκευών.

Βασικό αντικείμενο της συγκεκριμένης πτυχιακής εργασίας είναι ο σχεδιασμός και η ανάπτυξη μιας εφαρμογής λογισμικού για την μέτρηση της ηχητικής στάθμης (Sound Pressure Level) σε μια φορητή συσκευή κινητής τηλεφωνίας. Στις τεχνικές δυνατότητες της εφαρμογής συμπεριλαμβάνονται και τα χαρακτηριστικά ενός τυπικού συστήματος μέτρησης. Συγκεκριμένα, υποστηρίζονται οι εξής δυνατότητες:

- Λειτουργία καταγραφής fast / slow.
- Λειτουργία επιλογής time resolution.
- Μέτρηση στιγμιαίας ηχοστάθμης με χρήση καμπύλης A.
- Μέτρηση στιγμιαίας ηχοστάθμης με χρήση καμπύλης B.
- Μέτρηση στιγμιαίας ηχοστάθμης με χρήση καμπύλης C.
- Μέτρηση στιγμιαίας ηχοστάθμης σε χωρίς την χρήση καμπυλών, Lin.
- Μέτρηση μέσης ηχοστάθμης.
- Καταχώρηση ελάχιστης τιμής ηχοστάθμης.
- Καταχώρηση μέγιστης τιμής ηχοστάθμης.

Για την υλοποίηση των προαναφερθέντων τεχνικών προδιαγραφών απαιτήθηκε η θεωρητική μελέτη των μεθοδολογιών μέτρησης του επιπέδου ηχητικής πίεσης (SPL) και συναφών θεμάτων ψηφιακής επεξεργασίας ήχου (δειγματοληψία, ταχύς μετασχηματισμός Fourier, αναπαράσταση στο πεδίο των συχνοτήτων).

Η ανάπτυξη του συστήματος λογισμικού έγινε βάσει της μεθοδολογίας ICONIX [4]. Η μεθοδολογία αυτή βασίζεται στην αξιοποίηση ενός υποσυνόλου της UML για την δημιουργία

διαγραμμάτων ως ενδιάμεσα προϊόντα κατά την εξέλιξη του δυναμικού και στατικού μοντέλου του συστήματος που αναπτύσσεται. Συγκεκριμένα από το σύνολο των διαγραμμάτων της UML, χρησιμοποιούνται τα διαγράμματα περιπτώσεων χρήσης, τα διαγράμματα κλάσεων, τα διαγράμματα ευρωστίας και τα διαγράμματα ακολουθίας.

Κατά τη φάση της υλοποίησης έγινε η συγγραφή του κώδικα της εφαρμογής ο οποίος αποτυπώνει τα αποτελέσματα που έχουν παραχθεί κατά τη φάση του σχεδιασμού.

Η πτυχιακή εργασία ολοκληρώθηκε με τη χρησιμοποίηση του μετρητικού συστήματος που αναπτύχθηκε για την μέτρηση τυπικών μορφών θορύβου (π.χ. κυκλοφοριακού) και τη σύγκριση των λαμβανομένων τιμών ηχοστάθμης με αυτές που προκύπτουν από τη χρήση εξειδικευμένης μετρητικής συσκευής.

Οι φάσεις ανάπτυξης του συστήματος ηχομέτρησης παρουσιάζονται παρακάτω.

- ✓ Φάση Α - Ανάλυση: Επισήμανση στόχων, αντικειμένου, θεωρητική μελέτη SPL και επεξεργασίας ήχου.
- ✓ Φάση Β – Σχεδιασμός (εφαρμογή μεθόδου *ICONIX):* Ανάλυση απαιτήσεων (υψηλού επιπέδου και πεδίο προβλήματος), δημιουργία διαγραμμάτων (περιπτώσεων χρήσης, κλάσεων – αρχικό, ευρωστίας, ακολουθίας, κλάσεων – τελικό).
- ✓ Φάση Γ – Υλοποίηση / Ανάπτυξη: Επιλογή λειτουργικού συστήματος, εγκατάσταση και εξοικείωση με Software Developer Kit, συγγραφή κώδικα, γραφικό περιβάλλον – GUI.
- ✓ Φάση Δ – Ολοκλήρωση: Εγκατάσταση προγράμματος, έλεγχος καλής λειτουργίας, χρήση μετρητικού συστήματος για την μέτρηση τυπικών μορφών θορύβου και ταυτόχρονη σύγκριση των λαμβανομένων τιμών με την χρήση εξειδικευμένης μετρητικής συσκευής (ηχόμετρο) και εξειδικευμένης εφαρμογής για iPhone.

Πέραν του γενικού στόχου που αφορά στην υλοποίηση μιας εφαρμογής λογισμικού πάνω σε κινητή πλατφόρμα, η παρούσα εργασία θέτει ένα σύνολο από επιμέρους εκπαιδευτικούς στόχους.

Πρωταρχικός εκπαιδευτικός στόχος είναι η εμβάθυνση σε θέματα ψηφιακής επεξεργασίας σήματος που άπτονται άμεσα του θεωρητικού υπολογισμού του επιπέδου της ηχητικής πίεσης. Στην κατηγορία αυτή εντάσσονται οι έννοιες της δειγματοληψίας, της κβάντωσης, του υπολογισμού του ταχέως μετασχηματισμού Fourier, καθώς και του υπολογισμού συνελικτικών φίλτρων όπως είναι οι καμπύλες στάθμισης A, B και C. Η μελέτη των θεμάτων αυτών κρίνεται απαραίτητη καθώς προσδιορίζουν το σύνολο των λειτουργικών απαιτήσεων της εφαρμογής.

Η προγραμματιστική υλοποίηση των παραπάνω εννοιών στα πλαίσια μιας εφαρμογής που λειτουργεί σε πραγματικό χρόνο συνιστά ένα επιπλέον εκπαιδευτικό ζητούμενο. Λαμβάνοντας υπόψη την πολυπλοκότητα και τις ανάγκες μνήμης των υπολογισμών που θα πρέπει να πραγματοποιούνται σε πολύ μικρό χρονικό διάστημα, θα πρέπει να διερευνηθούν αποδοτικοί τρόποι υπερκέρασης αυτών των προβλημάτων μέσω της χρήσης εξειδικευμένων βιβλιοθηκών λογισμικού.

Η πραγματοποίηση του σχεδιασμού και της ανάλυσης ενός ολοκληρωμένου έργου λογισμικού μέσω μιας γλώσσας μοντελοποίησης αποτελεί έναν εξίσου σημαντικό εκπαιδευτικό στόχο που θέτει η παρούσα εργασία. Έχοντας κατά νου ότι απώτερος σκοπός είναι η παραγωγή μιας ανταγωνιστικής εμπορικής εφαρμογής κρίνεται απαραίτητη η χρήση της ενοποιημένης γλώσσας μοντελοποίησης (UML), προκειμένου να διασφαλιστεί η πληρότητα και η ποιότητα του τελικού προϊόντος. Μέσα σε ατό το πλαίσιο γίνεται κατανοητή η χρησιμότητα μιας αντικειμενοστραφούς μεθοδολογίας για την ανάπτυξη του λογισμικού. Επιπλέον, πραγματοποιείται με συστηματικό τρόπο η αντικειμενοστραφής ανάλυση και ο σχεδιασμός του προβλήματος κάνοντας χρήση των διαγραμμάτων της UML.

Ολοκληρώνοντας την αναφορά στους εκπαιδευτικούς στόχους της εργασίας, επισημαίνεται η απόκτηση εμπειρίας σε θέματα προγραμματισμού που αφορούν πλατφόρμες κινητής τηλεφωνίας. Στο συγκεκριμένο θέμα δίνεται ιδιαίτερη έμφαση εξαιτίας των εγγενών περιορισμών που προκύπτουν από την διαθέσιμη μνήμη και την υπολογιστική ισχύ των συσκευών αυτών.

Η συνεισφορά της παρούσας εργασίας εστιάζεται στην ενσωμάτωση μιας πληθώρας χαρακτηριστικών που μπορεί να συναντήσει κανείς μόνο μεμονωμένα σε αντίστοιχες υλοποιήσεις. Συγκεκριμένα η εφαρμογή αυτή έχει την δυνατότητα στάθμισης με χρήση του συνόλου των καθιερωμένων καμπυλών A, B, C καθώς και την δυνατότητα μέτρησης της ηχοστάθμης χωρίς την χρήση καμπυλών. Επιπλέον, το σύστημα είναι σε θέση να εναλλάσσει μεταξύ τριών διαφορετικών τρόπων για την γραφική απεικόνιση των μετρήσεων του στιγμιαίου επιπέδου της ηχητικής στάθμης. Στην κατηγορία αυτή η εφαρμογή παρέχει την μοναδική δυνατότητα απεικόνισης του σχετικού μεγέθους της τρέχουσας ηχητικής πίεσης, το οποίο υπολογίζεται ως η διαφορά μεταξύ της τρέχουσας και της ελάχιστης τιμής του SPL. Στα μοναδικά χαρακτηριστικά του συστήματος εντάσσεται και η επιλογή της τιμής του Time Resolution, που αποτελεί την αλλαγή της ταχύτητας της μέτρησης και καθορίζει το επίπεδο της λεπτομέρειάς της.

Η οργάνωση της εργασίας έχει ως ακολούθως. Στο κεφάλαιο 2 γίνεται συγκριτική μελέτη των τεχνικών χαρακτηριστικών και δυνατοτήτων που αφορούν στις φορητές πλατφόρμες και τα λειτουργικά συστήματα. Στο κεφάλαιο 3 πραγματοποιείται επισκόπηση των βασικών

εννοιών σημάτων που είναι απαραίτητες για τον θεωρητικό υπολογισμό του επιπέδου της ηχητικής πίεσης. Στο κεφάλαιο 4 η ανάλυση επικεντρώνεται στον ακουστικό θόρυβο και τις φυσικές ιδιότητες του. Μάλιστα δόθηκε ιδιαίτερη έμφαση στο μηχανισμό της ακοής και στις επιπτώσεις του θορύβου σε αυτόν. Επιπλέον γίνεται αναφορά τόσο στα καθιερωμένα όσο και σε εναλλακτικά μέτρα θορύβου. Το κεφάλαιο αυτό ολοκληρώνεται με την παρουσίαση της μεθοδολογίας για τον υπολογισμό του επιπέδου της ηχητικής πίεσης. Στο κεφάλαιο 5 αναπτύσσεται η εφαρμογή της μεθόδου ICONIX στον σχεδιασμό της εφαρμογής. Το Κεφάλαιο 6 αφορά στην ανάπτυξη του λογισμικού. Στο κεφαλιά 7 παρουσιάζεται η εφαρμογή που αναπτύχθηκε. Στο κεφάλαιο 8 παρουσιάζονται οι συγκριτικές δοκιμές που διενεργήθηκαν κατά την μέτρηση τυπικών μορφών θορύβου. Τέλος, στο κεφάλαιο 9 συνοψίζονται τα συμπεράσματα αυτής της εργασίας.

2 Φορητές πλατφόρμες – Λειτουργικά συστήματα

Στο παρόν κεφάλαιο γίνεται σύγκριση μεταξύ των χαρακτηριστικών τόσο στις φορητές πλατφόρμες όσο και στα λειτουργικά συστήματα, καθώς και μια αναφορά στη διαλειτουργικότητα.

2.1 Φορητές Πλατφόρμες

Μια φορητή πλατφόρμα είναι ένας υπολογιστής σε σχετικά μικρές διαστάσεις, συνήθως αποτελείται από μία οθόνη και ένα ενσωματωμένο πληκτρολόγιο, έτσι ώστε να μπορεί να μετακινείται εύκολα. Οι φορητές πλατφόρμες εμφανίστηκαν ευρέως στην καθημερινότητα των ανθρώπων από τη δεκαετία του 90. Αρχικά η χρήση τους περιοριζόταν στην φωνητική επικοινωνία και στην υπηρεσία SMS (Short Message Service). Λίγο αργότερα από τα κινητά τηλέφωνα, εμφανίστηκαν και οι πρώτοι προσωπικοί ψηφιακοί βοηθοί (PDAs), οι οποίοι είχαν ως βασική δυνατότητα τη σύνδεση στο διαδίκτυο. Ο συνδυασμός των δυο παραπάνω κατηγοριών δημιούργησε τα «έξυπνα κινητά τηλέφωνα» (smartphones) και κατ' επέκταση τις ταμπλέτες (Tablet PCs).

2.1.1 Κινητά Τηλέφωνα

Κινητό ονομάζεται το τηλέφωνο που δεν έχει καλωδιακή σύνδεση με το δίκτυο παροχής τηλεφωνίας και δεν εξαρτάται από κάποια τοπική ασύρματη συσκευή εκπομπής ραδιοφωνικού σήματος χαμηλής συχνότητας.

Σχήμα 1: Εξέλιξη κινητών τηλεφώνων

Η «γέννηση» της κινητής τηλεφωνίας έγινε στις 3 Απριλίου του 1973, όταν ο Martin Cooper [15] της Motorola έκανε το πρώτο δοκιμαστικό τηλεφώνημα από τη Νέα Υόρκη. Η συσκευή που κρατούσε είχε βάρος 900 γραμμάρια, ύψος 25 εκατοστά και κωδική ονομασία Dyna-Tac. Όμως το πρώτο αυτοματοποιημένο δίκτυο κινητής τηλεφωνίας λειτούργησε στις αρχές της δεκαετίας του 80 στη Σκανδιναβία όπου τα πρώτα κινητά τηλέφωνα ήταν εγκατεστημένα σε αυτοκίνητα. Οι συσκευές που κυκλοφόρησαν εκείνη την περίοδο αποτέλεσαν την πρώτη γενιά κινητών τηλεφώνων (1G). Στην αρχή της δεκαετίας του 90 υπήρξε ραγδαία εξέλιξη των κινητών τηλεφώνων με τη ψηφιοποίηση δικτύων (GSM) και συσκευών. Οι νέες συσκευές δεν ξεπερνούσαν σε βάρος τα

200 γραμμάρια και εκτός από τη φωνητική επικοινωνία είχαν τη δυνατότητα να αποστέλλουν σύντομα γραπτά μηνύματα (SMS). Με αυτές τις συσκευές ήμασταν πια στη δεύτερη γενιά κινητών τηλεφώνων (2G). Τέλος στις αρχές του 21ου αιώνα εμφανίστηκαν τα κινητά τρίτης γενιάς (3G) με αυξημένες δυνατότητες πολυμέσων όπως η βιντεοκλήση, MMS κ.ά.

2.1.2 Προσωπικοί Ψηφιακοί Βοηθοί (PDAs)

Προσωπικός ψηφιακός βοηθός (PDA) ή αλλιώς palmtop computer είναι μια μικρή φορητή πλατφόρμα που λειτουργεί σαν προσωπικός διαχειριστής πληροφοριών. Ουσιαστικά θα μπορούσε κανείς να πει ότι είναι ένας υπολογιστής τσέπης. Αν και δεν ήταν τόσο ισχυροί όσο οι επιτραπέζιοι ή οι φορητοί υπολογιστές, οι ψηφιακοί βοηθοί είναι χρήσιμοι για προγραμματισμό συναντήσεων, για αποθήκευση διευθύνσεων, αριθμών τηλεφώνων και για παιχνίδια. Οι συσκευές αυτές αντί για πληκτρολόγιο χρησιμοποιούσαν ένα ειδικό «στυλό» (γραφίδα) για μεγαλύτερη ευκολία. Χαρακτηριστικά των συσκευών αυτών ήταν η οθόνη αφής, η ενσύρματη σύνδεση στον υπολογιστή για συγχρονισμό, η αποθήκευση και ανάκτηση πληροφοριών, η ασύρματη σύνδεση στο διαδίκτυο και η ικανότητα υποδοχής καρτών μνήμης.

Ο πρώτος προσωπικός ψηφιακός βοηθός εμφανίστηκε το 1986 από την Psion και ονομαζόταν Psion organizer [16]. Τα PDAs έγιναν ευρύτερα γνωστά μετά της 7 Ιανουαρίου του 1992 όταν ο CEO της Apple, John Sculley παρουσίασε στο Consumer Electronics Show [20] που πραγματοποιήθηκε στο Λας Βέγκας της Νεβάδα, το Apple Newton. Τέσσερα χρόνια αργότερα μπήκε δυναμικά στην αγορά των PDAs η εταιρία Palm με μια σειρά που λεγόταν Pilot 1000. Την ίδια χρονιά η Nokia κυκλοφόρησε το «9000 Communicator» που ήταν ταυτόχρονα PDA και κινητό τηλέφωνο. Το «9000 Communicator» ήταν το PDA με τις περισσότερες πωλήσεις έως τότε. Με τη συσκευή αυτή στην ουσία η Nokia δημιούργησε μια νέα κατηγορία φορητής πλατφόρμας, που αργότερα ονομάσθηκαν έξυπνα κινητά τηλέφωνα (smartphones).

Σχήμα 2: Εξέλιξη PDAs

2.1.3 Έξυπνα Κινητά Τηλέφωνα (Smartphones)

Τα smartphones είναι μια κατηγορία φορητών πλατφόρμων που συνδυάζει τις δυνατότητες ενός κινητού τηλεφώνου και ενός προσωπικού ψηφιακού βοηθού (PDA). Το κυριότερο χαρακτηριστικό που διαφοροποιεί τα smartphones από ένα κινητό τηλέφωνο είναι η παρουσία ενός παραμετροποιήσιμου λειτουργικού συστήματος.

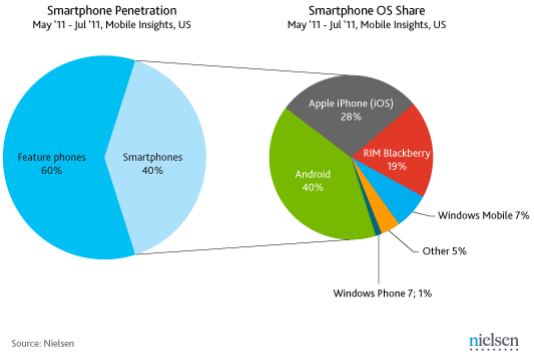
Τα smartphones προσφέρουν αναπτυγμένες πληροφοριακές δυνατότητες και συνδεσιμότητα, επιτρέποντας στον χρήση να χρησιμοποιεί πολλές εφαρμογές ταυτόχρονα, οι οποίες είναι φτιαγμένες έτσι ώστε να αξιοποιούν το υλικό (hardware) που διαθέτει κάθε συσκευή. Έκτος από την φωνητική επικοινωνία, έχουν επιπλέον μια πληθώρα δυνατοτήτων που τα κάνει να ξεχωρίζουν και να εξελίσσονται διαρκώς. Μερικές από τις δυνατότητες που έχουν είναι να στέλνουν και να δέχονται emails, να εμφανίζουν και να επεξεργάζονται κείμενα και λογιστικά φύλλα. Επίσης διαθέτουν υποδοχή καρτών μνήμης, συστήματα

Σχήμα 3: Smartphones

εντοπισμού (GPS), γυροσκόπιο και εξελιγμένους αισθητήρες κάμερας για να μπορούν να καταγράφουν εικόνες και βίντεο υψηλής ανάλυσης.

Το 1993 η IBM κυκλοφόρησε το Simone [17], ένα κινητό τηλέφωνο που είχε επιπλέον ημερολόγιο, βιβλίο διευθύνσεων, παγκόσμιο ρολόι, αριθμομηχανή, σημειωματάριο, e-mail client, δυνατότητα για αποστολή και λήψη φαξ, και παιχνίδια. Δεν διέθετε κουμπιά και χρησιμοποιούσε οθόνη αφής. Αυτή η συσκευή θεωρείτε από πολλούς ως το πρώτο έξυπνο κινητό τηλέφωνο. Όμως το ευρύ κοινό γνωρίζει τα έξυπνα κινητά το 1996 με την έλευση του «9000 Communicator», της Nokia ενώ ο όρος smartphones χρησιμοποιήθηκε για πρώτη φορά από την Ericsson το 1997. Από τότε έως σήμερα τα smartphones γίνονται όλο και πιο ελκυστικά για τους καταναλωτές καθώς αυξάνονται διαρκώς οι δυνατότητές τους ενώ παράλληλα γίνονται πιο προσιτά από οικονομικής άποψης.

Αργά αλλά σταθερά, τα έξυπνα τηλέφωνα συνεχίζουν να αυξάνουν το μερίδιό τους στη βιομηχανία κινητής τηλεφωνίας τόσο στις Ηνωμένες Πολιτείες όσο και στην Ευρώπη. Σύμφωνα με στατιστικά στοιχεία που αφορούν στη χρήση έξυπνων τηλεφώνων στις Ηνωμένες Πολιτείες, όπως αυτά παρουσιαστήκαν από την εταιρία Nilsen [18], τον Ιούλιο του 2011, το 40% του συνόλου των κινητών τηλεφώνων που χρησιμοποιούνται είναι έξυπνα τηλέφωνα. Το



Σχήμα 4: Στατιστικά στοιχεία διείσδυσης smartphones στις ΗΠΑ

υπόλοιπο 60% αφορά κινητά τηλέφωνα τα οποία δεν ανήκουν στην κατηγορία των έξυπνων τηλεφώνων π.χ. τηλέφωνα που υποστηρίζουν Java ME, αλλά το ποσοστό αυτό μειώνεται μέρα τη μέρα (Σχ. 4).

Στην Ευρώπη οι πρόσφατες εκτιμήσεις της εταιρίας ερευνών IDC, αναφέρει ότι το 47% των νέων συσκευών κατά το α' τρίμηνο του 2011, ήταν smartphones. Συνολικά, οι πωλήσεις smartphones στη Ευρώπη αυξήθηκαν κατά 76% σε σχέση με το α' τρίμηνο του 2011, φθάνοντας στα 21,2 εκατ. τεμάχια. Στο διάστημα αυτό ο κατασκευαστής με τις μεγαλύτερες πωλήσεις smartphones ήταν η Apple με μερίδιο 20,6%, ενώ ακολουθεί η Nokia με 19,6%. Στην τρίτη θέση «ισοβαθμούν» RIM (Blackberry) και HTC με 16,5% και την πεντάδα κλείνει η Samsung με 12,1%.

2.1.4 Ταμπλέτες (Tablet PCs)

Μια ταμπλέτα (Tablet PC) είναι ένας ηλεκτρονικός υπολογιστής σε μέγεθος μεγαλύτερο από ένα κινητό τηλέφωνο ή από ένα PDA όπου συνήθως το μέγεθός του φτάνει ακόμα και τις 10 ίντσες (Σχ. 5). Σχεδόν όλη του η επιφάνειά αποτελείται από μια οθόνη αφής και συνήθως δεν περιλαμβάνει φυσικό πληκτρολόγιο αλλά εικονικό για ευκολία στη χρήση. Τα χαρακτηριστικά και οι δυνατότητες αυτών των συσκευών είναι πιο κοντά στα smartphones, αν και τα περισσότερα δεν επιτρέπουν φωνητικές κλήσεις μέσω των δικτύων κινητής τηλεφωνίας. Οι ταμπλέτες λόγω του μεγέθους της οθόνης τους είναι ιδιαίτερα εύχρηστες και άνετες τόσο στην περιήγηση στο διαδίκτυο όσο και στην ανάγνωση ηλεκτρονικών βιβλίων (e-books).

Σχήμα 5: Tablet PCs

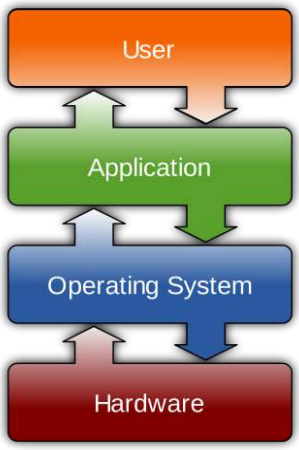
Κατά τη διάρκεια της δεκαετίας του 2000 η Microsoft χρησιμοποίησε για πρώτη φορά τον όρο Tablet PC [26] για ένα φορητό υπολογιστή που θα τον χρησιμοποιούσαν οι εργαζόμενοι στις επιχειρήσεις. Ο υπολογιστής αυτός δεν κατάφερε να γίνει ευρέως γνωστός λόγω του υψηλού κόστους του και των πολλών προβλημάτων χρηστικότητας. Τον Απρίλιο του 2010 η Apple κυκλοφόρησε ένα Tablet PC με το όνομα iPad. Η αυξημένη χρηστικότητα, το χαμηλό βάρος, η διάρκεια ζωής της μπαταρίας αλλά και η απλότητα της ταμπλέτας αυτής είχε τόσο μεγάλη απήχηση στους καταναλωτές που ταύτισαν το iPad με τον όρο Tablet PC. Ήταν τόσο μεγάλη η επιτυχία που στις 2 Μαρτίου του 2011 η Apple ανακοινώσε πως είχαν πουληθεί μέσα σε ένα χρόνο 15 εκατ. συσκευές ενώ λίγη ώρα αργότερα ανακοίνωσε τη κυκλοφορία μιας νέας

ταμπλέτας, του iPad 2. Χαρακτηριστικό είναι ότι στο Consumer Electronics Show του 2011 [20] παρουσιάστηκαν πάνω από 80 νέα μοντέλα ταμπλετών από διάφορες εταιρίες.

2.2 Λειτουργικά Συστήματα

Ένα λειτουργικό σύστημα (Operating System - OS) είναι ένα ολοκληρωμένο σύνολο ειδικών προγραμμάτων που χρησιμοποιείται για τη διαχείριση των «πόρων» του υπολογιστή και γενικά για τη δημιουργία κατάλληλου περιβάλλοντος για την εκτέλεση προγραμμάτων. Συγκεκριμένα επιτρέπει στον υπολογιστή να επιβλέπει και να συντονίζει αυτόματα, χωρίς την παρέμβαση του χρήστη, τις ίδιες τις λειτουργίες του. Οι λειτουργίες αυτές μπορεί να είναι η εκτέλεση των προγραμμάτων – εφαρμογών, η εκτέλεση βοηθητικών προγραμμάτων του ίδιου του OS, ο χειρισμός των δεδομένων εισόδου – εξόδου των προγραμμάτων κλπ. Τα λειτουργικά συστήματα για τις κινητές πλατφόρμες δεν είναι τόσο σύνθετα και είναι κατασκευασμένα να λειτουργούν με λιγότερους υπολογιστικούς πόρους.

Όταν πρωτοεμφανίστηκαν τα κινητά τηλέφωνα οι χρήστες τα επέλεγαν με βάση τα τεχνικά τους χαρακτηριστικά. Με την εμφάνιση των smartphones τα δεδομένα άλλαξαν και ένα μεγάλο ποσοστό χρηστών για την αγορά μιας νέας συσκευής άρχισε να εξετάζει και το λειτουργικό σύστημα που φέρει. Το έξυπνο κινητό τηλέφωνο για να λειτουργήσει χρειάζεται απαραίτητα κάποιο ισχυρό ενσωματωμένο λειτουργικό σύστημα, που ελέγχει τη φορητή πλατφόρμα, επιτρέποντας στους προγραμματιστές να αναπτύξουν τις εφαρμογές τους. Στην αρχή κάθε εταιρία, είτε ανέπτυσσε, είτε αγόραζε για λογαριασμό της το δικό της λειτουργικό σύστημα. Οι συσκευές όμως με τον καιρό έγιναν πολύπλοκες και η ανάπτυξη λογισμικού χρονοβόρα και πολυέξοδη με αποτέλεσμα οι εταιρίες να ενσωματώνουν στα κινητά τους έτοιμα λειτουργικά συστήματα. Σήμερα με την επιλογή ενός smartphone, γίνεται και επιλογή του λειτουργικού συστήματος στο οποίο τρέχει, όπως Android OS, Bada OS, BlackBerry OS, iOS, Symbian OS, WebOS και Windows Phone OS.



Σχήμα 6: Τυπική θέση του λειτουργικού συστήματος σε ένα υπολογιστικό σύστημα

2.2.1 Android OS

Σχήμα 7: Λογότυπο Android

Τον Οκτώβριο του 2003 στην California των ΗΠΑ γεννήθηκε ένα νέο λειτουργικό για κινητά τηλέφωνα με όνομα Android από τους Andy Rubin, Rich Miner, Nick Sears και Chris White. Ο πυρήνας αυτού του λειτουργικού συστήματος ήταν βασισμένος σε Linux. Δύο χρόνια αργότερα, τον Αύγουστο του 2005, εξαγοράστηκε από τη Google. Στις 5 Νοεμβρίου του 2007 γίνεται η πρώτη επίσημη παρουσίαση του Android OS. Για την ανάπτυξη και την προώθηση του λειτουργικού, η Google ανακοινώσε την ίδια μέρα την ίδρυση της εταιρείας Open Handset Alliance [22] υπό την ηγεσία της και σε συνεργασία με άλλα 34 μέλη συμπεριλαμβανομένων προγραμματιστών εφαρμογών, κατασκευαστών κινητών τηλεφώνων και chip. Μερικές από τις σημαντικότερες εταιρίες – μέλη ήταν οι HTC, Sony, Dell, Intel, Motorola, Qualcomm, Texas Instruments, η Samsung, LG, η T-Mobile, Nvidia, και Wind River Systems. Το πρώτο κινητό τηλέφωνο με Android OS (version 1.0) παρουσιάστηκε στις 23 Οκτωβρίου 2008 και ήταν το HTC Dream (G1), (Σχ. 8). Αξιοσημείωτη αναβάθμιση είναι αυτή της έκδοσης «cupcake» (version 1.5) στις 30 Απριλίου 2009 στην οποία υπήρχαν αλλαγές στο σύστημα διαχείρισης των μεταφορτώσεων (download manager), το framework, το Bluetooth, το λογισμικό συστήματος, το ραδιόφωνο και το σύστημα τηλεφωνίας, τα εργαλεία προγραμματισμού, το κυρίως σύστημα, διάφορες εφαρμογές καθώς και ένα πλήθος από διορθώσεις σφαλμάτων. Ακολούθησαν μια πληθώρα αναβαθμίσεων τόσο στον πυρήνα (Linux) του λογισμικού όσο και γενικά σε όλες τις λειτουργίες. Η τελευταία βασική έκδοση αποκλειστικά για κινητά τηλέφωνα παρουσιάστηκε στις 6 Δεκεμβρίου 2010 και είναι η Gingerbread (version 2.3).

Σχήμα 8: Το πρώτο κινητό με λειτουργικό Android

Στις 22 Φεβρουαρίου 2011 έγινε η πρώτη παρουσίαση της έκδοσης Honeycomb (version 3.0), η οποία απευθυνόταν σε συσκευές Tablet και η πρώτη συσκευή που χρησιμοποίησε αυτή την έκδοση ήταν το Motorola Xoom tablet (Σχ. 9). Δύο μήνες αργότερα η Google ανακοίνωσε το Ice Cream Sandwich (version 4.0) όπου ήταν το πρώτο Android OS που μπορεί να χρησιμοποιηθεί σε όλα τα είδη των φορητών συσκευών, δηλαδή τόσο σε smartphones όσο και σε tablets. Αυτή η ενοποίηση ήταν πολύ χρήσιμη για τους κατασκευαστές συσκευών και τους προγραμματιστές που πλέον θα χρησιμοποιούν ένα ενιαίο UI toolkit για να δημιουργούν εφαρμογές. Παράλληλα οι προγραμματιστές μπορούν να

δουλέψουν με πολλά νέα και βελτιωμένα APIs σε ένα ενιαίο περιβάλλον. Στις 19 Οκτωβρίου 2011 παρουσιάσθηκε το κινητό Galaxy Nexus που είναι η πρώτη συσκευή που λειτουργεί με την έκδοση Ice Cream Sandwich.

Το μεγαλύτερο μέρος του κώδικα Android είναι υπό την άδεια χρήσης Apache, μια άδεια χρήσης ελεύθερου λογισμικού που οι προγραμματιστές γράφουν κατά κύριο λόγο σε μια προσαρμοσμένη έκδοση της Java. Ο προγραμματισμός για Android OS δεν είναι μια δύσκολη διαδικασία, αρκεί ο προγραμματιστής να γνωρίζει την σύνταξη της γλώσσας προγραμματισμού Java. Φυσικά για να γίνει η ανάπτυξη μιας εφαρμογής χρειάζονται κάποια εργαλεία. Το πρώτο βήμα είναι η εγκατάσταση του Java Developer Kit (JDK), το οποίο είναι ένα περιβάλλον που μπορεί κάποιος να αναπτύξει και να εκτελέσει προγράμματα σε Java. Το δεύτερο βήμα είναι η εγκατάσταση ενός Integrated Development Environment (IDE) λογισμικού, που παρέχει τα προγραμματιστικά εργαλεία, όπως μεταφραστής, συντάκτης, διορθωτής κ.α. Το πιο διαδεδομένο λογισμικό IDE είναι το Eclipse, αν και υπάρχουν αρκετοί προγραμματιστές που χρησιμοποιούν το Netbeans. Το τρίτο βήμα είναι η εγκατάσταση εξειδικευμένων εργαλείων για τον προγραμματισμό Android και παρέχονται από το Android Software Development Kit (SDK). Το Android SDK περιέχει βιβλιοθήκες, βιβλιογραφία, παραδείγματα κώδικα, τον προσομοιωτή συσκευής και τον debugger.

Σχήμα 9: Xoom Motorola tablet

Ο «ανοιχτός» κώδικάς προσελκύει χιλιάδες προγραμματιστές να αναπτύξουν εφαρμογές, εργαλεία, παιχνίδια κλπ. και αυτό το κάνει το πιο γρήγορα αναπτυσσόμενο λειτουργικό σύστημα. Μέχρι τον Σεπτέμβριο του 2011 το Android OS, σύμφωνα με την εταιρία research2guidance [19], ξεπέρασε τις 500 χιλιάδες κατοχυρωμένες εφαρμογές στο Android Market, έχοντας ένα ποσοστό 37% στις αποσύρσεις εφαρμογών για διάφορους λογούς.

2.2.2 Bada OS

Σχήμα 10: Λογότυπο Bada OS

Το Bada OS είναι ένα σχετικά νέο λειτουργικό σύστημα για φορητές πλατφόρμες που αναπτύχθηκε από την Samsung Electronics και ο πυρήνας του είναι βασισμένος κυρίως σε Linux. Το Bada OS απευθύνεται τόσο σε smartphones όσο και σε Tablet PCs, αν και μέχρι στιγμής δεν έχει εμφανιστεί κάποιο Tablet PC με αυτό το λειτουργικό. Η λέξη Bada στα κορεατικά σημαίνει ωκεανός [24]. Την 1η Ιουνίου 2010 η Samsung παρουσίασε το πρώτο smartphone που «έτρεχε» το νέο

λειτουργικό, με όνομα Samsung Wave GT-S8500 (Σχ. 11). Εταιρίες όπως Twitter, EA, Capcom, Gameloft και Blockbuster δήλωσαν την υποστήριξή τους στο Bada OS. Τον Αύγουστο του 2010, η Samsung ανακοινώνει το δικό της Software Developer Kit (SDK) (version 1.0) για τους προγραμματιστές. Ένα χρόνο αργότερα κυκλοφορεί το SDK (version 2.0) με μεγάλο πλήθος διορθώσεων και καινοτομιών.

Σχήμα 11: Samsung Wave GT-S8500

Το πακέτο ανάπτυξης στο οποίο βασίζεται το Bada OS είναι η γλώσσα προγραμματισμού C/C++ και περιλαμβάνει ένα Eclipse - based IDE και έναν προσομοιωτή τηλεφώνου. Για να το αποκτήσει κάποιος πρέπει να συμφωνήσει με τους όρους που έχει θέσει η εταιρεία και για να διασφαλιστεί αυτό είναι απαραίτητο ο προγραμματιστής να δημιουργήσει δικό του λογαριασμό.

Στην μεσαίου-κόστους κατηγορία smartphones το Bada τα πηγαίνει καλά κατά τον 1ο χρόνο του και έχει δείξει μία εξαιρετική δυναμική στην αγορά. Η Samsung δηλώνει ότι οι χρήστες του Bada OS έχουν ξεπεράσει τα 10 εκατομμύρια downloads από το ηλεκτρονικό της κατάστημα, SamsungApps. Σύμφωνα με την Canalys, το 1^{ο} τρίμηνο του 2011, η Samsung είχε πουλήσει παγκοσμίως πάνω από 3,5 εκατομμύρια τηλέφωνα με το λειτουργικό Bada OS. Για την περαιτέρω ανάπτυξή του Bada OS η Samsung προγραμματίζει από το 2012 να το μετατρέψει σε ανοιχτού κώδικα.

2.2.3 BlackBerry OS

Σχήμα 13: Λογότυπο BlackBerry

Το BlackBerry OS είναι ένα λειτουργικό σύστημα για smartphones που αναπτύχθηκε από την καναδική εταιρία Research In Motion (RIM) το 1999 [36]. Η πρώτη συσκευή με BlackBerry OS παρουσιάστηκε το 1999 και ήταν το μοντέλο BlackBerry 850 (Σχ.13). Η πρώτη συσκευή είχε μονόχρωμη οθόνη αλλά όλα τα επόμενα μοντέλα είχαν έγχρωμες οθόνες. Επίσης σαν συσκευές ξεχώριζαν από το ενσωματωμένο «φυσικό» QWERTY πληκτρολόγιο. Τα smartphones BlackBerry έγιναν δημοφιλείς συσκευές ιδιαίτερα σε εταιρικούς χρήστες δεδομένου ότι είχαν τη δυνατότητα συγχρονισμού με τα Microsoft Exchange, Lotus Domino, Novell GroupWise e-mail και άλλα λογισμικά.

Σχήμα 12: BlackBerry 850

Η RIM πρόσφατα κυκλοφόρησε το πρώτο Tablet PC

που ονομάζεται BlackBerry Playbook. Το Playbook είναι η πρώτη συσκευή BlackBerry που «τρέχει» με το τελευταίο λειτουργικό σύστημα που ονομάζεται QNX OS και το οποίο αντικατέστησε το παλαιότερο BlackBerry OS. Τα καινούργια BlackBerry smartphones κυκλοφορούν με το νέο λειτουργικό QNX OS. Η RIM ανακοίνωσε ότι το 2012 όλες οι συσκευές της (BlackBerry smartphones και tablets) θα υποστηρίζονται από νέο, κοινό λειτουργικό σύστημα το οποίο θα λέγετε BBX. Το νέο λειτουργικό θα υποστηρίζει και cloud services.

Για την ανάπτυξη εφαρμογών για το BlackBerry OS χρησιμοποιείται η γλώσσα προγραμματισμού Java σε συνδυασμό με το πιο διαδεδομένο IDE (Integrated Development Environment) που είναι το Eclipse. Επιπρόσθετα είναι απαραίτητη η εγκατάσταση του εργαλείου BlackBerry Java SDK. Με την έλευση του BBX, θα υποστηρίζεται το περιβάλλον ανάπτυξης εφαρμογών τόσο για προγραμματιστές σε HTML 5 όσο και native. Με ένα πρόσθετο εργαλείο, το BlackBerry Web Works, οι προγραμματιστές που ανέπτυσσαν εφαρμογές, τόσο για τα BlackBerry 6 & 7, όσο και για τα Playbook, μπορούν ήδη να προσαρμόζουν τις εφαρμογές τους για όλα τα λειτουργικά.

Όπως και στα περισσότερα λειτουργικά, έτσι και στο BlackBerry OS, οι ενημερώσεις για το λειτουργικό σύστημα μπορούν να γίνουν και over-the-air. Πάνω από 15 χιλιάδες εφαρμογές είναι διαθέσιμες στο BlackBerry App World.

Σχήμα 14: BlackBerry Smartphone και tablet

2.2.4 iOS

Σχήμα 15: Apple iOS

Το iOS ή iPhone OS είναι το λειτουργικό σύστημα που χρησιμοποιείται από την Apple σε όλες τις εκδόσεις του iPod Touch, iPhone και iPad. Παρουσιάστηκε στο Macworld Conference & Expo για πρώτη φορά μαζί με τη συσκευή iPhone (Σχ. 16), στις 9 Ιανουαρίου 2007 [37]. Είναι ένα λειτουργικό σύστημα σχεδιασμένο από τη Apple αποκλειστικά για τις δικές της συσκευές με κλειστό κώδικα. Προέρχεται από το Mac OS X, συνεπώς βασίζεται σε Unix.

Στην πρώτη του έκδοση το iOS δεν μπορούσε να συναγωνιστεί σε δυνατότητες τα άλλα λειτουργικά συστήματα των υπόλοιπων smartphones που υπήρχαν στην αγορά σχεδόν μια δεκαετία. Παρόλα αυτά σημείωσε εμπορική επιτυχία πουλώντας σε 74 μέρες ένα εκατομμύριο συσκευές. Οι λόγοι που οδήγησαν σε αυτή την επιτυχία δεν ήταν λίγοι. Ένας

Σχήμα 17: iPhone (πρώτη γενιά)

Σχήμα 17: Εφαρμογή Siri για το iPhone 4S (iOS 5)

από τους βασικούς και ίσως ο σημαντικότερος ήταν η ιδιαίτερα απλή χρήση του iOS, λόγω της εύκολης αλληλεπίδρασης του χρήστη με την οθόνη αφής της συσκευής, χωρίς την χρήση γραφίδας και με την παρουσία μόλις ενός Hardware κουμπιού. Ο χειρισμός του smartphone γίνεται ευχάριστος και γρήγορος καθώς ο χρήστης αλληλεπιδρά με φυσικότητα με τα αντικείμενα που προβάλλονται στην οθόνη. Για παράδειγμα ο χρήστης μέσω της οθόνης αφής πολλαπλών σημείων μπορεί να χρησιμοποιεί διάφορες κινήσεις των δακτύλων του και να παίρνει άμεσα τα αποτελέσματα στην οθόνη, έτσι ζουμάρει σε μια φωτογραφία με το άνοιγμα των δυο δακτύλων του ή αλλάζει φωτογραφίες με μια απλή κίνηση του δακτύλου του από δεξιά προς τα αριστερά. Επίσης το γραφικό περιβάλλον του iOS ήταν ιδιαίτερα όμορφο και ελκυστικό για τον χρήστη. Ένα επιπλέον πλεονέκτημα ήταν η ενσωμάτωση γνωστών εφαρμογών από παλαιότερα επιτυχημένα προϊόντα της Apple όπως το iPod, δίνοντάς του μια επιπλέον δυναμική. Τέλος ένας εξίσου σημαντικός λόγος της επιτυχίας του ήταν το ηλεκτρονικό κατάστημα App Store μέσω του οποίου διέθετε αποκλειστικά εφαρμογές για τους χρήστες του iOS. Ακολούθησαν με ένα χρόνο διαφορά το iPhone 3G και το iPhone 3GS, με το iOS να φτάνει στην έκδοση 4. με εξίσου μεγάλη επιτυχία. Με την κυκλοφορία του iPhone 4 τον Ιούνιο του 2010, η Apple κατάφερα να φτάσει και να ξεπεράσει σε δυνατότητες του λειτουργικού συστήματος όλες τις ανταγωνίστριες εταιρίες. Η τελευταία έκδοση iOS 5 κυκλοφόρησε παράλληλα με το iPhone 4S τον Οκτώβριο του 2011, έχοντας επιπλέον δυνατότητα το iCloud και το Siri (Σχ. 17).

Η Apple συνδύασε απευθείας το λειτουργικό iOS με την πρώτη της ταμπλέτα που ήταν το iPad και παρουσιάσθηκε τον Απρίλιο του 2010. Το iPad με λειτουργικό iOS 4 είχε τόσο μεγάλη απήχηση στο κοινό που πούλησε 14,8 εκατομμύρια συσκευές μέσα στο 2010, δηλαδή

το 75% από τα όλα τα Tablet PC που είχαν πουληθεί έως τότε. Το Μάρτιο του 2011 κυκλοφόρησε το iPad 2 (Σχ. 18) με ακόμα μεγαλύτερη επιτυχία, και το οποίο τον Οκτώβριο αναβαθμίστηκε σε iOS 5.

Η Apple έχει συγκεντρώσει σε ένα πακέτο όλα τα εργαλεία που χρειάζεται ένας προγραμματιστής για να δημιουργήσει εφαρμογές για Mac, iPhone και iPad, το Xcode Developer Tools. Το Xcode είναι στενά συνδεδεμένο με το Cocoa και Cocoa Touch Framework, δημιουργώντας ένα παραγωγικό και εύκολο στη χρήση περιβάλλον ανάπτυξης το οποίο είναι αρκετά ισχυρό τόσο για την παραγωγή εφαρμογών για Mac Os X όσο και για το iOS. Το σύνολο των εργαλείων Xcode για iOS αποτελείται από έναν Builder, τον LLVM compiler, τον iOS simulator και το Instruments (εργαλείο παρατήρησης και δοκιμών).

Σχήμα 18: iPad 2

Το μοναδικό λειτουργικό στο οποίο «τρέχει» το Xcode είναι το Mac Os X. Η Apple επειδή δεν χορηγεί άδεια για εγκατάσταση του Mac Os X σε τρίτους κατασκευαστές, αναγκάζει στην ουσία τους προγραμματιστές σε αγορά υπολογιστή της εταιρίας της.

Σύμφωνα με την Apple το iOS ξεπερνά τις 600 χιλιάδες εφαρμογές στο App Store έχοντας ένα ποσοστό 24% στις αποσύρσεις για διάφορους λόγους.

2.2.5 Symbian OS

Σχήμα 19: Λογότυπο Symbian OS

Το Symbian OS δημιουργήθηκε με τη γλώσσα προγραμματισμού C++ από τη Symbian Ltd και είναι ένα λειτουργικό σύστημα για φορητές συσκευές. Αποτελεί μια εξέλιξη του λειτουργικού συστήματος EPOC της Psion και «τρέχει» αποκλειστικά σε ARM processors [38]. Οι πρώτες εκδόσεις του Symbian OS ήταν βασισμένες στον πυρήνα EKA1, ενώ σήμερα έχει αντικατασταθεί από τον σταθερότερο και σαφώς ισχυρότερο EKA2. Στο Symbian OS έχουν βασιστεί αρκετές πλατφόρμες όπως οι Nokia (Series 60, Series 80 και Series 90), Sony Ericsson (UIQ), Benq-Siemens, Samsung, Motorola και NTT DoCoMo. Τον Δεκέμβριο του 2008 η Nokia αποφάσισε να αγοράσει την Symbian Ltd. Το λειτουργικό της Symbian - στις διάφορες μορφές του – υπάρχει σε χιλιάδες συσκευές κινητών τηλεφώνων από διάφορους κατασκευαστές. Σύμφωνα με ορισμένες εκτιμήσεις οι συσκευές που έχουν πουληθεί με Symbian OS μέχρι το τέλος του 2010 αγγίζουν τα 400 εκατομμύρια. Στις 11 Φεβρουαρίου 2011 η Nokia αποφάσισε να εγκαταλείψει το Symbian OS και στις νέες smartphone συσκευές

της να έχει το λειτουργικό Windows Phone 7 σε συνεργασία με την Microsoft. Όμως δεν εγκατέλειψε εντελώς το Symbian OS, καθώς η εταιρεία θα συνεχίσει να το χρησιμοποιήσετε για τηλέφωνα που δεν είναι smartphones. Στις 5 Απριλίου 2011 πάρθηκε η απόφαση να μετατραπεί το μέχρι πρότινος Open-Source Symbian σε Closed-Source λειτουργικό. Πλέον η ανάπτυξη του λειτουργικού περνά αποκλειστικά στα χέρια της φινλανδικής εταιρίας, με τις εταιρίες και τους προγραμματιστές να μην μπορούν να συνεισφέρουν στην βελτίωση του πηγαίου κώδικα του συστήματος. Στις 22 Ιουνίου 2011 η φινλανδική εταιρεία, ανέθεσε πλέον εξολοκλήρου την ανάπτυξη του Symbian στην Accenture.

Για την ανάπτυξη εφαρμογών παλαιότερα ήταν απαραίτητο το Carbide IDE. Στη συνέχεια και αναλόγως την έκδοση Symbian (Symbian 60 1st, 2nd ή 3rd Edition) που γινόταν η ανάπτυξη του λογισμικού, έπρεπε να εγκατασταθεί το ανάλογο SDK. Η βασική γλώσσα προγραμματισμού είναι η C++. Εκτός από τη C++ οι προγραμματιστές με τη βοήθεια ενός add-on του Visual Studio.NET, του AppForge Crossfire μπορούσαν να αναπτύξουν την εφαρμογή τους σε Visual BASIC.NET και C#. Τον Ιούνιο του 2010 η Nokia προσφέρει στους προγραμματιστές για Symbian OS ένα ολοκληρωμένο περιβάλλον ανάπτυξης, το Qt SDK 1.0 σε συνδυασμό με το Qt Creator. Το Qt SDK 1.0 είναι εξοπλισμένο με έναν compiler που επιτρέπει την δημιουργία πακέτων αρχείων .sis (Symbian), έναν ανιχνευτή σφαλμάτων απευθείας στη συσκευή και έναν νέο και γρήγορο προσομοιωτή.

Σχήμα 20: Nokia – N8

Τον Απρίλιο του 2011 η Nokia ανακοινώσε ότι από το ηλεκτρονικό της κατάστημα Ovi Store γίνονται 5 εκατομμύρια downloads ημερησίως. Το μέλλον του Symbian smartphone στον κόσμο έχει σχεδόν τελειώσει, με την εγκατάλειψη της Nokia και τους περισσότερους χρήστες να στρέφονται σε Android, iOS και Windows Phone 7 σε παγκόσμιο επίπεδο.

2.2.6 WebOS

Σχήμα 22: Λογότυπο Web OS

Το WebOS είναι λειτουργικό σύστημα βασισμένο σε πυρήνα Linux. Το WebOS αναπτύχθηκε αρχικά από την εταιρία Palm ως διάδοχος του λειτουργικού συστήματος Palm OS που ήταν ιδιαίτερα διαδεδομένο για τους υπολογιστές παλάμης της σειράς Palm. Η πρώτη έκδοση του Palm OS το 1996 ήταν η Palm OS 1.0 που χρησιμοποιήθηκε στα μοντέλα Palm 1000

Σχήμα 21: Κεντρική οθόνη Web OS

και Palm 5000 και περιείχε εφαρμογές για διαχείριση επαφών και υποχρεώσεων, ημερολόγιο, σημειωματάριο, αριθμομηχανή και μια εφαρμογή ασφαλείας. Το λειτουργικό σύστημα τελικά, αποκτήθηκε από την Hewlett-Packard (HP) η οποία χρησιμοποιεί το WebOS σε μια σειρά από συσκευές, συμπεριλαμβανομένων πολλών smartphones και ενός Table PC. Η HP με την κυκλοφορία του webOS 3.x. έχει προωθήσει το λειτουργικό σε επιχειρήσεις κινητής τηλεφωνίας στην αγορά, εστιάζοντας στη βελτίωση των χαρακτηριστικών ασφαλείας και διαχείρισης.

Τον Ιούλιο του 2011 κυκλοφόρησε η HP το πρώτο Tablet PC με το λειτουργικό Web OS, που ονομαζόταν HP TouchPad [25] (Σχ. 23). Η έκδοση του λειτουργού ήταν η WebOS 3.0 το οποίο πρόσφερε video chat, ασύρματη εκτύπωση, email, eBooks, Web browsing, επεξεργασία εγγράφων και πρόσβαση στο "HP Catalog" για απόκτηση επιπλέον εφαρμογών.

Σχήμα 23: HP TouchPad

Για την ανάπτυξη του WebOS χρησιμοποιούνται τεχνολογίες προγραμματισμού WEB (JavaScript, CSS, jQuery, HTML 5, SQL), με ένα επιπλέον API, το Mojo. Το web development έχει μια πληθώρα εργαλείων οικείων στους περισσότερους προγραμματιστές, γεγονός που θέλει να εκμεταλλευτεί η HP. Επιπλέον, είναι απαραίτητη η εγκατάσταση του WebOS SDK έτσι ώστε να υπάρχει δυνατότητα ανάπτυξης τόσο σε JavaScript όσο και σε C/C++. Τα εργαλεία που περιλαμβάνει είναι GCC compiler, SDL και OpenGL βιβλιοθήκες κώδικα, πολλά δείγματα κώδικα και εντοπισμό σφαλμάτων.

Οι διαθέσιμες εφαρμογές για το λειτουργικό σύστημα WebOS δεν ξεπερνούν τις δύο χιλιάδες. Στις 18 Αυγούστου 2011 η HP ανακοίνωσε ότι σταματά την παραγωγή των WebOS συσκευών, συγκεκριμένα το TouchPad και τα WebOS smartphones, καθιστώντας το μέλλον του λειτουργικού συστήματος αβέβαιο.

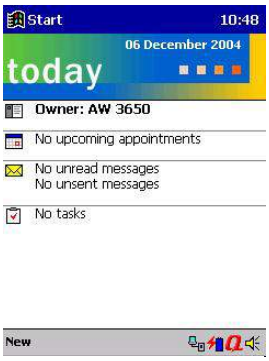
2.2.7 Windows Phone OS

Σχήμα 24: Λογότυπο Windows Phone

Το Windows Phone OS είναι το λειτουργικό σύστημα της Microsoft που χρησιμοποιείται σε φορητές συσκευές. Το Windows Phone OS είναι η εξέλιξη του λειτουργικού Windows Mobile OS. Στις 19 Απριλίου του 2000 η Microsoft παρουσίασε το Pocket PC 2000 (Σχ. 25) το οποίο βασιζόταν στο Windows CE 3.0 [26]. Ήταν η αρχή ενός λειτουργικού συστήματος που αργότερα ονομάστηκε Windows Mobile. Τον Ιούνιο του 2003 χρησιμοποιήθηκε από την Microsoft για πρώτη φορά η ονομασία «Windows Mobile» με την παρουσίαση του λειτουργικού συστήματος Windows Mobile 2003. Οι συσκευές με το λειτουργικό Windows Mobile έκαναν σχεδόν ότι και τα

σύγχρονα λειτουργικά συστήματα των smartphones, έτσι υποστήριζαν multitasking, εγκατάσταση εφαρμογών, πλήρη πρόσβαση στο σύστημα αρχείων, πρόσβαση στο μητρώο, προσωπικές πληροφορίες συγχρονισμού και μια πλήρη σουίτα γραφείου. Η εγκατάσταση μιας «βαριάς» έκδοσης των Windows Mobile σε μια συσκευή ήταν τόσο απλή όσο η σύνδεση της με τον υπολογιστή μέσω USB και η λειτουργία ενός πρότυπου οδηγού εγκατάστασης λογισμικού στον υπολογιστή. Οι κύριοι λόγοι που το λειτουργικό Windows Mobile δεν έγινε ευρέως γνωστό ήταν ότι δεν υπήρχε πλατφόρμα ανάπτυξης εφαρμογών, όπου θα ενθάρρυνε του προγραμματιστές να αναπτύξουν τις δικές τους εφαρμογές. Επίσης δεν υπήρχε εύκολη διεπαφή με τον χρήστη λόγω της απαιτούμενης χρήσης γραφίδας ή περιττών κουμπιών. Κάποιες σημαντικές αναβαθμίσεις ακολούθησαν το 2005 και το 2007, οι εκδόσεις Windows Mobile 5 (Σχ. 26) και Windows Mobile 6 αντίστοιχα, όμως παρόλο τις προσδοκίες της εταιρίας, δεν άλλαξαν πολύ τα πράγματα. Τον Οκτώβριο του 2009 , η Microsoft έστω και καθυστερημένα,

Σχήμα 25: Αρχική οθόνη Windows Mobile 5



Σχήμα 27: Pocket PC 2000

ανακοίνωσε το Windows Marketplace, ένα ηλεκτρονικό κατάστημα όπου μπορούσε ο χρήστης να «κατεβάζει» αποκλειστικά εφαρμογές για την συσκευή του. Έναν χρόνο αργότερα, τον Οκτώβριο του 2010 η Microsoft ανακοίνωσε ένα νέο λειτουργικό σύστημα για φορητές συσκευές, στοχευμένο στο καταναλωτικό κοινό αυτή την φορά και όχι στις επιχειρήσεις, το όνομα του Windows Phone 7 (Σχ. 27). Σύμφωνα με την Microsoft το νέο λειτουργικό προσφέρει ένα πιο φιλικό περιβάλλον για τον χρήστη, με μία νέα σχεδιαστική γλώσσα, με το όνομα Metro. Επιπλέον, έβαλε κανόνες χαρακτηριστικών στο hardware των συσκευών της, όπως capacitive οθόνη τουλάχιστον 4 ιντσών, camera τουλάχιστον 5 MP, Sensors (GPS, Accelerometer, light), Codec Accelerator, τουλάχιστον 256 MB RAM, τουλάχιστον GPU DirectX 9, τουλάχιστον CPU ARM v7 και 3 hardware buttons (Start,Search, Back).

Σχήμα 26: Windows Phone 7

Για την ανάπτυξη μιας εφαρμογής για Mobile Phone OS, είναι απαραίτητη η χρήση του ολοκληρωμένου εργαλείου ανάπτυξης της Microsoft, με όνομαVisual Studio .NET. Οι περισσότεροι προγραμματιστές προτιμούν την συγγραφή κώδικα με την γλώσσα προγραμματισμού C#, αλλά δεν είναι και λίγοι εκείνοι που χρησιμοποιούν την Visual basic

.NET. Επίσης, η Microsoft προσφέρει ελεύθερα το Microsoft Visual Studio 2010 express for Windows Phone, το οποίο περιλαμβάνει το XNA Game Studio 4.0 και έναν προσομοιωτή τηλεφώνου.

Σύμφωνα με την Microsoft, σήμερα, το MarketPlace έχει ήδη ξεπεράσει το φράγμα των 25 χιλιάδων εφαρμογών, ενώ υποστηρίζει ότι το 51% των εφαρμογών που διαθέτει είναι δωρεάν. Επίσης με μια σχετικά πρόσφατη έρευνα της εταιρίας IDC, φαίνεται ότι μέχρι το 2015 θα φτάσει στο 20,9% το ποσοστό στην αγορά, γεγονός που θα την κάνει να βρεθεί στη δεύτερη θέση [27].

2.3 Διαλειτουργικότητα

Με την ανάλυση που έγινε παραπάνω γίνεται εύκολα αντιληπτή η ποικιλομορφία των λειτουργικών συστημάτων. Υπάρχουν περισσότερες από επτά μεγάλες πλατφόρμες όπως Android OS, Bada OS, BlackBerry OS, iOS, Symbian OS, WebOS και Windows Phone OS, με διάφορες γλώσσες προγραμματισμού και User Interfaces (UI). Είναι βέβαιο, ότι κατά την παραγωγή ενός λογισμικού, μας ενδιαφέρει τόσο η διείσδυσή του στη αγορά, όσο και το κόστος ανάπτυξής του. Στην περίπτωση που ένας προγραμματιστής θέλει να απευθυνθεί στην πλειοψηφία των χρηστών, θα πρέπει να αναπτύξει το λογισμικό του, για κάθε μια πλατφόρμα ξεχωριστά, αυξάνοντας το κόστος και τη διάρκεια ανάπτυξής του. Έτσι, πολλοί προγραμματιστές άρχισαν να στρέφουν το ενδιαφέρον τους, στην ανάπτυξη λογισμικού με διαλειτουργικότητα, το λεγόμενο Cross Platform (ή Multi Platform) λογισμικό.

Το σημαντικότερο θέμα που ανακύπτει κατά την ανάπτυξη μιας διαλειτουργικής εφαρμογής για κινητά είναι ο χειρισμός των έντονων διαφοροποιήσεων που υπάρχουν μεταξύ των εγγενών χαρακτηριστικών που διαθέτει η κάθε κινητή πλατφόρμα, για την αλληλεπίδραση του χρήστη με αυτή, τα προαναφερθέντα User Interfaces. Οι βασικότερες στρατηγικές για την αντιμετώπιση αυτού του προβλήματος είναι δύο. Η πρώτη αφορά στην ανάπτυξη μιας εφαρμογής με χρήση ξεχωριστών User Interface χαρακτηριστικών για κάθε διαφορετική πλατφόρμα με την επιβάρυνση του επιπλέον προγραμματιστικού κόστους που συνεπάγεται. Στο κόστος αυτό θα πρέπει κανείς να συνυπολογίσει το γεγονός ότι παρόμοια User Interface controls ενδέχεται να έχουν διαφορετική συμπεριφορά από πλατφόρμα σε πλατφόρμα. Η δεύτερη αφορά στην εξομοίωση των User Interface controls της κάθε πλατφόρμας μέσω της χρήσης πρωτογενών γραφικών, η οποία όμως ενέχει το επιπλέον κόστος της ανάπτυξης ενός προσωπικού συστήματος User Interface από την πλευρά του προγραμματιστή. Στα αρνητικά της χρήσης εξομοιούμενων User Interface controls θα πρέπει

να συγκαταλεγούν τα ζητήματα αξιοπιστίας τους, καθώς η λειτουργικότητα που προσφέρουν είναι εν γένει διαφορετική από αυτήν των εγγενών χαρακτηριστικών της κάθε πλατφόρμας.

Η εναλλακτική προσέγγιση που βασίζεται στην εξομοίωση των εγγενών User Interface controls καλύπτει ένα ευρύ φάσμα διαφορετικών μεθοδολογιών. Η προφανέστερη από αυτές σχετίζεται με την ανάπτυξη Mobile Web Applications καθώς η μόνη προϋπόθεση που θέτει για τις διάφορες κινητές πλατφόρμες είναι η υποστήριξη ενός web browser, καθώς η προγραμματιστική βάση για την ανάπτυξη αυτών των εφαρμογών είναι η HTML, η JavaScript και τα CSS. Βασικό πλεονέκτημα αυτής της μεθοδολογίας είναι το ότι δεν υπάρχει ανάγκη για κάποιο συγκεκριμένο SDK για την ανάπτυξη της εφαρμογής. Αυτό σημαίνει πως με μόλις μια εφαρμογή μπορεί να καλύψει κανείς την πλειοψηφία των συσκευών και των browser που υπάρχουν. Οι WEB εφαρμογές ωστόσο, είναι κατά κανόνα πιο αργές καθώς κάθε ενέργεια του χρήστη απαιτεί την αποστολή κάποιου αιτήματος σε έναν εξυπηρετητή και βέβαια την αναμονή για την απόκρισή του. Επιπλέον, οι εφαρμογές αυτές δεν έχουν την αντίστοιχη λειτουργικότητα των εφαρμογών που βασίζονται στα εγγενή UI, καθώς ο browser της εκάστοτε συσκευής δεν έχει πρόσβαση στα εξελιγμένα χαρακτηριστικά της (πχ GPS, πυξίδα, γυροσκόπιο κ.α.).

Μια επιπλέον δυνατότητα είναι η ανάπτυξη υβριδικών εφαρμογών με χρήση εξειδικευμένων SDKs (PhoneGup, Titanium, Rhodes) τα οποία παρέχουν διεπαφές σε JavaScript για την πρόσβαση στα εγγενή χαρακτηριστικά της κάθε κινητής πλατφόρμας. Ο τρόπος ανάπτυξης μιας υβριδικής εφαρμογής βασίζεται και πάλι στην συγγραφή του κώδικα σε HTML, JavaScript και CSS, ο οποίος όμως στην συνέχεια μεταγλωττίζεται σε εξειδικευμένες standalone εφαρμογές για κάθε πλατφόρμα ξεχωριστά. Παρά τα πλεονεκτήματά της, η συγκεκριμένη προσέγγιση έχει ένα θεμελιώδες μειονέκτημα, καθώς είναι άμεσα εξαρτημένη από το SDK περιβάλλον που έχει επιλέξει ο κάθε προγραμματιστής. Στο ίδιο πνεύμα οι προγραμματιστές είναι αναγκασμένοι να χρησιμοποιούν τα ειδικά χαρακτηριστικά του κάθε περιβάλλοντος, τα οποία σε πολλές περιπτώσεις αδυνατούν να καλύψουν την τεχνολογική εξέλιξη των κινητών συσκευών, όπως για παράδειγμα οθόνες ειδικής υψηλής ανάλυσης (π.χ. iPhone Retina Display).

Συμπερασματικά, η ανάπτυξη διαλειτουργικών εφαρμογών είναι αδιέξοδη καθώς είναι εξαιρετικά δύσκολο να εξομοιώσει κανείς τα εγγενή χαρακτηριστικά της κάθε κινητής πλατφόρμας. Για τον λόγο αυτό είναι προτιμότερο να επενδύσει κανείς, για την ανάπτυξη ποιοτικού λογισμικού, στην ορθή ανάπτυξη εγγενών εφαρμογών εκμεταλλευόμενος στο έπακρο το εύρος των δυνατοτήτων που προσφέρουν.

3 Βασικές Έννοιες Σημάτων

Το παρόν κεφάλαιο πραγματεύεται βασικά στοιχεία της ψηφιακής επεξεργασίας σήματος τα οποία είναι απαραίτητα για την υλοποίηση της εφαρμογής [5],[9].

3.1 Η Έννοια του Σήματος

Ο όρος σήμα (signal) υποδηλώνει μια συνάρτηση που απεικονίζει μια ή περισσότερες ανεξάρτητες μεταβλητές ενός συνόλου I (πεδίο ορισμού) σε μια ή περισσότερες εξαρτημένες μεταβλητές ενός συνόλου U (πεδίο τιμών).

Ένα μονοδιάστατο σήμα εκφράζει τη μεταβολή μιας μεταβλητής, έστω x, σε σχέση με μια άλλη μεταβλητή, έστω t. Δηλαδή, ορίζεται ως μια συνάρτηση x(•) τέτοια ώστε t—>x(t). Συνήθως, η ανεξάρτητη μεταβλητή t αναφέρεται στο χρόνο και λαμβάνει τιμές από το πεδίο ορισμού I του σήματος, ενώ η τιμή του σήματος τη χρονική στιγμή t συμβολίζεται με x(t). Οι τιμές του σήματος περιορίζονται σε κάποιο πεδίο τιμών του U.

Αν η ανεξάρτητη μεταβλητή ενός μονοδιάστατου σήματος μεταβάλλεται σε ένα συνεχές διάστημα, δηλαδή το πεδίο ορισμού I είναι ένα διάστημα (πιθανώς απείρου μήκους) της ευθείας $\mathbb{R}$ των πραγματικών αριθμών, τότε το σήμα ονομάζεται σήμα συνεχούς χρόνου. Στις πιο συνηθισμένες περιπτώσεις σημάτων συνεχούς χρόνου, $I = \mathbb{R}$ ή $I = \mathbb{R}^+$, όπου $\mathbb{R}^+$ είναι το σύνολο των μη αρνητικών πραγματικών αριθμών.

3.2 Συνάρτηση Δέλτα Συνεχούς Χρόνου

Υπάρχει μια ειδική "συνάρτηση" η οποία είναι χρήσιμη στην ανάλυση σημάτων. Η συνάρτηση αυτή δεν μπορεί να οριστεί με μαθηματική αυστηρότητα ως μία συνήθης συνάρτηση, αλλά μόνο ως μια γενικευμένη συνάρτηση. Εδώ, θα εκφραστεί ως το "όριο" μιας ακολουθίας (συνήθων) συναρτήσεων σε τρόπο ώστε να επιτευχθεί απλότητα και χρησιμότητα, χωρίς τις έννοιες της αυστηρής θεωρίας των γενικευμένων συναρτήσεων.

Πρώτα χρειάζεται να οριστεί η συνάρτηση rect (μοναδιαίος τετραγωνικός παλμός, unit square pulse):

$$rect(t) = \begin{cases} 1, & |t| < 1/2 \\ 0, & \text{οπουδήποτε αλλού.} \end{cases} \quad (1)$$

Με τη βοήθεια της συνάρτησης αυτής ορίζεται μια ακολουθία συναρτήσεων, η οποία δίνεται από τον τύπο $\delta(t) = erect(\varepsilon t)$. Οι τετραγωνικοί παλμοί οι οποίοι αποτελούν τις γραφικές παραστάσεις των συναρτήσεων της ακολουθίας αυτής έχουν εμβαδόν ίσο με τη

μονάδα. Το ύψος τους αυξάνεται καθώς $\varepsilon \to \infty$, ενώ αντίστοιχα το πλάτος τους μειώνεται. Το "όριο" αυτής της ακολουθίας συναρτήσεων είναι ένα ορθογώνιο με άπειρο ύψος και απειροστό πλάτος έτσι ώστε κατά κάποιο τρόπο το εμβαδόν του να παραμένει ίσο με τη μονάδα. Το "όριο" αυτό συμβολίζεται με δ(t). Η εξήγηση αυτή οδηγεί στον "ορισμό" της συνάρτησης δέλτα (delta function) ή συνάρτησης Dirac (Dirac function) με "τύπο":

$$\delta(t) = \lim_{\varepsilon \to \infty} \delta_\varepsilon(t) \quad (2),$$

η οποία ικανοποιεί την σχέση: $\int_{-\infty}^{\infty} \delta_\varepsilon(t)dt = 1$ (3).

Η συνάρτηση δέλτα, δ(•), ονομάζεται και κρουστική συνάρτηση συνεχούς χρόνου (continuous time impulse function) και ικανοποιεί την ακόλουθη ιδιότητα "δειγματοληψίας":

$$\int_{-\infty}^{\infty} x(t)\delta(t - t')dt = x(t') \quad (4),$$

όπου, το δ(t-t΄) αποτελεί μία κρουστική συνάρτηση μετατοπισμένη στη θέση t=t'. Από αυτήν την ιδιότητα της συνάρτησης δέλτα, μπορεί να θεωρηθεί ότι το δ(t — t') "δειγματίζει" ένα σήμα x(•) στο σημείο t = t'.

3.3 Ο Μετασχηματισμός Fourier

Ο μετασχηματισμός Fourier είναι ένα ισχυρό εργαλείο για ανάλυση σημάτων και συστημάτων. Επιτρέπει την ποσοτική περιγραφή της δειγματοληψίας σημάτων συνεχούς χρόνου, της λειτουργίας τεχνολογικών προϊόντων που περιλαμβάνουν συνδεσμολογίες συνελικτικών φίλτρων και του θορύβου. Θα αναπτύξουμε τις βασικές ιδιότητες του μετασχηματισμού Fourier για μονοδιάστατα σήματα, τόσο συνεχούς όσο και διακριτού χρόνου.

Ο μετασχηματισμός Fourier είναι ένας ολοκληρωτικός μετασχηματισμός που, στη γενική περίπτωση, απεικονίζει μια μιγαδική συνάρτηση n μεταβλητών σε μια άλλη μιγαδική συνάρτηση n μεταβλητών. Ο μετασχηματισμός Fourier ενός μονοδιάστατου συνεχούς σήματος συνεχούς χρόνου f(t) ορίζεται ως ένα νέο συνεχές σήμα συνεχούς χρόνου F(s) με βάση τη σχέση:

$$F\{f(t)\} = F(s) = \int_{-\infty}^{\infty} f(t)e^{i2\pi st}dt \quad (5),$$

όπου, $i = \sqrt{-1}$.

Ο αντίστροφος μετασχηματισμός Fourier του σήματος F(s) ορίζεται ως:

$$F^{-1}\{F(s)\} = \int_{-\infty}^{\infty} F(s)e^{-i2\pi st}ds \quad (6).$$

Η μόνη διαφορά ανάμεσα στον ευθύ και τον αντίστροφο μετασχηματισμό Fourier βρίσκεται στο πρόσημο του εκθέτη στο μιγαδικό εκθετικό.

Η σχέση ανάμεσα στον ευθύ και τον αντίστροφο μετασχηματισμό Fourier δίνεται από τον τύπο αντιστροφής:

$$f(t) = F^{-1}\{F\{f(t)\}\}\ (7), \qquad δηλαδη\ f(t) = \left\{F^{-1}\{F(s)\}\right\}\ (8).$$

Τα σήματα f(t) και F(s) ονομάζονται ζεύγη μετασχηματισμού Fourier και η μεταξύ τους αντιστοιχία είναι ένα προς ένα.

3.3.1 Ο Διακριτός Μετασχηματισμός Fourier

Θεωρούμε το σήμα διακριτού χρόνου f_n, n=0,1,2, …, N-1 με διάρκεια N δειγμάτων. Ορίζουμε τον ευθύ διακριτό μετασχηματισμό Fourier του σήματος {f_n, n=0,1,2, …, N-1} ως:

$$F_m = \sum_{n=0}^{N-1} f_n e^{i2\pi\frac{mn}{N}}\ , m = 0,1,2,\dots,N-1\ (9)$$

και τον αντίστροφο διακριτό μετασχηματισμό Fourier του σήματος {F_m, m=0,1,2, …, N-1} ως:

$$g_n = \frac{1}{N}\sum_{m=0}^{N-1} F_m e^{-i2\pi\frac{mn}{N}}\ , n = 0,1,2,\dots,N-1\ (10),$$

όπου, ισχύει $g_n=f_n$, n=0,1,2, …, N-1.

Ο διακριτός μετασχηματισμός Fourier μπορεί να θεωρηθεί ως μια προσέγγιση του συνεχούς μετασχηματισμού Fourier. Πραγματικά,

$$F(s) = \int_{-\infty}^{\infty} f(t)e^{i2\pi st}dt\ (11).$$

Επομένως,

$$F\left(\frac{m}{T}\right) = \int_{-\infty}^{\infty} f(t)e^{i2\pi\frac{mt}{T}}dt\ (12),$$

όπου T είναι το μήκος του διαστήματος του πεδίου ορισμού όπου το σήμα είναι σημαντικά διαφορετικό από το μηδέν. Υποθέτουμε ότι με αρκετά μεγάλο αριθμό δειγμάτων N ισχύει T=NΔt καθώς και η προσέγγιση:

$$\int_{-\infty}^{\infty} f(t)e^{i2\pi\frac{mt}{T}}dt \approx \Delta t \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(n\Delta t)e^{i2\pi\frac{mn\Delta t}{T}} = \Delta t \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f(n\Delta t)e^{i2\pi\frac{mn}{N}}\ (13).$$

Αφού το σήμα έχει μηδενική τιμή εκτός ενός διαστήματος μήκους T, μπορούμε να υποθέσουμε χωρίς βλάβη της γενικότητας ότι το σήμα επαναλαμβάνεται περιοδικά με

περίοδο T και, αντίστοιχα, τα δείγματα του επαναλαμβάνονται περιοδικά με περίοδο N. Επομένως, έχουμε:

$$\int_{-\infty}^{\infty} f(t)e^{i2\pi\frac{mt}{T}}dt \approx \Delta t \sum_{n=0}^{N-1} f(n\Delta t)e^{i2\pi\frac{mn}{N}} \quad (14).$$

Το οποίο σημαίνει ότι $F(m) \equiv F\left(\frac{m}{T}\right)$

$$\approx \Delta t \Delta MF\left\{f_n = f(n\Delta t), \qquad n = -\frac{N}{2}+1, -\frac{N}{2}+2, \dots, \frac{N}{2}\right\} \quad (15).$$

Για να ισχύει η προσέγγιση, θα πρέπει το Δt να είναι αρκετά μικρό και το N αρκετά μεγάλο, ώστε να καλύπτεται το σήμα συνεχούς χρόνου με αρκετά καλή διακριτική ικανότητα και σε όλη τη διάρκειά του. Το θεώρημα δειγματοληψίας θα ποσοτικοποιήσει τη σχέση αυτή.

3.3.2 Ταχείς Αλγόριθμοι Διακριτού Μετασχηματισμού Fourier

Για τον υπολογισμό του διακριτού μετασχηματισμού Fourier ενός σήματος N δειγμάτων, η υπολογιστική πολυπλοκότητα εφαρμογής του ορισμού είναι της τάξης του N^2 και επομένως αρκετά σημαντική. Ευτυχώς, υπάρχουν αλγόριθμοι που έχουν πολυπλοκότητα μόνο της τάξης του $N\log_2 N$. Οι αλγόριθμοι αυτοί είναι γνωστοί ως ταχείς μετασχηματισμοί Fourier και παρουσιάζουν την ταχύτερη συμπεριφορά όταν το N είναι δύναμη του δύο. Για N = 1024, το υπολογιστικό κέρδος $\frac{N^2}{\log_2 N} \approx 100$. Το υπολογιστικό κέρδος αυξάνεται όσο μεγαλώνει ο αριθμός N των δειγμάτων.

Η βασική ιδέα πίσω από τους ταχείς μετασχηματισμούς Fourier βρίσκεται στο ότι ο διακριτός μετασχηματισμός Fourier μπορεί να εκφραστεί ως πολλαπλασιασμός πινάκων:

$$\begin{bmatrix} F_0 \\ \cdot \\ \cdot \\ F_{N-1} \end{bmatrix} = \boldsymbol{W} \begin{bmatrix} f_0 \\ \cdot \\ \cdot \\ f_{N-1} \end{bmatrix}, \text{όπου } \boldsymbol{W} = [W_{nm}] = \left[e^{i2\pi\frac{mn}{N}}\right] \quad (16).$$

Όμοια,

$$\begin{bmatrix} f_0 \\ \cdot \\ \cdot \\ f_{N-1} \end{bmatrix} = \frac{1}{N}\boldsymbol{W}^{\dagger} \begin{bmatrix} F_0 \\ \cdot \\ \cdot \\ F_{N-1} \end{bmatrix}, \text{όπου } \boldsymbol{W}^{\dagger} = \boldsymbol{W}^{*\boldsymbol{T}} = \boldsymbol{W}^{*} \quad (17).$$

Οι πίνακες $\boldsymbol{W}$ και $\boldsymbol{W}^{\dagger}$ έχουν πολλές συμμετρίες και, όταν N = 2^p, μπορούν να παραγοντοποιηθούν σε p πίνακες τάξης N x N που περιέχουν επαναλαμβανόμενες τιμές, μηδενικά και μονάδες. Αυτή η ιδιαίτερη δομή των πινάκων $\boldsymbol{W}$ και $\boldsymbol{W}^{\dagger}$ επιτρέπει τη λειτουργία ταχέων αλγορίθμων.

3.3.3 Ταχύς Μετασχηματισμός Fourier με Αποδεκάτιση Χρόνου

Έστω x(n) ένα σήμα πεπερασμένης διάρκειας, όπου 0≤n≤N - 1 και N είναι άρτιος. Αν $x_1(n)$ και $x_2(n)$ είναι τα σήματα που αποτελούνται από τις άρτιες και τις περιττές τιμές του x(n) αντίστοιχα ισχύει:

$$x_1(n) = x(2n) \text{ όπου } n = 0,1,2,\dots,\frac{N}{2}-1 \quad (18),$$

$$x_2(n) = x(2n+1) \text{ όπου } n = 0,1,2,\dots,\frac{N}{2}-1 \quad (19).$$

Τότε λέγεται ότι τα σήματα $x_1(n)$ και $x_2(n)$ λαμβάνονται με αποδεκάτιση (decimation) του σήματος x(n) με το 2. Και τα δύο σήματα έχουν διάρκεια $\frac{N}{2}$. Ορίζοντας $\omega_N = e^{i\frac{2\pi}{N}}$, ο διακριτός μετασχηματισμός Fourier του x(n) μπορεί να γραφεί ως εξής:

$$X(k) = \sum_{n=0,\pm2,\pm4,\dots} x(n)\omega_N^{nk} + \sum_{n=1,\pm3,\pm5,\dots} x(n)\omega_N^{nk} \quad (20)$$

ή

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n)\omega_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\omega_N^{2(n+1)k} \quad (21).$$

Επειδή $\omega_N^2 = e^{-i\frac{2\pi}{N}2} = \omega_{N/2}$ η εξίσωση γίνεται

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x_1(n)\omega_{N/2}^{nk} + \omega_N^k \sum_{n=0}^{\frac{N}{2}-1} x_2(n)\omega_{N/2}^{nk} \quad (22),$$

οπότε,

$$X(k) = X_1(k) + \omega_N^k X_2(k), \qquad k = 0,1,2\dots,\frac{N}{2}-1 \quad (23).$$

Με τη χρήση της τελευταίας εξίσωσης οι πρώτες $\frac{N}{2}$ τιμές του διακριτού μετασχηματισμού Fourier μπορούν να υπολογιστούν από το διακριτό μετασχηματισμό Fourier των $\frac{N}{2}$ αποδεκατισθέντων σημάτων $x_1(n)$ και $x_2(n)$. Λόγω της περιοδικότητας του διακριτού μετασχηματισμού Fourier ισχύει:

$$X_1\left(k+\frac{N}{2}\right) = X_1(k) \quad (24), \qquad X_2\left(k+\frac{N}{2}\right) = X_2(k) \quad (25).$$

Επίσης, επειδή $\omega_N^{\frac{N}{2}} = e^{-i\pi} = -1$, ισχύει ότι, $\omega_N^{k+\frac{N}{2}} = \omega_N^k \omega_N^{\frac{N}{2}} = -\omega_N^k$ (26),

έτσι συνάγονται οι ακόλουθες σχέσεις:

$$X(k) = X_1(k) + \omega_N^k X_2(k) \text{ όπου } 0 \le k \le \frac{N}{2} - 1 \quad (27),$$

$$X\left(k + \frac{N}{2}\right) = X_1(k) - \omega_N^k X_2(k) \text{ όπου } 0 \le k \le \frac{N}{2} - 1 \quad (28).$$

Οπότε, με τη χρήση των δυο τελευταίων εξισώσεων επιτυγχάνεται ελάττωση κατά περίπου 50% των υπολογισμών σε σχέση με τον απευθείας υπολογισμό του διακριτού μετασχηματισμού Fourier από τον ορισμό του.

3.3.4 Χρήσιμες Ιδιότητες Μετασχηματισμού Fourier

1. Διατήρηση της ενέργειας, δηλαδή

$$\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} |f(m,n)|^2 = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} |F(k,l)|^2 \quad (29).$$

2. Συμπίεση της ενέργειας, δηλαδή το φαινόμενο μόνο ένας σχετικά μικρός αριθμός από τους συντελεστές του ορθομοναδιαίου μετασχηματισμού να λαμβάνουν απόλυτη τιμή εμφανώς μεγαλύτερη από τους υπόλοιπους των οποίων η απόλυτη τιμή είναι πρακτικά μηδενική.
3. Στατιστική αποσυσχέτιση, δηλαδή το φαινόμενο οι συντελεστές του ορθομοναδιαίου μετασχηματισμού, θεωρούμενοι ως τυχαίες μεταβλητές, να είναι στατιστικώς αμοιβαία ανεξάρτητοι σε ευρείες κλάσεις σημάτων.

3.4 Δειγματοληψία Σημάτων Συνεχούς Χρόνου

Μια σημαντική κατηγορία σημάτων διακριτού χρόνου προκύπτει από τη δειγματοληψία (sampling) σημάτων συνεχούς χρόνου. Η διαδικασία αυτή είναι απαραίτητη όταν σήματα συνεχούς χρόνου πρόκειται να υποστούν επεξεργασία με ψηφιακά μέσα και πραγματοποιείται από ειδικές διατάξεις οι οποίες καλούνται αναλογικό ψηφιακοί μετατροπείς (analog-to-digital converters).

Όταν ένα σήμα συνεχούς χρόνου πρόκειται να ψηφιοποιηθεί και αναζητείται η άριστη τεχνική δειγματοληψίας, πρέπει να μελετηθούν προσεκτικά το φάσμα του σήματος, ο αποδεκτός μέσος ρυθμός δειγματοληψίας και ο τρόπος επεξεργασίας του σήματος.

Η διαδικασία της δειγματοληψίας πρέπει να εκτελεστεί έτσι ώστε η ακολουθία των δειγμάτων που θα προκύψει να αναπαριστά το αρχικό σήμα, με όσο το δυνατόν μικρότερη απώλεια πληροφορίας. Αν το φάσμα του σήματος μπορεί να περιοριστεί, όπως απαιτείται από το θεώρημα δειγματοληψίας, τότε η περιοδική δειγματοληψία εφαρμόζεται για διάφορους

λόγους. Πρώτον, γιατί είναι η απλούστερη μέθοδος και μπορεί να υλοποιηθεί εύκολα. Δεύτερον, γιατί οι περιοδικές ακολουθίες δειγμάτων σήματος υφίστανται εύκολα ψηφιακή επεξεργασία. Τέλος, γιατί έχουν αναπτυχθεί πολλοί ταχείς και αποτελεσματικοί αλγόριθμοι για την υλοποίηση αντίστοιχων ψηφιακών φίλτρων.

3.4.1 Περιοδική Δειγματοληψία

Θεωρώντας ένα θετικό αριθμό T και ένα σήμα συνεχούς χρόνου x(t) λαμβάνονται δείγματα του x(t) ανά T χρονικές μονάδες. Προκύπτει ένα σήμα διακριτού χρόνου $\{x_n, n=0,\pm1, \pm2, \ldots\}$ που δίνεται από τη σχέση $x_n=x(nt)$. Καταχρηστικά, γράφεται $x_n=x(nT)$. Ένα σήμα διακριτού χρόνου που προκύπτει από σήμα συνεχούς χρόνου μέσω δειγματοληψίας ονομάζεται δειγματοποιημένο (sampled signal). Η ποσότητα T ονομάζεται περίοδος δειγματοληψίας (sampling period) και 1/T είναι ο ρυθμός (sampling rate) ή συχνότητα δειγματοληψίας (sampling frequency). Για τη δειγματοληψία σημάτων συνεχούς χρόνου ισχύει το θεώρημα της επόμενης ενότητας που αποδίδεται στους Whittaker, Shannon και Nyquist.

3.4.2 Το Θεώρημα Δειγματοληψίας

Θεωρούμε το σήμα συνεχούς χρόνου x(•) και την ακολουθία $\{x_n, n=0,\pm1, \pm2, \ldots\}$ δειγμάτων του που λαμβάνονται περιοδικά με περίοδο T_s, δηλαδή $x_n=x(nT_s)$ για κάθε n. Ισχύει το ακόλουθο θεώρημα δειγματοληψίας:

Θεώρημα Whittaker, Shannon, Nyquist:

Αν το σήμα x(•) έχει μετασχηματισμό Fourier $X(s) = \int_{-\infty}^{\infty} x(t)e^{i2\pi st}dt$ τέτοιο ώστε X(s) = 0 όταν $|s| \geq s_0$για κάποια συχνότητα s_0 και η συχνότητα δειγματοληψίας του σήματος είναι τουλάχιστον διπλάσια της μέγιστης συχνότητας του $s_0(\frac{1}{T_s} \geq 2s_0)$τότε η ακολουθία $\{x_n=x(nT_s), n=0,\pm1, \pm2, \ldots\}$ επαρκεί για να ανακατασκευαστεί το σήμα x(•) σε οποιαδήποτε χρονική στιγμή. Πιο συγκεκριμένα:

$$X(t) = \sum_{n=-\infty}^{\infty} x_n \frac{\sin[\frac{\pi(t-nT_s)}{T_s}]}{\frac{\pi(t-nT_s)}{T_s}} = \sum_{n=-\infty}^{\infty} x(nT_s)\sin c(\frac{t-nT_s}{T_s}) \quad (30),$$

όπου $\sin c(n) = \frac{\sin(\pi u)}{\pi u}$.

3.5 Κβάντωση Σημάτων

Στην ιδανική περίπτωση, η δειγματοληψία ενός σήματος συνεχούς χρόνου παράγει ένα σήμα διακριτού χρόνου αποτελούμενο από μια ακολουθία δειγμάτων, κάθε ένα από τα οποία είναι γνωστό με άπειρη ακρίβεια. Στην πράξη, όμως, η ιδανική αυτή κατάσταση προσεγγίζεται από μια διαδικασία η οποία μετατρέπει ένα σήμα συνεχούς χρόνου σε ένα ψηφιακό σήμα, δηλαδή σε μια ακολουθία δειγμάτων πεπερασμένης ακρίβειας. Κάθε δείγμα πεπερασμένης ακρίβειας αναφέρεται ως κβαντισμένο δείγμα (quantized sample) και παράγεται στην έξοδο ενός κβαντωτή (quantizer), δηλαδή ενός μη γραμμικού συστήματος το οποίο μετασχηματίζει (απεικονίζει) ένα δείγμα άπειρης ακρίβειας σε μια τιμή από ένα πεπερασμένο σύνολο προκαθορισμένων τιμών. Οι κβαντωτές ορίζουν ομοιόμορφα ή μη ομοιόμορφα κατανεμημένα επίπεδα κβάντωσης, αλλά περισσότερο συνηθισμένοι είναι οι κβαντωτές του πρώτου τύπου. Στην ουσία, ένας κβαντωτής με ομοιόμορφα κατανεμημένα επίπεδα κβάντωσης στρογγυλοποιεί τις τιμές του δείγματος εισόδου στο πλησιέστερο επίπεδο κβάντωσης.

3.6 Γραμμικά Χρονικά Αναλλοίωτα Συστήματα

3.6.1 Η Ιδιότητα της Γραμμικότητας

Ένα σύστημα είναι γραμμικό όταν το σήμα εξόδου που αντιστοιχεί στο γραμμικό συνδυασμό δύο σημάτων εισόδου είναι ο ίδιος γραμμικός συνδυασμός των σημάτων εξόδου που αντιστοιχούν σε καθένα από τα σήματα εισόδου. Πιο αυστηρά, το σύστημα S(•) είναι γραμμικό αν και μόνο αν $S(a_1n_1+a_2n_2) = a_1S(u_1) + a_2S(u_2)$ για κάθε a_1, a_2 και οποιαδήποτε σήματα εισόδου u_1, u_2.

3.6.2 Κρουστική Απόκριση και Συνέλιξη

Όταν ένα γραμμικό σύστημα είναι και χρονικά αναλλοίωτο, τότε η περιγραφή του γίνεται πλήρως μόνο από την κρουστική του απόκριση.

Κρουστική Απόκριση

Έστω x(•) το σήμα εισόδου ενός γραμμικού, χρονικά αναλλοίωτου συστήματος με αντίστοιχο σήμα εξόδου y(•). Από τον "ορισμό" της συνάρτησης δ, έχουμε:

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau \,, για\ κάθε\ t \quad (31).$$

Έστω ότι διεγείρουμε το σύστημα με σήμα εισόδου μια συνάρτηση δέλτα και αυτό αποκρίνεται με σήμα εξόδου μια συνάρτηση h(•). Επειδή το σύστημα είναι χρονικά

αναλλοίωτο, η απόκρισή του σε διέγερση δ(• — τ) θα είναι h(• — τ). Παράλληλα, επειδή είναι γραμμικό, η απόκρισή του στο σήμα εισόδου x(•) θα είναι:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \quad (32),$$

όποιο και αν είναι το σήμα εισόδου x(•).

Η συνάρτηση h(•), δηλαδή η απόκριση του συστήματος σε σήμα εισόδου δ(•), ονομάζεται κρουστική απόκριση (impulse response) του συστήματος και καθορίζει πλήρως το σύστημα με την έννοια ότι προσδιορίζει την απόκριση σε οποιοδήποτε σήμα εισόδου. Το σήμα εξόδου είναι μια υπέρθεση (με βάρη που καθορίζονται από τις τιμές του σήματος εισόδου) χρονικά μετατοπισμένων κρουστικών αποκρίσεων. Παρατηρούμε όμως ότι:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau = \int_{-\infty}^{\infty} x(t-\tau)h(\tau)d\tau \quad (33),$$

δηλαδή το σήμα εξόδου είναι υπέρθεση (με βάρη που καθορίζονται από τις τιμές της κρουστικής απόκρισης) χρονικά μετατοπισμένων σημάτων εισόδου.

Συνέλιξη

Ένα ολοκλήρωμα της μορφής:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau = \int_{-\infty}^{\infty} x(t-\tau)h(\tau)d\tau \quad (34),$$

ονομάζεται ολοκλήρωμα συνέλιξης (convolution integral) των συναρτήσεων x(•) και h(•). Η συνάρτηση ονομάζεται συνέλιξη των συναρτήσεων χ(•) και h(•) και συμβολίζεται ως y = x*h.

Γραμμικά, χρονικά αναλλοίωτα συστήματα διακριτού χρόνου

Ένα γραμμικό, χρονικά αναλλοίωτο σύστημα διακριτού χρόνου χαρακτηρίζεται από μια κρουστική απόκριση h_n, έτσι ώστε το σήμα εξόδου y_n που αντιστοιχεί σε σήμα εισόδου x_n να δίνεται από τη συνέλιξη διακριτού χρόνου:

$$y_n = \sum_{n'=-\infty}^{\infty} x_{n'}h_{n-n'} = \sum_{n'=-\infty}^{\infty} x_{n-n'}h_{n'} \quad (35)$$

3.6.3 Αρμονική Απόκριση και Συνάρτηση Μεταφοράς

Θεωρούμε το γραμμικό, χρονικά αναλλοίωτο σύστημα συνεχούς χρόνου με κρουστική απόκριση h(•),

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau = \int_{-\infty}^{\infty} x(t-\tau)h(\tau)d\tau \quad (36)$$

Ως σήμα εισόδου, θεωρούμε το μιγαδικό εκθετικό σήμα $x_s(t) = e^{-i2\pi st}$συχνότητας s. Έχουμε:

$$y_s(t) = \int_{-\infty}^{\infty} e^{-i2\pi s(t-\tau)}h(\tau)d\tau = e^{-i2\pi st} \int_{-\infty}^{\infty} e^{-i2\pi st}h(\tau)d\tau \quad (37),$$

αλλά,

$\int_{-\infty}^{\infty} e^{-i2\pi s\tau}h(\tau)d\tau = H$ (38)

και επομένως

$y_s(t) = H(s) \cdot x_s(t)$ (39).

Παρατηρούμε τα εξής:

1. Η απόκριση (σήμα εξόδου) οποιουδήποτε γραμμικού, χρονικά αναλλοίωτου συστήματος σε σήμα εισόδου $x_s(t) = e^{-i2\pi st}$ είναι το ίδιο το σήμα εισόδου πολλαπλασιασμένο με μια (γενικά μιγαδική σταθερά) H(s).
2. Η μιγαδική σταθερά H(s) είναι ακριβώς η τιμή του μετασχηματισμού Fourier της κρουστικής απόκρισης h(•) του συστήματος για συχνότητα s.
3. Τα μιγαδικά εκθετικά σήματα $x_s(t) = e^{-i2\pi st}$αποτελούν ιδιοσήματα οποιουδήποτε γραμμικού, χρονικά αναλλοίωτου συστήματος με αντίστοιχη ιδιοτιμή H(s).
4. $H(s) = \int_{-\infty}^{\infty} h(t)e^{i2\pi st}dt$ (40),

   $h(t) = \int_{-\infty}^{\infty} H(s)e^{-i2\pi st}ds$ (41).

Η συνάρτηση H(s) ονομάζεται συνάρτηση μεταφοράς (transfer function) του συστήματος.

Ας θεωρήσουμε τώρα ένα οποιοδήποτε σήμα εισόδου x(•) με μετασχηματισμό Fourier $X_s(t) = \int_{-\infty}^{\infty} x(t)e^{i2\pi st}dt$.

Έχουμε:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \quad (42)$$

και

$$Y(s) = \int_{-\infty}^{\infty} y(t)e^{i2\pi st}dt \quad (43)$$

άρα,

$$Y(s) = \int_{-\infty}^{\infty} e^{i2\pi st}[\int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau]dt = \int_{-\infty}^{\infty} x(\tau)[\int_{-\infty}^{\infty} h(t-\tau)e^{i2\pi st}dt]d\tau$$

$$= \left[\int_{-\infty}^{\infty} x(\tau)e^{i2\pi s\tau}d\tau\right] \cdot H(s) = X(s)H(s) \quad (44).$$

Δηλαδή, ο μετασχηματισμός Fourier Y(•) της απόκρισης (σήματος εξόδου) ενός γραμμικού, χρονικά αναλλοίωτου συστήματος είναι ίσος με το γινόμενο του μετασχηματισμού Fourier X(•) του σήματος και της συνάρτησης μεταφοράς H(•) του συστήματος. Με άλλα λόγια, η πράξη της συνέλιξης στο πεδίο του χρόνου αντιστοιχεί σε πολλαπλασιασμό στο πεδίο της συχνότητας (και αντίστροφα).

4 Ακουστικός Θόρυβος

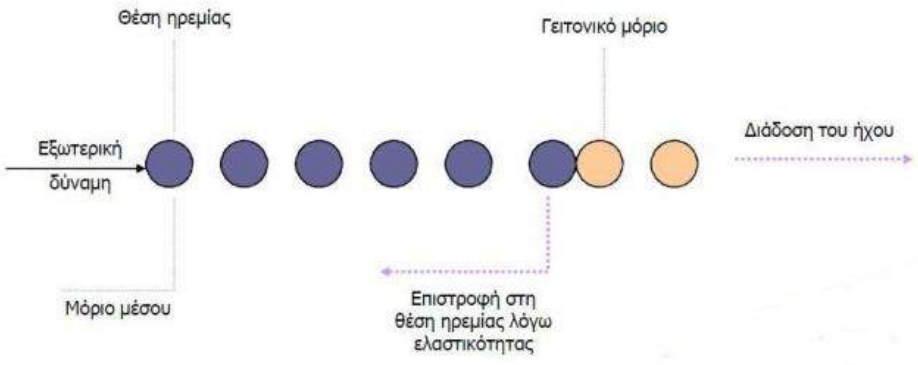
Στο κεφάλαιο αυτό η ανάλυση επικεντρώνεται στον ακουστικό θόρυβο και τις φυσικές ιδιότητες του. Μάλιστα δόθηκε ιδιαίτερη έμφαση στο μηχανισμό της ακοής και στις επιπτώσεις του θορύβου σε αυτόν. Επιπλέον γίνεται αναφορά τόσο στα καθιερωμένα όσο και σε εναλλακτικά μέτρα θορύβου. Η ανάλυση ολοκληρώνεται με την παρουσίαση της μεθοδολογίας για τον υπολογισμό του επιπέδου της ηχητικής πίεσης.

4.1 Φυσικές Ιδιότητες Θορύβου

4.1.1 Δονήσεις και Κύματα

Σύμφωνα με τον ΕΛΟΤ 263.1 οι ορισμοί του θορύβου είναι οι εξής:
«Θόρυβος ονομάζεται κάθε απεριοδικός σύνθετος ήχος που η στιγμιαία τιμή του αυξομειώνεται με τυχαίο τρόπο» και «Θόρυβος ονομάζεται κάθε δυσάρεστος ή ανεπιθύμητος ήχος». Ενώ στο ίδιο πρότυπο ορίζεται και ο ήχος ως «η μηχανική διαταραχή που διαδίδεται με ορισμένη ταχύτητα μέσα σε ένα μέσο που μπορεί να αναπτύξει εσωτερικές δυνάμεις (π.χ. Ελαστικότητας, εσωτερικής τριβής) κι έχει τέτοιο χαρακτήρα, ώστε μπορεί να διεγείρει το αισθητήριο της ακοής και να προκαλέσει ακουστικό αίσθημα» [10]. Η φυσική περιγραφή του ήχου είναι δυνατή μέσω της μηχανικής δόνησης ενός αέριου, υγρού ή στερεού μέσου κατά την οποία πραγματοποιείται μεταφορά ενέργειας από μία πηγή υπό την μορφή προοδευτικών κυματισμών. Με άλλα λόγια ο ήχος θα μπορούσε να οριστεί ως το μικρό τμήμα της ενέργειας που χάνεται σε κάθε περίπτωση κίνησης ή δόνησης ενός αντικειμένου [1].

Προκειμένου να γίνει καλύτερα αντιληπτή η φυσική διαδικασία παραγωγής του ήχου θεωρούμε ένα σωματίδιο του μέσου αρκετά μεγάλο ώστε να είναι αντιπροσωπευτικό των φυσικών ιδιοτήτων του, αλλά μικρότερο σε σχέση με τις τυπικές διαστάσεις της ακουστικής διαταραχής, όπως αυτή καθορίζεται από το μήκος κύματος που σχετίζεται με την δόνηση του



Σχήμα 28: Κίνηση σωματιδίων κατά την διάδοση του ήχου [1]

μορίου. Συγκεκριμένα, η μετατόπιση ενός μορίου από τη θέση ισορροπίας του έχει σαν

αποτέλεσμα την σύγκρουσή του με το γειτονικό του μόριο στο οποίο προκαλεί αντίστοιχη μετατόπιση, ενώ το ίδιο επανακτά την αρχική θέση ισορροπίας του. Η συγκεκριμένη διαδικασία επαναλαμβάνεται μέσω των διαδοχικών ταλαντώσεων των μορίων καθώς η διαταραχή μεταδίδεται από μόριο σε μόριο του ελαστικού μέσου. Κατά τη διάδοση της διαταραχής η μόνη φυσική ποσότητα που μεταφέρεται είναι η ενέργεια καθώς τα μόρια του ελαστικού μέσου τίθενται το ένα μετά το άλλο σε ταλάντωση περί την αρχική θέση ισορροπίας τους (Σχ. 28).

Ο χρόνος που απαιτείται προκειμένου η κίνηση να μεταδοθεί μεταξύ δυο διαδοχικών μορίων, καθορίζοντας την ταχύτητα διάδοσης της διαταραχής, εξαρτάται από την ελαστικότητα του μέσου και δίνεται από την ακόλουθη εξίσωση:

$$\boldsymbol{c = k\sqrt{E/\rho}} \quad (45),$$

όπου k είναι μια σταθερά, E είναι ο συντελεστής ελαστικότητας του μέσου και ρ είναι η πυκνότητα του μέσου. Για τον αέρα, υπό κανονικές συνθήκες, η συγκεκριμένη ταχύτητα είναι ίση με 344 m/sec στους 20^{o}C. Με βάση τη σχέση (45), μπορούμε να υπολογίσουμε το μήκος κύματος που αντιστοιχεί στην διαδιδόμενη διαταραχή σύμφωνα με την εξίσωση:

$$\boldsymbol{\lambda = cT = \frac{c}{f}} \quad (46),$$

όπου λ είναι το μήκος κύματος, T είναι ο χρόνος μεταξύ διαδοχικών ταλαντώσεων των μορίων, c είναι η ταχύτητα του ήχου στον αέρα και f η συχνότητα της διαταραχής.

Η παραπάνω ανάλυση αναφέρεται στην απλούστερη μορφή ακτινοβολούμενου κύματος, το επίπεδο κύμα, το οποίο αντλεί την ονομασία του από το γεγονός ότι μεταδίδεται από την πηγή προς μία και μόνο διεύθυνση καθώς τα μέτωπα του κύματος παραμένουν παράλληλα το ένα ως προς το άλλο. Οι διαταραχές που αντιστοιχούν στα επίπεδα κύματα αποσβένονται κυρίως λόγω απωλειών κατά την μετάδοσή τους και της διαβάθμισης της θερμοκρασίας εντός του ίδιου του μέσου.

Παρά το γεγονός ότι η εκτίμηση του μεγέθους του ηχητικού κύματος μπορεί να πραγματοποιηθεί με μια πλειάδα διαφορετικών τρόπων, είναι συνηθέστερη η μέτρηση της ακουστικής πίεσης καθώς είναι πολύ δυσκολότερο να προσδιορίσει κανείς παραμέτρους που σχετίζονται με την μετατόπιση ή την ταχύτητα των μορίων του ελαστικού μέσου. Ο καθορισμός των συγκεκριμένων παραμέτρων είναι αναγκαίος υπό κανονικές συνθήκες μόνο σε περιπτώσεις κατά τις οποίες οι μετρήσεις πραγματοποιούνται κοντά στο πεδίο της ηχητικής πηγής. Αυτό συμβαίνει διότι, το διάνυσμα της ταχύτητας των σωματιδίων του ελαστικού μέσου δεν είναι απαραίτητα παράλληλο προς στην κατεύθυνση προς την οποία ταξιδεύει το ηχητικό κύμα με αποτέλεσμα το μέγεθος της ηχητικής πίεσης να παρουσιάζει σημαντικές διαφοροποιήσεις σε μικρά διαστήματα κατά την διεύθυνση διάδοσης του

ηχητικού κύματος. Η ακουστική ένταση μπορεί να υπολογισθεί ως το μέσο του τετραγώνου της ηχητικής πίεσης σε πεδία που βρίσκονται σε μεγαλύτερες αποστάσεις από την ηχητική πηγή. Το Επίπεδο της Ηχητικής Πίεσης (Sound Pressure Level) αποτελεί μια εύκολα μετρήσιμη παράμετρο. Για το λόγο αυτό, το Επίπεδο της Ηχητικής Πίεσης (SPL) έχει καθιερωθεί ως ο πιο συνηθισμένος τρόπος έκφρασης του μεγέθους του ακουστικού πεδίου.

4.1.2 Ακουστική Ισχύς, Ενεργειακή Πυκνότητα και Ένταση

Ακουστική Ενέργεια

Κάθε πηγή θορύβου έχει μια χαρακτηριστική ακουστική ενέργεια ενώ τα Επίπεδα της Ηχητικής Πίεσης που προκαλεί εξαρτώνται από πολλούς εξωγενείς παράγοντες μεταξύ των οποίων συμπεριλαμβάνονται η απόσταση και ο προσανατολισμός του δέκτη, οι διαβαθμίσεις της θερμοκρασίας και της ταχύτητας εντός του ελαστικού μέσου, καθώς και το περιβάλλον. Η ακουστική ενέργεια, είναι μια σημαντική παράμετρος η οποία χρησιμοποιείται ευρέως για την απόλυτη εκτίμηση και σύγκριση των ακουστικών πηγών.

Πυκνότητα Ακουστικής Ενέργειας

Η ακουστική ενέργεια η οποία περικλείεται στη μονάδα όγκου του ελαστικού μέσου αποτελεί μια θεμελιώδη παράμετρο για οποιοδήποτε τύπο ακουστικού πεδίου. Ο επίσημος όρος που χρησιμοποιείται για την εν λόγω ποσότητα είναι η ενεργειακή πυκνότητα και συνδέεται με την ακουστική πίεση μέσω της ακόλουθης εξίσωσης:

$$\boldsymbol{D = \frac{{p_{rms}}^2}{\rho c^2}} \quad (17).$$

Ένταση

Η ένταση ορίζεται μέσω της ενέργειας που διαρρέει μια μοναδιαία περιοχή του ακουστικού πεδίου στη μονάδα του χρόνου και διαφέρει ανάλογα με τον τύπο του ακουστικού πεδίου. Στην περίπτωση ενός ελεύθερου πεδίου όπου τα ηχητικά κύματα καταφθάνουν μόνο από την διεύθυνση της πηγής, η ένταση δίνεται από την παρακάτω σχέση:

$$I = \frac{{p_{rms}}^2}{4\rho c^2} \quad (28).$$

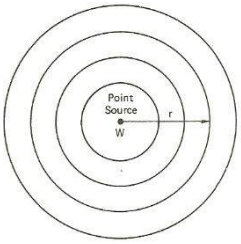
Στην περίπτωση ενός ακουστικού πεδίου, στο οποίο η πιθανότητα ο ήχος να καταφθάνει από οποιαδήποτε διεύθυνση είναι ίση, η καθαρή ένταση είναι μηδενική. Ωστόσο, η ένταση του ήχου που διαπερνά ένα επίπεδο μοναδιαίας επιφάνειας από μία και μόνο πλευρά θα δίνεται από τη σχέση:

$$I = \frac{W}{4\pi r^2} \quad (49),$$

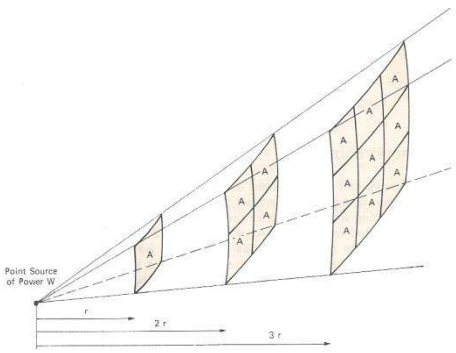
όπου το γινόμενο ρc είναι η χαρακτηριστική ακουστική αντίσταση του μέσου, η οποία για τον αέρα σε θερμοκρασία 20^{o}C είναι 407rayls (Kg m^{-2} sec^{-1}.)

4.1.3 Σημειακή Πηγή

Μια ηχητική πηγή μπορεί να θεωρηθεί ως σημειακή όταν οι διαστάσεις της είναι μικρές σε σχέση με την απόστασή της από τον δέκτη. Ως σημειακές πηγές μπορούν να θεωρηθούν πολλές από τις συνηθισμένες πηγές θορύβου όπως είναι οι βιομηχανικές εγκαταστάσεις, τα αεροσκάφη και τα μεμονωμένα οχήματα. Στο Σχ. 29 παρουσιάζεται μια ιδανική σημειακή πηγή η οποία μπορεί να θεωρηθεί ότι παράγει μια ακολουθία από σφαιρικά μέτωπα κύματος ως αποτέλεσμα των διαδοχικών διαταραχών της. Στην περίπτωση μιας καθαρής ημιτονοειδούς διαταραχής, η απόσταση ανάμεσα σε δύο διαδοχικά μέτωπα κύματος, τα οποία αντιστοιχούν σε διαδοχικές μέγιστες τιμές της πίεσης, θα είναι ίση με το μήκος κύματος. Το συγκεκριμένο γεγονός έχει ιδιαίτερη σημασία σε περιπτώσεις κατά τις οποίες θα πρέπει να συνυπολογιστούν φαινόμενα ανάκλασης εντός του ηχητικού πεδίου. Όπως γίνεται αντιληπτό με βάση το Σχ. 30, η ηχητική ενέργεια διαδίδεται ισότροπα προς όλες τις διευθύνσεις καθώς απομακρύνεται όλο και περισσότερο από το κέντρο της σφαίρας στο οποίο βρίσκεται η ηχητική πηγή. Ο υπολογισμός της έντασης μιας σημειακής πηγής μπορεί να πραγματοποιηθεί θεωρώντας πως η ενεργειακή της έξοδος διαπερνά εξολοκλήρου ένα σφαιρικό κέλυφος ακτίνας r, ως το πηλίκο της εκπεμπόμενης ενέργειας προς το εμβαδόν της το εμβαδόν της σφαιρικής επιφάνειας μέσω της ακόλουθης σχέσης:



Σχήμα 29: Διάδοση σφαιρικών μετώπων κύματος από σημειακή πηγή [1]



Σχήμα 30: Διασπορά ήχου από σημειακή πηγή [1]

$$\boldsymbol{I = \frac{W}{4\pi r^2}} \quad (50).$$

Η σχέση (50) υποδεικνύει πως η ένταση της ηχητικής πηγής είναι αντιστρόφως ανάλογη προς το τετράγωνο της απόστασης από την πηγή. Με άλλα λόγια, η ένταση εξασθενεί 6dB για κάθε διπλασιασμό της απόστασης.

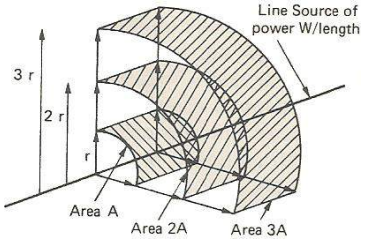
4.1.4 Γραμμική Πηγή

Βασικό χαρακτηριστικό μιας γραμμικής ηχητικής πηγής είναι ότι η ενέργειά της εκπέμπεται κατά συνεχή τρόπο στο χώρο, όπως για παράδειγμα συμβαίνει με ένα αγωγό που

μεταφέρει ένα ταραχώδες υγρό. Ως γραμμική μπορεί να θεωρηθεί μια ηχητική πηγή η οποία προκύπτει ως το αποτέλεσμα της συνισταμένης δράσης ενός αριθμού από σημειακές πηγές, οι αποστάσεις μεταξύ των οποίων είναι αρκετά μικρές ώστε να εκπέμπουν με συνεχή τρόπο πάνω σε μια νοητή γραμμή που να τις συνδέει. Στην κατηγορία των γραμμικών πηγών συγκαταλέγονται οι εργοστασιακές πηγές, όπου μηχανές και ιμάντες μεταφοράς λειτουργούν σε πολύ κοντινές αποστάσεις, καθώς και δύο εξαιρετικά σημαντικές πηγές περιβαλλοντικού θορύβου που είναι οι δρόμοι και οι σιδηρόδρομοι.

Συγκεκριμένα, ο θόρυβος που παράγεται από έναν δρόμο υψηλής κυκλοφορίας μπορεί να θεωρηθεί ως το αποτέλεσμα της δράσης μιας γραμμικής πηγής, παρά ως μια ακολουθία μεμονωμένων γεγονότων, εξαιτίας της συνεχούς διέλευσης οχημάτων. Οι σιδηρόδρομοι θεωρούνται συνήθως ως γραμμικές πηγές θορύβου σε απόσταση από τις γραμμές ώστε να διευκολυνθεί η μελέτη του φαινομένου η οποία επιβάλλεται από την έντονη κοινωνική ενόχληση που προκαλεί. Ωστόσο, η δράση του θορύβου που προκαλείται από ένα σιδηρόδρομο δεν μπορεί να θεωρηθεί γραμμική σε πολύ κοντινές ή πολύ μακρινές αποστάσεις από τις γραμμές καθώς στις περιπτώσεις αυτές η πολυπλοκότητα του φαινομένου αυξάνεται.

Το Σχ. 31 παρουσιάζει ένα τμήμα μιας γραμμικής πηγής άπειρου μήκους η οποία εκπέμπει μια σταθερή ποσότητα ενέργειας ανά μονάδα μήκους. Τα μέτωπα των κυμάτων που εκπέμπονται από τη συγκεκριμένη πηγή διαδίδονται αποκλειστικά κατά μήκος της διάστασης που είναι κάθετη στον άξονα κίνησης της πηγής, έτσι ώστε δύο οποιαδήποτε σημεία που ισαπέχουν από την νοητή ευθεία της να βρίσκονται πάντοτε στο ίδιο μέτωπο κύματος και να έχουν τις ίδιες ιδιότητες. Αυτό έχει σαν αποτέλεσμα τα μέτωπα των εκπεμπόμενων κυμάτων να σχηματίζουν κυλινδρικές επιφάνειες γύρω από τον άξονα της γραμμικής πηγής. Η ενέργεια που εκλύεται σε αυτή την περίπτωση ανά μονάδα μήκους της πηγής σε μία μονάδα χρόνου διαπερνά το ίδιο μήκος κυλινδρικής επιφάνειας για όλες τις ακτίνες. Η ένταση επομένως, της ηχητικής πηγής σε μια δοσμένη ακτίνα r μπορεί να υπολογισθεί ως το πηλίκο της ενέργειας που εκλύεται από το αντίστοιχο στοιχειώδες μήκος της γραμμικής πηγής προς το εμβαδόν της αντίστοιχης κυλινδρικής επιφάνειας σύμφωνα με τη σχέση:



Σχήμα 31: Διασπορά ήχου από γραμμική πηγή [1]

$$\boldsymbol{I = \frac{W}{2\pi r}} \quad (51).$$

Η σχέση (51) υποδεικνύει πως η ένταση της ηχητικής πηγής είναι αντιστρόφως ανάλογη προς την απόσταση από την πηγή. Με άλλα λόγια, η ένταση εξασθενεί 3dB για κάθε διπλασιασμό της απόστασης.

4.1.5 Κλίμακες Θορύβου

Η διάδοση μιας διαταραχής εντός ενός ελαστικού μέσου πραγματοποιείται υπό τη μορφή ενός κύματος, το μέγεθος του οποίου περιγράφεται συνήθως μέσω της τετραγωνικής ρίζας του μέσου τετραγώνου του πλάτους του (Arms). Η συγκεκριμένη ποσότητα εκφράζεται συνήθως σε τιμές η μέτρηση των οποίων απαιτεί τη χρήση κατάλληλων μονάδων και κλιμάκων. Η παράμετρος που χρησιμοποιείται συχνότερα κατά τη διενέργεια μετρήσεων θορύβου είναι η ηχητική πίεση παρά η ένταση του ηχητικού πεδίου. Η μέτρηση της ηχητικής πίεσης εκφράζεται σε μονάδες δύναμης ανά μονάδα επιφάνειας, η οποία στο σύστημα SI πραγματοποιείται με χρήση μονάδων Pascal (Pa), όπου 1 Pa αντιστοιχεί σε δύναμη 1 Newton ανά τετραγωνικό μέτρο [3].

Η κανονική διαδικασία μέτρησης της πίεσης πάνω σε μια γραμμική κλίμακα προκαλεί συγκεκριμένα προβλήματα τα οποία σχετίζονται με τις επιδόσεις της ανθρώπινης ακοής. Ο ασθενέστερος ήχος που μπορεί να γίνει αντιληπτός από ένα μέσο ανθρώπινο αυτί είναι της τάξης των 1000 Hz και έχει βρεθεί ότι η συχνότητα αυτή αντιστοιχεί σε πίεση 20 μPa. Η συγκεκριμένη ποσότητα πίεσης έχει καθιερωθεί ως η ονομαστική τιμή του ακουστικού κατωφλίου για την διενέργεια μετρήσεων του επιπέδου ηχητικής πίεσης. Στο άλλο άκρο της κλίμακας βρίσκεται το κατώφλι του πόνου, το οποίο αντιστοιχεί προσεγγιστικά σε μια τιμή πίεσης της τάξης των 100 Pa. Το εύρος των τιμών που καλύπτει η συγκεκριμένη κλίμακα ορίζει ένα λόγο της μέγιστης προς την ελάχιστη τιμή της τάξης του 1 εκατομμύριο προς 1, καθιστώντας προφανείς τους λόγους για τους οποίους η μέτρηση της ηχητικής πίεσης δεν συνίσταται να πραγματοποιείται με απευθείας εφαρμογή γραμμικών κλιμάκων οι οποίες περιλαμβάνουν τεράστια πλήθη αριθμών τα οποία είναι εξαιρετικά δύσκολο να τα διαχειριστεί κανείς.

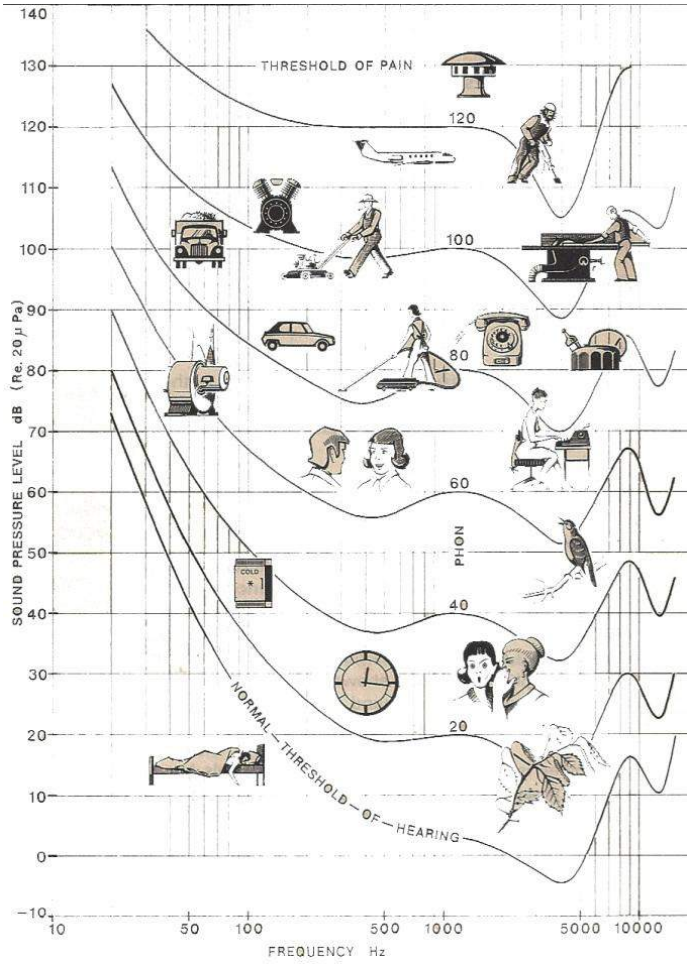
Επιπλέον, έχει αποδειχθεί ότι η απόκριση του ανθρώπινου αυτιού στα διάφορα ερεθίσματα είναι λογαριθμική και όχι γραμμική. Για τους παραπάνω λόγους, έχει βρεθεί ότι είναι πιο πρακτική η έκφραση των ακουστικών παραμέτρων μέσω του λογαριθμικού λόγου της μετρούμενης τιμής προς μια καθιερωμένη τιμή. Η συγκεκριμένη πρακτική έχει σαν αποτέλεσμα την ελάττωση του πλήθους των διαθέσιμων αριθμητικών τιμών της κλίμακας σε ευκολότερα διαχειρίσιμες αναλογίες. Η αντίστοιχη μονάδα μέτρησης ορίζεται ως ο δεκαδικός λογάριθμος του λόγου δύο ακουστικών πιέσεων ή εντάσεων και ονομάζεται Bel από το όνομα του Alexander Graham Bel. Στην πράξη χρησιμοποιείται το ένα δέκατο αυτής της

κλίμακας, μιας και βρέθηκε να είναι αρκετά μεγάλη για τους γενικότερους σκοπούς χρήσης της. Το γεγονός ότι η ακουστική ένταση, η ενέργεια δηλαδή που διέρχεται από μια περιοχή μοναδιαίου εμβαδού, είναι ανάλογη του τετραγώνου της ακουστικής πίεσης τεκμηριώνει την καταλληλότητα της κλίμακας που ορίζεται μέσω της ακόλουθης σχέσης:

Sound Pressure Level $$L_p = \mathbf{10}\log_{10}(\frac{p(t)}{p_0})^2 = \mathbf{20}\log_{10}(\frac{p(t)}{p_0}) \quad (52),$$

για τη διενέργεια ακουστικών μετρήσεων. Ως p(t) συμβολίζεται η μετρούμενη στιγμιαία ηχητική πίεση ενώ p_0 είναι η ακουστική πίεση αναφοράς, η οποία έχει συνήθως την τιμή των 20μPa.

Η χρήση του όρου επίπεδο στην έννοια του Επιπέδου της Ηχητικής Πίεσης (Sound Pressure Level) δεν είναι τυχαία αλλά επισημαίνει το γεγονός ότι η εν λόγω ποσότητα αντιστοιχεί σε ένα συγκεκριμένο επίπεδο πάνω από κάποια προκαθορισμένη τιμή αναφοράς.



Σχήμα 32: Τυπικά επίπεδα ηχητικής πίεσης συνηθισμένων πηγών θορύβου [1]

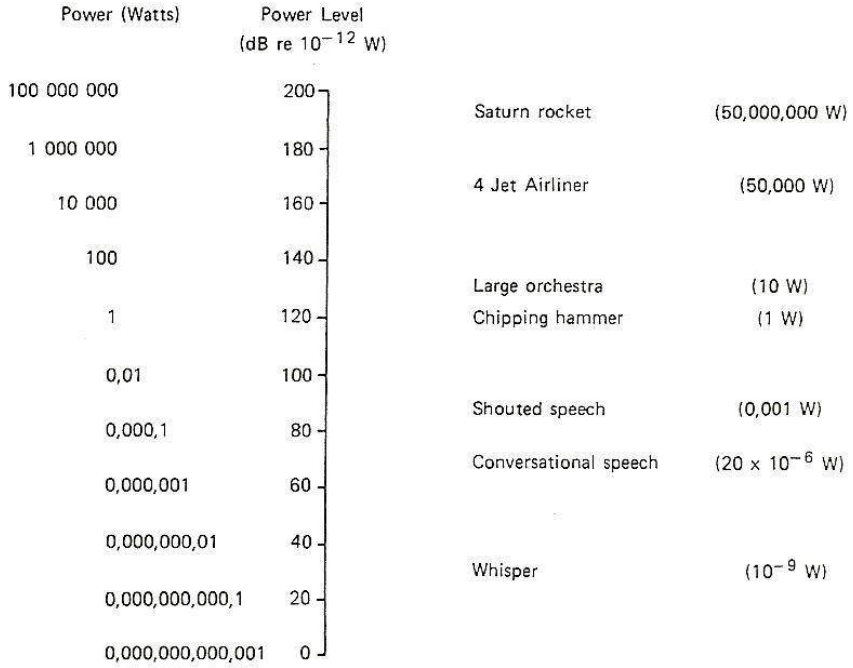
Η κλίμακα decibel μπορεί να χρησιμοποιηθεί για την πραγματοποίηση οποιασδήποτε μέτρησης, ανεξάρτητα της μονάδας στην οποία μετριέται το υπό εξέταση μέγεθος, αρκεί να γίνεται ρητή αναφορά στην μονάδα στην οποία εκφράζεται η απόλυτη τιμή αναφοράς της λογαριθμικής κλίμακας. Η χρήση της κλίμακας decibel έχει σαν συνέπεια τον περιορισμό του δυναμικού εύρους των ηχητικών πιέσεων από ένα λόγο της τάξης του 1 εκατομμυρίου προς 1 σε ένα λόγο της τάξης του 0 προς 120, όπου το 0 υποδηλώνει το ελάχιστο κατώφλι αναφοράς ενώ το 120 υποδηλώνει ένα προσεγγιστικό κατώφλι του πόνου. Το Σχ. 32 παρουσιάζει τα τυπικά επίπεδα της ηχητικής πίεσης για διάφορες συνηθισμένες πηγές θορύβου.

Η Ακουστική Ισχύς εκφράζεται συνήθως στην κλίμακα decibel εξαιτίας του τεράστιου εύρους των ισχύων που μπορεί να συναντήσει κανείς σε τυπικά προβλήματα

θορύβου. Το Επίπεδο της Ηχητικής Ισχύος (Sound Power Level) ορίζεται ως το δεκαπλάσιο του δεκαδικού λογαρίθμου που αντιστοιχεί στο πηλίκο μεταξύ της μετρούμενης ηχητικής ισχύος και μιας προκαθορισμένης ισχύος αναφοράς της τάξης των 10^{-12} Watt και θα δίνεται από την ακόλουθη σχέση:

$$\text{Sound Power Level } \boldsymbol{L_W = 10\log_{10}(\frac{W}{W_0})^2} \quad (53).$$

Στην σχέση (53) ο όρος W αντιστοιχεί στο επίπεδο της μετρούμενης ισχύος ενώ ο όρος Wo αντιστοιχεί στην τιμή της ισχύος αναφοράς. Ο πίνακας (Σχ.33) παρουσιάζει την έξοδο ισχύος για μερικές τυπικές μορφές θορύβου.



Σχήμα 33: Έξοδος ακουστικής ενέργειας τυπικών θορύβων [1]

Η σωστή διαχείριση ποσοτήτων οι οποίες σχετίζονται με την κλίμακα decibel προϋποθέτει την σαφή κατανόηση μερικών θεμελιωδών γεγονότων. Είναι εξαιρετικά σημαντικό, να μην παραγνωρίζει κανείς το γεγονός ότι η κλίμακα decibel είναι μια λογαριθμική κλίμακα για την οποία δεν ισχύουν οι συνήθεις ιδιότητες που χαρακτηρίζουν γραμμικές ποσότητες. Μέσα σε αυτό το πλαίσιο, το μηδενικό επίπεδο στην κλίμακα decibel δεν συνεπάγεται την απουσία θορύβου, αλλά το γεγονός ότι το επίπεδο της μετρούμενης ηχητικής πίεσης είναι ίσο με το επίπεδο αναφοράς. Εξίσου σημαντικό είναι να κατανοήσει κανείς τον τρόπο υπολογισμού του επιπέδου της ηχητικής πίεσης σε περιπτώσεις όπου έχουμε την ταυτόχρονη δράση δύο ή και περισσότερων πηγών θορύβου. Η περιγραφή του τρόπου υπολογισμού του επιπέδου της

ηχητικής πίεσης για την γενική περίπτωση της ταυτόχρονης δράσης n συσχετισμένων πηγών, όπου $p_k(t),\ k \in [n]$ η στιγμιαία πίεση που εκλύεται από την k-οστή πηγή θορύβου, προϋποθέτει των υπολογισμό του μέσου τετραγώνου της συνολικής πίεσης η οποία δίνεται από την παρακάτω εξίσωση:

$$\boldsymbol{p_{tot}(t) = \sum_{k=1}^{n} p_k(t)} \quad (54).$$

Με βάση τη σχέση (54), η μέση τετραγωνική πίεση κατά τη διάρκεια του χρόνου της μέτρησης δίνεται από την παρακάτω εξίσωση:

$$\overline{p_{tot}^2} = \frac{1}{T}\int_0^T \left\{\sum_{k=1}^{n} p_k^2(t)\right\} dt \Leftrightarrow \overline{p_{tot}^2} = \frac{1}{T}\int_0^T \left\{\sum_{k=1}^{n}\sum_{m=1}^{n} p_k(t)\, p_m(t)\right\} dt \Leftrightarrow$$

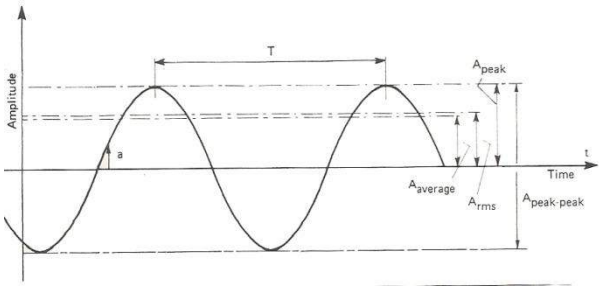
$$\boldsymbol{\overline{p_{tot}^2} = \frac{1}{T}\int_0^T \left\{\sum_{k=1}^{n} p_k^2(t) + \sum_{k\neq m} 2p_k(t)p_m(t)\right\} dt \Leftrightarrow \overline{p_{tot}^2} = \sum_{k=1}^{n} \overline{p_k^2} + \sum_{k\neq m} 2\overline{p_k}\,\overline{p_m}} \quad (55).$$

Η σχέση (55) καθορίζει τον τρόπο υπολογισμού του συνολικού επιπέδου της ηχητικής πίεσης κατά την από κοινού δράση δύο ή και περισσότερων συσχετισμένων πηγών. Όπως είναι προφανές, η μέση ολική τετραγωνική πίεση δεν είναι εξολοκλήρου ίση με το άθροισμα των μέσων τετραγώνων των επιμέρους πιέσεων αλλά εξαρτάται από τον τρόπο αλληλεπίδρασης του κάθε ξεχωριστού ζεύγους πηγών. Ωστόσο, στην περίπτωση όπου οι ηχητικές πηγές είναι ανά δύο ασυσχέτιστες, η σχέση (55) θα μπορούσε να γραφεί ως εξής:

$$\boldsymbol{\overline{p_{tot}^2} = \sum_{k=1}^{n} \overline{p_k^2}} \quad (56).$$

4.1.6 Χαρακτηριστικά Ενθόρυβων Σημάτων

Κάθε ηχητικό σήμα χαρακτηρίζεται από συγκεκριμένες φυσικές παραμέτρους οι οποίες είναι σχετικά απλές για τις περιπτώσεις σημάτων σταθερού πλάτους και συχνότητας ενώ η πολυπλοκότητά τους αυξάνεται σημαντικά όταν η συχνότητα και το πλάτος των σημάτων μεταβάλλεται με το χρόνο. Η απλούστερη μορφή ενός ηχητικού σήματος είναι αυτή που αντιστοιχεί σε έναν καθαρό τόνο με σταθερή συχνότητα και



Σχήμα 34: Ημιτονοειδές σήμα και διάφορα μέτρα του πλάτους του

ημιτονοειδώς μεταβαλλόμενο πλάτος όπως φαίνεται στο Σχ. 34. Η χρονική εξέλιξη του εν λόγω σήματος επαναλαμβάνεται με σταθερή περίοδο Τ. Η έννοια της φάσης δεν είναι σημαντική για την ανάλυση μεμονωμένων ενθόρυβων σημάτων καθώς προκύπτουν κατά κανόνα ως το αποτέλεσμα της τυχαίας σύνθεσης πολλών συχνοτικών συνιστωσών.

Το πλάτος ενός ημιτονοειδούς σήματος μπορεί να εκφραστεί ως συνάρτηση των παραμέτρων που παρουσιάζονται στο Σχ. 34 οι οποίες συνδέονται μεταξύ τους μέσω των εξισώσεων που ακολουθούν.

Η τιμή που αντιστοιχεί στην τετραγωνική ρίζα του μέσου τετραγώνου ενός οποιουδήποτε σήματος κατά τη διάρκεια μιας περιόδου είναι ανάλογη του ενεργειακού του περιεχομένου και αποτελεί μία από τις σημαντικότερες και συχνότερα χρησιμοποιούμενες παραμέτρους για την ποσοτική περιγραφή του πλάτους ενός σήματος. Η συγκεκριμένη ποσότητα περιγράφεται μαθηματικά από την σχέση:

$$\boldsymbol{A_{rms}} = \sqrt{\frac{\mathbf{1}}{\boldsymbol{T}}\int_{\mathbf{0}}^{\boldsymbol{T}} \boldsymbol{a^2(t)dt}} \quad (57),$$

όπου Τ είναι η περίοδος του ημινονοειδούς σήματος και a(t) το στιγμιαίο πλάτος του. Ένας εναλλακτικός τρόπος περιγραφής του πλάτους ενός σήματος είναι μέσω του υπολογισμού της μέσης απόλυτης τιμής του, η οποία θα δίνεται από την παρακάτω σχέση:

$$\boldsymbol{A_{average}} = \frac{\mathbf{1}}{\boldsymbol{T}}\int_{\mathbf{0}}^{\boldsymbol{T}} \boldsymbol{|a(t)|dt} \quad (58).$$

Εκτός από τις σχέσεις (57) και (58) οι οποίες περιγράφουν ποσοτικά τη μέση συμπεριφορά του πλάτους του σήματος κατά τη χρονική διάρκεια της μιας περιόδου, είναι δυνατό να αναφερθεί κανείς και στην μέγιστη τιμή του πλάτους του σήματος για το ίδιο χρονικό διάστημα με βάση τη σχέση:

$$\boldsymbol{A_{peak}} = \mathbf{max}\{\boldsymbol{a(t)}\}, \boldsymbol{t} \in \boldsymbol{T} \quad (59).$$

Στην περίπτωση ενός απλού ημιτονοειδούς σήματος οι ποσότητες Arms, Aaverage και Apeak που εμφανίζονται στις σχέσεις (57), (58) και (59) συνδέονται μέσω της ακόλουθης εξίσωσης:

$$\boldsymbol{A_{rms}} = \frac{\boldsymbol{\pi}}{\mathbf{2}\sqrt{\mathbf{2}}}\boldsymbol{A_{average}} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}}}\boldsymbol{A_{peak}} \quad (60).$$

Δύο εξίσου σημαντικοί παράγοντες που σχετίζονται ιδιαίτερα με την περιγραφή της κυματομορφής ενός ενθόρυβου σήματος είναι ο συντελεστής της κορυφογραμμής που δίνεται από τη σχέση:

$$\text{Crest Factor } \boldsymbol{F_c} = \frac{\boldsymbol{A_{peak}}}{\boldsymbol{A_{rms}}} \quad (61)$$

και ο συντελεστής του σχήματος που δίνεται από τη σχέση:

Form Factor $\boldsymbol{F_f} = \frac{\boldsymbol{A_{rms}}}{\boldsymbol{A_{average}}}$ (62),

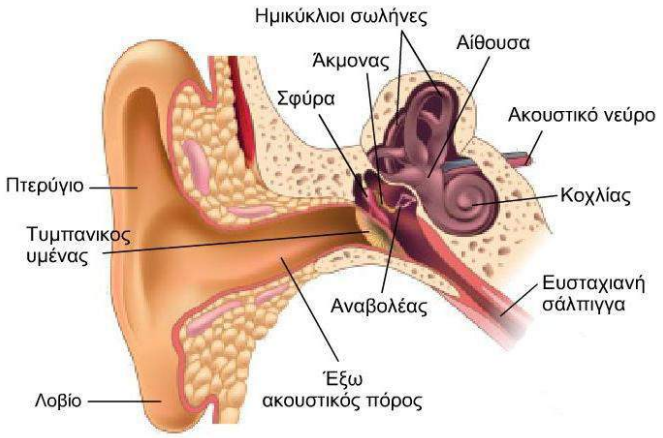
όπου $F_c = 1.414 \approx 3dB$ και $F_f = 1.11 \approx 1dB$

Δυστυχώς στις περισσότερες των περιπτώσεων, τα σήματα θορύβου που μελετώνται δεν είναι καθαρά ημιτονοειδή, καθώς το πλάτος τους και η συχνότητά τους μεταβάλλονται από κοινού με το χρόνο. Αυτό έχει σαν συνέπεια την μη ύπαρξη απλών μαθηματικών σχέσεων που να συνδέουν τις ποσότητες Arms, Aaverage και Apeak, οι οποίες όμως εξακολουθούν να συνιστούν σημαντικούς περιγραφείς τη χρονικής εξέλιξης ενός σήματος. Ειδικότερα, η μέτρηση της ποσότητας Apeak σε καθαρά τυχαία θορυβικά σήματα δεν έχει κανένα νόημα εκτός εάν πρόκειται για περιπτώσεις που έχουν να κάνουν με αιφνίδια θορυβικά γεγονότα ελάχιστης χρονικής διάρκειας.

4.2 Ψυχοακουστική - Μέτρα Θορύβου

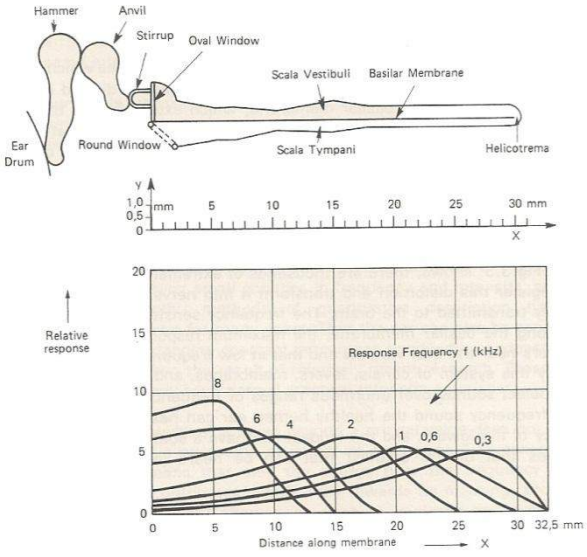
4.2.1 Ο Μηχανισμός της Ακοής

Το ανθρώπινο αυτί αποτελείται από τρία κύρια μέρη όπως φαίνονται στο Σχ. 35. Το εξωτερικό και το μέσο αυτί είναι τα τμήματα εκείνα που συλλέγουν τα αερομεταφερόμενα κύματα του ήχου, οδηγώντας τα στο γεμάτο με υγρό εσωτερικό αυτί, το οποίο λειτουργεί ως μετατροπέας, μετασχηματίζοντας τις μηχανικές δονήσεις των σημάτων σε νευρικές ωθήσεις οι οποίες μεταφέρουν την ακουστική πληροφορία στον εγκέφαλο. Το εξωτερικό αυτί αποτελείται από τον λοβό και το ακουστικό κανάλι με κύρια αποστολή την συλλογή των αερομεταφερόμενων κυμάτων του ήχου τα οποία διοχετεύει μέσω του ακουστικού καναλιού στο τύμπανο του αυτιού, θέτοντάς το σε ταλάντωση. Το μέσο αυτί δρα ως μια συσκευή σύνθετης αντίστασης αποτελούμενο από τρία μικρά οστά τα οποία λειτουργούν σαν ένα σύνολο μοχλών, με ένα μηχανικό όφελος κατά προσέγγιση της τάξης του 3:1, μεταφέροντας τις δονήσεις του τυμπάνου του αυτιού στο εσωτερικό αυτί.



Σχήμα 35: Κύρια μέρη αυτιού [21]

Η τελική σύνδεση μεταξύ του ακουστικού γεγονότος και της αντίστοιχης νευρικής ώθησης πραγματοποιείται στο εσωτερικό αυτί το οποίο αποτελείται από δύο ξεχωριστά συστήματα, τα ημικυκλικά κανάλια και τον κοχλία. Τα ημικυκλικά κανάλια έχουν πρωτίστως την μέριμνα της ισορροπίας ενώ η δράση του κοχλία αφορά αυτή καθεαυτή την ακοή. Η γεμάτη υγρό κοιλότητα του κοχλία διαιρείται σε δύο διαμήκη κανάλια από την βασική μεμβράνη η οποία εκτείνεται κατά μήκος ολόκληρου του κοχλία με εξαίρεση ένα μικρό κενό που βρίσκεται στο βάθος του κοχλία και ονομάζεται ελικότρημα (helicotrema).



Σχήμα 36:Διαμήκη διατομή του κοχλία [1]

Τα Σχ. 35 και Σχ. 36 παρουσιάζουν τη διαμήκη και την εγκάρσια τομή του κοχλία σε ανεπτυγμένη μορφή αντίστοιχα. Η απόκριση του αναβολέα σε ένα ηχητικό ερέθισμα προκαλεί την μετακίνηση του ωοειδούς παραθύρου η οποία με την σειρά της έχει σαν αποτέλεσμα την μετάδοση της διαταραχής του υγρού κατά μήκος του ακουστικού καναλιού διαμέσω του ελικοτρήματος στο κατώτερο κανάλι και τελικά στο κυκλικό παράθυρο το οποίο εκτρέπεται προκειμένου να φιλοξενήσει την συγκεκριμένη διαταραχή. Η διάδοση της διαταραχής μέσω των προαναφερθέντων καναλιών επιφέρει την στρέβλωση της βασικής μεμβράνης, στην ανώτερη επιφάνεια της οποίας υπάρχουν εκατοντάδες εξαιρετικά ευαίσθητα τριχοειδή κύτταρα (Σχ. 36) τα οποία καταγράφουν την παραμόρφωσή της μετατρέποντάς τη σε νευρικές ωθήσεις που μεταδίδονται τελικά στον εγκέφαλο. Η ευαισθησία του ανθρώπινου μηχανισμού ακοής ποικίλει κατά μήκος της βασικής μεμβράνης, όπου η μέγιστη απόκριση στις υψηλές συχνότητες συμβαίνει στο ωοειδούς παράθυρο ενώ για τις χαμηλές συχνότητες συμβαίνει κοντά στο ελικότρημα. Το ανθρώπινο αυτί εκμεταλλευόμενο αυτό το σύστημα καναλιών, αναλογέων, μεμβρανών και τριχοειδών κυττάρων έχει την ικανότητα να αντιλαμβάνεται ένα τεράστιο εύρος ηχητικών συχνοτήτων και εντάσεων. Συγκεκριμένα το ανθρώπινο σύστημα ακοής μπορεί να συλλάβει ήχους υψηλής συχνότητας, η μέγιστη τιμή των οποίων μπορεί να είναι 1000 φορές μεγαλύτερη από την αντίστοιχη ελάχιστη τιμή των ήχων χαμηλής συχνότητας. Αντίστοιχα, η πίεση των ισχυρότερων ήχων που συλλαμβάνει το ανθρώπινο αυτί μπορεί να είναι και 1 εκατομμύριο φορές μεγαλύτερη από την ένταση των ασθενέστερων αντιληπτών ήχων.

4.2.2 Επιπτώσεις Θορύβου και Επιτρεπτά Επίπεδα

Ο θόρυβος οφείλεται στις ηχητικές συνθήκες του χώρου και προκαλείται από την συμβολή πολλών ηχογόνων παραγόντων όπως καταιγίδα, κυκλοφοριακή κίνηση, μέσα ψυχαγωγίας, ανθρώπινη δραστηριότητα κ.ά [2]. Οι παράγοντες που καθορίζουν την επικινδυνότητα του θορύβου είναι:

- Η στάθμη ηχητικής πίεσης (dB).
- Η συχνότητα του ήχου.
- Η διάρκεια της έκθεσης.

Οι επιπτώσεις του θορύβου στον άνθρωπο είναι πολλαπλές και ποικίλες και επηρεάζοντας τις καθημερινές του δραστηριότητες. Ξεκινούν από ένα απλό εκνευρισμό ή δυσφορία και μπορούν να καταλήξουν σε μόνιμες βλάβες του οργανισμού. Οι κύριοι κίνδυνοι του θορύβου για την υγεία που παρουσιάζονται από τον Παγκόσμιο Οργανισμό Υγείας (WHO) είναι οι εξής:

- Πόνος και ακροαστική κόπωση.
- Ενόχληση.
- Επιρροή στην κοινωνική συμπεριφορά (επιθετικότητα).
- Παρεμπόδιση της επικοινωνίας μέσω ομιλίας.
- Διαταραχή του ύπνου.
- Καρδιαγγειακές επιπτώσεις.
- Ορμονικές αντιδράσεις και τις πιθανές του συνέπειες στον ανθρώπινο μεταβολισμό και το ανοσοποιητικό σύστημα.
- Μειωμένη απόδοση σε δουλειά και εργασία.

Λόγω της σοβαρότητας των επιπτώσεων του θορύβου τα κράτη, όπως και η Ελλάδα έχουν θεσπίσει μια σειρά νομοθετικών και διοικητικών μέτρων για τον περιορισμό της ηχορύπανσης [33]. Κυρίως στους χώρους εργασίας, όπου είναι σε λειτουργία πολλά μηχανήματα που παράγουν θόρυβο, εγκυμονεί πολλαπλούς κινδύνους για την υγεία την ασφάλεια των εργαζομένων:

- Απώλεια ακοής: ο υπερβολικός θόρυβος προκαλεί βλάβες στα τριχωτά κύτταρα στον κοχλία, στο έσω ους, με αποτέλεσμα την απώλεια της ακοής.
- Άγχος στην εργασία: το άγχος στην εργασία σπανίως οφείλεται σε ένα μόνον αίτιο και συνήθως προκαλείται από την αλληλεπίδραση περισσοτέρων παραγόντων κινδύνου. Ο θόρυβος στο περιβάλλον εργασίας μπορεί να είναι παράγοντας άγχους, ακόμη και όταν βρίσκεται σε χαμηλά επίπεδα.

- Αυξημένος κίνδυνος ατυχημάτων: τα υψηλά επίπεδα θορύβου δυσχεραίνουν την επικοινωνία μεταξύ των μελών του προσωπικού, αυξάνοντας την πιθανότητα ατυχημάτων.

Ο Παγκόσμιος Οργανισμός Υγείας συνιστά στο χώρο εργασίας ο θόρυβος σε σταθερό επίπεδο να μην υπερβαίνει τα 85 dB(A) και στιγμιαία όχι περισσότερο από 120 dB(A). Αντιστοίχως στο χώρο του ύπνου, σε σταθερό επίπεδο λιγότερο από 30 dB(A) και όχι περισσότερο από 45dB(A). Σύμφωνα με τον WHO οι μέγιστες επιτρεπόμενες τιμές για την ένταση του θορύβου σε κάποιους περιβάλλοντες χώρους παρουσιάζονται στον πίνακα 1.

Πίνακας 1: Οδηγός μέγιστων επιτρεπτών τιμών για την ηχορύπανση σε συγκεκριμένα περιβάλλοντα (WHO, 2008a) [23]

Περιβάλλον	**Επιπτώσεις στην υγεία**	**Ένταση θορύβου (dB)**	**Διάρκεια έκθεσης (h)**	**Μέγιστη στιγμιαία τιμή (dB)**
Εξωτερικοί χώροι	Σοβαρή ενόχληση ημέρα & νύχτα	55	16	-
Εξωτερικοί χώροι	Μικρή ενόχληση ημέρα & νύχτα	50	16	-
Κατοικίες εσωτερικοί χώροι	Κατανόηση ομιλίας, μικρή ενόχληση ημέρα & νύχτα	35	16	45
Δωμάτια ύπνου	Διαταραχή ύπνου τη νύχτα	45	8	60
Σχολικές αίθουσες	Ενόχληση στην κατανόηση ομιλίας	35	Διάρκεια μαθήματος	
Δωμάτια ύπνου για προσχολική ηλικία	Διαταραχή ύπνου	30	Διάρκεια ύπνου	45
Σχολικές αυλές	Ενόχληση	55	Διάρκεια ημέρας	-
Νοσοκομειακοί θάλαμοι	Διαταραχή ύπνου	30	8	40
Νοσοκομειακά ιατρεία		30	16	
Βιομηχανία, εμπορικές επιχειρήσεις, μαγαζιά, συγκοινωνίες	Επίδραση στην ακοή	70	24	110
Τελετές, φεστιβάλ, συναυλίες κλπ		100	4	110

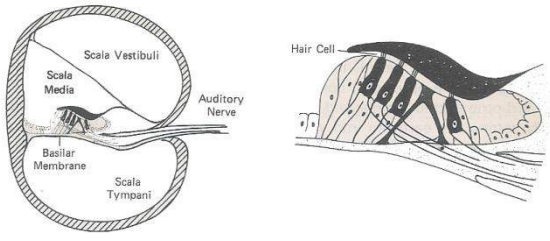
Συγκεντρώσεις σε κλειστό χώρο		85	1	110
Μουσική και άλλοι ήχοι από ηχεία και ακουστικά		85	1	110
Σειρήνες από παιχνίδια, πυροσβεστική κλπ				140

Τα ανώτερα επιτρεπτά επίπεδα θορύβου (ηχοστάθμης) συναρτήσει των συνεχόμενων ωρών έκθεσης, για να μην προκληθεί μη αναστρέψιμη βλάβη της ακοής παρουσιάζονται στον πίνακα 2.

Πίνακας 2: Επίπεδα θορύβου πέραν των οποίων συντρέχει κίνδυνος απώλειας της ακοής [7]

Διάρκεια έκθεσης (Ώρες ανά ημέρα)	Μέγιστη επιτρεπτή ηχοστάθμη dB(A)
8	87
4	90
5	93
1	96
1/2	99
1/4	102

4.2.3 Εναλλακτικά Μέτρα Θορύβου



Σχήμα 37: Εγκάρσια διατομή του κοχλία

Το συνολικό επίπεδο ηχητικής πίεσης (SPL) όπως αυτό περιγράφεται από τη σχέση (52) αποτελεί το απλούστερο θορυβικό μέτρο που μπορεί να χρησιμοποιήσει κανείς χωρίς να πραγματοποιήσει στάθμιση του μετρούμενου ηχητικού σήματος, εκμεταλλευόμενος στο ακέραιο το φάσμα των συχνοτήτων που μπορεί να γίνουν αντιληπτοί από το ανθρώπινο σύστημα ακοής (συνήθως από 20Hz έως 20KHz). Ωστόσο, το συγκεκριμένο μέτρο χρησιμοποιείται σπανίως εξαιτίας της εξαιρετικά μικρής του συσχέτισης με τον υποκειμενικό τρόπο αντίληψης του ήχου από τον άνθρωπο.

Leq - Ισοδύναμο Επίπεδο Ηχητικής Πίεσης

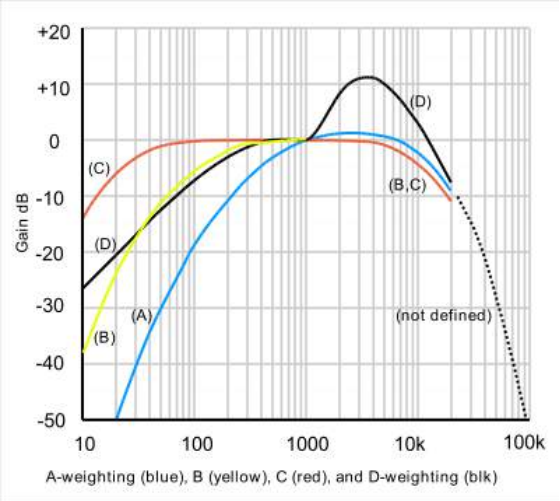
Η ποσότητα Leq αντιστοιχεί στο μέσο επίπεδο του θορύβου, το οποίο προκύπτει ως η αναμενόμενη τιμή του Επιπέδου της Ηχητικής Πίεσης (SPL) κατά τη χρονική διάρκεια της μιας περιόδου [6]. Η μαθηματική περιγραφή της εν λόγω ποσότητας θα δίνεται από την παρακάτω σχέση:

$$\boldsymbol{L_{eq} = 10\log_{10}\frac{1}{T}\int_0^T (\frac{p(t)}{p_0})^2\, dt} \quad (63),$$

όπου T είναι ο συνολικός χρόνος μέτρησης, p(t) η στιγμιαία τιμή της ηχητικής πίεσης ενώ po είναι η τιμή της πίεσης αναφοράς η οποία είναι της τάξης των 20μPa.

A - Σταθμισμένο Επίπεδο Ηχητικής Πίεσης: και άλλες συχνοτικές σταθμίσεις

Η A - Στάθμιση αποτελεί την πιο συχνά χρησιμοποιούμενη καμπύλη για την μέτρηση του Επιπέδου της Ηχητικής Πίεσης (SPL) από μια οικογένεια καμπυλών που ορίζονται με βάση το διεθνές πρότυπο IEC 61672. Η συγκεκριμένη ποσότητα αποτελεί τη συνήθη έξοδο των διάφορων μετρητικών συσκευών όπως είναι τα ηχόμετρα, των οποίων η συχνοτική απόκριση έχει ρυθμιστεί έτσι ώστε να ακολουθεί κατά προσέγγιση την ισοδύναμη καμπύλη ηχητικής έντασης της τάξης των 40 phons. Το A - Σταθμισμένο Επίπεδο της Ηχητικής Πίεσης εκφράζεται σε db(A) και έχει αποδειχθεί ότι παρουσιάζει υψηλή συσχέτιση με την υποκειμενική απόκριση του ανθρώπινου συστήματος ακοής στα διάφορα ηχητικά ερεθίσματα. Μάλιστα, η καλύτερη απόδοση του συγκεκριμένου τρόπου στάθμισης σε σχέση με άλλες κλίμακες θορύβου σε συνδυασμό με την ευκολία υπολογισμού του, τον καθιέρωσε σε πολλά εθνικά και διεθνή πρότυπα. Η καμπύλη που αντιστοιχεί στο A - Σταθμισμένο φίλτρο ορίζεται στο διάστημα από 20Hz έως 20KHz και φαίνεται στο Σχ. 38. Εκτός από την A - Στάθμιση υπάρχουν και άλλες συχνοτικές σταθμίσεις όπως είναι η B - Στάθμιση και η C - Στάθμιση οι οποίες ακολουθούν τις ισοδύναμες καμπύλες ηχητικής έντασης της τάξης των 70 και 100 phons αντίστοιχα. Οι καμπύλες των B και C φίλτρων στάθμισης φαίνονται και αυτές στο Σχ. 38. Φίλτρο A χρησιμοποιείται για μεσαίες εντάσεις ενώ τα φίλτρα B, C για υψηλές εντάσεις [35].



Σχήμα 38: Φασματικές καμπύλες στάθμισης τύπου A, B, C και D

4.3 Υπολογισμός Επιπέδου Ηχητικής Πίεσης

Θεωρούμε ένα ηχογραφημένο ηχητικό σήμα x(t): $0 \leq t \leq T$ όπου T είναι ο συνολικός χρόνος καταγραφής του σήματος σε seconds. Το συγκεκριμένο ηχητικό σήμα θεωρούμε πως έχει δειγματιστεί με συχνότητα δειγματοληψίας Fs (Samples / sec), έτσι ώστε να αναπαρίσταται τελικά ως ένα σύνολο δειγμάτων $Fs \cdot T$.

Ο υπολογισμός του SPL σε db, dbA, dbB, dbC για το παραπάνω ηχητικό σήμα ακολουθεί την εξής διαδικασία:

1. Το σύνολο των ηχητικών δειγμάτων χωρίζεται σε μη επικαλυπτόμενα παράθυρα των $\Delta t \in \{125 \text{ msec}, 1 \text{ sec}\}$ ανάλογα με τον τρόπο λειτουργίας fast ή slow.
2. Υπολογίζουμε τον FFT του σήματος ($X(k)$, $0 \leq k \leq N-1$) για το συγκεκριμένο παράθυρο χρόνου σύμφωνα με τη διαδικασία που περιγράφεται στην παράγραφο 3.3.3, όπου N το πλήθος των δειγμάτων.
3. Υπολογίζουμε:
   - την στάθμιση A για το συγκεκριμένο παράθυρο χρόνου σύμφωνα με τις εξισώσεις: $X_A(k) = \alpha_A(f_k) \cdot X(k)$ για $f_k = k \cdot \Delta f$ με $\Delta f = F_s / N$ και $\alpha_A(f) = \frac{12200^2 \cdot f^4}{(f^2+20{,}6^2)\cdot(f^2+12200^2)\cdot((f^2+107{,}7^2)^{0{,}5})\cdot((f^2+737.9^2)^{0{,}5})}$ (64).
   - την στάθμιση B για το συγκεκριμένο παράθυρο χρόνου σύμφωνα με τις εξισώσεις: $X_B(k) = \alpha_B(f_k) \cdot X(k)$ για $f_k = k \cdot \Delta f$ με $\Delta f = F_s / N$ και $\alpha_B(f) = \frac{12200^2 \cdot f^3}{(f^2+20{,}6^2)\cdot(f^2+12200^2)\cdot((f^2+158{,}5^2)^{0{,}5})}$ (65).
   - την στάθμιση C για το συγκεκριμένο παράθυρο χρόνου σύμφωνα με τις εξισώσεις: $X_C(k) = \alpha_C(f_k) \cdot X(k)$ για $f_k = k \cdot \Delta f$ με $\Delta f = F_s / N$ και $\alpha_C(f) = \frac{12200^2 \cdot f^2}{(f^2+20{,}6^2)\cdot(f^2+12200^2)}$ (66).
4. Υπολογίζουμε την ενέργεια του σήματος για το συγκεκριμένο παράθυρο χρόνου σύμφωνα με την σχέση (29) όπου λαμβάνοντας υπόψη την χρησιμοποιούμενη στάθμιση θα έχουμε ότι:
   - Χωρίς στάθμιση $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X(k)|^2$ (67).
   - Στάθμιση A $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_A(k)|^2$ (68).

- Στάθμιση B $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_B(k)|^2$ (69).

- Στάθμιση C $\varepsilon_x = \frac{1}{N}\sum_{k=0}^{N-1}|X_C(k)|^2$ (70).

Επομένως, η μέση ενέργεια του σήματος για κάθε δυνατή στάθμιση θα δίνεται από τις σχέσεις:

- Χωρίς στάθμιση $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X(k)|^2$ (71).
- Στάθμιση A $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_A(k)|^2$ (72).
- Στάθμιση B $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_B(k)|^2$ (73).
- Στάθμιση C $\bar{\varepsilon}_x = \frac{1}{N*\Delta t}\sum_{k=0}^{N-1}|X_C(k)|^2$ (74).

5. Το SPL θα δίνεται με βάση την εξίσωση 10 ως εξής:

$$SPL = 10log_{10}\left(\frac{\bar{\varepsilon}_x}{\bar{\varepsilon}_{ref}}\right) = 10log_{10}(\bar{\varepsilon}_x) - 10log_{10}(\bar{\varepsilon}_{ref}) \quad (75),$$

όπου, οι μονάδες ανάλογα με τον τύπο στάθμισης θα είναι σε db, dbA, dbB, dbC.

Η ποσότητα $\bar{\varepsilon}_{ref}$ είναι η ενέργεια του επιπέδου αναφοράς η οποία είναι μία σταθερά, γεγονός που σημαίνει πως ο όρος $10log_{10}(\bar{\varepsilon}_{ref})$ μπορεί να αντικατασταθεί από μία σταθερά C η οποία θα ενσωματώνει και την απόκλιση από τις πραγματικές τιμές του επιπέδου τις ηχητικής πίεσης. Έτσι, θα έχουμε ότι:

$$SPL = 10log_{10}\left(\frac{\bar{\varepsilon}_x}{\bar{\varepsilon}_{ref}}\right) = 10log_{10}(\bar{\varepsilon}_x) + C \quad (76),$$

όπου C είναι η σταθερά βαθμονόμησης.

Ο υπολογισμός της σταθεράς βαθμονόμησης προϋποθέτει την ύπαρξη μετρήσεων για το SPL του αρχικού σήματος ανά Δt. Γνωρίζοντας την σταθερά βαθμονόμησης C και εφαρμόζοντας την παραπάνω διαδικασία για το σύνολο των χρονικών παραθύρων για την συνολική χρονική διάρκεια T του σήματος μπορούμε να υπολογίσουμε την χρονική εξέλιξη του SPL.

4.4 Ηχόμετρα

Για την μέτρηση του ήχου χρησιμοποιούνται ειδικά όργανα μέτρησης που ονομάζονται ηχόμετρα [35]. Το ηχόμετρο είναι όργανο σχεδιασμένο για να ανταποκρίνεται στον ήχο κατά τον ίδιο τρόπο όπως το ανθρώπινο αυτί και να παρέχει αντικειμενικά και επαναλήψιμα αποτελέσματα μετρήσεων της στάθμης ηχητικής πίεσης Lp, η οποία αποτελεί το αποτέλεσμα μέτρησης των μεταβολών της πίεσης του αέρα. Η μέτρηση της στιγμιαίας ηχητικής στάθμης δεν είναι εύκολη και για το λόγο αυτό χρησιμοποιούμε ένα άλλο μέγεθος, την ισοδύναμη στάθμη θορύβου (Leq) με μονάδα μέτρησης το dB.

Στην αγορά υπάρχουν αρκετά ηχόμετρα που μας βοηθάνε να μετρήσουμε τον ήχο και να εξάγουμε είτε την στιγμιαία τιμή θορύβου, είτε την μέση τιμή. Ένα τυπικό ηχόμετρο αποτελείται από το μικρόφωνο, την μονάδα επεξεργασίας και τη μονάδα απεικόνισης αποτελεσμάτων μετρήσεων.

Το **μικρόφωνο** μετατρέπει το ηχητικό σήμα σε ισοδύναμο ηλεκτρικό. Ο καταλληλότερος τύπος μικροφώνου είναι το μικρόφωνο πυκνωτικού τύπου, το οποίο συνδυάζει την ακρίβεια με την σταθερότητα και την αξιοπιστία. Το σήμα είναι δυνατόν να υποστεί διαφόρων τύπων επεξεργασία, στην **μονάδα επεξεργασίας,** όπου συνήθως διέρχεται από κατάλληλο σταθμιστικό κύκλωμα, προκειμένου να ληφθεί υπόψη η ιδιαίτερα περίπλοκη απόκριση του ανθρώπινου αυτιού κατά συχνότητα του ακουστού ηχητικού φάσματος.

Σχήμα 39: Ηχόμετρο

Σταθμιστικό κύκλωμα είναι το κύκλωμα που παρεμβάλλεται στα όργανα μέτρησης του ήχου και έχει την ιδιότητα να περιορίζει ή και να ενισχύει, κατά πλάτος και σε διαφορετικό βαθμό, τις διάφορες συνιστώσες του ηχητικού σήματος. Είναι, δηλαδή, ένα ηλεκτρονικό κύκλωμα του οποίου η ευαισθησία μεταβάλλεται ανάλογα με το φίλτρο A, B, C που θέλουμε να χρησιμοποιήσουμε (Ενότητα 4.2.2). Το σταθμιστικό κύκλωμα όπου χρησιμοποιείται κυρίως για τις μετρήσεις θορύβου είναι το A, οπότε τα αποτελέσματα των μετρήσεων ηχητικής πίεσης παρέχονται ως A – ηχοστάθμη ή ως A – στάθμη ηχητικής πίεσης και εκφράζονται σε dB(A).

Στην **μονάδα απεικόνισης** αποτελεσμάτων, που στα σύγχρονα ηχόμετρα είναι μια ψηφιακή οθόνη, απεικονίζεται η μετρούμενη τιμή. Οι περισσότεροι προς μέτρηση ήχοι είναι κυμαινόμενης στάθμης και για την ορθή μέτρηση του ήχου πρέπει οι διακυμάνσεις της στάθμης του να μετρηθούν όσο το δυνατόν ακριβέστερα. Για το σκοπό αυτό έχουν τυποποιηθεί δυο χαρακτηριστικά απόκρισης τα οποία είναι γνωστά ως «Fast», «Slow»

χαρακτηριστικά χρονικής στάθμισης. Στη λειτουργία «Fast», το χρονικό πλαίσιο μέσα στο οποίο γίνεται η εκτίμηση της τιμής της στάθμης ηχητικής πίεσης είναι τα 125 ms και στην «Slow», είναι το 1 sec. Ένα ηχόμετρο για να είναι αξιόπιστο θα πρέπει να καλύπτει τις προδιαγραφές του προτύπου IEC61672 [11], [12], [13]. Το πρότυπο με γενικό τίτλο «Electroacoustics – sound level meters». αποτελείται από τρία μέρη τα οποία είναι:

- *Μέρος 1*: Προδιαγραφές, περιγράφει τις τεχνικές λεπτομέρειες που πρέπει να έχει ένα ηχόμετρο ανάλογα με τον τύπο του, τις ηχοστάθμες και τις κλάσεις των ηχομέτρων.
- *Μέρος 2*: Δοκιμές αξιολόγησης, παρέχει λεπτομέρειες για τις αναγκαίες δοκιμές για να εξακριβώσει τη συμμόρφωση προς όλες τις προδιαγραφές που προβλέπονται στο IEC 61672-1 για τα ηχόμετρα όλων των τύπων.
- *Μέρος 3*: Περιοδικοί έλεγχοι, περιγράφει τις διαδικασίες για περιοδικές δοκιμές των ηχομέτρων.

Σχήμα 40: Βαθμονομητής ηχητικής στάθμης

Τα κύρια χαρακτηριστικά που πρέπει να εξετάζει κάποιος για την αγορά ενός ηχόμετρου, ανάλογα με τον χώρο που θέλει να το χρησιμοποιήσει είναι η κλάση του οργάνου (που αφορά την ακρίβεια του οργάνου), η κλίμακα μέτρησης, ο χρόνος λήψης μετρήσεων, οι δυνατότητες στάθμισης, και οι δυνατότητες μετρήσεων (πχ Μέγιστο, Ελάχιστο, Μέσος όρος κ.α.). Σημαντική παράμετρος για τις σωστές μετρήσεις είναι η βαθμονόμηση του οργάνου. Η βαθμονόμηση του ηχομέτρου γίνεται σύμφωνα με τις υποδείξεις του κατασκευαστή μέσω κατάλληλης ηχητικής πηγής, βαθμονομητή ηχητικής στάθμης, που προσαρμόζεται στο μικρόφωνο του ηχομέτρου και παράγει ένα ή περισσότερα γνωστά επίπεδα ηχητικής πίεσης σε μια ή περισσότερες συγκεκριμένες συχνότητες. Ο χρήστης του ηχομέτρου προσαρμόζει την ένδειξη του ηχομέτρου στην σωστή γνωστή τιμή που εκπέμπει ο βαθμονομητής. Οι προδιαγραφές του βαθμονομητή ηχητικής στάθμης προσδιορίζονται από το πρότυπο IEC 60942 "Electroacoustics - sound calibrators" [14].

4.5 Μικρόφωνα

Η καρδιά ενός ηχομέτρου είναι το μικρόφωνό του, τα χαρακτηριστικά του οποίου επηρεάζουν σημαντικά τα αποτελέσματα των μετρήσεων [8], [6]. Τα μικρόφωνα είναι διατάξεις μετατροπής της μηχανικής ενέργειας σε ηλεκτρική. Αποτελούνται από μια μονάδα

που μετατρέπει την ταλάντωση των μορίων του αέρα σε μηχανική ταλάντωση (διάφραγμα), μια μονάδα που μετατρέπει τη μηχανική ταλάντωση του διαφράγματος σε ηλεκτρικό σήμα και ένα ηλεκτρονικό κύκλωμα. Η ακρίβεια με την οποία αναπαράγεται το ηχητικό κύμα σε ηλεκτρικό σήμα εξαρτάται από τον τύπο και την ποιότητα κατασκευής του μικροφώνου, αλλά και την ταυτοποίηση των απαιτήσεων της κάθε εφαρμογής με το βέλτιστο τύπο μικροφώνου.

Οι βασικότεροι τύποι μικροφώνων είναι τα :

- Πυκνωτικά μικρόφωνα (condenser microphones).
- Δυναμικά μικρόφωνα (dynamic microphones).
- Μικρόφωνα ταινίας (ribbon microphones).
- Μικρόφωνα άνθρακα (carbon microphones).
- Πιεζοηλεκτρικά μικρόφωνα (piezoelectric microphones).

Κατά την επιλογή του κατάλληλου μικροφώνου για μια εφαρμογή εξετάζονται πέραν από τον τύπο του μικροφώνου και μια σειρά από άλλα ηλεκτρακουστικά χαρακτηριστικά.

- **Προενίσχυση (preamplifier)**

Το πλάτος του σήματος που παράγει είναι πολύ μικρό, λόγω του ότι το διάφραγμα του μικροφώνου είναι πολύ μικρό. Δεδομένου της αντίστασης των καλωδίων, του ηλεκτρικού και ηλεκτρομαγνητικού θορύβου, το σήμα αυτό εάν δεν ενισχυθεί θα εξασθενίσει περιορίζοντας δραστικά το δυναμικό εύρος χάνοντας σημαντικό ποσοστό προτού διανύσει την απόσταση μέχρι την επόμενη βαθμίδα. Για το λόγο αυτό επιβάλλεται η ενίσχυση του σήματος με τη χρήση προενισχυτή μικροφώνου.

- **Εξωτερική τάση πόλωσης (Phantom Power)**

Αφορά τα πυκνωτικά μικρόφωνα και είναι η συνεχής τάση (12 – 48V) που χρειάζεται το ηλεκτρικό κύκλωμα του μικροφώνου για να τεθεί σε λειτουργία και να δημιουργηθεί τάση πόλωσης στον πυκνωτή.

- **Ευαισθησία (G) (sensitivity)**

Είναι ο λόγος της τάσης εξόδου ενός μικροφώνου προς μια σταθερή ηχητική στάθμη εισόδου, συνήθως στα 1000Hz. Όσο πιο μεγάλο σήμα εξόδου δίδει ένα μικρόφωνο για δεδομένη ηχητική στάθμη εισόδου τόσο πιο ευαίσθητο είναι το μικρόφωνο αυτό. Η στάθμη ευαισθησίας υπολογίζεται σε dB_{SPL} και δίνεται από τον τύπο:

$$G_E = 20 \log \frac{T_E}{T_0} \quad (77),$$

όπου $T_E = \frac{\tilde{u}}{\tilde{p}}$ η ευαισθησία και u η σωματιδιακή ταχύτητα, p η ηχητική πίεση

T_0 η τιμή αναφοράς της ευαισθησίας (T_0 = 1V / Pa).

- **Κατευθυντικότητα ελεύθερου πεδίου (directional effect)**

Είναι ο δείκτης της μεταβολής της τάσης εξόδου του μικροφώνου ως προς την κατεύθυνση από την οποία προέρχεται ο ήχος για ηχητική έκθεση σε περιβάλλον ελεύθερου πεδίου (χωρίς δηλαδή ανακλάσεις).

- **Κατευθυντικότητα διάχυτου πεδίου (diffuse field directivity)**

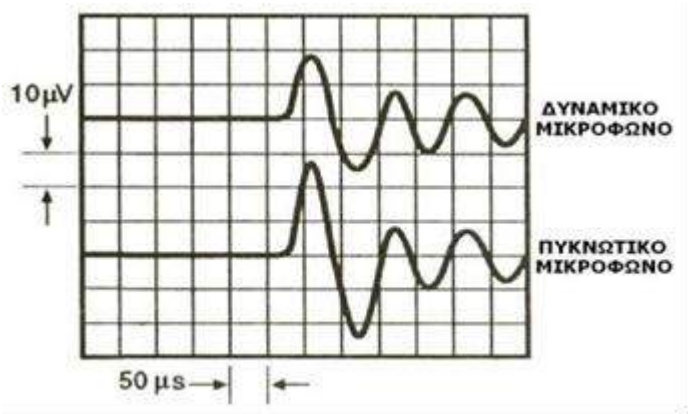
Είναι δείκτες της μεταβολής της τάσης εξόδου του μικροφώνου σε συνάρτηση με την κατεύθυνση του ήχου μεταξύ συνθηκών ελεύθερου και διάχυτου πεδίου (όπου η ηχητική ενέργεια κατευθύνεται προς το διάφραγμα του μικροφώνου με ίδια πιθανότητα από κάθε διεύθυνση του χώρου).

- **Απόκριση συχνοτήτων (frequency response)**

Είναι η συνάρτηση που αντιστοιχεί στην τιμή του σήματος εξόδου του μικροφώνου (εκφρασμένη σε dB) σε κάθε συχνότητα ως προς την τιμή εξόδου που αντιστοιχεί σε κάποια καθορισμένη συχνότητα αναφοράς, όταν για είσοδο υπάρχει ηχητικό πεδίο σταθερής ηχητικής στάθμης ανά συχνότητα. Η συνηθέστερη μορφή απεικόνισης της συχνοτικής απόκρισης είναι υπό τη μορφή διαγράμματος. Το εύρος συχνοτήτων στο οποίο η συχνοτική απόκριση του μικροφώνου είναι επίπεδη ονομάζεται συχνοτική περιοχή λειτουργίας του μικροφώνου.

- **Στιγμιαία απόκριση (transient response)**

Είναι μέτρο της ικανότητας του μικροφώνου να αποκρίνεται σε γρήγορες αλλαγές του πλάτους του ηχητικού κύματος. Καθορίζεται από την ηλεκτρική, μηχανική και ακουστική εμπέδηση του μικροφώνου. Αφορά τόσο στο χρόνο μέγιστης απόκριση μετά την έναρξη όσο και στο χρόνο σίγασης (ηρεμίας) μετά τη διακοπή ενός απειροστού παλμού (συνάρτηση δέλτα). Είναι το βασικό στοιχείο διαφοροποίησης των πυκνωτικών από τα δυναμικά μικρόφωνα. Τα πυκνωτικά μικρόφωνα έχουν ελαφρύτερο διάφραγμα, άρα μικρότερη μηχανική αδράνεια και κατά συνέπεια καλύτερη στιγμιαία απόκριση (Σχ. 41).



Σχήμα 41: Στιγμιαία απόκριση μικροφώνου

- **Εμπέδηση (impedance)**

Είναι η ολική σύνθετη αντίσταση εξόδου του μικροφώνου. Μετριέται σε Ohm, είναι μιγαδικό μέγεθος και μεταβάλλεται ανάλογα με τη συχνότητα.

Ως προς την εμπέδηση υπάρχουν τρεις βασικές κατηγορίες «τάξεις» μικροφώνων:

- Χαμηλής εμπέδησης: $Z < 600\Omega$, με τυπική τιμή τα 200Ω.
- Μεσαίας εμπέδησης: $600\Omega < Z < 10.000\Omega$, με τυπική τιμή τα 2kΩ.
- Υψηλής εμπέδησης: $Z > 10.000\Omega$, με τυπική τιμή τα 25kΩ.

- **Εσωτερική στάθμη θορύβου (internal noise level)**

Είναι η τάση που εμφανίζεται στην έξοδο του μικροφώνου εκφρασμένη σε dBV, στην απουσία ακουστικού σήματος, είναι δηλαδή μέτρο του θερμικού θορύβου που δημιουργείται από τα στοιχεία του κυκλώματος και εξαρτάται από τον τύπο του μικροφώνου, την εσωτερική του αντίσταση και τη θερμοκρασία.

- **Υπερφόρτωση (overload sound pressure)**

Είναι η μεγαλύτερη ηχητική πίεση που μπορεί να καταγράψει το μικρόφωνο πέραν της οποίας η παραμόρφωση υπερβαίνει κάποια καθορισμένα όρια. Στα περισσότερα μικρόφωνα το όριο αυτό είναι μεταξύ των 120 και 140dB (στη θέση του διαφράγματος).

- **Δυναμικό εύρος (dynamic range)**

Είναι η διαφορά ανάμεσα στη μέγιστη τάση εξόδου που δίδει το μικρόφωνο χωρίς παραμόρφωση και χωρίς κίνδυνο καταστροφής και στην εσωτερική στάθμη θορύβου του.

5 Σχεδιασμός

Ο σχεδιασμός και η ανάλυση του συστήματος πραγματοποιήθηκε με χρήση της Unified Modeling Language (UML). Η UML αποτελεί μια γλώσσα αντικειμενοστρεφούς μοντελοποίησης ενός πληροφοριακού συστήματος. Η μοντελοποίηση ενός συστήματος παρέχει τη δυνατότητα του πειραματισμού με διαφορετικές λύσεις ή προσεγγίσεις για το ίδιο πρόβλημα. Επίσης, καθιστά εφικτή τη δυνατότητα ανάλυσης, σχεδιασμού καταγραφής και παρακολούθησης της προόδου ενός έργου πληροφορικής, προσφέροντας μια κοινή γλώσσα για την επικοινωνία όσων εμπλέκονται στην κατασκευή του συστήματος. Είναι σαφές, ότι χωρίς ένα μοντέλο δεν είναι δυνατόν να προσεγγίσει κανείς την πολυπλοκότητα των συγχρόνων πληροφοριακών συστημάτων. Συγκεκριμένα, ο σχεδιασμός βασίστηκε στην μεθοδολογία αντικειμενοστρεφούς ανάπτυξης ICONIX. Προτιμήθηκε η μεθοδολογία ICONIX διότι είναι απλούστερη και συντομότερη από την πολύ εκτενή ενοποιημένη προσέγγιση Unified Process (UP). Η ICONIX χρησιμοποιεί την UML σαν γλώσσα έκφρασης των απαιτήσεων και των προδιαγραφών, του υπό σχεδιασμό λογισμικού και στηρίζεται πάρα πολύ στις περιπτώσεις χρήσεις.

5.1 Απαιτήσεις Υψηλού Επιπέδου

Το λογισμικό που αναπτύχθηκε αφορά σε ένα σύστημα ηχομέτρησης, το οποίο θα λειτουργεί σε φορητή πλατφόρμα κινητής τηλεφωνίας. Συνεπώς, το σύστημα θα πρέπει να παρέχει τις βασικές λειτουργίες ενός τυπικού ηχομέτρου.

Συγκεκριμένα ο χρήστης του συστήματος θα έχει την δυνατότητα να επιλέξει μεταξύ των δύο βασικών τρόπων καταγραφής ενός τυπικού ηχομέτρου που αντιστοιχούν στις λειτουργίες Fast και Slow. Οι λειτουργίες αυτές αναφέρονται στο μέγεθος του χρονικού παραθύρου, βάσει του οποίου θα πραγματοποιούνται οι υπολογισμοί του στιγμιαίου επιπέδου της ηχητικής πίεσης για το καταγραφόμενο ηχητικό σήμα. Η λειτουργία Slow σχετίζεται με ένα χρονικό παράθυρο το μέγεθος του οποίου είναι 8πλάσιο του αντίστοιχου χρονικού παραθύρου για την λειτουργία Fast.

Η συχνότητα δειγματοληψίας του ηχητικού σήματος θα είναι 44100 δείγματα το δευτερόλεπτο και αντιστοιχεί αυτήν που χρησιμοποιείται για την δημιουργία ενός μουσικού CD.

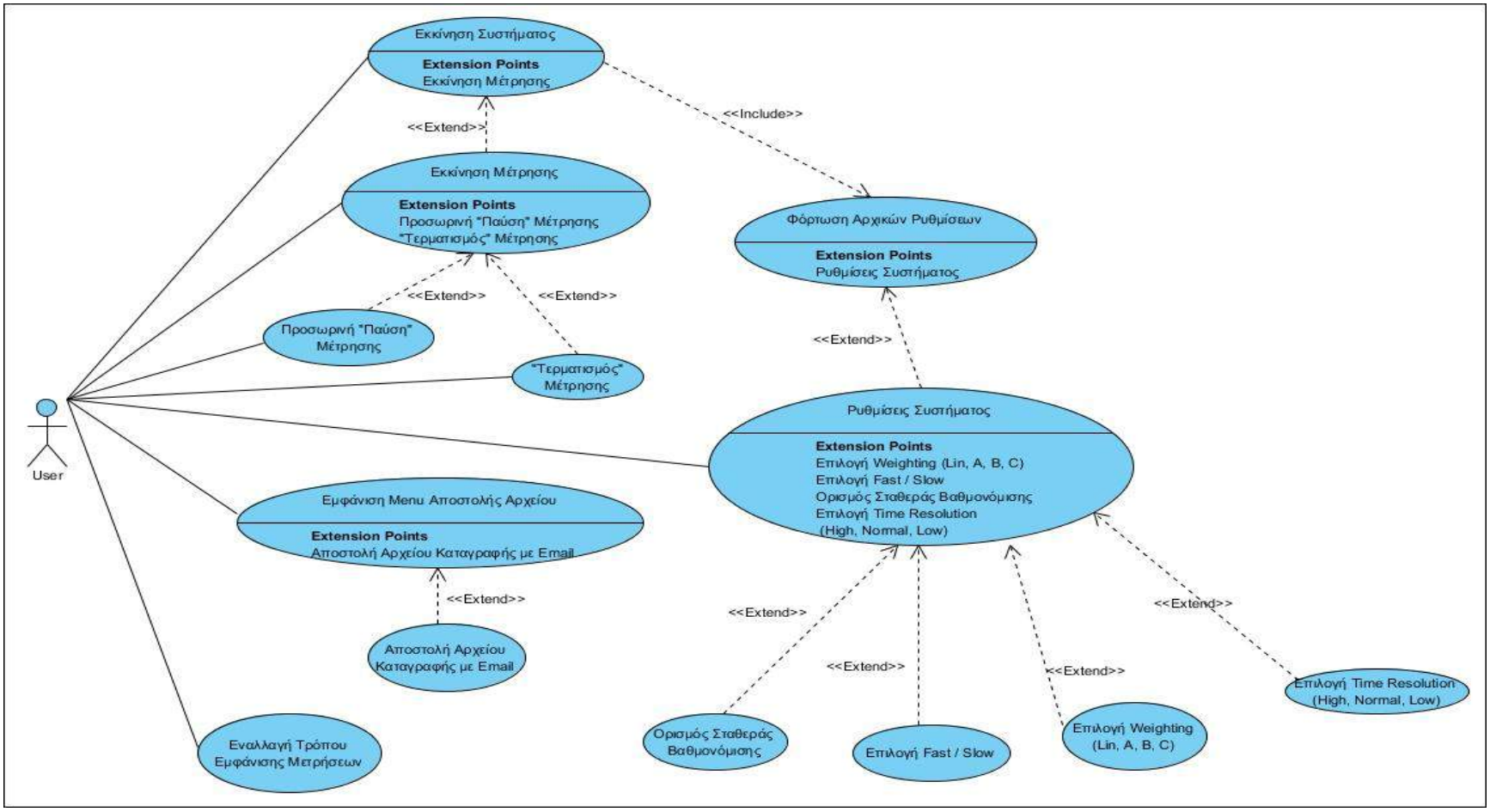
Το σύστημα θα μετράει την στιγμιαία ηχοστάθμη σε dB και θα υπολογίζει τη μέση, την ελάχιστη και τη μέγιστη τιμή της. Οι παραπάνω μετρήσεις θα είναι δυνατόν να πραγματοποιηθούν επιπλέον και με τη χρήση κατάλληλων φίλτρων στάθμισης που είναι τα

A, B και C, οι μετρήσεις των οποίων θα γίνονται σε dB(A), dB(B) και dB(C) αντίστοιχα. Ο χρήστης θα πληροφορείται για τις παραπάνω μετρήσεις μέσα από ένα σύνολο διαφορετικών επιλογών εμφάνισης των αποτελεσμάτων, όπως για παράδειγμα ψηφιακή / αναλογική ένδειξη, λεκτικά τυπικών επιπέδων θορύβου και γραφήματα επιπέδου ηχητικής πίεσης συναρτήσει του χρόνου καθώς και για τις τρέχουσες ρυθμίσεις του συστήματος.

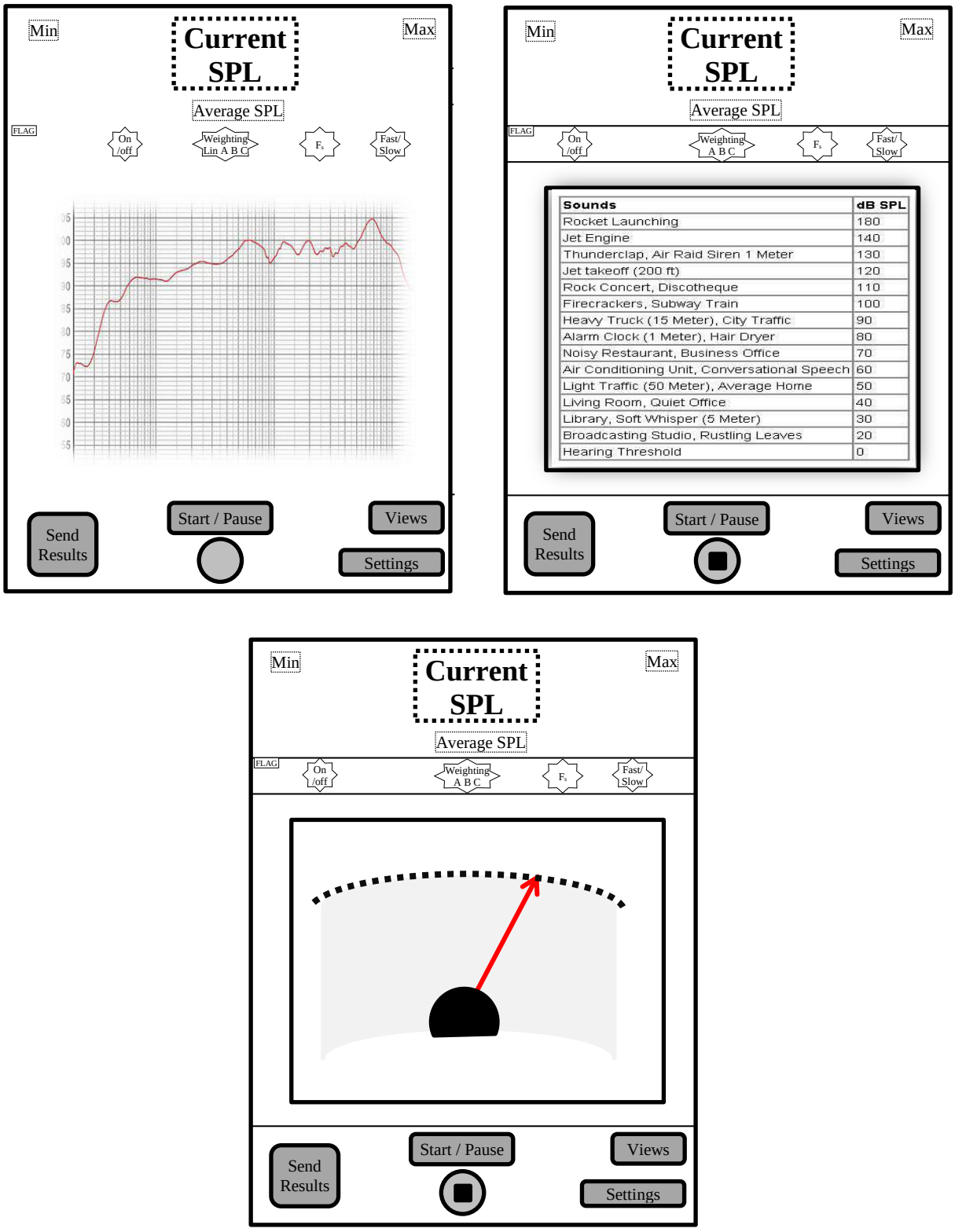
Επιπρόσθετα το σύστημα θα παρέχει τη δυνατότητα στον χρήστη να ξεκινά, να «παγώνει» και να σταματά τη διαδικασία της μέτρησης κατά την διάρκεια της οποίας θα υπάρχει ένδειξη η οποία θα σηματοδοτεί ότι το σύστημα βρίσκεται σε λειτουργία. Στις παρεχόμενες υπηρεσίες του συστήματος θα είναι η βαθμονόμησή του και η εξαγωγή αρχείου αποτελεσμάτων με τα καταγραφόμενα στοιχεία της μέτρησης, το οποίο θα είναι δυνατό να αποσταλεί μέσω ηλεκτρονικού ταχυδρομείου. Τέλος ο χρήστης θα μπορεί να επαναφέρει το σύστημα στις αρχικές του ρυθμίσεις.

5.2 Μοντέλο Περιπτώσεων Χρήσης (Use Case Model)

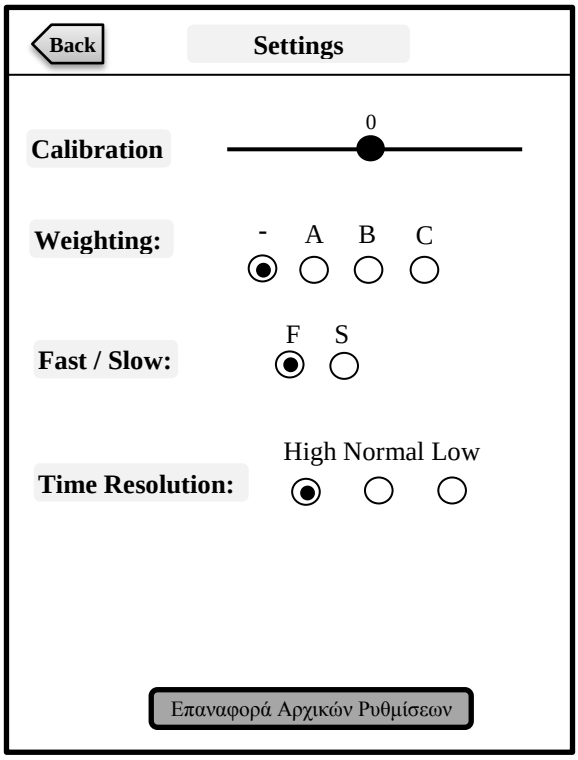
5.2.1 Διάγραμμα Περιπτώσεων Χρήσης (Use Case Diagram)



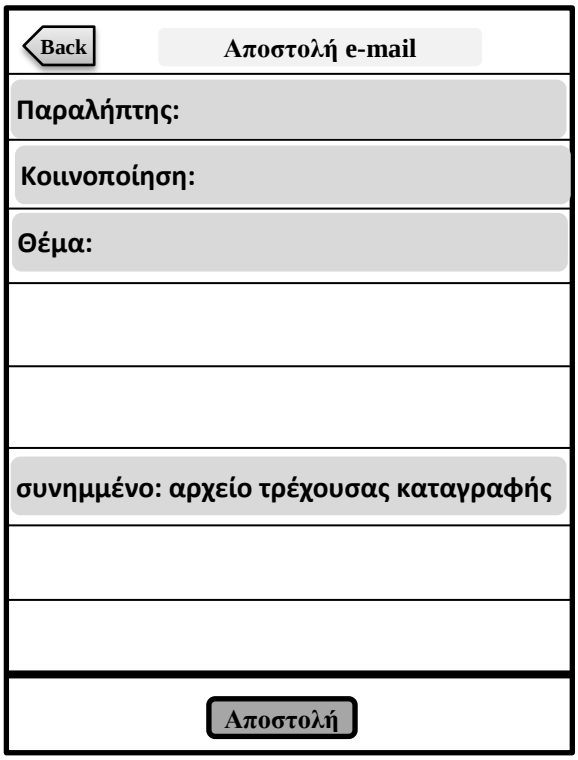
5.2.2 Ενδεικτικές Οθόνες του Συστήματος



Σχήμα 42: Εναλλακτικές οθόνες παρουσίασης αποτελεσμάτων



Σχήμα 44: Οθόνη Επιλογών Συστήματος



Σχήμα 43: Οθόνη Αποστολής Αρχείου Καταγραφής

5.2.3 Τεκμηρίωση Περιπτώσεων Χρήσης

Τίτλος: Εκκίνηση μέτρησης

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης ξεκινάει την εφαρμογή μέτρησης ηχητικής πίεσης.

Προϋποθέσεις:

- Φόρτωση προκαθορισμένων παραμέτρων.

Μετασυνθήκες:

- Το σύστημα καταγράφει τα δείγματα του ηχητικού σήματος.

Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί της εκκίνησης μετρήσεων που βρίσκεται στην κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Το σύστημα πληροφορεί τον χρήστη για τις τιμές της στιγμιαίας, μέσης, μέγιστης και ελάχιστης ηχητικής στάθμης.

Εναλλακτικές Ροές:

Εναλλακτική ροή Α: Δεν λειτουργεί το μικρόφωνο.

2α. Εμφάνιση κατάλληλου μηνύματος που αναφέρει ότι δεν λειτουργεί σωστά το μικρόφωνο.

2β. Ο έλεγχος επιστρέφει στο βήμα 1.

Τίτλος: Προσωρινή παύση μέτρησης

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης "παγώνει" τις μετρήσεις του συστήματος.

Προϋποθέσεις:

- Ο χρήστης έχει θέσει σε λειτουργία το σύστημα μέτρησης.

Μετασυνθήκες:

- Δεν υπάρχουν.

Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί της προσωρινής παύσης μετρήσεων που βρίσκεται στην κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Το σύστημα παραμένει αδρανές και πληροφορεί τον χρήστη για τις τελευταίες ενδείξεις.

Εναλλακτικές Ροές:

- Δεν υπάρχουν.

Τίτλος: Τερματισμός μέτρησης

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης σταματά τις μετρήσεις.

Προϋποθέσεις:

- Ο χρήστης έχει θέσει σε λειτουργία το σύστημα μέτρησης.

Μετασυνθήκες:

- Δεν υπάρχουν.

Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί τερματισμού των μετρήσεων που βρίσκεται στην κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Το σύστημα σταματά τις μετρήσεις και την αντίστοιχη εμφάνιση των αποτελεσμάτων.
3. Το σύστημα δημιουργεί αρχείο καταγραφής όπου καταχωρεί τις τιμές των μετρήσεων που πραγματοποιήθηκαν, καθώς και τις αντίστοιχες ρυθμίσεις.

Εναλλακτικές Ροές:

Εναλλακτική ροή Α: Είναι επιλεγμένο το κουμπί αποθήκευσης από την οθόνη ρυθμίσεων.

- Δεν υπάρχουν.

Τίτλος: Ορισμός ρυθμίσεων συστήματος μέτρησης

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης μπορεί να καθορίσει τις ρυθμίσεις του συστήματος.

Προϋποθέσεις:

- Το σύστημα σταματάει την καταγραφή.

Μετασυνθήκες:

- Το σύστημα αποθηκεύει τις επιλεγμένες ρυθμίσεις του χρήστη.

Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί ρυθμίσεων που βρίσκεται στην κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Εμφανίζεται η οθόνη ρυθμίσεων.
3. Ο χρήστης μπορεί να επιλέξει μεταξύ των λειτουργιών Fast / Slow.
4. Ο χρήστης μπορεί να επιλέξει το Time Resolution μεταξύ των τιμών High, Normal, Low.
5. Ο χρήστης μπορεί να επιλέξει μεταξύ του τρόπου στάθμισης Lin, A, B, C.
6. Ο χρήστης μπορεί να επαναφέρει τις προκαθορισμένες ρυθμίσεις.
7. Ο χρήστης μπορεί να διορθώσει την σταθερά βαθμονόμησης.

8. Ο χρήστης επιστρέφει στην κεντρική οθόνη.

Εναλλακτικές Ροές:

- Δεν υπάρχουν.

Τίτλος: Αποστολή αρχείου καταγραφής

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης μπορεί να στείλει με email το τρέχον αρχείο καταγραφής.

Προϋποθέσεις:

- Το σύστημα έχει διακόψει/σταματήσει την καταγραφή.

Μετασυνθήκες:

- Δεν υπάρχουν.

Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί αποστολής αρχείου καταγραφής που βρίσκεται στην κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Ο χρήστης αποστέλλει το αρχείο.
3. Ο χρήστης επιστρέφει στην κεντρική οθόνη.

Εναλλακτικές Ροές:

Εναλλακτική ροή Α: Δεν υπάρχει πρόσβαση στο email.

2α. Εμφάνιση κατάλληλου μηνύματος που αναφέρει ότι δεν υπάρχει πρόσβαση στο email.

2β. Ο έλεγχος επιστρέφει στο βήμα 2.

Τίτλος: Εναλλαγή τρόπου εμφάνισης μετρήσεων

Χρήστης: Κύριος Χρήστης.

Περιγραφή: Ο χρήστης μπορεί να εναλλάσσεται στις οθόνες εμφάνισης των μετρήσεων της στιγμιαίας ηχητικής πίεσης.

Προϋποθέσεις:

- Δεν υπάρχουν.

Μετασυνθήκες:

- Δεν υπάρχουν.

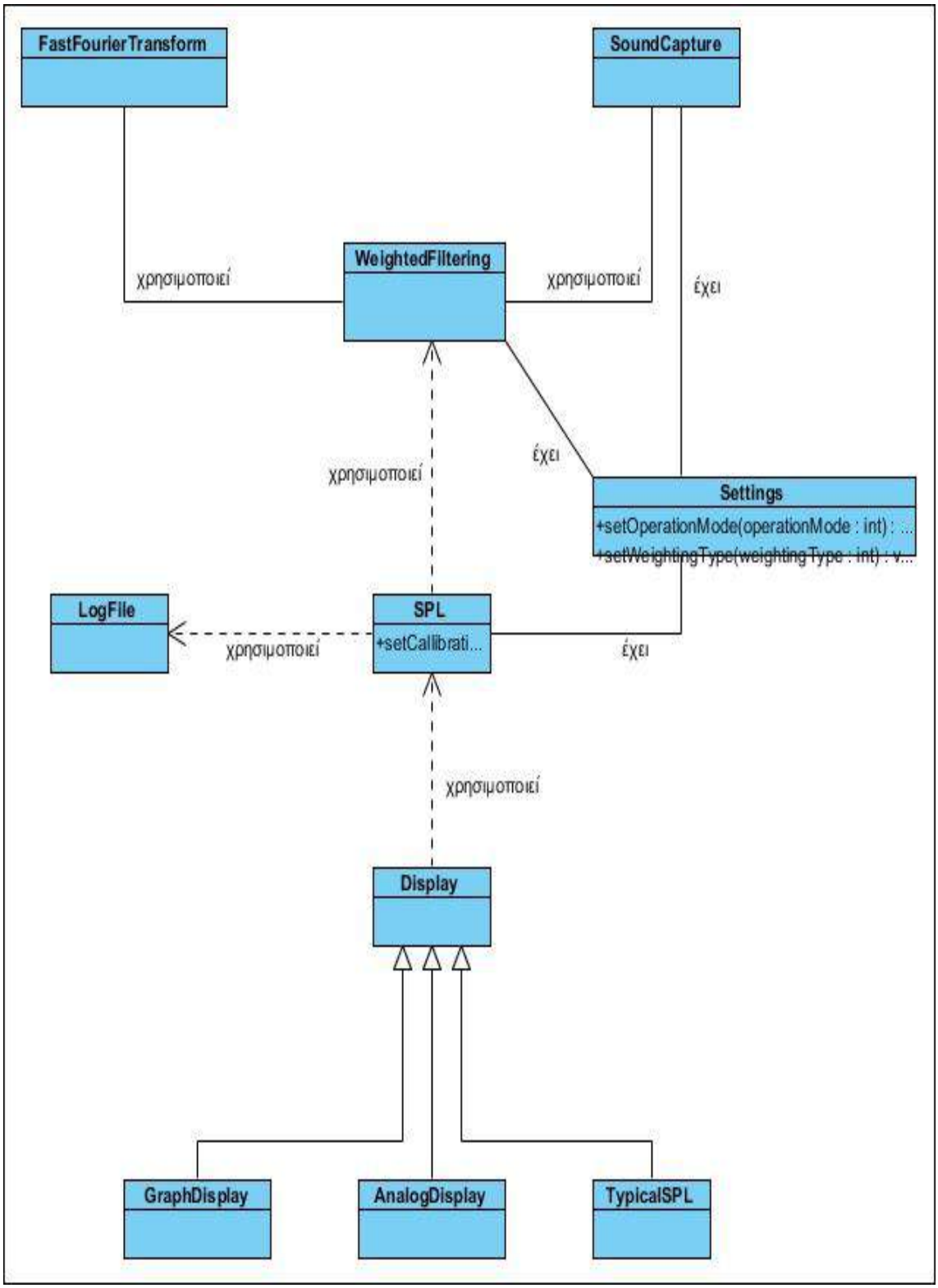
Βασική Ροή:

1. Ο χρήστης επιλέγει το κουμπί εναλλαγής τρόπου εμφάνισης των μετρήσεων από την κεντρική οθόνη εφαρμογής με τις επιλεγμένες ρυθμίσεις.
2. Το σύστημα εμφανίζει την αντίστοιχη οθόνη παρουσίασης των αποτελεσμάτων.

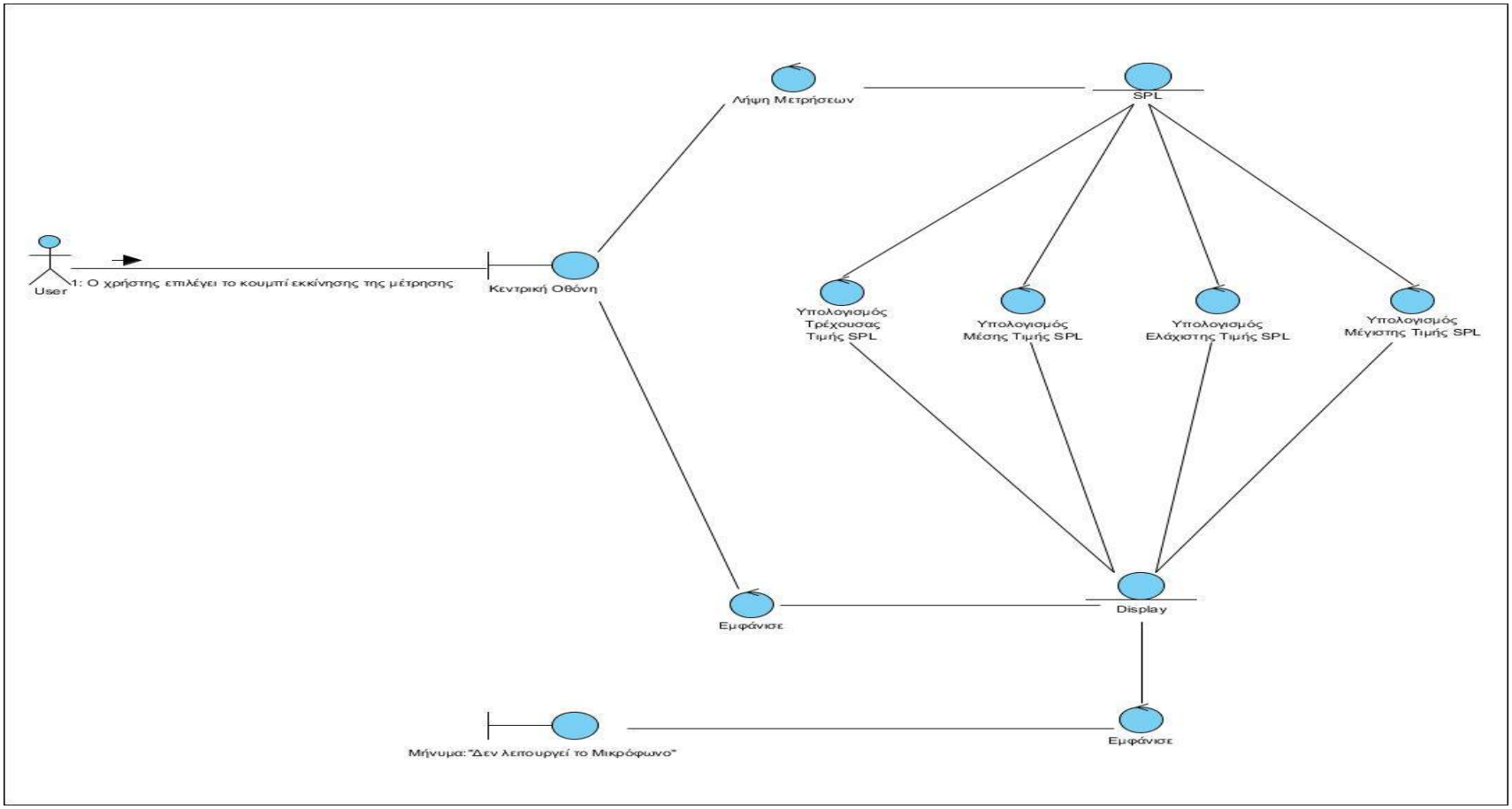
Εναλλακτικές Ροές:

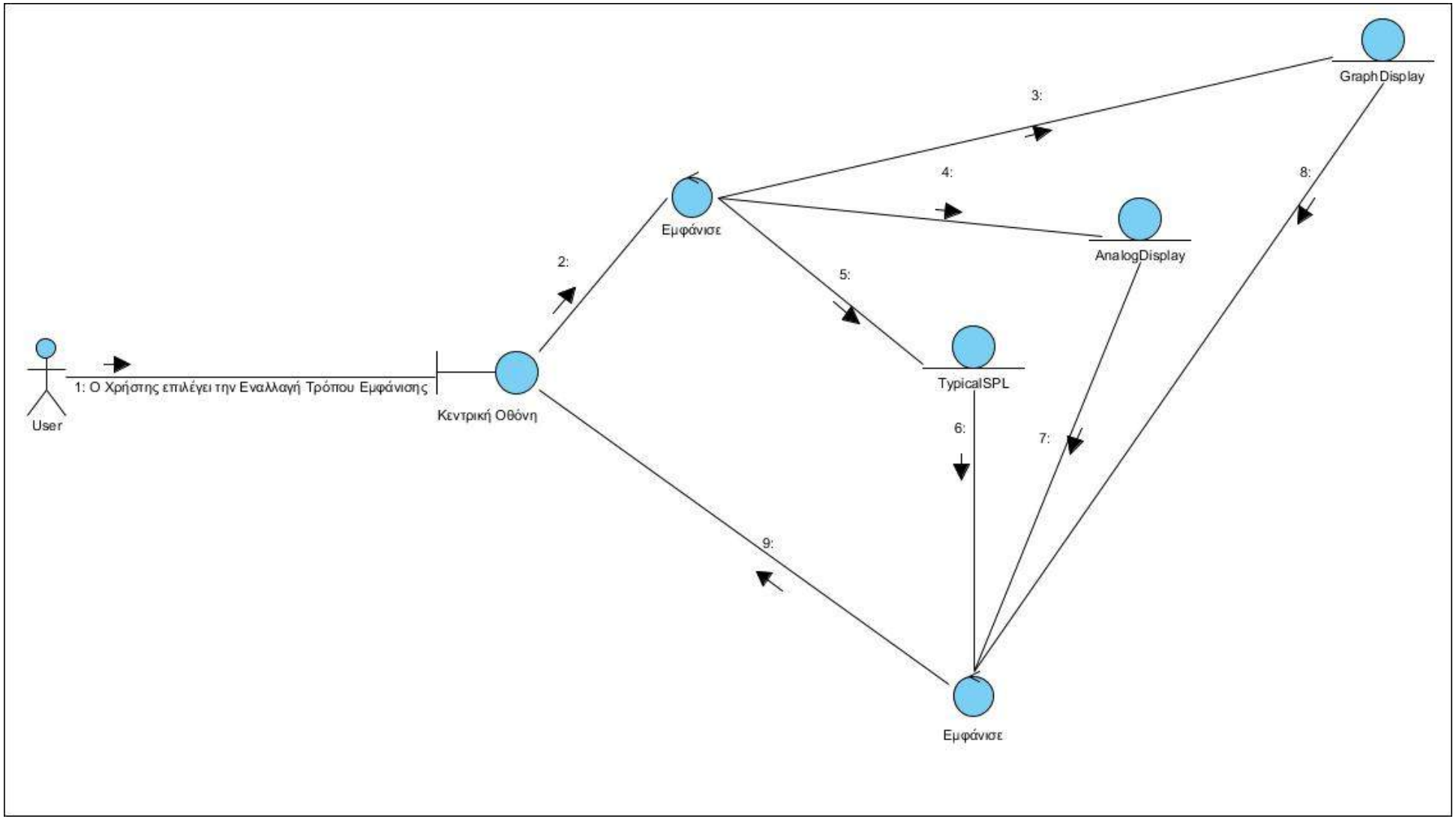
- Δεν υπάρχουν.

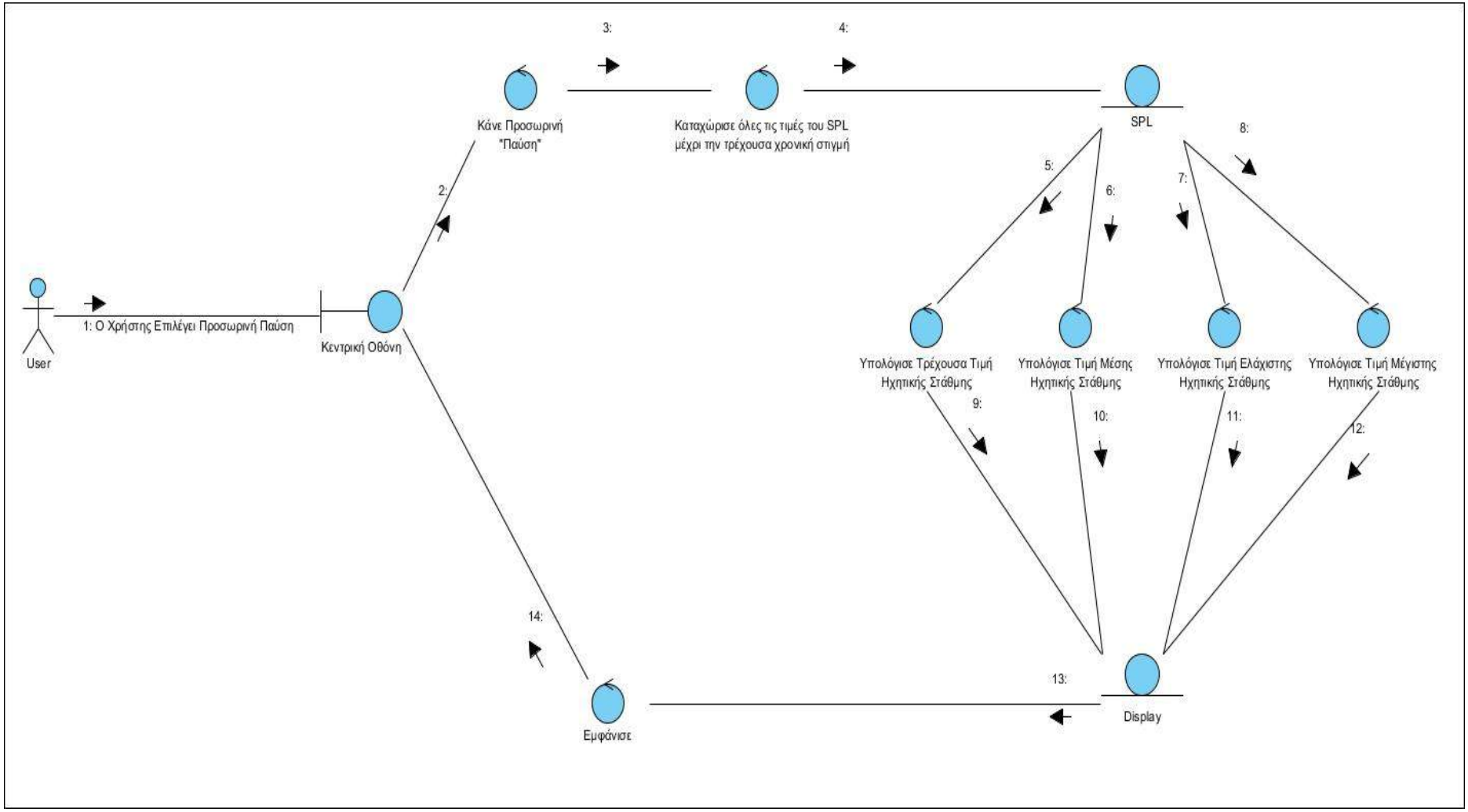
5.3 Διάγραμμα Πεδίου Προβλήματος (Αρχικό)

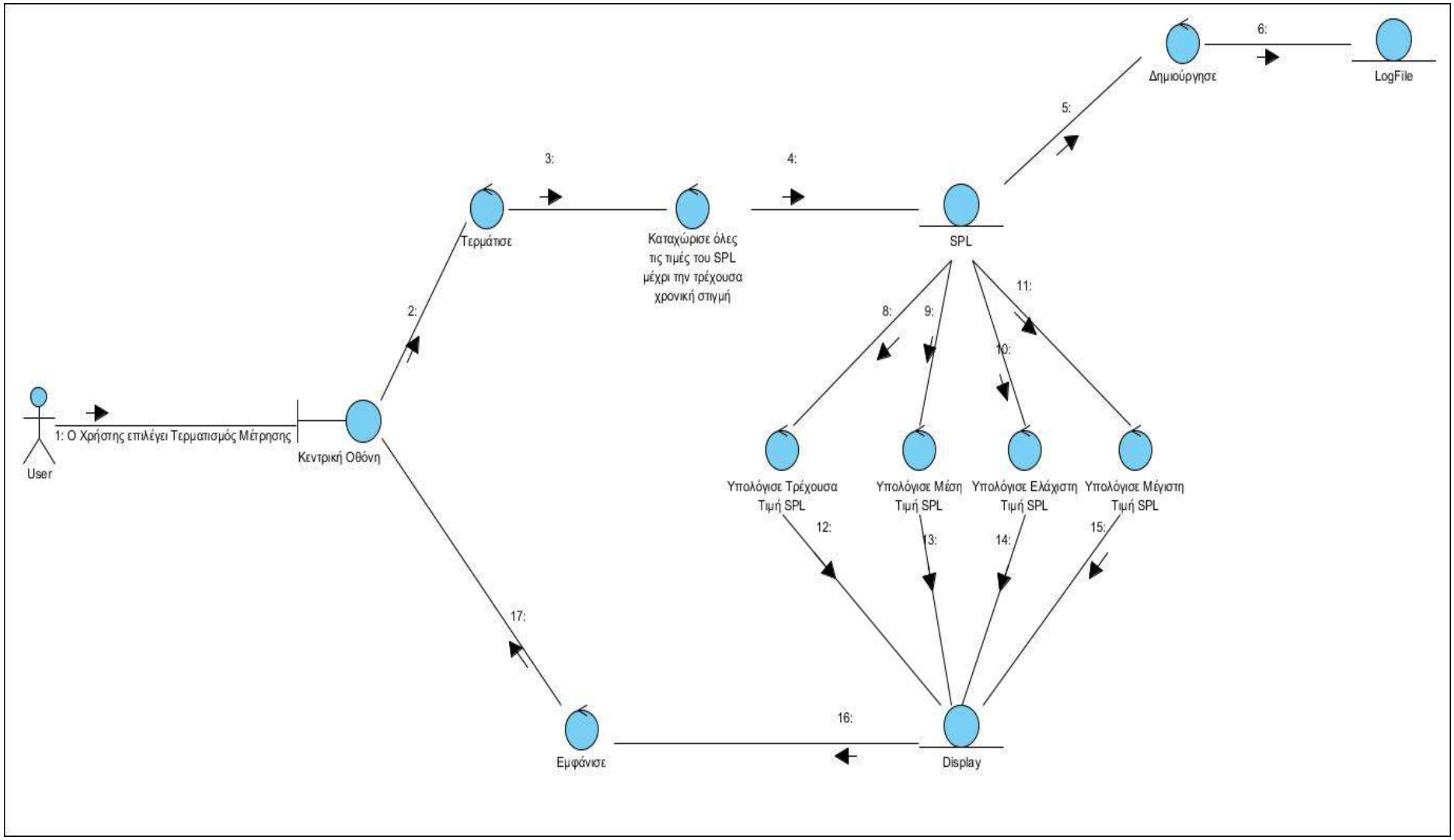


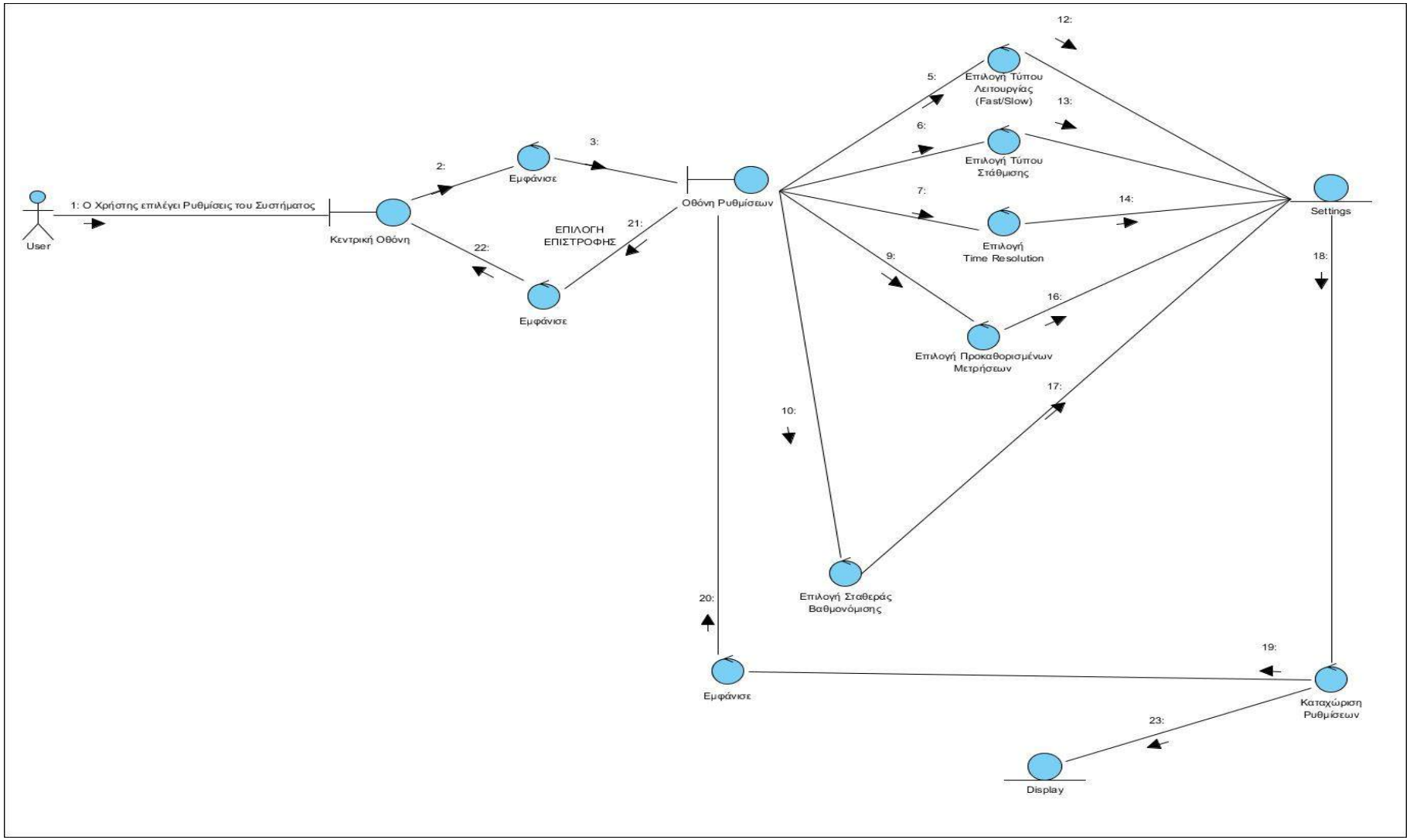
5.4 Διαγράμματα Ευρωστίας Συστήματος

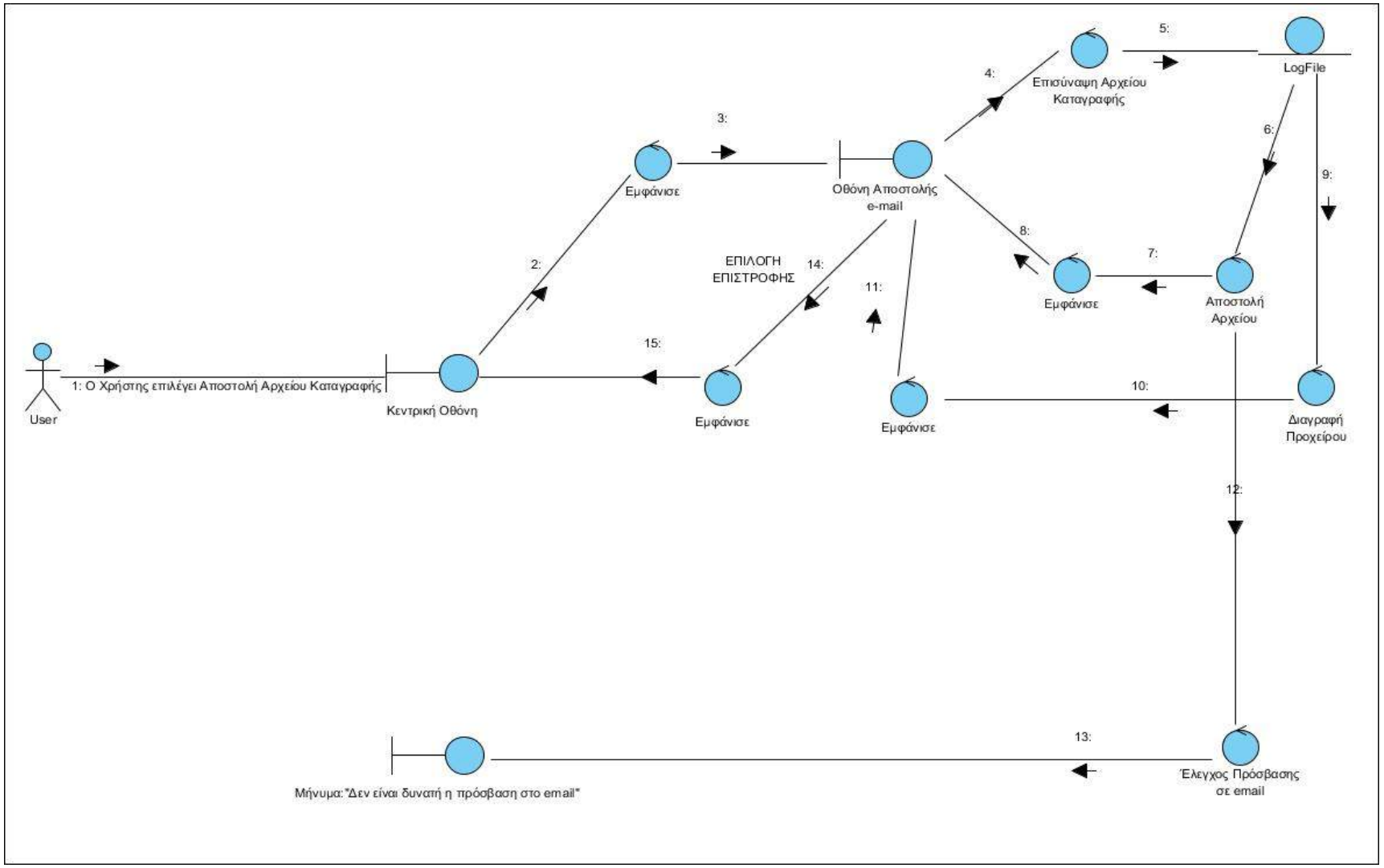



User
1: Ο Χρήστης επιλέγει την Εναλλαγή Τρόπου Εμφάνισης
Κεντρική Οθόνη
2:
Εμφάνισε
3:
GraphDisplay
4:
AnalogDisplay
5:
TypicalSPL
6:
7:
8:
9:
Εμφάνισε

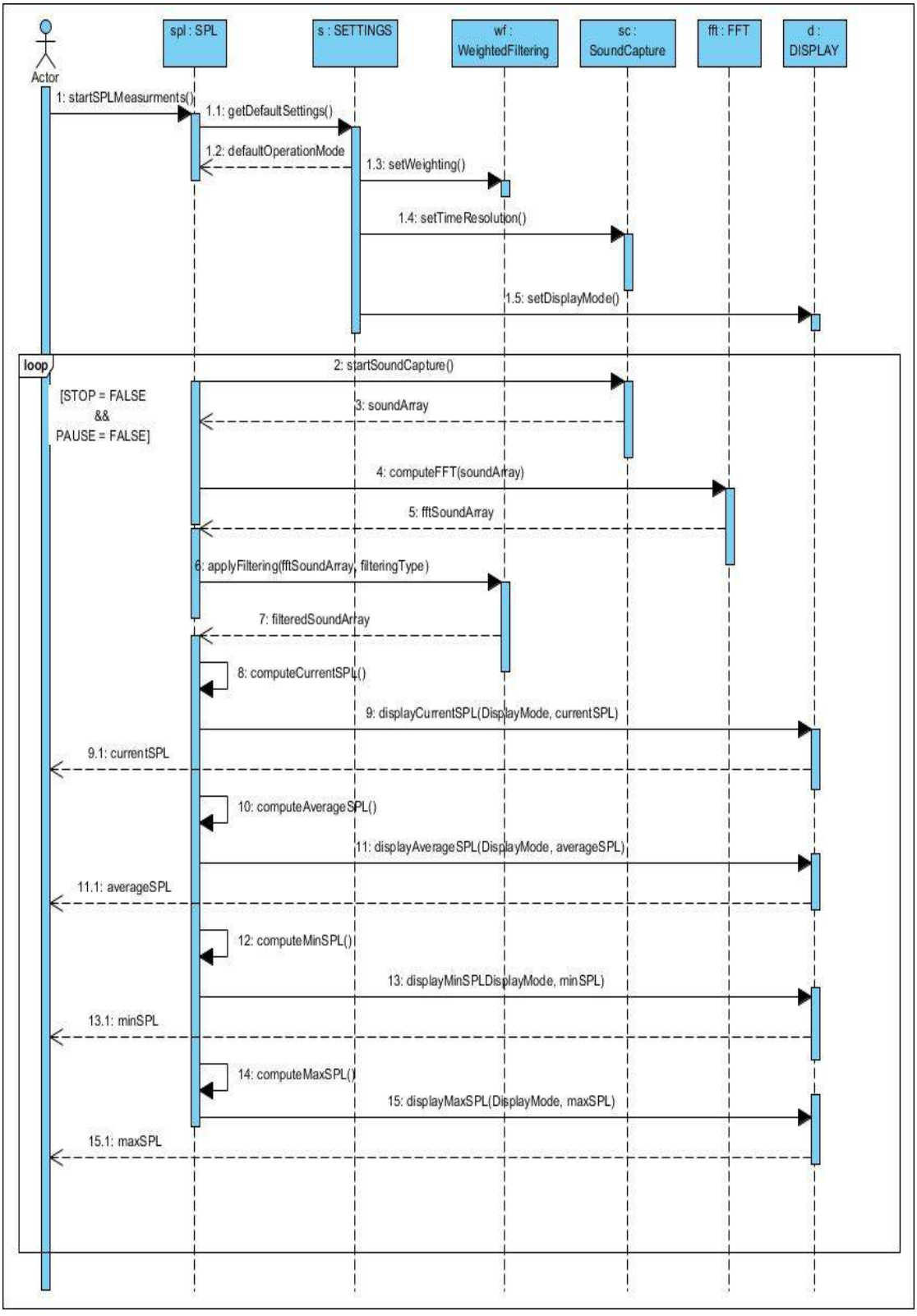

3:
4:
Κάνε Προσωρινή "Παύση"
Καταχώρισε όλες τις τιμές του SPL μέχρι την τρέχουσα χρονική στιγμή
SPL
8:
5:
6:
7:
2:
User
1: Ο Χρήστης Επιλέγει Προσωρινή Παύση
Κεντρική Οθόνη
Υπολόγισε Τρέχουσα Τιμή Ηχητικής Στάθμης
Υπολόγισε Τιμή Μέσης Ηχητικής Στάθμης
Υπολόγισε Τιμή Ελάχιστης Ηχητικής Στάθμης
Υπολόγισε Τιμή Μέγιστης Ηχητικής Στάθμης
9:
10:
11:
12:
14:
13:
Εμφάνισε
Display

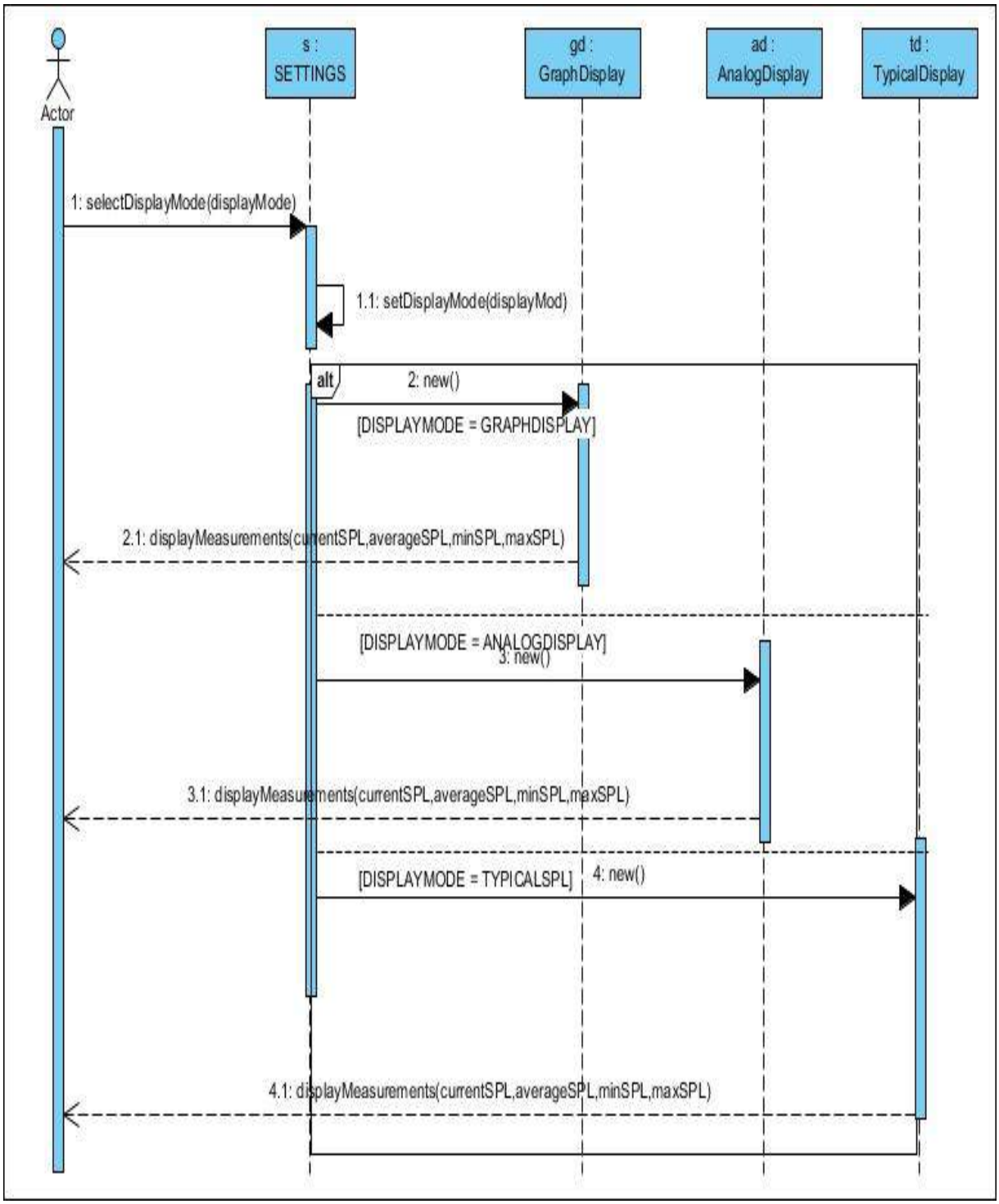

User
1: Ο Χρήστης επιλέγει Τερματισμός Μέτρησης
Κεντρική Οθόνη
2:
Τερμάτισε
3:
Καταχώρισε όλες τις τιμές του SPL μέχρι την τρέχουσα χρονική στιγμή
4:
SPL
5:
Δημιούργησε
6:
LogFile
8:
9:
10:
11:
Υπολόγισε Τρέχουσα Τιμή SPL
Υπολόγισε Μέση Τιμή SPL
Υπολόγισε Ελάχιστη Τιμή SPL
Υπολόγισε Μέγιστη Τιμή SPL
12:
13:
14:
15:
16:
Display
17:
Εμφάνισε

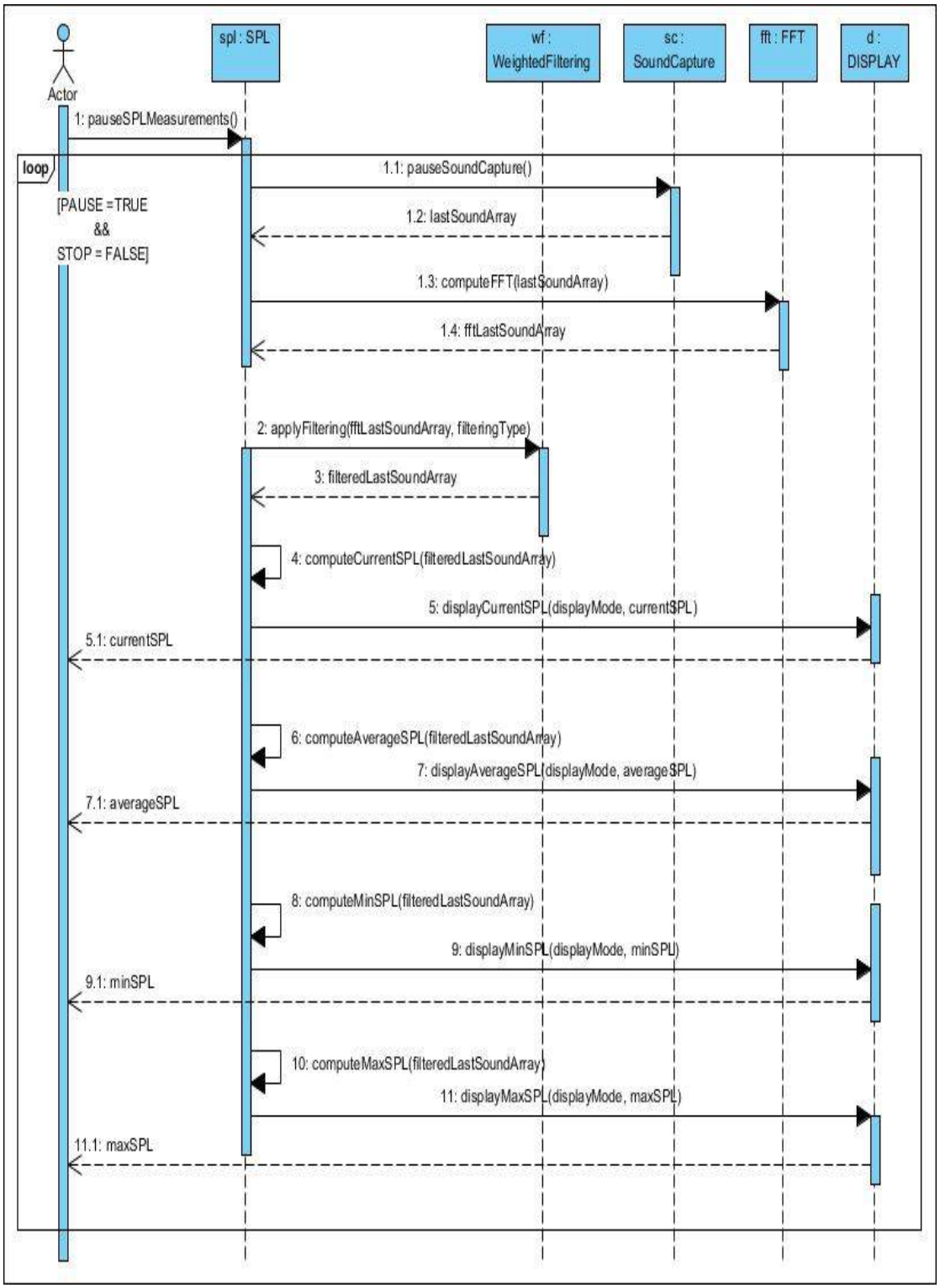

User
1: Ο Χρήστης επιλέγει Ρυθμίσεις του Συστήματος
Κεντρική Οθόνη
2:
Εμφάνισε
3:
Οθόνη Ρυθμίσεων
ΕΠΙΛΟΓΗ ΕΠΙΣΤΡΟΦΗΣ
21:
22:
Εμφάνισε
5:
Επιλογή Τύπου Λειτουργίας (Fast/Slow)
12:
13:
6:
Επιλογή Τύπου Στάθμισης
7:
Επιλογή Time Resolution
14:
9:
Επιλογή Προκαθορισμένων Μετρήσεων
16:
Settings
18:
10:
Επιλογή Σταθεράς Βαθμονόμισης
17:
20:
Εμφάνισε
19:
Καταχώριση Ρυθμίσεων
23:
Display

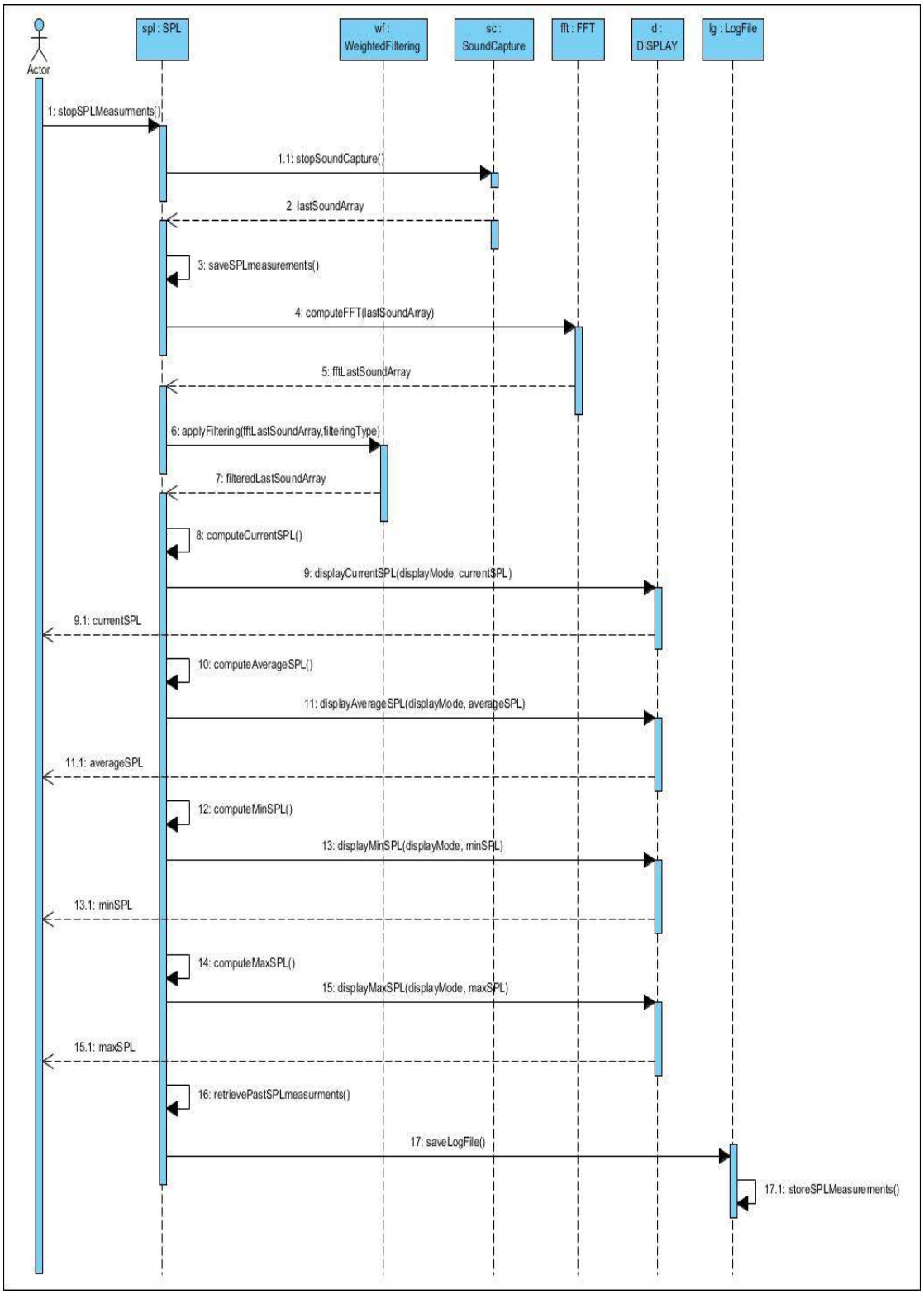

User
1: Ο Χρήστης επιλέγει Αποστολή Αρχείου Καταγραφής
Κεντρική Οθόνη
2:
Εμφάνισε
3:
Οθόνη Αποστολής e-mail
4:
Επισύναψη Αρχείου Καταγραφής
5:
LogFile
6:
9:
Αποστολή Αρχείου
7:
Εμφάνισε
8:
ΕΠΙΛΟΓΗ ΕΠΙΣΤΡΟΦΗΣ
14:
11:
15:
Εμφάνισε
Εμφάνισε
10:
Διαγραφή Προχείρου
12:
13:
Έλεγχος Πρόσβασης σε email
Μήνυμα: "Δεν είναι δυνατή η πρόσβαση στο email"

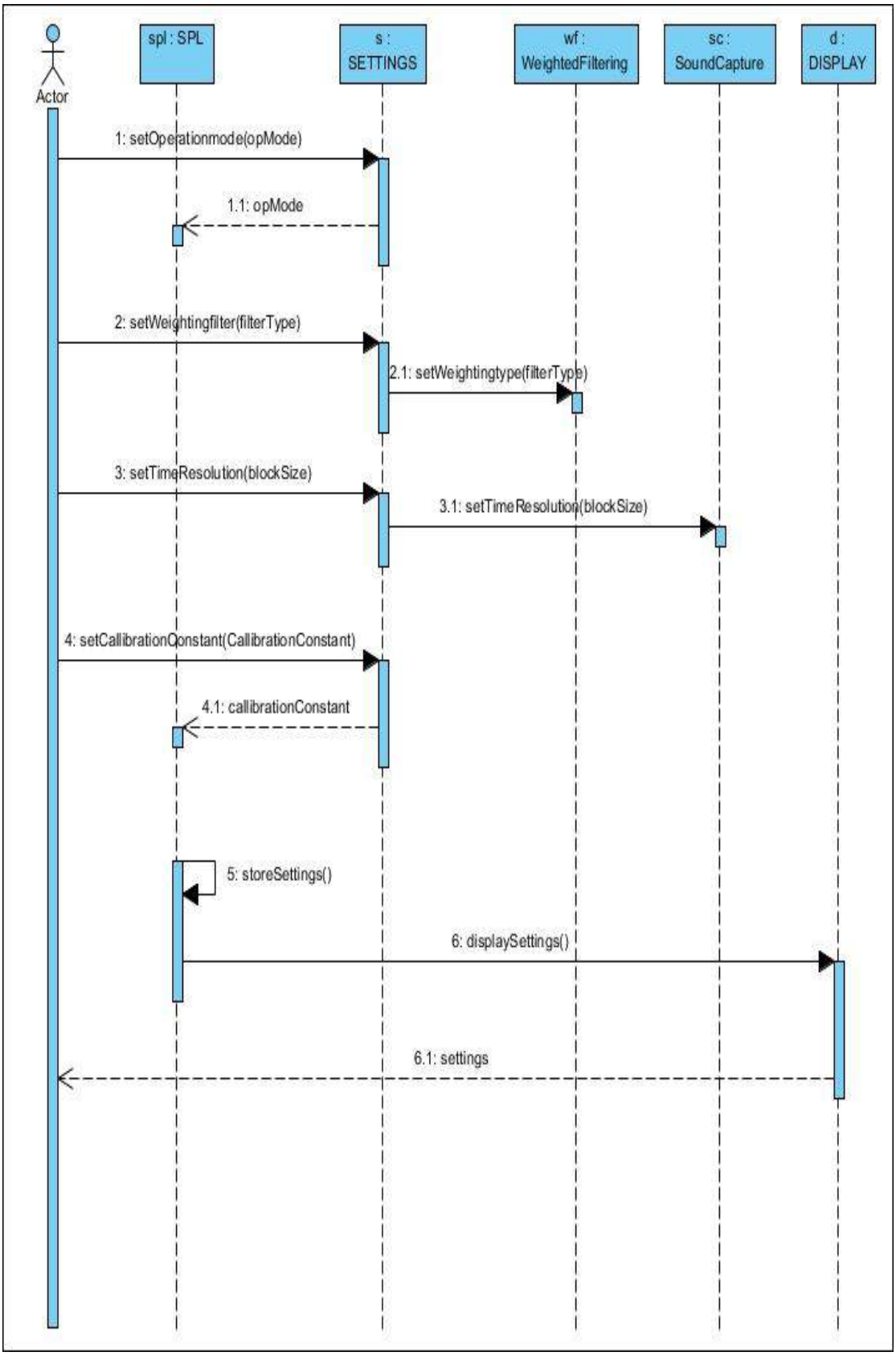
5.5 Διάγραμμα Ακολουθίας Συστήματος

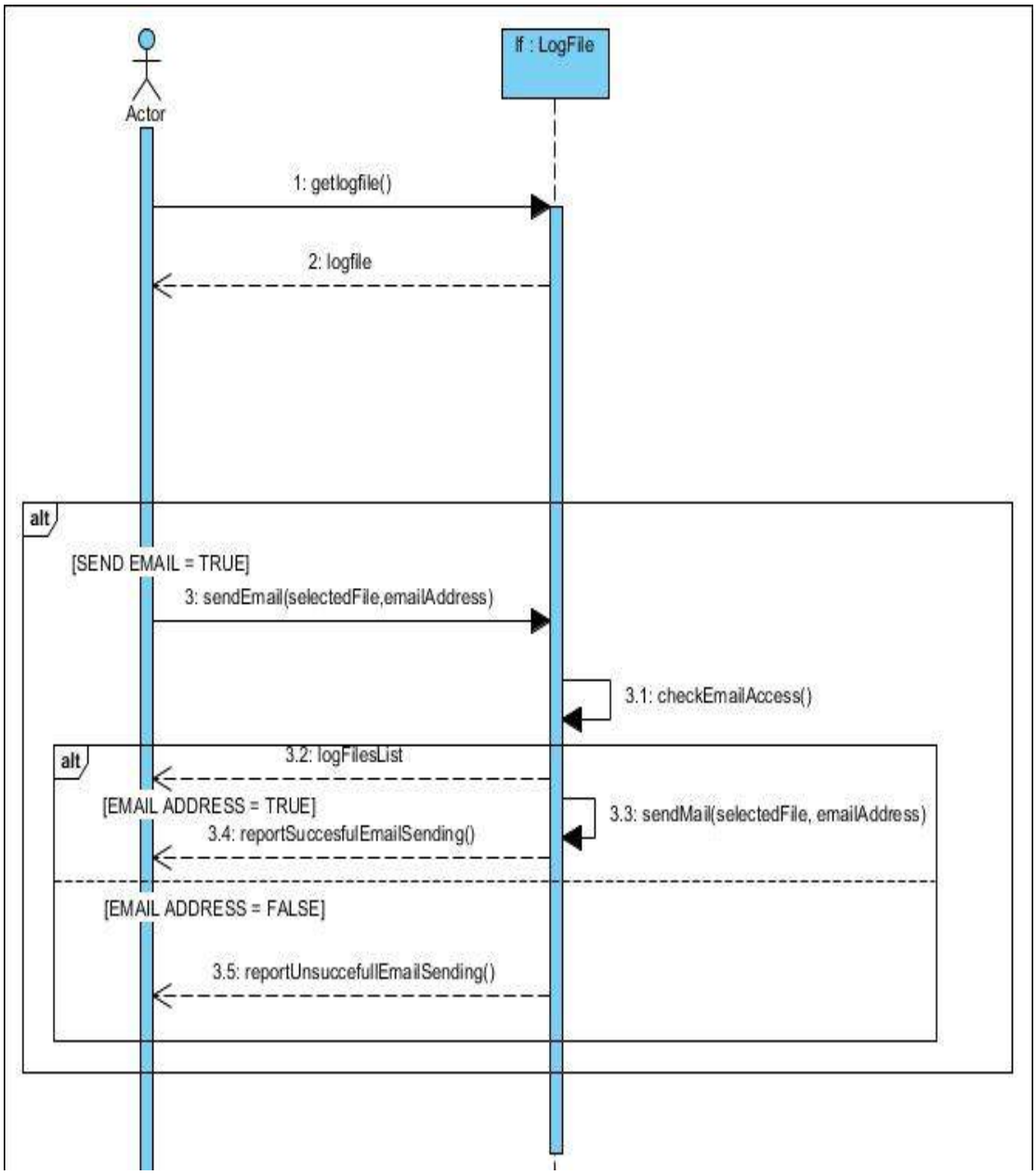



Actor
s : SETTINGS
gd : GraphDisplay
ad : AnalogDisplay
td : TypicalDisplay
1: selectDisplayMode(displayMode)
1.1: setDisplayMode(displayMod)
alt
2: new()
[DISPLAYMODE = GRAPHDISPLAY]
2.1: displayMeasurements(currentSPL,averageSPL,minSPL,maxSPL)
[DISPLAYMODE = ANALOGDISPLAY]
3: new()
3.1: displayMeasurements(currentSPL,averageSPL,minSPL,maxSPL)
[DISPLAYMODE = TYPICALSPL]
4: new()
4.1: displayMeasurements(currentSPL,averageSPL,minSPL,maxSPL)


Actor
spl : SPL
wf : WeightedFiltering
sc : SoundCapture
fft : FFT
d : DISPLAY
1: pauseSPLMeasurements()
loop
[PAUSE = TRUE && STOP = FALSE]
1.1: pauseSoundCapture()
1.2: lastSoundArray
1.3: computeFFT(lastSoundArray)
1.4: fftLastSoundArray
2: applyFiltering(fftLastSoundArray, filteringType)
3: filteredLastSoundArray
4: computeCurrentSPL(filteredLastSoundArray)
5: displayCurrentSPL(displayMode, currentSPL)
5.1: currentSPL
6: computeAverageSPL(filteredLastSoundArray)
7: displayAverageSPL(displayMode, averageSPL)
7.1: averageSPL
8: computeMinSPL(filteredLastSoundArray)
9: displayMinSPL(displayMode, minSPL)
9.1: minSPL
10: computeMaxSPL(filteredLastSoundArray)
11: displayMaxSPL(displayMode, maxSPL)
11.1: maxSPL

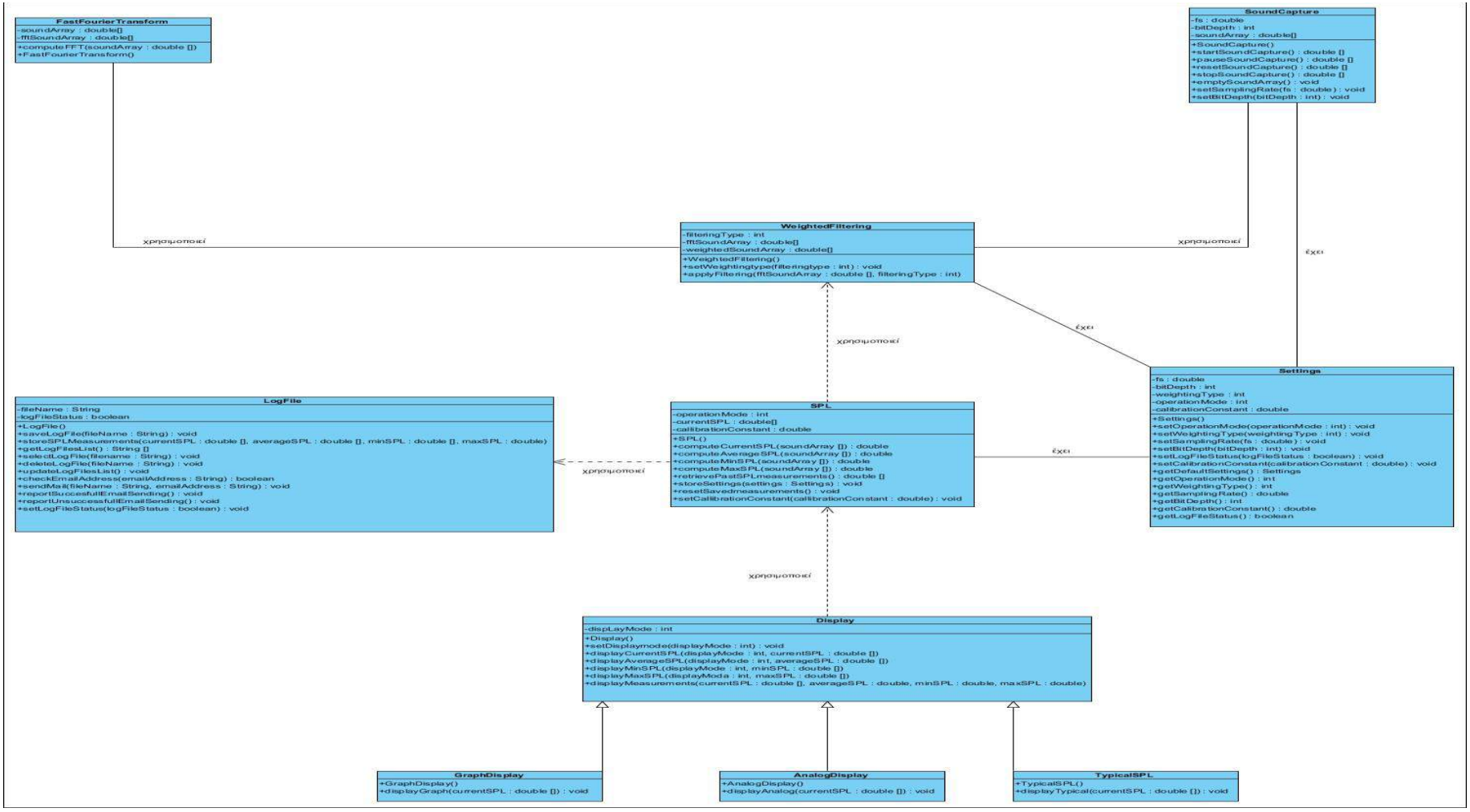

Actor
spl : SPL
wf : WeightedFiltering
sc : SoundCapture
fft : FFT
d : DISPLAY
lg : LogFile
1: stopSPLMeasurments()
1.1: stopSoundCapture()
2: lastSoundArray
3: saveSPLmeasurements()
4: computeFFT(lastSoundArray)
5: fftLastSoundArray
6: applyFiltering(fftLastSoundArray,filteringType)
7: filteredLastSoundArray
8: computeCurrentSPL()
9: displayCurrentSPL(displayMode, currentSPL)
9.1: currentSPL
10: computeAverageSPL()
11: displayAverageSPL(displayMode, averageSPL)
11.1: averageSPL
12: computeMinSPL()
13: displayMinSPL(displayMode, minSPL)
13.1: minSPL
14: computeMaxSPL()
15: displayMaxSPL(displayMode, maxSPL)
15.1: maxSPL
16: retrievePastSPLmeasurments()
17: saveLogFile()
17.1: storeSPLMeasurements()


Actor
spl : SPL
s : SETTINGS
wf : WeightedFiltering
sc : SoundCapture
d : DISPLAY
1: setOperationmode(opMode)
1.1: opMode
2: setWeightingfilter(filterType)
2.1: setWeightingtype(filterType)
3: setTimeResolution(blockSize)
3.1: setTimeResolution(blockSize)
4: setCallibrationConstant(CallibrationConstant)
4.1: callibrationConstant
5: storeSettings()
6: displaySettings()
6.1: settings

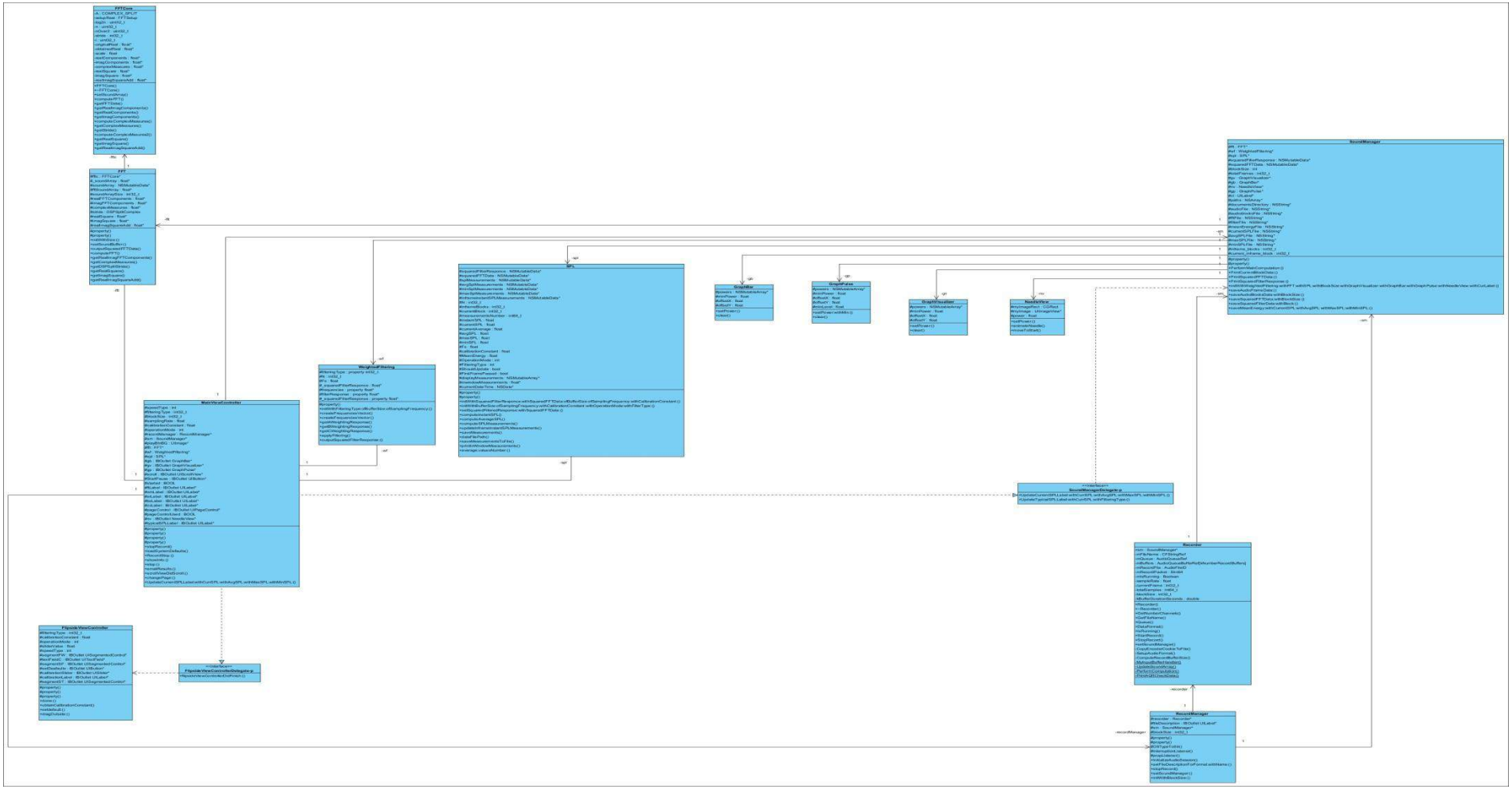


Ο κύριος σκοπός δημιουργίας των διαγραμμάτων ακολουθίας που παρουσιάστηκαν είναι η κατανομή της λειτουργικότητας (μεθόδων) στις κλάσεις του συστήματος και κατά δεύτερο λόγο ο εντοπισμός τυχόν κλάσεων, ιδιοτήτων και μεθόδων που δεν αναδείχθηκαν στο στάδιο της ανάλυσης. Το στατικό μοντέλο του υπό ανάπτυξη συστήματος, δηλαδή το διάγραμμα κλάσεων που περιγράφει την αρχιτεκτονική του είναι επομένως αναμενόμενο να αναθεωρείται μετά την ολοκλήρωση των διαγραμμάτων ακολουθίας. Στο αναθεωρημένο διάγραμμα κλάσεων που παρουσιάζεται στην ενότητα 5.6 περιλαμβάνονται οι επιπλέον μέθοδοι κάθε κλάσης με την πλήρη υπογραφή τους, δηλαδή μαζί με τις παραμέτρους που απαιτούνται και τον επιστρεφόμενο τύπο τους. Τέλος, το τελικό διάγραμμα κλάσεων που παρουσιάζεται στην ενότητα 5.7 προκύπτει από την εξειδίκευση του αναθεωρημένου διαγράμματος κλάσεων κατά την διαδικασία συγγραφής του κώδικα.

5.6 Διάγραμμα Κλάσεων (Αναθεωρημένο)



5.7 Διάγραμμα Κλάσεων (Τελικό)

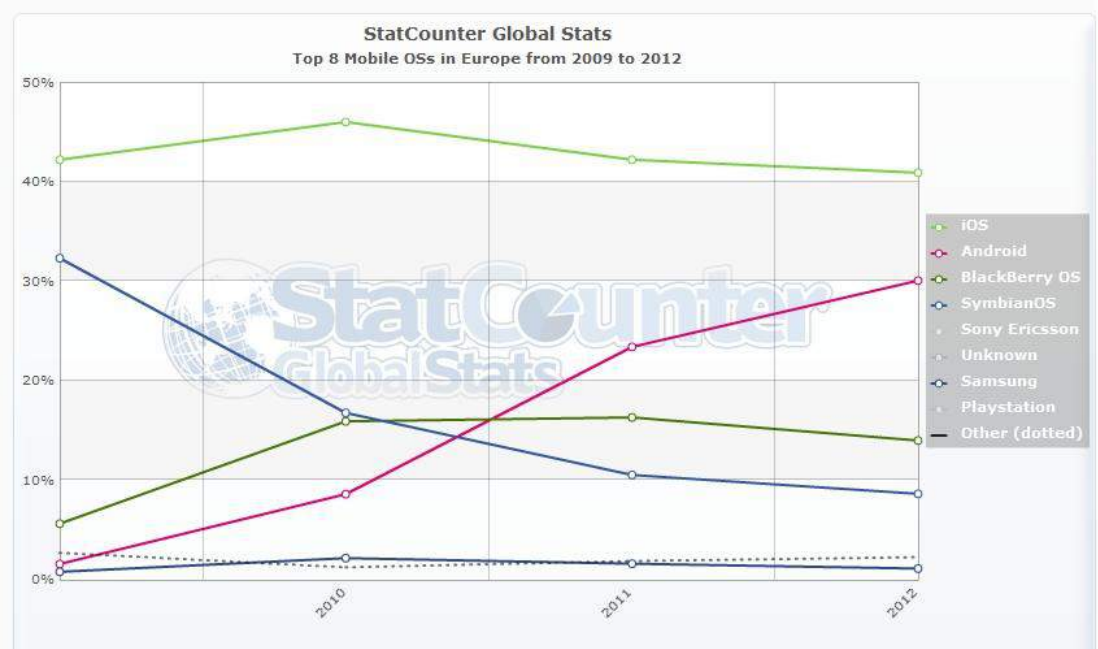


6 Ανάπτυξη Λογισμικού

Στο κεφάλαιο αυτό αναλύονται οι λόγοι που οδήγησαν στην επιλογή του λειτουργικού συστήματος iOS, της γλώσσας προγραμματισμού Objective-C και του περιβάλλοντος ανάπτυξης xCode. Επιπλέον γίνεται η επισκόπηση του συστήματος και η τεχνική ανάλυση αυτού.

6.1 Επιλογή Λειτουργικού Συστήματος

Μετά την ολοκλήρωση της μελέτης των λειτουργικών συστημάτων και του σχεδιασμού της εφαρμογής, βάσει της μεθοδολογίας ICONIX, θα πρέπει να ξεκινήσει η ανάπτυξη της. Σύμφωνα με την έρευνα της StatCounter [28]από το 2009 έως σήμερα, στην Ευρώπη φαίνεται να βρίσκεται διαρκώς στην πρώτη θέση το iOS με ποσοστό σήμερα 40,88%. Το Symbian OS μετά από μια μεγάλη πτώση από 32,32% (2009) και τη δεύτερη θέση, έχει βρεθεί σήμερα στο 8,64% και την τέταρτη θέση. Το Android OS έχοντας μόνο άνοδο έχει φτάσει σήμερα το 30,06% καταλαμβάνοντας την δεύτερη θέση και το BlackBerry OS μετά από μια μικρή πτώση τον τελευταίο χρόνο έχει ποσοστό 13,98%.

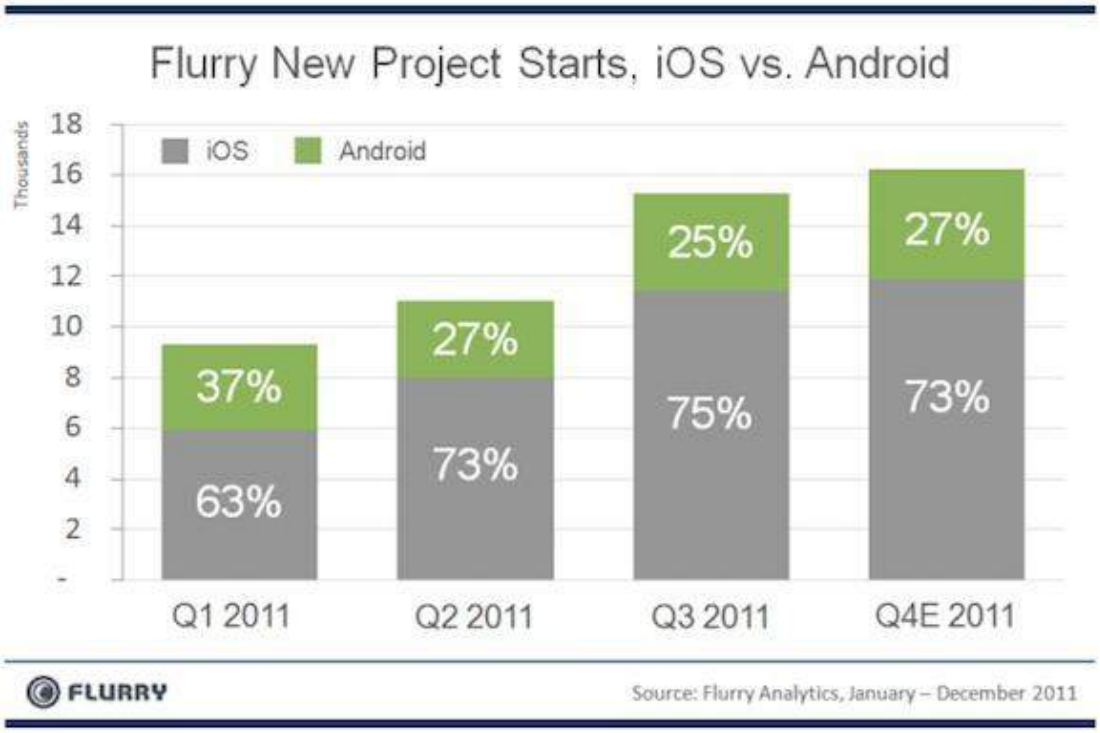


Σχήμα 45: Top Mobile OS 2009 - 2012

Συνοψίζοντας την ανάλυση των λειτουργικών συστημάτων καταλήγουμε ότι τα δύο πιο διαδεδομένα λειτουργικά συστήματα που κατέχουν και το μεγαλύτερο μέρος της αγοράς είναι το iOS και το Android OS, επομένως η τελική επιλογή θα γίνει ανάμεσα σε αυτά τα δυο λειτουργικά συστήματα.

Σύμφωνα με τις στατιστικές μελέτες και παρόλο που το Android OS χρησιμοποιείται από μεγαλύτερο αριθμό συσκευών της αγοράς, οι προγραμματιστές προτιμούν να αναπτύσσουν εφαρμογές σε περιβάλλον iOS. Στο παρακάτω γράφημα φαίνεται το ποσοστό των νέων projects που έκαναν χρήση της εταιρείας αναλύσεων Flurry το 2011 κατά τη διάρκεια του οποίου οι προγραμματιστές δημιούργησαν περίπου 50,000 εφαρμογές [30].

Το κλειστό λειτουργικό σύστημα του iOS και η εφαρμογή του μόνο σε συγκεκριμένες συσκευές (iPhone – iPad – iPod) της Apple προστατεύει τον προγραμματιστή από τις ασυμβατότητες που μπορεί να δημιουργηθούν όταν η ίδια εφαρμογή πρέπει να λειτουργήσει σε κινητά διαφορετικών κατασκευαστών, όπως συμβαίνει με το Android OS.



Σχήμα 46: Flurry New Project Starts

Σε αναφορά της η McAfee [29] υπογραμμίζει την ανωτερότητα του iOS έναντι του Android όσον αφορά την ασφάλεια του γενικότερου λειτουργικού. Η McAfee αναφέρει ότι η Apple έχει κάνει τόσο καλή δουλειά με το λειτουργικό που δεν έχουν υπάρξει μέχρι στιγμής αναφορές για κάποιο malware που να επηρεάζει τους χρήστες του iOS. Ένας λόγος που το iOS είναι τόσο ασφαλές είναι και ο περιορισμός του τρόπου που οι χρήστες κατεβάζουν εφαρμογές, δηλαδή το AppStore, επομένως αρκετοί προγραμματιστές προτιμούν να αναπτύσσουν εφαρμογές σε ένα πιο ασφαλές περιβάλλον.

Η εταιρεία της Apple παρέχει πολύ καλή υποστήριξη τόσο στους χρήστες όσο και στους προγραμματιστές του iOS κρατώντας τους άμεσα ενήμερους για τις εξελίξεις και τις αναβαθμίσεις που πραγματοποιούνται στο iOS. Επίσης, διαθέτει και ένα οργανωμένο δίκτυο

που πραγματοποιεί σεμινάρια και καθοδηγεί τους προγραμματιστές σε όλα τα μέρη του κόσμου. Αντίθετα όταν η Google αναβαθμίσει το λογισμικό της με καινούργια έκδοση, είναι στη κρίση της κάθε κατασκευάστριας εταιρείας πότε θα το βγάλει προς χρήση για την δικιά της συσκευή. Επίσης η εφαρμογή που θα αναπτυχθεί για ένα κινητό iPhone με μικρές προγραμματιστικές αλλαγές μπορεί να χρησιμοποιηθεί και σε ταμπλέτες iPad οι οποίες κατέχουν, συμφώνα με τα αποτελέσματα της εταιρείας αναλύσεων IDC για το τρίτο τρίμηνο του 2011 το 61.5% της αγοράς, ενώ στη δεύτερη θέση βρίσκετε η Samsung με ποσοστό μόλις 5.6%.

Σύμφωνα, με την παραπάνω ανάλυση επιλέχθηκε το λειτουργικό iOS της Apple και κατά προέκταση η γλώσσα προγραμματισμού Objective C, για την ανάπτυξη συστήματος λογισμικού το οποίο θα χρησιμοποιείται για την πραγματοποίηση μετρήσεων ηχοστάθμης.

6.2 iOS Software Development Kit

Για την ανάπτυξη Native εφαρμογών σε κινητές πλατφόρμες, η Apple από τον Μάρτιο του 2008 προσφέρει το iOS SDK. Παράλληλα, διένειμε και το "iOS Simulator", όπου μπορούν οι προγραμματιστές να δοκιμάζουν εικονικά της εφαρμογές τους, όμως ο προσομοιωτής δεν υποστηρίζει κάποια από τα εξαρτήματα του κινητού όπως κάμερα και accelerometer. Η φόρτωση της εφαρμογής στην συσκευή είναι δυνατή, μόνο με την συμμετοχή στο "iOS Developer Program", όπου είναι απαραίτητη η ετήσια συνδρομή.

Σχήμα 47: iOS Software Development Kit

Το iOS SDK θα μπορούσε κανείς να αναφέρει ότι αποτελείται από 4 βασικά μέρη, το Cocoa Touch, τα Media, τα Core Services και τον Πυρήνα του Mac OS X. Το Cocoa Touch είναι ένα Application Programming Interface (API) που χρησιμοποιείται για τον σχεδιασμό σε συσκευές με λειτουργικό σύστημα iOS. Είναι υπεύθυνο για το Accelerometer, την camera, την ιεραρχία στις αλλαγές που γίνονται μεταξύ των προβολών στην οθόνη, τα events που έχουν να κάνουν με την αφή μας στην οθόνη (π.χ. multi-touch) και οι υπηρεσίας τοποθεσίας της συσκευής. Τα Media είναι μια οικογένεια από APIs που είναι υπεύθυνα για την διαχείριση όλων συνολικά των πολυμέσων της συσκευής (OpenAL, OpenGL, Core

Animation, Images, Quartz, Audio, Video). Τα Core Services είναι άλλη μία οικογένεια από APIs που είναι υπεύθυνα για την διαχείριση του δικτύου και γενικότερα των δεδομένων (Networking, SQLite, Core Location, CoreMotion, Threads). Τέλος, ο πυρήνας του Mac OS X που είναι υπεύθυνο για το TCP/IP, File System, Security, Sockets και Power Management.

Το επίσημο περιβάλλον ανάπτυξης για το iOS SDK, είναι το ίδιο με αυτό που χρησιμοποιείται στο Mac OS X και ονομάζεται Xcode. Η εγκατάστασή του μπορεί να γίνει επίσημα, αποκλειστικά στο λειτουργικό σύστημα Mac OS X της Apple. Συνεπώς για να μπορέσει κάποιος να αναπτύξει μια εφαρμογή για iOS θα πρέπει να διαθέτει, computer της Apple, με λειτουργικό σύστημα Mac OS X. Το περιβάλλον ανάπτυξης Xcode είναι εύκολο να το αποκτήσει κάποιος, αφού διατίθεται δωρεάν από το AppStore. Η βασική γλώσσα προγραμματισμού που χρησιμοποιείται τόσο στις εφαρμογές για Mac OS X, όσο και για το iOS είναι η Objective-C.

6.3 Η Γλώσσα Προγραμματισμού Objective-C

Σχήμα 48: Objective-C

Η Objective-C είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού η οποία δημιουργήθηκε το 1986 από τους Brad Cox και Tom Love [31]. Έχει μεγάλες επιρροές από την SmallTalk και την ANSI C και χρησιμοποιείται κυρίως για προγραμματισμό συσκευών της Apple και γενικά συστημάτων που βασίζονται στο OpenStep. Πάνω στην Objective-C βασίζονται πολλές αντικειμενοστραφείς γλώσσες μεταξύ των οποίων η JAVA.

Ως υπερσύνολο της γλώσσας προγραμματισμού C, η Objective-C υποστηρίζει την ίδια βασική σύνταξη, δηλαδή τα γνωστά είδη μεταβλητών (int, float, ..), δομές, λειτουργίες και δείκτες. Επιπρόσθετα περιλαμβάνει τα ακόλουθα χαρακτηριστικά:

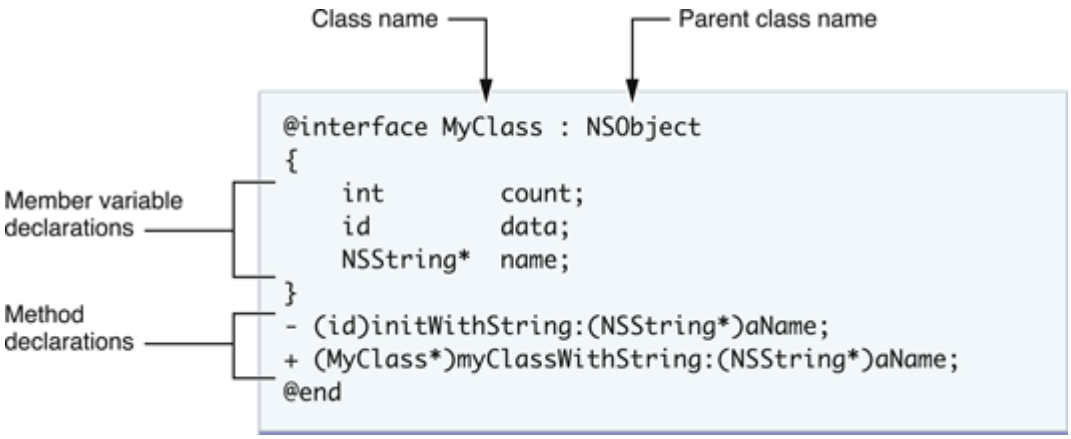
- Καθορισμός νέων κλάσεων
- Καθορισμός μεθόδων
- Μπλοκ τμήματα κώδικά που μπορούν να εκτελεστούν ανά πάσα στιγμή
- Επεκτάσεις στην κύρια γλώσσα όπως τα πρωτόκολλα και τις κατηγορίες

Όπως όλες οι αντικειμενοστραφείς γλώσσες προγραμματισμού έτσι και η Objective-C οργανώνει το πρόγραμμα σε αντικείμενα. Ένα αντικείμενο είναι μια μονάδα που αποτελείται από την περιγραφή κάποιων δεδομένων των λειτουργιών που μπορούν να εφαρμοστούν σε αυτά. Ένα αντικειμενοστραφές πρόγραμμα αποτελείται από διάφορα αντικείμενα που αλληλεπιδρούν μεταξύ τους. Ο καθορισμός μιας κλάσης απαιτεί δυο διαφορετικά τμήματα κώδικα, της διεπαφής και της υλοποίησης. Το μέρος της διεπαφής περιέχει τον ορισμό της κλάσης και καθορίζει τις μεταβλητές στιγμιότυπα καθώς και τις μεθόδους που είναι συσχετισμένες με αυτή την κλάση. Όπως συμβαίνει και με τον κώδικα της C ορίζονται αρχεία κεφαλίδων και αρχεία πηγαίου κώδικα σε ξεχωριστές "public" δηλώσεις. Τα συγκεκριμένα αρχεία και οι αντίστοιχες επεκτάσεις τους αναφέρονται στον πίνακα 3.

Πίνακας 3 : Τύποι αρχείων

Προέκταση	**Τύπος αρχείου**
.h	Κεφαλίδα. Τα αρχεία κεφαλίδας περιέχουν κλάσεις, τύπους, μεθόδους και δηλώσεις σταθερών.
.m	Πηγαίος κώδικας. Ένα αρχείο πηγαίου κώδικα με αυτή την επέκταση μπορεί να περιέχει κώδικα γραμμένο στις γλώσσες Objective-C και C
.mm	Πηγαίος κώδικας. Ένα αρχείο πηγαίου κώδικα με αυτή την επέκταση μπορεί να περιέχει επιπλέον κώδικα γραμμένο σε C++, τα αρχεία αυτά θα πρέπει να χρησιμοποιούνται στις περιπτώσεις όπου ο κώδικας σε Objective-C αναφέρεται σε κλάσεις υλοποιημένες σε C++

Όταν θέλουμε να εισάγουμε μια κεφαλίδα στον πηγαίο κώδικα, χρησιμοποιούμε το #import και στην συνέχεια προσθέτουμε σε εισαγωγικά το όνομά της. Η δήλωση μιας διεπαφής ξεκινάει με @interface, ακολουθεί το όνομά της και στην συνέχεια στην περίπτωση κληρονομικότητας, μετά από άνω-κάτω τελεία δηλώνεται το όνομα της κλάσης από την οποία κληρονομεί. Στο παρακάτω σχήμα φαίνεται η τυπική σύνταξη μιας κλάσης, που κληρονομεί από την βασική κλάση NSObject. Στην Objective-C υπάρχει μονή κληρονομικότητα, έτσι δεν μπορεί μια κλάση να κληρονομήσει από περισσότερες της μίας, κλάσεις. Ακολουθούν οι δηλώσεις των μεταβλητών μέσα σε αγκύλες στην περίπτωση που έχουμε περισσότερες από μία. Ακολουθούν οι επικεφαλίδες των μεθόδων και τελειώνει με @end.

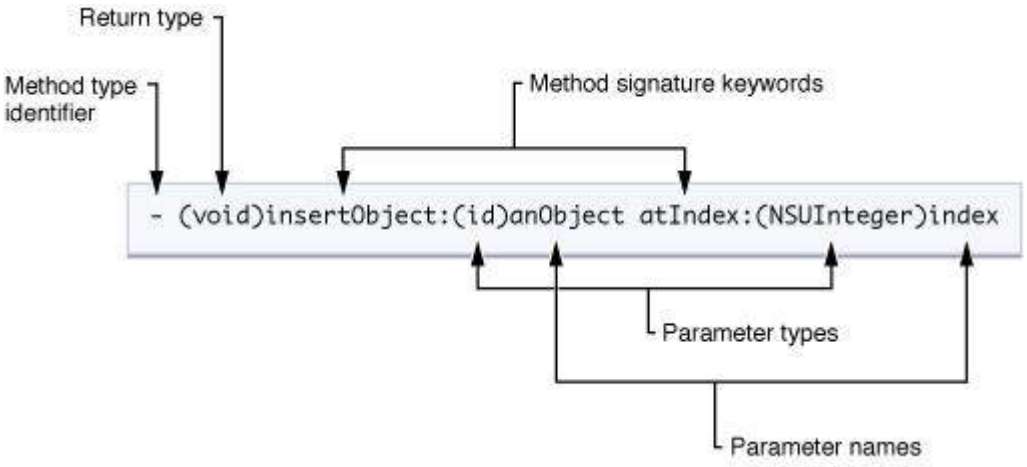


Σχήμα 49: Δήλωση Interface

Η σύνταξη για την υλοποίηση μιας κλάσης είναι παρόμοια. Ξεκινά με την οδηγία μεταγλωττιστή @implementation ακολουθούμενη από το όνομα της κλάσης και τελειώνει με την οδηγία μεταγλώττισης @end. Η υλοποίηση των μεθόδων μιας κλάσης πραγματοποιείται μεταξύ των παραπάνω οδηγιών. Ο κώδικας που αφορά στην υλοποίηση μιας κλάσης θα πρέπει να ενσωματώνει πάντοτε το αντίστοιχο αρχείο διεπαφής στις πρώτες γραμμές του.

```
#import "MyClass.h"
@implementation MyClass
- (id)initWithString:(NSString *)aName
{
// code goes here
}
+ (MyClass *)myClassWithString:(NSString *)aName
{
   // code goes here
}
@end
```

Υπάρχουν δυο τύποι μεθόδων στην Objective-C ο πρώτος τύπος αφορά τις μεθόδους που μπορούν να κληθούν από ένα στιγμιότυπο κάποιας κλάσης (instance method) και τις μεθόδους εκείνες που η κλήση τους δεν προϋποθέτει την ύπαρξη αντικειμένου κάποιας κλάσης (class method). Ο ορισμός μιας μεθόδου αποτελείται από το αναγνωριστικό του τύπου της μεθόδου, του τύπου επιστροφής, ενός ή περισσότερων λέξεων κλειδιά που αφορούν την υπογραφή της μεθόδου, του τύπου της αντίστοιχης παραμέτρου και του ονόματός της. Παρακάτω φαίνεται η δήλωση της instance method insertObject:atIndex



Σχήμα 50: Δήλωση Μεθόδου

Μια ιδιότητα υπό την ευρεία έννοια αποτελεί μια προγραμματιστική οντότητα η οποία ενθυλακώνει την κατάσταση ενός αντικειμένου. Συγκεκριμένα αποτελεί είτε ένα χαρακτηριστικό όπως για παράδειγμα το όνομα ή το χρώμα είτε δηλώνει την σχέση με ένα ή περισσότερα αντικείμενα. Η κλάση ενός αντικειμένου είναι αυτή που καθορίζει την διεπαφή με την οποία οι χρήστες μπορούν να προσπελαύνουν (getter methods) και να τροποποιούν (setter methods) τις τιμές των ιδιοτήτων της. Οι μέθοδοι που πραγματοποιούν αυτή την λειτουργία ονομάζονται μέθοδοι προσπέλασης (accessor methods).

Υπάρχουν δυο τύποι μεθόδων προσπέλασης και καθένας από αυτούς θα πρέπει να είναι σύμφωνος με τις σχετικές συμβάσεις ονοματολογίας. Μια μέθοδος τύπου getter η οποία επιστρέφει την τιμή μιας ιδιότητας θα πρέπει να έχει το ίδιο όνομα με την ιδιότητα αυτή. Αντίστοιχα μια μέθοδος τύπου setter η οποία τροποποιεί την τιμή μιας ιδιότητας με όνομα propertyName θα πρέπει να είναι της μορφής setPropertyName: όπου το πρώτο γράμμα του ονόματος της ιδιότητας θα είναι κεφαλαίο. Η δήλωση μιας ιδιότητας φαίνεται παρακάτω:

```
@property (nonatomic, copy) NSString *userName;
```

Η δήλωση ιδιοτήτων εκμηδενίζει την ανάγκη υλοποίησης μεθόδων getter και setter για κάθε ιδιότητα που δηλώνεται σε μια κλάση. Αντίθετα η επιθυμητή συμπεριφορά προσδιορίζεται μέσω της δήλωσης μιας ιδιότητας και της αντίστοιχης σύνθεσής της με την οδηγία μεταγλώττισης @synthesize. Η σύνθεση της ιδιότητας που ορίστηκε παραπάνω θα μπορούσε να γίνει ως εξής : @synthesize;

Η δήλωση ιδιοτήτων ελαττώνει το συνολικό όγκο του κώδικα καθιστώντας τον περισσότερο ευανάγνωστο και κατά συνέπεια μειώνοντας τα πιθανά σφάλματα. Οι δηλώσεις

ιδιοτήτων όπως και οι δηλώσεις των μεθόδων μιας κλάσης πραγματοποιούνται στο αντίστοιχο αρχείο διεπαφής.

Ένα πρωτόκολλο είναι μια συλλογή μεθόδων οι οποίες μπορούν να υλοποιηθούν από μια οποιαδήποτε κλάση, ακόμα και από κλάσεις που δεν κληρονομούν από την ίδια υπερκλάση. Οι μέθοδοι ενός πρωτοκόλλου καθορίζουν ένα είδος συμπεριφοράς το οποίο είναι ανεξάρτητο από οποιαδήποτε συγκεκριμένη κλάση. Τα πρωτόκολλα στην ουσία καθορίζουν μια διεπαφή, η υλοποίηση της οποίας αποτελεί ευθύνη των κλάσεων που συμμορφώνονται με το συγκεκριμένο πρωτόκολλο. Στην πράξη ένα πρωτόκολλο καθορίζει ένα σύνολο μεθόδων οι οποίες εγκαθιδρύουν μια συμφωνία μεταξύ αντικειμένων χωρίς να είναι απαραίτητο να αποτελούν στιγμιότυπα κάποιας συγκεκριμένης κλάσης. Η συγκεκριμένη συμφωνία παρέχει την δυνατότητα επικοινωνίας μεταξύ των εμπλεκόμενων αντικειμένων. Για παράδειγμα η κλάση UIApplication υλοποιεί την επιθυμητή συμπεριφορά μιας εφαρμογής. Στην συγκεκριμένη περίπτωση η χρήση πρωτοκόλλων εξυπηρετεί την επικοινωνία με το αντικείμενο της εφαρμογής χωρίς να είναι απαραίτητη η δημιουργία ενός νέου αντικειμένου που θα κληρονομεί από την κλάση UIApplication. Η επικοινωνία αυτή εξασφαλίζεται καλώντας τις μεθόδους που είναι συσχετισμένες με το αντικείμενο delegate της κλάσης UIApplication. Η δήλωση του πρωτοκόλλου με το οποίο συμμορφώνεται μια συγκεκριμένη κλάση μπορεί να γίνει μέσα σε brackets (<...>) όπως φαίνεται στον παρακάτω πίνακα.

```
@interface HelloWorldViewController : UIViewController <UITextFieldDelegate> {

}

@end
```

Επιπλέον οι μέθοδοι του πρωτοκόλλου που υλοποιούνται δεν είναι απαραίτητο να δηλώνονται στο αντίστοιχο αρχείο κεφαλίδας. Η δήλωση ενός πρωτοκόλλου είναι παρόμοια με αυτήν της διεπαφής μιας κλάσης με την διαφορά ότι τα πρωτόκολλα δεν κληρονομούν από κάποια κλάση αλλά και ούτε περιέχουν μεταβλητές στιγμιότυπα, αν και μπορούν να περιέχουν τους ορισμούς ιδιοτήτων. Το παρακάτω παράδειγμα παρουσιάζει την δήλωση ενός απλού πρωτοκόλλου με μία μέθοδο.

```
@protocol MyProtocol

- (void)myProtocolMethod;

@end
```

Η υιοθέτηση ενός συγκεκριμένου πρωτοκόλλου ανάγεται στην υλοποίηση των μεθόδων που ορίζονται μέσα σε αυτό. Τέλος υπάρχουν πρωτόκολλα για τα οποία η υλοποίηση κάποιων μεθόδων είναι προαιρετική.

```
@interface NSDate (NSDateCreation)
```

Η παραπάνω γραμμή κώδικα αποτελεί παράδειγμα δήλωσης μιας κατηγορίας μέσω της συντακτικής σύμβασης κατά την οποία το όνομα μιας κατηγορίας εμπεριέχεται μέσα σε παρενθέσεις. Οι κατηγορίες αποτελούν ένα χαρακτηριστικό της γλώσσας Objective-C το οποίο παρέχει την δυνατότητα επέκτασης της διεπαφής μιας κλάσης χωρίς να είναι απαραίτητη η δήλωση ενός αντικειμένου που κληρονομεί από αυτή. Οι μέθοδοι που συμπεριλαμβάνονται σε μία κατηγορία αποτελούν τμήμα του τύπου μιας κλάσης και κληρονομούνται από όλες τις υποκλάσεις της. Η χρήση κατηγοριών αποτελεί ένα μέσο ομαδοποίησης των δηλώσεων που αφορούν σχετικές μεθόδους στο ίδιο αρχείο κεφαλίδας. Στη πράξη μπορεί κανείς να πραγματοποιήσει τη δήλωση διαφορετικών ιδιοτήτων σε ξεχωριστά αρχεία κεφαλίδας.

6.4 Εργαλεία Προγραμματισμού

Το Xcode είναι μια σουίτα από εργαλεία προγραμματισμού, που προσφέρεται από την Apple, για την ανάπτυξη λογισμικού τόσο για το λειτουργικό Mac OS X όσο και για το iOS [32]. Ο χώρος εργασίας του Xcode χωρίζεται σε τρία βασικά παράθυρα, έναν χώρο πλοήγησης, την περιοχή σύνταξης κειμένου και τον βοηθητικό χώρο όπου επιτρέπει την εύκολη οργάνωση βοηθητικών ενεργειών, όπως την άμεση πρόσβαση της βιβλιοθήκης, την εύκολη δοκιμή της εφαρμογής στην συσκευή και την προετοιμασία αυτής για την υποβολή στο App Store. Ο χώρος εργασίας μπορεί να προσαρμοστεί ανάλογα όπως για παράδειγμα να γίνει απόκρυψη του πλοηγού ή εμφάνισης μόνο της σύνταξης, ανάλογα με τις ανάγκες του προγραμματιστή.

Μέσω της σουίτας του Xcode έχουμε τη δυνατότητα να τρέχουμε την εφαρμογή ή να κάνουμε αποσφαλμάτωση με τη βοήθεια του προσομοιωτή, iOS Simulator. Χρησιμοποιώντας τον προσομοιωτή είμαστε βέβαιοι ότι η εφαρμογή μας θα λειτουργήσει με τον τρόπο που θέλουμε. Ο οδηγός αποσφαλμάτωσης βρίσκεται ενσωματωμένος στο Xcode. Όταν ο προγραμματιστής επιλέξει ένα σημείο διακοπής (break point), η εκτέλεση του προγράμματος σταματά προσωρινά στο σημείο αυτό, δίνοντας τη δυνατότητα να ακολουθήσουμε ένα συγκεκριμένο νήμα εκτέλεσης βήμα προς βήμα. Επίσης παρέχεται η δυνατότητα να εκτελεστεί η εφαρμογή μας απευθείας στη φορητή συσκευή, αφού αυτή

συνδεθεί στον υπολογιστή μας. Για να γίνει αυτό θα πρέπει να έχει εγκατασταθεί ένα "associated provisioning profile" στην συσκευή μας. Πολλές φορές αυτό είναι αναγκαίο διότι ο προσομοιωτής δεν παρέχει όλες τις δυνατότητες μιας πραγματικής φορητής συσκευής.

Ο έλεγχος της απρόσκοπτης λειτουργίας της εφαρμογής μπορεί να πραγματοποιηθεί κάνοντας ανάλυση της απόδοσής της. Αυτό επιτυγχάνεται με τη βοήθεια του Instruments application. Το εργαλείο αυτό συγκεντρώνει στοιχεία, κατά τη διάρκεια λειτουργίας της εφαρμογής και τα παρουσιάζει σε ένα γραφικό χρονοδιάγραμμα. Τα στοιχεία που μπορεί να απεικονίσει είναι η μνήμη της εφαρμογής, η δραστηριότητα του δίσκου, η δραστηριότητα του δικτύου, η απόδοση γραφικών και άλλων μετρήσεων. Προβάλλοντας τα παραπάνω δεδομένα μαζί μπορούμε να αναλύσουμε χρονικά την συμπεριφορά απόδοσης της εφαρμογής, εντοπίζοντας πιθανούς τομείς βελτίωσης. Με την συνεχή χρήση του εργαλείου διαπιστώνεται εύκολα, αν οι αλλαγές βελτιώνουν τις επιδόσεις της εφαρμογής. Στην περίπτωση που κάτι δεν λειτουργεί σωστά, μετά από αλλαγές στον κώδικα, η λειτουργία "snapshot" του Xcode δίνει την δυνατότητα επαναφοράς του project σε προηγούμενη κατάσταση. Το snapshot αποθηκεύει στον δίσκο την τρέχουσα κατάσταση του project αυτόματα, σε διάφορες στιγμές όπως πριν από κάθε εκτέλεση, αναζήτηση, αντικατάσταση και άλλες ειδικές περιπτώσεις, για την πιθανή αποκατάσταση αργότερα. Τέλος, οποιαδήποτε στιγμή θελήσει ο προγραμματιστής μπορεί να πραγματοποιήσει snapshots χειροκίνητα.

6.5 Επισκόπηση Συστήματος

Συνοπτικά η λειτουργία του συστήματος περιγράφεται παρακάτω. Το σύστημα δέχεται συνολικά από το μικρόφωνο 44100 τιμές ανά δευτερόλεπτο (μεταξύ [-32768,+32767] Sint16), οι οποίες όμως λαμβάνονται σε block, το μέγεθος των οποίων και η συχνότητα λήψης τους εξαρτάται από την επιλογή του χρήστη ανάμεσα σε slow ή fast για τον τρόπο λειτουργίας του μετρητικού συστήματος και ανάμεσα σε high ή normal ή low ανάλογα με την επιθυμητή τιμή του time resolution. Συγκεκριμένα, οι δυνατές τιμές για το μέγεθος του block και της αντίστοιχης συχνότητας λήψης φαίνονται στον πίνακα 4.

Πίνακας 4: Time resolution

	High		**Normal**		**Low**	
	samples	**msec**	**samples**	**msec**	**samples**	**msec**
Fast	512	11.60997732	1024	23.21995465	2048	46.4399092
Slow	4096	92.87981859	8192	185.7596372	16384	371.5192744

Οι λαμβανόμενες τιμές ανά block κανονικοποιούνται στο διάστημα [-1,1] και υπολογίζονται ο FFT (όπως φαίνεται στην παράγραφο 3.3.3) και η συχνοτική απόκριση του φίλτρου (A, B, C ή Lin [χωρίς φίλτρο]) (με βάση τις σχέσεις (64), (65), (66)). Στην συνέχεια υπολογίζονται τα τετράγωνα των συντελεστών Fourier και τα τετράγωνα της απόκρισης του φίλτρου για το συγκεκριμένο block επεξεργασίας. Οι δύο πίνακες τετραγώνων πολλαπλασιάζονται ανά στοιχείο και τελικά υπολογίζεται το άθροισμα των τιμών του πίνακα που προκύπτει (με βάση τις σχέσεις (67), (68), (69), (70)). Το άθροισμα αυτό είναι ίσο με την συνολική ενέργεια του σταθμισμένου σήματος. Από την συνολική ενέργεια του σταθμισμένου σήματος υπολογίζεται η τιμή της μέσης ενέργειας (με βάση τις σχέσεις (71), (72), (73), (74)) από την οποία προκύπτει τελικά η τιμή του στιγμιαίου SPL (με βάση τις σχέσεις (75), (76) σε κάθε block ηχητικών δειγμάτων.

Οι τιμές του στιγμιαίου SPL που υπολογίζονται σε κάθε block συνδυάζονται ώστε να προκύψει η τιμή του τρέχοντος SPL. Συγκεκριμένα, η τιμή του τρέχοντος SPL προκύπτει ως ο μέσος όρος των n τελευταίων μετρήσεων του στιγμιαίου SPL, όπου n είναι το πλήθος των block που αντιστοιχούν στην χρονική διάρκεια ενός δευτερολέπτου.

6.6 Τεχνική Ανάλυση Λογισμικού

Οι κλάσεις που υλοποιήθηκαν για την ανάπτυξη της εφαρμογής μέτρησης της ηχητικής στάθμης (SPL) σύμφωνα με την ανάλυση απαιτήσεων που έγινε σε προηγούμενο κεφάλαιο είναι οι εξής:

Ονομασία Κλάσης	**Γλώσσα Προγραμματισμού**
AppDelegate	Objective-C
MainViewController	Objective-C
FlipsideViewController	Objective-C
RecordManager	Objective-C
Recorder	Objective-C++
SoundManager	Objective-C
SPL	Objective-C
WeightedFiltering	Objective-C
FFT	Objective-C
FFTCore	Objective-C++

GraphBar	Objective-C
GraphPulse	Objective-C
NeedleView	Objective-C
GraphVisualizer	Objective-C

AppDelegate

Η συγκεκριμένη κλάση αποτελεί τον πυρήνα της εφαρμογής που υλοποιήθηκε καθώς αναλαμβάνει την διαχείριση του κεντρικού παραθυρικού αντικειμένου που αντιστοιχεί στην κλάση MainViewController. Μεταξύ των δύο αυτών κλάσεων υπάρχει απλή κατευθυνόμενη συσχέτιση από την κλάση AppDelegate προς την κλάση MainViewController. Η πληθυκότητα της παραπάνω συσχέτισης είναι 1-1 καθώς σε κάθε στιγμή εκτέλεσης του προγράμματος ένα αντικείμενο της AppDelegate είναι συσχετισμένο με ένα και μόνο αντικείμενο της MainViewController. Η κατεύθυνση αυτής της συσχέτισης δεν είναι αμφίδρομη καθώς μόνο το αντικείμενο της AppDelegate έχει πρόσβαση στο αντικείμενο της MainViewController, το οποίο και δημιουργεί με την έναρξη του προγράμματος, και όχι το αντίστροφο. Συμπληρωματικά, η AppDelegate είναι επιφορτισμένη με τις λειτουργίες που αφορούν την λογική του Multiview Application με την οποία είναι υλοποιημένη η παρούσα εφαρμογή.

MainViewController

Η κλάση αυτή ενθυλακώνει το σύνολο της παραθυρικής λειτουργικότητας που παρέχεται από την εφαρμογή, καθώς αναλαμβάνει την διαχείριση της γραφικής διεπαφής με τον χρήστη. Είναι άμεσα συσχετισμένη με το αντίστοιχο αρχείο γραφικού περιεχομένου .xib στο οποίο πραγματοποιήθηκε ο σχεδιασμός του γραφικού περιβάλλοντος της κεντρικής φόρμας της εφαρμογής. Η συγκεκριμένη κλάση είναι συσχετισμένη μέσω ενός πρωτοκόλλου με την κλάση FlipsideViewController η οποία είναι η δεύτερη παραθυρική απεικόνιση της εφαρμογής. Μεταξύ των δύο αυτών παραθυρικών κλάσεων υπάρχει κατευθυντικότητα, καθώς η MainViewController είναι αυτή που δημιουργεί το αντικείμενο της FlipsideViewController ανταποκρινόμενη σε αντίστοιχη ενέργεια του χρήστη. Ωστόσο η επικοινωνία των δύο κλάσεων εξασφαλίζεται μέσω της χρήσης ενός πρωτοκόλλου και των αντίστοιχων μεθόδων που δεσμεύεται να υλοποιήσει η FlipsideViewController. Επιπλέον η κλάση MainViewController έχει την ευθύνη της δημιουργίας των αντικειμένων όλων των υπόλοιπων κλάσεων της εφαρμογής με τις οποίες συνδέεται με απλή κατευθυνόμενη συσχέτιση πληθυκότητας 1-1. Αυτό συμβαίνει διότι η κλάση MainViewController έχει την δυνατότητα πρόσβασης στα αντικείμενα των υπόλοιπων κλάσεων και όχι το αντίστροφο. Επιπρόσθετα, αναγνωρίζεται η πληθυκότητα 1-1 καθώς σε κάθε στιγμή εκτέλεσης του

προγράμματος, το μοναδικό αντικείμενο της MainViewController είναι συσχετισμένο με ένα και μόνο αντικείμενο των υπολοίπων κλάσεων.

FlipsideViewController

Η κλάση αυτή συμπληρώνει την γραφική διεπαφή με την εφαρμογή καθώς είναι υπεύθυνη για την συλλογή των ατομικών προτιμήσεων του χρήστη για τον τρόπο λειτουργίας του συστήματος. Είναι άμεσα συσχετισμένη με το δεύτερο αρχείο γραφικού περιεχομένου .xib μέσω του οποίου ο χρήστης μπορεί να παραμετροποιήσει:

- Την ταχύτητα της μέτρησης της ηχητικής στάθμης (Fast / Slow)
- Το Time Resolution (High, Normal, Low)
- Την χρήση καμπύλης στάθμισης (Lin[χωρίς στάθμιση], A, B, C.)
- Την τιμή της σταθεράς βαθμονόμησης από ένα εύρος επιτρεπτών τιμών (Calibration constant)
- Την επαναφορά των προεπιλεγμένων ρυθμίσεων του συστήματος.

Οι παραπάνω επιλογές του χρήστη μεταφέρονται στο αντικείμενο της κλάσης MainViewController μέσω του πρωτοκόλλου. Η ονομασία της προκύπτει από τον τρόπο μετάβασης από το κεντρικό παράθυρο της εφαρμογής προς σε αυτήν και αντίστροφα.

RecordManager

Η κλάση αυτή αναλαμβάνει την διαμεσολάβηση με το αντικείμενο της κλάσης Recorder που είναι υλοποιημένη σε Objective-C++. Ο ρόλος της ανάγεται στην διεπαφή του συστήματος με τις χαμηλότερου επιπέδου λειτουργίες που αφορούν στην συλλογή των ηχητικών δειγμάτων από το μικρόφωνο της συσκευής. Η συγκεκριμένη κλάση έχει απλή κατευθυνόμενη συσχέτιση πληθυκότητας 1-1 με την κλάση Recorder. Η συγκεκριμένη συσχέτιση αναγνωρίζεται καθώς το μοναδικό αντικείμενο της Recorder που υπάρχει σε κάθε στιγμή εκτέλεσης του προγράμματος έχει δημιουργηθεί κατά την αρχικοποίηση της RecordManager. Η συσχέτιση αυτή δεν είναι αμφίδρομη καθώς το αντικείμενο της Recorder δεν έχει πρόσβαση στο αντικείμενο της RecordManager. Η παρεχόμενη λειτουργικότητα από αυτή την κλάση, συμπληρώνεται από την διαχείριση της ηχητικής συνεδρίας (Audio Session) την οποία ξεκινά ή σταματά ανάλογα με τις ενέργειες του χρήστη. Η κλάση αυτή καθορίζει σημαντικές παραμέτρους της ηχητικής καταγραφής που πραγματοποιεί το σύστημα όπως για παράδειγμα το PCM Linear που είναι ο επιθυμητός τρόπος αναπαράστασης του ηχητικού σήματος που δειγματίζεται. Επιπλέον, μέσω της υλοποίησης κατάλληλων μεθόδων παρακολούθησης ηχητικών γεγονότων (Audio Event Listeners) έχει την δυνατότητα διαχείρισης ανεπιθύμητων αλλαγών κατά την διάρκεια της ηχητικής συνεδρίας. Τέλος, η κλάση RecordManager έχει απλή κατευθυνόμενη συσχέτιση με την

κλάση SoundManager πληθυκότητας 1-1, καθώς είναι αυτή που αναλαμβάνει την σύνδεση του αντικειμένου της δεύτερης με την κλάση Recorder.

Recorder

Η συγκεκριμένη κλάση είναι επιφορτισμένη με τις χαμηλού επιπέδου λειτουργίες διαχείρισης που αφορούν στη συλλογή των ηχητικών δειγμάτων από το υλικό της φορητής συσκευής. Η υλοποίηση αυτών των λειτουργιών είναι απαραίτητο να γίνεται όσο το δυνατό γρηγορότερα και για τον λόγο αυτό ο κώδικας της κλάσης Recorder είναι γραμμένος σε Objective-C++, εκμεταλλευόμενος τις εγγενείς δυνατότητες του υλικού. Το σύνολο των μεθόδων που υλοποιούνται από την κλάση Recorder αντιστοιχεί στις λειτουργίες εκείνες που προδιαγράφονται στο CoreAudio Framework. Η συλλογή των ηχητικών δειγμάτων από την εν λόγω κλάση, πραγματοποιείται μέσω της χρήσης τριών audio buffers τα οποία αποτελούν και το κεντρικό σημείο επικοινωνίας μεταξύ του λογισμικού και του υλικού της φορητής συσκευής. Συγκεκριμένα, κατά την διάρκεια της ηχοληψίας κάθε φορά που ένα audio buffer γεμίζει με το επιθυμητό πλήθος δειγμάτων του ήχου καλείται μία μέθοδος callback η οποία σηματοδοτεί την ολοκλήρωση της συλλογής του ηχητικού σήματος. Μέσα στα πλαίσια αυτής της μεθόδου, η κλάση Recorder τροφοδοτεί τα δείγματα του ήχου στην κλάση SoundManager προκειμένου να ολοκληρωθεί ο υπολογισμός του επιπέδου της ηχητικής στάθμης. Μεταξύ των κλάσεων Recorder και SoundManager υπάρχει αμφίδρομη συσχέτιση, καθώς η μία κλάση μπορεί να προσπελάσει τα αντικείμενα της άλλης. Η πληθυκότητα της συσχέτισης είναι 1-1 καθώς ανά πάσα στιγμή της εκτέλεσης του προγράμματος υπάρχει ένα και μόνο αντικείμενο από κάθε κλάση. Η συλλογή των ηχητικών δειγμάτων γίνεται με βάση το ρυθμό δειγματοληψίας που καθορίζεται από το υλικό της φορητής συσκευής και συμπληρώνεται από τις αντίστοιχες ενέργειες της κανονικοποίησης και του zero-padding που είναι απαραίτητες για την συγκεκριμένη υλοποίηση.

SoundManager

Η κλάση αυτή έχει κεντρικό ρόλο στον υπολογισμό του επιπέδου της ηχητικής στάθμης, καθώς αναλαμβάνει τον συντονισμό των επιμέρους κλάσεων που εμπλέκονται σε αυτόν. Αυτό συμβαίνει διότι σε πρώτη φάση επικοινωνεί με την κλάση Recorder από την οποία λαμβάνει κάθε φορά τα τρέχοντα δείγματα του ήχου και σε δεύτερη φάση τα τροφοδοτεί στην κλάση FFT προκειμένου να υπολογιστούν οι αντίστοιχοι συντελεστές του μετασχηματισμού Fourier. Στη συνέχεια επικοινωνώντας με τις κλάσεις WeightedFiltering και SPL ολοκληρώνει τον υπολογισμό του σταθμισμένου επιπέδου ηχητικής στάθμης. Μεταξύ της κλάσης SoundManager και των κλάσεων SPL, WeightedFiltering και FFT υπάρχει σχέση απλής μονόδρομης συσχέτισης 1-1. Η συγκεκριμένη συσχέτιση

αναγνωρίζεται καθώς η κλάση SoundManager έχει αποκλειστική πρόσβαση στα μοναδικά αντικείμενα των κλάσεων SPL, WeightedFiltering και FFT που υπάρχουν ανά πάσα στιγμή εκτέλεσης του προγράμματος. Η διενέργεια του υπολογισμού του επιπέδου της ηχητικής στάθμης από τη SoundManager γίνεται μέσα στα πλαίσια της μεθόδου callback της κλάσης Recorder με την οποία υπάρχει αμφίδρομη επικοινωνία. Η κλάση SoundManager δέχεται ως είσοδο ένα σύνολο ηχητικών δειγμάτων το οποίο καθορίζεται από το επιλεγμένο μέγεθος παραθύρου και υπολογίζει την αντίστοιχη τιμή του τρέχοντος SPL.

WeightedFiltering

Η κλάση αυτή αναλαμβάνει τον υπολογισμό της συχνοτικής απόκρισης των χρησιμοποιούμενων φίλτρων στάθμισης του συστήματος. Οι τέσσερις διαφορετικοί τρόποι στάθμισης που υλοποιούνται είναι οι παρακάτω:

- Απουσία στάθμισης, η οποία αντιστοιχεί με πολλαπλασιασμό του ηχητικού σήματος με έναν πίνακα μονάδων προκειμένου να μην το μεταβάλει.
- Σταθμίσεις A, B, C, οι οποίες αντιστοιχούν με πολλαπλασιασμό του ηχητικού σήματος με πίνακες κατάλληλων τιμών σύμφωνα με την μαθηματική περιγραφή της συγκεκριμένης στάθμισης που επιλέγεται.

Η κλάση WeightedFiltering δέχεται ως είσοδο από την κλάση SoundManager το παράθυρο χρόνου για το οποίο θα πρέπει να υπολογίσει την αντίστοιχη απόκριση του φίλτρου που έχει επιλέξει ο χρήστης.

FFT

Η συγκεκριμένη κλάση αναλαμβάνει τη διεπαφή με την κλάση FFTCore η οποία είναι υλοποιημένη σε Objective-C++. Μεταξύ των κλάσεων αυτών υπάρχει απλή κατευθυνόμενη συσχέτιση από την κλάση FFT προς την κλάση FFTCore καθώς μόνο η πρώτη μπορεί να προσπελάσει το αντικείμενο της δεύτερης. Η πληθυκότητα της εν λόγω συσχέτισης είναι 1-1, καθώς ένα μόνο αντικείμενο από κάθε κλάση υπάρχει ανά πάσα στιγμή εκτέλεσης του προγράμματος. Η κλάση FFT δέχεται ως είσοδο από την κλάση SoundManager ένα συγκεκριμένο block ηχητικών δειγμάτων και με κατάλληλες κλήσεις μεθόδων της FFTCore επιστρέφει ένα διάνυσμα με τους αντίστοιχους συντελεστές του μετασχηματισμού Fourier.

FFTCore

Η κλάση αυτή έχει την ευθύνη του υπολογισμού του ταχέως μετασχηματισμού Fourier σε ένα συγκεκριμένο block ηχητικών δειγμάτων. Η ανάγκη για την εκτέλεση του υπολογισμού αυτού σε πραγματικό χρόνο και με βάση τον ρυθμό συλλογής των ηχητικών δειγμάτων από το μικρόφωνο, υπαγορεύουν την υλοποίησή της σε Objective-C++. Συγκεκριμένα ο υπολογισμός του FFT πραγματοποιείται, χρησιμοποιώντας ένα σύνολο μεθόδων που

παρέχονται από το Accelerate Framework. Η βασική λογική για την γρήγορη εκτέλεση αυτού του υπολογισμού είναι η αναγωγή του σε ένα σύνολο πράξεων μεταξύ διανυσμάτων και η αποφυγή χρήσεων επαναληπτικών δομών. Η κλάση FFTCore αναλαμβάνει την δημιουργία κατάλληλων προγραμματιστικών δομών, όπως αυτές ορίζονται στο Accelerate Framework, ώστε να καταστεί δυνατή η εκτέλεση του υπολογισμού και η ανάκτηση των αποτελεσμάτων του.

SPL

Σκοπός της κλάσης αυτής είναι η εκτίμηση της τρέχουσας τιμής της ηχητικής στάθμης από το σύνολο των στιγμιαίων τιμών που υπολογίζονται για κάθε block ηχητικών δειγμάτων. Η κλάση SPL δέχεται ως είσοδο την συχνοτική απόκριση του φίλτρου στάθμισης που χρησιμοποιείται από την κλάση WeightedFiltering και τους συντελεστές Fourier που αντιστοιχούν στα ηχητικά δείγματα του κάθε block από την κλάση FFT. Η SPL έχει επιπλέον την ευθύνη του υπολογισμού της μέσης, της μέγιστης και της ελάχιστης τιμής του επιπέδου της ηχητικής πίεσης με βάση τις τρέχουσες τιμές που καταγράφονται. Το σύνολο των εμπλεκόμενων υπολογισμών και σε αυτή την κλάση γίνεται με χρήση του Accelerate Framework για λόγους ταχύτητας.

GraphBar

Σκοπός της κλάσης αυτής είναι η γραφική απεικόνιση της τρέχουσας τιμής του επιπέδου ηχητικής πίεσης. Η κλάση GraphBar δέχεται ως είσοδο την τιμή του τρέχοντος SPL από την κλάση SoundManager, την οποία απεικονίζει τελικά με μία μπάρα (bar) παράλληλη προς τον άξονα y, μεταβάλλοντας αναλόγως το ύψος της, κάθετα στον άξονα x. Μεταξύ των κλάσεων SoundManager και GraphBar υπάρχει απλή κατευθυνόμενη συσχέτιση από την πρώτη προς την δεύτερη, καθώς μόνο η κλάση SoundManager μπορεί να προσπελάσει το αντικείμενο της GraphBar. Η πληθυκότητα της συσχέτισης αυτής είναι 1-1 καθώς υπάρχει ένα μόνο αντικείμενο από κάθε κλάση, ανά πάσα στιγμή εκτέλεσης του προγράμματος.

GraphPulse

Σκοπός της κλάσης αυτής είναι η γραφική απεικόνιση της τρέχουσας τιμής του επιπέδου ηχητικής πίεσης. Η κλάση GraphPulse δέχεται ως είσοδο την τρέχουσα και την ελάχιστη τιμή του SPL από την κλάση SoundManager, την οποία απεικονίζει τελικά με μία μπάρα (bar) παράλληλη προς τον άξονα y, μεταβάλλοντας αναλόγως το ύψος της, κάθετα στον άξονα x. Το ύψος της μπάρας υπολογίζεται ως η διαφορά μεταξύ της τρέχουσας και της ελάχιστης τιμής του SPL. Μεταξύ των κλάσεων SoundManager και GraphPulse υπάρχει απλή κατευθυνόμενη συσχέτιση από την πρώτη προς την δεύτερη, καθώς μόνο η κλάση SoundManager μπορεί να προσπελάσει το αντικείμενο της GraphPulse. Η πληθυκότητα της

συσχέτισης αυτής είναι 1-1 καθώς υπάρχει ένα μόνο αντικείμενο από κάθε κλάση, ανά πάσα στιγμή εκτέλεσης του προγράμματος.

GraphVisualizer

Σκοπός της κλάσης αυτής είναι η γραφική απεικόνιση της τρέχουσας τιμής του επιπέδου ηχητικής πίεσης. Η κλάση GraphVisualizer δέχεται ως είσοδο την τιμή του τρέχοντος SPL από την κλάση SoundManager, την οποία απεικονίζει τελικά με μία συνεχή γραμμή που ενώνει τις διαδοχικές τιμές του τρέχοντος SPL. Μεταξύ των κλάσεων SoundManager και GraphVisualizer υπάρχει απλή κατευθυνόμενη συσχέτιση από την πρώτη προς την δεύτερη, καθώς μόνο η κλάση SoundManager μπορεί να προσπελάσει το αντικείμενο της GraphVisualizer. Η πληθυκότητα της συσχέτισης αυτής είναι 1-1 καθώς υπάρχει ένα μόνο αντικείμενο από κάθε κλάση, ανά πάσα στιγμή εκτέλεσης του προγράμματος.

NeedleView

Σκοπός της κλάσης αυτής είναι η απεικόνιση της τρέχουσας τιμής του επιπέδου ηχητικής πίεσης. Η κλάση NeedleView δέχεται ως είσοδο την τιμή του τρέχοντος SPL από την κλάση SoundManager, την οποία απεικονίζει τελικά με την βοήθεια μια βελόνας, προσομοιώνοντας μια αναλογική ένδειξη. Η γωνία στροφής της βελόνας αλλάζει ανάλογα με την τιμή του τρέχοντος SPL. Μεταξύ των κλάσεων SoundManager και NeedleView υπάρχει απλή κατευθυνόμενη συσχέτιση από την πρώτη προς την δεύτερη, καθώς μόνο η κλάση SoundManager μπορεί να προσπελάσει το αντικείμενο της NeedleView. Η πληθυκότητα της συσχέτισης αυτής είναι 1-1 καθώς υπάρχει ένα μόνο αντικείμενο από κάθε κλάση, ανά πάσα στιγμή εκτέλεσης του προγράμματος.

6.7 Προγραμματιστικές Διεπαφές

Υπηρεσίες Ηχητικών Ουρών (Audio Queue Services)

Οι υπηρεσίες ηχητικών ουρών παρέχουν έναν ευθύ και χαμηλής υπολογιστικής επιβάρυνσης τρόπο για την καταγραφή και αναπαραγωγή του ήχου στα λειτουργικά συστήματα iOS και MAC OS X. Οι συγκεκριμένες υπηρεσίες αποτελούν την προτεινόμενη τεχνολογία που θα πρέπει να χρησιμοποιήσει κανείς, προκειμένου να ενσωματώσει βασικές λειτουργίες ηχογράφησης και αναπαραγωγής σε εφαρμογές που εκτελούνται στα παραπάνω λειτουργικά συστήματα. Οι υπηρεσίες ηχητικών ουρών επιτρέπουν την καταγραφή και αναπαραγωγή του ήχου σε οποιοδήποτε από τους παρακάτω τύπους διαμόρφωσης:

- Γραμμική Παλμοκωδική Διαμόρφωση (Linear PCM).

- Όλοι οι τύποι συμπιεσμένης διαμόρφωσης, που υποστηρίζεται εγγενώς από την πλατφόρμα της Apple.
- Κάθε τύπος διαμόρφωσης για τον οποίο έχει εγκατασταθεί ο αντίστοιχος codec.

Οι παρεχόμενες υπηρεσίες συνιστούν υψηλού επιπέδου διεπαφές για την διαχείριση των συσκευών του υλικού που είναι υπεύθυνες για την καταγραφή και αναπαραγωγή του ήχου (π.χ. μικρόφωνα και ηχεία) χωρίς να γνωρίζουν τις αντίστοιχες διεπαφές του υλικού. Επιπλέον παρέχουν την δυνατότητα χρήσης εξειδικευμένων codecs, χωρίς να γνωρίζουν τον ακριβή τρόπο λειτουργίας τους. Οι υπηρεσίες ηχητικών ουρών αποτελούν μια διεπαφή γραμμένη αποκλειστικά σε C για την ανάπτυξη cocoa εφαρμογών και γενικότερα εφαρμογών σε MAC OS X.

Μία ηχητική ουρά είναι ένα αντικείμενο λογισμικού το οποίο χρησιμοποιείται για την καταγραφή και αναπαραγωγή του ήχου στα λειτουργικά συστήματα iOS και MAC OS X. Το συγκεκριμένο αντικείμενο λογισμικού αναπαρίσταται μέσω της κλάσης AudioQueueRef η οποία είναι δηλωμένη στο αρχείο κεφαλίδας AudioQueue.h. Μία ηχητική ουρά είναι υπεύθυνη για τις παρακάτω εργασίες:

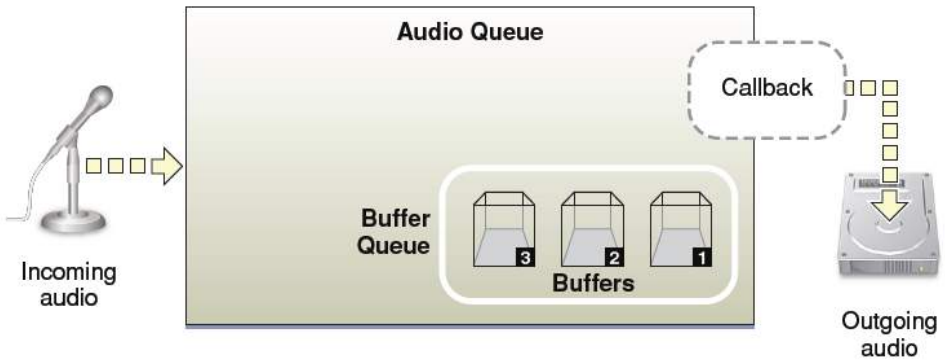
- Σύνδεση με τις συσκευές ήχου
- Διαχείριση Μνήμης
- Εφαρμογή κατάλληλων codec ανάλογα με τον τύπο συμπίεσης
- Διαμεσολάβηση ηχογράφησης ή αναπαραγωγής

Όλες οι ηχητικές ουρές έχουν την ίδια γενική δομή, η οποία αποτελείται από τα εξής μέρη:

- Ένα σύνολο από Audio Queue Buffers καθένα από τα οποία αποτελεί μια προσωρινή αποθήκη για λαμβανόμενα ηχητικά δεδομένα
- Μία Buffer Queue η οποία αποτελεί μία διατεταγμένη λίστα από Audio Queue Buffers
- Μία μέθοδο Audio Queue Callback η υλοποίηση της οποίας ανήκει στην ευθύνη του προγραμματιστή

Η αρχιτεκτονική των ηχητικών ουρών ποικίλει ανάλογα με το εάν αυτή χρησιμοποιείται για ηχογράφηση ή για αναπαραγωγή. Οι διαφορές τους εντοπίζονται κυρίως στον τρόπο σύνδεσης μεταξύ της εισόδου και της εξόδου της ηχητικής ουράς καθώς και στον ρόλο που επιτελεί η μέθοδος Callback.

Η δομή μιας ηχητικής ουράς για την καταγραφή ήχου φαίνεται στο παρακάτω σχήμα:

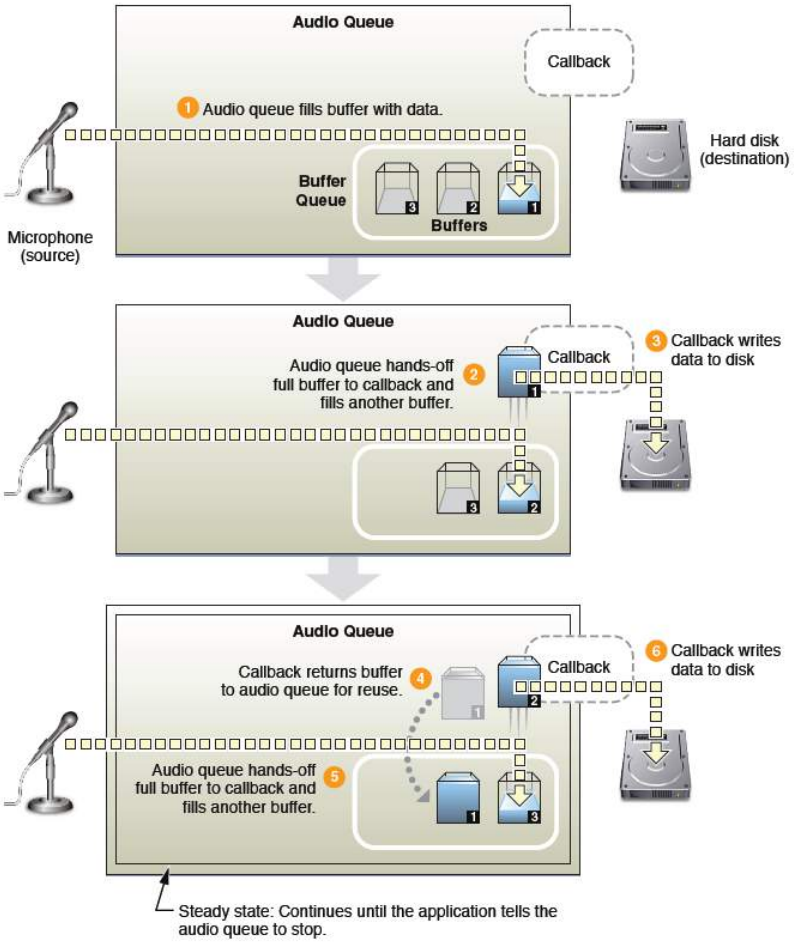


Σχήμα 51: Audio Queue

Η είσοδος μιας ουράς ηχογράφησης συνδέεται συνήθως με εξωτερικές ηχητικές συσκευές, όπως για παράδειγμα το εσωτερικό μικρόφωνο. Η έξοδος της ουράς είναι αυτή που χρησιμοποιεί την μέθοδο callback που υλοποιεί ο προγραμματιστής. Στην περίπτωση που η ηχογράφηση αποθηκεύεται στον δίσκο, η μέθοδος callback αποθηκεύει τα νέα ηχητικά δεδομένα σε buffers όπως τα λαμβάνει από την ηχητική ουρά σε ένα αρχείο ήχου. Κάθε ηχητική ουρά, είτε πρόκειται για ηχογράφηση, είτε για αναπαραγωγή περιλαμβάνει ένα ή περισσότερα audio queue buffers τα οποία είναι διατεταγμένα σε μια συγκεκριμένη σειρά που καλείται buffer queue. Στην παραπάνω εικόνα τα audio queue buffers είναι αριθμημένα σύμφωνα με την σειρά που γεμίζονται, η οποία είναι η ίδια με την σειρά με την οποία παραδίδονται για χρήση από την μέθοδο callback.

Ένα audio queue buffer είναι μια δομή δεδομένων που αντιστοιχεί στον τύπο AudioQueueBuffer ο οποίος δηλώνεται στο αρχείο κεφαλίδας AudioQueue.h. Μια εφαρμογή μπορεί να περιλαμβάνει έναν οποιοδήποτε αριθμό από buffers, ωστόσο μια τυπική επιλογή είναι τα 3 buffers. Στην περίπτωση αυτή είναι δυνατό το ένα buffer να είναι επιφορτισμένο με την καταγραφή των δεδομένων στον δίσκο, ενώ κάποιο άλλο να δέχεται τα νέα ηχητικά δεδομένα. Με τον τρόπο αυτό το τρίτο buffer είναι διαθέσιμο προκειμένου να αντιμετωπιστούν προβλήματα καθυστέρησης που σχετίζονται με λειτουργίες εισόδου και εξόδου. Το πλήθος των buffers εξαρτάται από την υπολογιστική ισχύ της συσκευής στην οποία εκτελείται η εφαρμογή. Όσο μικρότερη είναι η υπολογιστική ισχύς της συσκευής τόσο μεγαλύτερο πρέπει να είναι το πλήθος των buffers ώστε να μεγαλώνει ο χρόνος – κύκλος επαναχρησιμοποίησης του ίδιου buffer. Στην συγκεκριμένη υλοποίηση επιλέχθηκαν 5 buffers, λόγω της ανάγκης εκτέλεσης μιας σειράς πολύ γρήγορων υπολογισμών.

Κατά την ηχογράφηση ένα audio queue buffer γεμίζει με δεδομένα ήχου που συλλέγονται από την συσκευή εισόδου, ενώ τα υπόλοιπα buffers στοιχίζονται πίσω από το τρέχον αναμένοντας την σειρά τους για να γεμιστούν. Η ηχητική ουρά παραδίδει τα γεμισμένα buffer στην μέθοδο callback με την σειρά με την οποία τα συνέλεξε. Οι παρακάτω εικόνες περιγράφουν την σχετική διαδικασία.



Σχήμα 52: Λειτουργία Audio Queue

Στο πρώτο βήμα, η ηχογράφηση ξεκινά και η audio queue γεμίζει ένα buffer με τα δεδομένα που συνέλεξε. Στο δεύτερο βήμα, το πρώτο buffer είναι γεμάτο και η audio queue καλεί την μέθοδο callback παραδίδοντάς της το γεμισμένο buffer (buffer 1). Στο βήμα 3 η μέθοδος callback καταχωρεί τα δεδομένα του buffer σε ένα αρχείο ήχου και την ίδια στιγμή η Audio queue ξεκινά να γεμίζει ένα άλλο buffer (buffer 2) με νέα δεδομένα. Στο βήμα 4 η μέθοδος callback ελευθερώνει το buffer 1 στοιχίζοντάς το στην αρχική του θέση προκειμένου να χρησιμοποιηθεί εκ νέου. Στο βήμα 5 η audio queue καλεί και πάλι την μέθοδο callback Παραδίδοντάς της το επόμενο buffer (buffer 2). Κατά το βήμα 6 η μέθοδος callback γράφει τα περιεχόμενα αυτού του buffer στο αρχείο ήχου. Αυτή η επαναληπτική διαδικασία θα συνεχίζεται μέχρι ο χρήστης να διακόψει την ηχογράφηση. Η μέθοδος

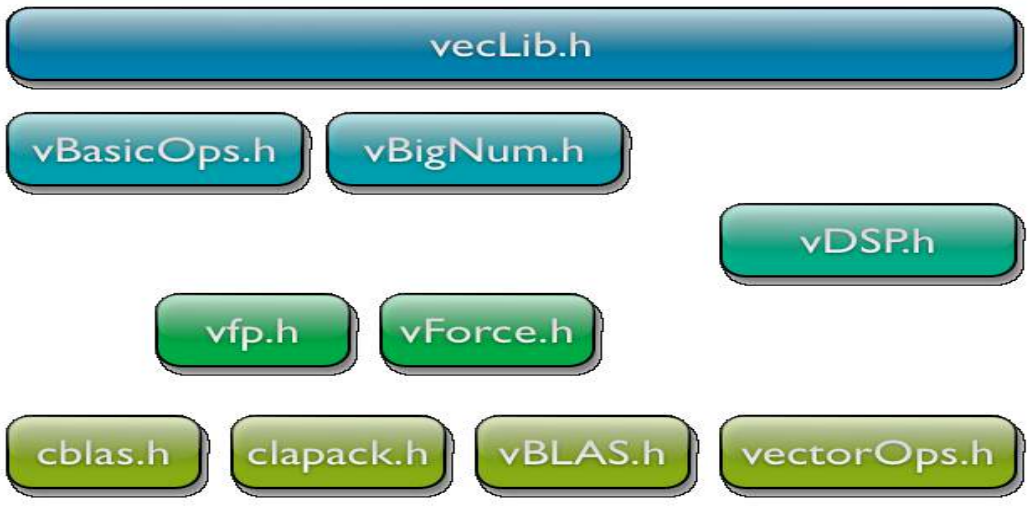
callback είναι αυτή που έχει το περισσότερο προγραμματιστικό φορτίο κατά την χρήση των υπηρεσιών ηχητικών ουρών. Κατά την διάρκεια της ηχογράφησης η Audio queue callback καλείται επαναλαμβανόμενα από την audio queue που έχει την κυριότητα της. Ο χρόνος που μεσολαβεί μεταξύ δύο διαδοχικών κλήσεων εξαρτάται από την χωρητικότητα των Audio queue buffers η οποία κυμαίνεται συνήθως από μισό έως μερικά δευτερόλεπτα.

Διεπαφή Ψηφιακής Επεξεργασίας Σήματος (vDSP - API)

Η διεπαφή vDSP παρέχει μαθηματικές συναρτήσεις για την ανάπτυξη εφαρμογών που σχετίζονται με την επεξεργασία φωνής, ήχου, βίντεο, διαγνωστικής ιατρικής απεικόνισης, σημάτων ραντάρ, σεισμικής ανάλυσης και επιστημονικής επεξεργασίας δεδομένων. Οι συναρτήσεις της διεπαφής, λειτουργούν τόσο σε πραγματικούς όσο και σε μιγαδικούς τύπους δεδομένων. Μεταξύ των παρεχόμενων συναρτήσεων συμπεριλαμβάνονται συναρτήσεις για την μετατροπή του τύπου των δεδομένων, υπολογισμού του ταχέως μετασχηματισμού Fourier (FFT) και λειτουργίες τύπου διάνυσμα προς διάνυσμα και διάνυσμα προς βαθμωτό. Οι συναρτήσεις αυτές υλοποιούνται με δύο τρόπους:

- μέσω διανυσματικού κώδικα, ο οποίος χρησιμοποιεί διανυσματικές εντολές προς τον επεξεργαστή
- μέσω βαθμωτού κώδικα, ο οποίος δεν κάνει χρήση τέτοιων εντολών

Η διεπαφή vDSP χρησιμοποιεί τον καταλληλότερο τρόπο, ανάλογα με τα δοσμένα ορίσματα και τις διανυσματικές δυνατότητες του επεξεργαστή. Η δήλωση της βρίσκεται στο αρχείο κεφαλίδας vDSP.h όπου ορίζονται οι χρησιμοποιούμενοι τύποι των δεδομένων και τα αποδεκτά σύμβολα για τις παρεχόμενες συναρτήσεις. Όπως φαίνεται και στο παρακάτω σχήμα, αποτελεί τμήμα της βιβλιοθήκης vecLib.h η οποία με την σειρά της είναι τμήμα του



Σχήμα 53: vecLib

Accelerate Framework του MAC OS X.

Η ενσωμάτωση των συγκεκριμένων συναρτήσεων θα πρέπει να γίνεται με την δήλωση #include <Accelerate/Accelerate.h>. Οι περισσότερες συναρτήσεις της vDSP λειτουργούν δεχόμενες δεδομένα εισόδου για τα οποία παράγουν τα αντίστοιχα δεδομένα εξόδου. Το σύνολο αυτών των ορισμάτων είτε πρόκειται για διανυσματικά, είτε για βαθμωτά δεδομένα, περνούν προς τις συναρτήσεις αυτές μέσω αναφορών (by reference). Άλλοι τύποι ορισμάτων που περνούν προς τις συναρτήσεις είναι οι παρακάτω:

- address strides, που καθορίζουν το βήμα επεξεργασίας των δεδομένων εισόδου και εξόδου.
- flags, που επηρεάζουν τον τρόπο λειτουργίας των συναρτήσεων
- element counts, που καθορίζουν το πλήθος των στοιχείων προς επεξεργασία

Η διεπαφή vDSP χρησιμοποιήθηκε για τον υπολογισμό του FFT που αντιστοιχεί στα εισερχόμενα δείγματα του ήχου καθώς και για τον υπολογισμό των φίλτρων στάθμισης. Η χρήση της κρίθηκε αναγκαία, λόγω της απαίτησης πολύ γρήγορων υπολογισμών στην μονάδα του χρόνου.

7 Η εφαρμογή iSPL PRO

Η εφαρμογή που αναπτύχθηκε στα πλαίσια της πτυχιακής εργασίας ονομάσθηκε iSPL PRO. Το όνομά της είναι μια σύνθεση του i, που παραπέμπει στο iOS, του SPL, που είναι τα αρχικά του Sound Pressure Level, και του PRO από το professional.

Σχήμα 54: Εικονίδιο iSPL PRO

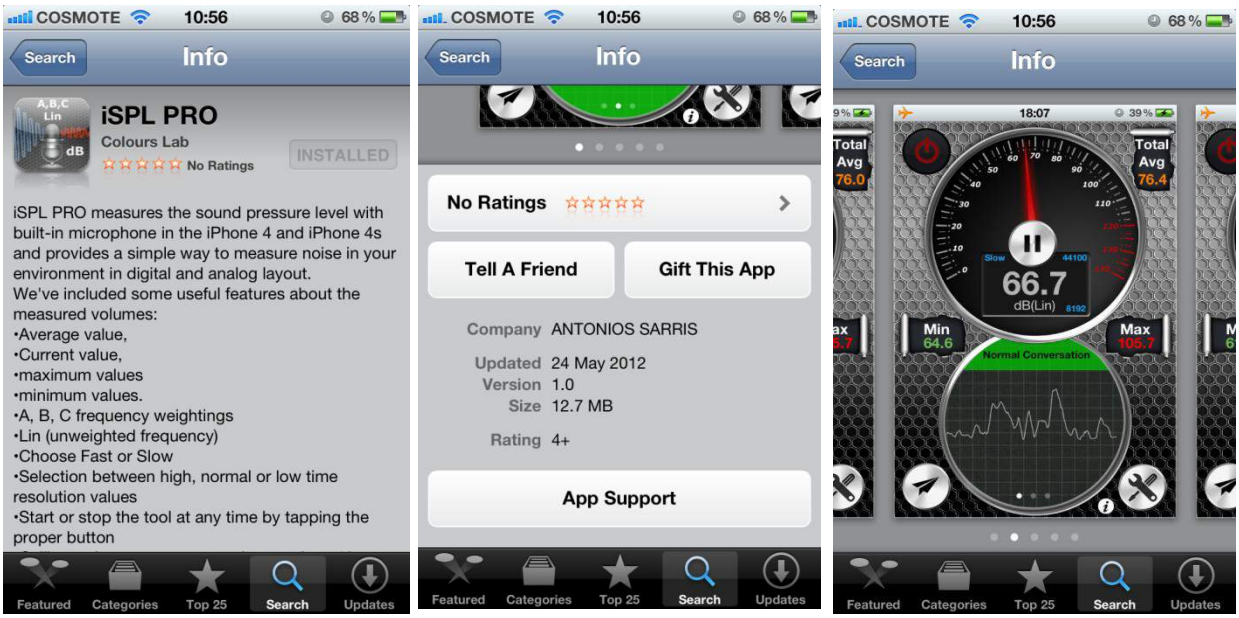
7.1 Εγκατάσταση εφαρμογής

Για την εγκατάσταση της εφαρμογής σε μια συσκευή με λειτουργικό iOS πρέπει να γίνει αίτηση για την διάθεση αυτής από το App Store. Το App Store είναι ένα ηλεκτρονικό κατάστημα στο οποίο έχουν πρόσβαση όλες οι iOS συσκευές. Το μόνο που έχει να κάνει ο χρήστης είναι να επιλέξει το εικονίδιο του App Store (Σχ. 55) από τη συσκευή του, έχοντας την απαραίτητη σύνδεση με το internet. Στη συνέχεια επιλέγει την καρτέλα "Search" όπου πληκτρολογεί το όνομα της εφαρμογής iSPL PRO και κάνοντας αναζήτηση παρουσιάζονται

Σχήμα 56: Εικονίδιο App Store

Σχήμα 56: Βήματα αναζήτησης της εφαρμογής

όλες οι σχετικές με τον τίτλο εφαρμογές, κάθε εφαρμογή που έχει και ένα αντιπροσωπευτικό εικονίδιο. Χαρακτηριστικές εικόνες από την διαδικασία αυτή παρουσιάζονται στο Σχ. 56. Επιλέγοντας την εφαρμογή εμφανίζονται κάποιες πληροφορίες σχετικές με αυτήν (Σχ. 57). Συγκεκριμένα υπάρχει μια βασική περιγραφή της και μια σύντομη αναφορά στις λειτουργίες της. Επιπλέον δίνεται η δυνατότητα στον χρήστη να δει πέντε printscreen της εφαρμογής. Με αυτό τον τρόπο ο χρήστης μπορεί να έχει μια πρώτη εντύπωση για την εφαρμογή που προτίθεται να αγοράσει πριν αυτή εγκατασταθεί στη συσκευή του. Όταν ο χρήστης αποφασίσει ότι θέλει να αγοράσει την εφαρμογή επιλέγει το κατάλληλο κουμπί και η εφαρμογή εγκαθίσταται στη συσκευή του.



Σχήμα 57: Παρουσίαση εφαρμογής στο App Store

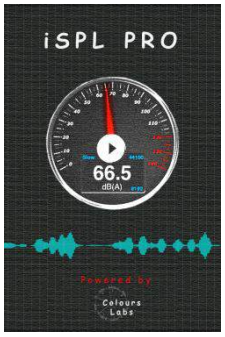
7.2 Χρήση της εφαρμογής

Η εφαρμογής iSPL PRO αποτελείται από τρείς βασικές οθόνες, την αρχική (οθόνη μετρήσεων), την οθόνη επιλογών – ρυθμίσεων και την οθόνη πληροφοριών.

7.2.1 Αρχική οθόνη

Όταν επιλεγεί το εικονίδιο της εφαρμογής (Σχ. 54) και μέχρι να φορτωθεί το πρόγραμμα εμφανίζεται η οθόνη εκκίνησης (splash screen) (Σχ. 58). Η αρχική οθόνη είναι η πρώτη οθόνη που βλέπει ο χρήστης μετά την φόρτωση του προγράμματος. Η αρχική οθόνη αποτελείται από πέντε εμφανή κουμπιά, τρεις ξεχωριστές ετικέτες και τέσσερις κατηγορίες παρουσίασης μετρήσεων.

Τα κουμπιά της αρχικής οθόνης είναι:



Σχήμα 58: Οθόνη εκκίνησης (splash screen)


Κουμπί - ένδειξη λειτουργίας μέτρησης (ON/OFF)


Κουμπί εκκίνησης / παύσης μέτρησης (Play / Pause)


Κουμπί ρυθμίσεων


Κουμπί αποστολής μετρήσεων με email


Κουμπί πληροφοριών

Κουμπί - ένδειξη λειτουργίας μέτρησης (ON/OFF): Το κουμπί αυτό όταν έχει την φωτεινή κόκκινη ένδειξη υποδεικνύει ότι το σύστημα είναι σε κατάσταση μέτρησης. Ο χρήστης έχει την δυνατότητα πατώντας το να σταματήσει την μέτρηση και το κουμπί παύει να είναι φωτεινό.

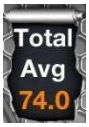
Κουμπί εκκίνησης / παύσης μέτρησης (Play / Pause): Ο χρήστης αρχικά έχει την δυνατότητα πατώντας το κουμπί εκκίνησης να ξεκινήσει τις μετρήσεις. Με το πάτημα, το κουμπί μετατρέπεται από κουμπί εκκίνησης σε παύσης. Όταν ο χρήσης πατήσει το κουμπί της παύσης σταματούν προσωρινά οι μετρήσεις, μέχρι ο χρήστης να ξαναπατήσει το κουμπί εκκίνησης για την συνέχιση της μέτρησης.

Κουμπί ρυθμίσεων: Το κουμπί αυτό όταν πατηθεί μεταφέρει τον χρήστη από την οθόνη εκκίνησης στην οθόνη ρυθμίσεων.

Κουμπί αποστολής μετρήσεων με email: Τι κουμπί αυτό δίνει την δυνατότητα στον χρήστη να αποστείλει σε κάποιο email τις τελευταίες μετρήσεις που κατέγραψε το σύστημα. Οι μετρήσεις αυτές έχουν καταγραφεί από το σύστημα σε ένα txt αρχείο το οποίο επισυνάπτεται στο email.

Κουμπί πληροφοριών: Το κουμπί αυτό όταν πατηθεί μεταφέρει τον χρήστη από την οθόνη εκκίνησης στην οθόνη πληροφοριών. Με άγγιγμα στο πάνω αριστερό μέρος της οθόνης ο χρήστης επανέρχεται στην αρχική οθόνη .

Οι ετικέτες είναι:



Ένδειξη συνολικού μέσου όρου

Ένδειξη ελάχιστου

Ένδειξη μέγιστου

Ένδειξη συνολικού μέσου όρου: Εμφανίζεται ο μέσος όρος όλων των επιμέρους τιμών.

Ένδειξη ελάχιστου: Εμφανίζεται η μικρότερη τιμή που έχει καταγραφεί από το σύστημα.

Ένδειξη μέγιστου: Εμφανίζεται η μεγαλύτερη τιμή που έχει καταγραφεί από το σύστημα.

Οι τρείς κατηγορίες παρουσίασης των αποτελεσμάτων είναι η ψηφιακή, η αναλογική, η γραφική και η λεκτική / περιγραφική.

Στη ψηφιακή απεικόνιση των μετρήσεων εμφανίζεται σε μια οθόνη (display) η τιμή της στιγμιαίας ηχοστάθμης σε dB, σε ψηφιακή μορφή και με ακρίβεια ενός δεκαδικού ψηφίου.

Στην συγκεκριμένη απεικόνιση εμφανίζονται στο χρήστη και κάποιες επιπλέον πληροφορίες για την μέτρηση σε μπλε χρώμα. Οι πληροφορίες αυτές είναι:

Fast/Slow	Ένδειξη αργής/γρήγορης μέτρησης
Sampling Rate	Ένδειξη ρυθμού δειγματοληψίας
Blocks	Ένδειξη πλήθους των blocks της μέτρησης

Σχήμα 59: Πληροφορίες μέτρησης

Στη αναλογική απεικόνιση των μετρήσεων εμφανίζεται η τιμή της στιγμιαίας ηχοστάθμης σε dB, με την βοήθεια ενός αριθμημένου καντράν και μιας κινητής βελόνας για την ένδειξη.

Στη γραφική απεικόνιση των μετρήσεων, ο χρήστης μπορεί να επιλέξει μεταξύ τριών (3) γραφικών παραστάσεων. Η μετακίνηση ανάμεσα στις γραφικές παραστάσεις πραγματοποιείται με την βοήθεια του Scroll View, ενός χαρακτηριστικού του iOS, όπου ο χρήστης μπορεί με μια απλή κίνηση με τα δάκτυλά του (είτε δεξιά, είτε αριστερή), να αλλάξει την γραφική απεικόνιση της μέτρησης, χωρίς αυτή να διακοπεί. Η πρώτη γραφική παράσταση απεικονίζει μια συνεχή γραμμή που ενώνει τις διαδοχικές τιμές του τρέχοντος SPL. Η δεύτερη γραφική παράσταση απεικονίζει μια μπάρα παράλληλη προς τον άξονα y μεταβάλλοντας αναλόγως το ύψος της κάθετα στον άξονα x. Τέλος η τρίτη γραφική παράσταση απεικονίζει μια μπάρα παράλληλη προς τον άξονα y, μεταβάλλοντας αναλόγως το ύψος της κάθετα στον άξονα x. Το ύψος της μπάρας υπολογίζεται ως η διαφορά μεταξύ της τρέχουσας και της ελάχιστης τιμής του SPL.

Η λεκτική απεικόνιση των μετρήσεων εμφανίζεται μόνο όταν οι μετρήσεις γίνονται χωρίς στάθμιση (Linear). Ανάλογα με την τιμή της στιγμιαίας ηχοστάθμης σε dB (Lin)

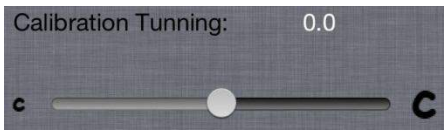
Σχήμα 60: Στιγμιότυπα αρχικής οθόνης

εμφανίζεται λεκτικά αλλά και χρωματικά η τυπική μορφή θορύβου, σύμφωνα με τον πίνακα που εμφανίζεται στην οθόνη πληροφοριών.

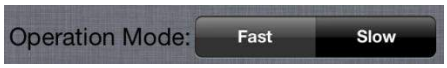
7.2.2 Οθόνη επιλογών - ρυθμίσεων

Η οθόνη επιλογών – ρυθμίσεων δίνει τη δυνατότητα στο χρήστη να αλλάξει κάποιες παραμέτρους του συστήματος. Οι επιλογές αυτές παρουσιάζονται στον παρακάτω πίνακα.

Επιλογές ακύρωσης / αποθήκευσης



Βαθμονόμηση (±10)



Λειτουργία καταγραφής Fast/ slow

Επιλογή στάθμισης

Επιλογή μεγέθους block μέτρησης

Επαναφορά στις αρχικές ρυθμίσεις

Επιλογές ακύρωσης / αποθήκευσης: Ο χρήστης για να αποθηκεύσει τις αλλαγές που θα πραγματοποιήσει σε αυτή την οθόνη, θα πρέπει να επιλέξει το πλήκτρο Done το οποίο αποθηκεύει τις αλλαγές και επαναφέρει τον χρήστη στην αρχική οθόνη. Σε διαφορετική περίπτωση μπορεί να πατήσει το πλήκτρο Cancel επιστρέφοντας στην αρχική οθόνη χωρίς να έχει γίνει η αποθήκευση των τελευταίων επιλογών.

Βαθμονόμηση (±10): Ο χρήστης στην περίπτωση που συγκρίνει τις ενδείξεις με κάποιο ηχόμετρο και παρατηρήσει διαφορές μπορεί να βαθμονομήσει την εφαρμογή έχοντας τη δυνατότητα αλλαγής από -10 έως + 10. Η εφαρμογή όμως έχει ήδη βαθμονομηθεί με τη βοήθεια του ηχομέτρου BRUEL & KJAER, TYPE 2250 Light.

Λειτουργία καταγραφής Fast/ slow: Η επιλογή της καταγραφής fast ή slow αφορά στην γρήγορή ή την αργή απόκριση του συστήματος. Έτσι για ένα στιγμιαίο κρότο κατάλληλη είναι η επιλογή fast ενώ στη αντίθετη περίπτωση για ένα παρατεταμένο θόρυβο κατάλληλη είναι η επιλογή slow.

Επιλογή στάθμισης: Ο χρήστης μπορεί να επιλέξει το κατάλληλο φίλτρο στάθμισης μεταξύ των A, B, C, οι μετρήσεις των οποίων θα γίνονται σε dB(A), dB(B) και dB(C) αντίστοιχα. Επίσης έχει την δυνατότητα να μετρήσει χωρίς στάθμιση σε dB(Lin).

Επιλογή μεγέθους block μέτρησης: Η επιλογή αυτή μπορεί να καθορίσει με λεπτομέρεια τον αριθμό των block υπολογισμού που επιθυμεί ο χρήστης σε κάθε slow ή fast μέτρηση.

Έτσι ο χρήστης έχει τη δυνατότητα εκτός από την επιλογή fast(Normal) και slow(Normal), να έχει τις επιλογές fast(High) και slow(High), fast(Low) και slow(Low). Οι ακριβείς επιλογές τόσο των blocks όσο και του χρόνου φαίνονται στον πίνακα 4 του κεφαλαίου 6.5.

Επαναφορά στις αρχικές ρυθμίσεις: Με επιλογή του κουμπιού ο χρήστης επαναφέρει το σύστημα στις αρχικές του ρυθμίσεις, οι οποίες παρουσιάζονται στον παρακάτω πίνακα:

Calibration Tuning	0
Operation Mode	Slow
Weighting Filter	A
Time Resolution	Normal

7.2.3 Οθόνη πληροφοριών

Η οθόνη πληροφοριών χωρίζεται σε τρεις επιμέρους καρτέλες. Την καρτέλα για την βοήθεια (Help), την καρτέλα τυπικών πηγών θορύβου (Typical noise) και την καρτέλα με τις πληροφορίες του συστήματος (About us).

Η καρτέλα βοήθειας περιέχει ένα σύντομο οδηγό για την λειτουργία του συστήματος. Η καρτέλα των τυπικών πηγών θορύβου περιέχει τον πίνακα με τις τυπικές πηγές θορύβου που χρησιμοποιεί το σύστημα. Ο πίνακας φαίνεται στο Σχ. 61.

Τέλος η τρίτη καρτέλα περιέχει πληροφορίες για τον προγραμματιστη καθώς και έναν κατάλογο των λειτουργιών του συστήματος.

Σχήμα 61: Στιγμιότυπα οθόνης πληροφοριών

8 Συγκριτικές Δοκιμές – Μετρήσεις Τυπικών Μορφών Θορύβου

Η εφαρμογή που αναπτύχθηκε για την μέτρηση της ηχοστάθμης χρησιμοποιήθηκε για να γίνουν μετρήσεις τυπικών μορφών θορύβου και σύγκριση των αποτελεσμάτων με εξειδικευμένες μετρητικές συσκευές. Στόχος των συγκριτικών δοκιμών που πραγματοποιήθηκαν ήταν η επαλήθευση της ορθής λειτουργίας του παρεχόμενου μετρητικού συστήματος και ο καθορισμός της ακρίβειάς του. Κύριο μέλημα ήταν η εξακρίβωση του κατά πόσο μπορεί κανείς να πραγματοποιήσει μετρήσεις ικανοποιητικής ακρίβειας με μια συσκευή χαμηλού κόστους.

Πίνακας 5: χαρακτηριστικά συναφών εφαρμογών

Όνομα Εφαρμογής	**Λειτουργία Fast/slow**	**Φίλτρα**	**Βαθμονόμηση**	**Καταγραφή & αποστολή email**	**Γραφική παράσταση**	**Βελόνα**	**Διάφορα**
dB Volume	Ναι	A, B, C, Lin	Όχι	Όχι	Ναι	Όχι	hold
Decibel 10	Sensitivity	Δεν γίνεται αναφορά	Ναι		Ναι. Λεκτική περιγραφή	Ναι	Max, Peak
Decibel Ultra	rate	C	Ναι	Ναι	2 γραφικές	Ναι	Peak, offset
Decibels	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι		Ναι	
Sound Level	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι		Ναι	
Audio Tool	Sensitivity	A, B, C, Lin	Ναι	Όχι		Ναι	Max, peak
Decibel Meter	Ναι	Δεν γίνεται αναφορά	Όχι	Όχι	Ναι	Όχι	peak
Noise meter EoE	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι		Ναι	Peak, eye on earth
Noise Meter	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι	Λεκτική περιγραφή	Όχι	Peak
UE SPL	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι		Ναι	Peak
SoundMeter	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι	Ναι	Όχι	Max, peak
Decibelite	Όχι	Δεν γίνεται αναφορά	Όχι	Όχι		Ναι	Max
Ηχόμετρο	Ναι	A	Όχι	Όχι	Ναι	Όχι	Peak

Η επιλογή της εφαρμογής που χρησιμοποιήθηκε για τις συγκριτικές μετρήσεις έγινε μετά από μελέτη των δυνατοτήτων των πιο διαδεδομένων εφαρμογών ηχομέτρησης στο App Store. Οι περισσότερες εφαρμογές δεν διευκρινίζουν αν οι μετρήσεις είναι με τη χρήση φίλτρων ή όχι και είναι προφανές ότι δεν μπορούν να χρησιμοποιηθούν για τη σύγκριση.

Από τις εφαρμογές που αναφέρουν το φίλτρο που χρησιμοποιούν υπάρχουν μόνο δύο (dB Volume, Audio Tool) οι οποίες δίνουν τη δυνατότητα στον χρήστη να επιλέξει ανάμεσα στα φίλτρα A, B, C και Lin (Πίνακας 5).

Από τις εφαρμογές αυτές μόνο η Audio Tool δίνει τη δυνατότητα βαθμονόμησης και καταχώρισης μέγιστης τιμής. Οι δύο αυτές δυνατότητες είναι ιδιαίτερα χρήσιμές για τις συγκριτικές δοκιμές. Η βαθμονόμηση χρησιμοποιήθηκε ώστε να βαθμονομηθεί η συσκευή με ένα πραγματικό ηχόμετρο και η μέγιστή τιμή βοήθησε στη σύγκριση διότι στην καταγραφή της τιμής δεν υπεισέρχεται ο χρήστης. Από τα παραπάνω γίνεται φανερό ότι θα χρησιμοποιηθεί η εφαρμογή Audio Tool για τις συγκριτικές δοκιμές.

8.1 Βαθμονόμηση – Αρχικές Μετρήσεις

Η βαθμονόμηση της εφαρμογής και μετρήσεις, πραγματοποιήθηκαν στις 12 Απριλίου 2012 στις εγκαταστάσεις της εταιρείας QAC – ΚΑΡΙΝΙΩΤΑΚΗΣ Κ. & ΣΙΑ ΕΕ (Ακουστικές Μετρήσεις – Δοκιμές – Πιστοποίηση). Το ηχόμετρο που χρησιμοποιήθηκε ως πρότυπο αναφοράς είχε σαφώς πολύ καλύτερα μετρολογικά χαρακτηριστικά (ακρίβεια, επαναληψιμότητα, διακριτική ικανότητα) και ήταν το BRUEL & KJAER, TYPE 2250 Light (το συγκεκριμένο ηχόμετρο είναι και αναλυτής συχνοτήτων). Οι προδιαγραφές του ηχομέτρου παρουσιάζονται παρακάτω [34]:

Σχήμα 62: Συγκριτικές δοκιμές στις εγκαταστάσεις της εταιρείας QAC

- ✓ Συμμόρφωση σύμφωνα με τα Εθνικά και διεθνή πρότυπα:
  - IEC61672 –1 (2002–05) Class 1.
  - IEC60651 (1979) plus Amendment 1 (1993–02) and Amendment 2 (2000–10), Type 1.
  - IEC60804 (2000–10), Type 1.
  - IEC61252, Electroacoustics – Specifications for Personal Sound Exposure Meters.
  - DIN 45657 (1997–07).
  - ANSIS1.4–1983 plus ANSI S1.4A–1985 Amendment, Type 1.

- ANSIS1.43–1997, Type 1.

✓ Όρια μέτρησης:

- Δυναμική περιοχή: καθαρό σήμα ήχου, στάθμιση A: 16,4 - 140 dB.
- Εύρος Δείκτης: Σύμφωνα με το IEC 60651, στάθμιση A: 23,9 dB έως 123 dB.
- Εύρος Γραμμικότητας: Σύμφωνα με το IEC 60804, στάθμιση A 21,8 dB έως 140 dB.
- Γραμμικό Εύρος λειτουργίας: Σύμφωνα με το IEC 61672, στάθμιση A: 1 kHz: 25,0 dB έως 140 dB.
- Peak C Εύρος: Σύμφωνα με το IEC 61672: 43,0 dB έως 143 dB.

✓ Μικρόφωνο

Ονομαστική ευαισθησία: 50 mV/Pa (corresponding to -26 dB re 1 V/Pa) ± 1.5 dB.

Χωρητικότητα: 14 pF (at 250 Hz).

✓ Οθόνη χειρισμού: Ασπρόμαυρη οθόνη αφής με δυνατότητα εμφάνισης μετρήσεων σε διάφορα μεγέθη.

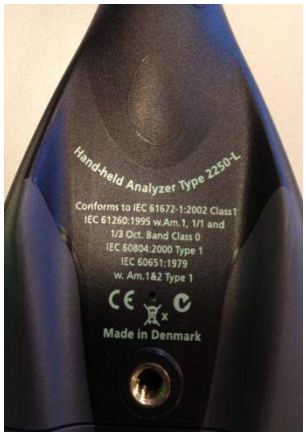
✓ A, C, Z Frequency Weighting.

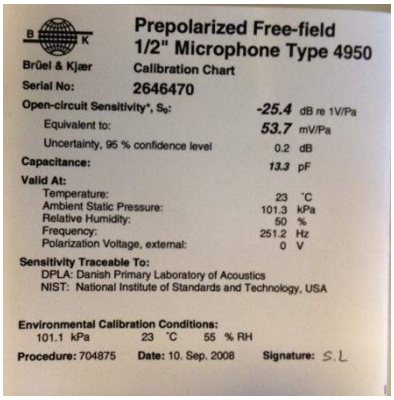
✓ Fast and Slow Time Weighting.

✓ Max/Min and Hold Functions.

✓ Auto Shut-Off Feature.

✓ Περιέχει: Wind Shield, Carrying Case and Battery.





Σχήμα 63: Τεχνικά χαρακτηριστικά ηχομέτρου, μικροφώνου και βαθμονόμηση ηχομέτρου

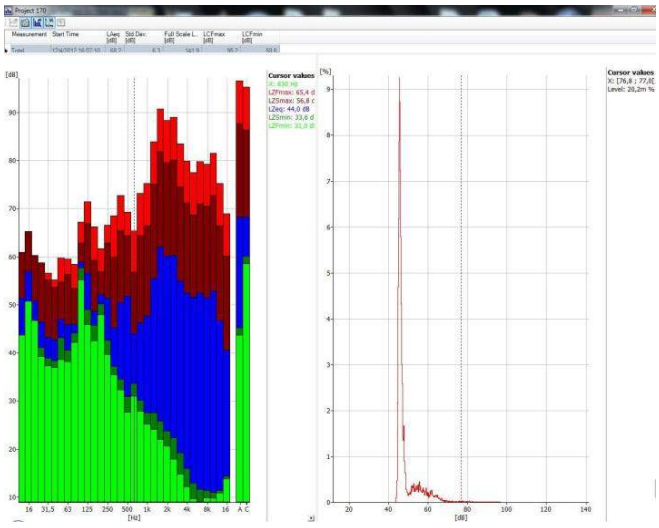
Λόγω του ότι το ηχόμετρο δεν μπορούσε να χρησιμοποιηθεί εκτός του Εργαστηρίου, και για να γίνουν αξιόπιστες συγκριτικές δοκιμές και σε άλλους χώρους, βαθμονομήθηκε και η μετρητική εφαρμογή (AudioTool) η οποία είχε εγκατασταθεί σε συσκευή κινητής τηλεφωνίας iPhone 4S. Η εφαρμογή που δημιουργήθηκε στα πλαίσια αυτής της εργασίας είχε εγκατασταθεί σε συσκευή κινητής τηλεφωνίας iPhone 4.

Για την βαθμονόμηση των εφαρμογών που ήταν εγκατεστημένα στις συσκευές κινητής τηλεφωνίας ακολουθήθηκαν τα παρακάτω βήματα:

1. Τοποθετήθηκαν οι συσκευές δίπλα στο ηχόμετρο ώστε να έχουν την ίδια απόσταση από την πηγή παραγωγής σήματος.
2. Παράχθηκε τόνος συχνότητας 1kHz.
3. Καταγράφηκαν οι ενδείξεις του ηχομέτρου και των συσκευών.
4. Υπολογίσθηκε από τις διαφορές των ενδείξεων η τιμή της σταθεράς βαθμονόμησης.

Λόγω του ότι δεν υπήρχε διαθέσιμος ανηχοϊκός θάλαμος η συγκεκριμένη πειραματική διαδικασία ολοκληρώθηκε σχετικά σύντομα ώστε να ελαχιστοποιηθεί η πιθανότητα λανθασμένης μέτρησης λόγω κάποιου εξωτερικού θορύβου που απροσδόκητα θα μπορούσε να συμβαίνει κατά τη διάρκεια των μετρήσεων.

Στο χώρο του Εργαστηρίου, για να εξακριβωθεί η ορθότητα και η επαναληψημότητα των μετρήσεων της εφαρμογής με την χρήση διαφορετικών φίλτρων και διαφορετικών χρονικών παραθύρων πραγματοποιήθηκαν 6 δοκιμές διάρκειας 2 λεπτών η καθεμία. Τα συγκεντρωτικά δεδομένα της καταγραφής του ηχομέτρου αποθηκεύονται στη μνήμη του και με το πρόγραμμα BZ-5503 Utility software for hand-held Analyzers μεταφέρονται στον υπολογιστή. Το πρόγραμμα αυτό εκτός από την παρουσίαση των συγκεντρωτικών αποτελεσμάτων έχει και τη δυνατότητα παρουσίασης στατιστικών. Εκτός από τα συγκεντρωτικά αποτελέσματα γίνεται καταγραφή των αποτελεσμάτων ανά 15sec σε φύλλο εργασίας.



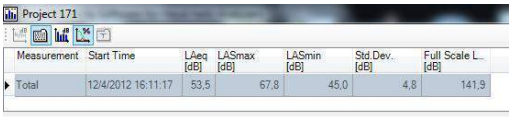
Σχήμα 64: Πρόγραμμα BZ-5503 Utility software for hand-held Analyzers

Οι ρυθμίσεις του ηχομέτρου και της εφαρμογής για κάθε δοκιμή παρουσιάζονται παρακάτω. Η εφαρμογή είχε πάντα την ανάλυση "Normal".

Δοκιμή 1:

Ηχόμετρο: Στάθμιση A, slow (project 171)

Εφαρμογή: Στάθμιση A, slow



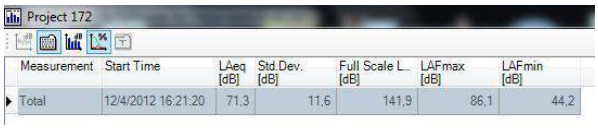
Χρόνος	LASmax (db)	LASmin (db)	LAS(db) 00:00:15	LAS(db) 00:00:30	LAS(db) 00:00:45	LAS(db) 00:01:00	LAS(db) 00:01:15	LAS(db) 00:01:30	LAS(db) 00:01:45	LAS(db) 00:02:00
Ηχόμετρο	67.8	45	47.4	50.2	65.8	51.3	49.4	53.2	46.8	54.8

Εφαρμογή	64.3	44.2	46.2	48.8	62.2	48.3	47	50.3	45.4	52.6

Δοκιμή 2:

Ηχόμετρο: Στάθμιση A, Fast (project 172)

Εφαρμογή: Στάθμιση A, Fast



Χρόνος	LAFmax (db)	LAFmin (db)	LAF(db) 00:00:15	LAF(db) 00:00:30	LAF(db) 00:00:45	LAF(db) 00:01:00	LAF(db) 00:01:15	LAF(db) 00:01:30	LAF(db) 00:01:45	LAF(db) 00:02:00
Ηχόμετρο	86.1	44.2	46.3	80.3	85.8	64.7	59.5	56.3	57.1	48.2
Εφαρμογή	79,1	43.8	44.8	76.9	78.2	61.7	56.9	53.9	54.7	46.6

Δοκιμή 3:

Ηχόμετρο: Στάθμιση C, slow (project 173)

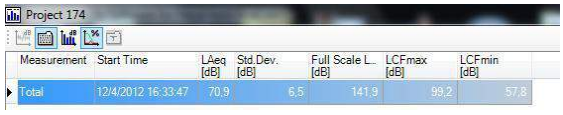
Εφαρμογή: Στάθμιση C, slow

Χρόνος	LCSmax (db)	LCSmin (db)	LCS(db) 00:00:15	LCS(db) 00:00:30	LCS(db) 00:00:45	LCS(db) 00:01:00	LCS(db) 00:01:15	LCS(db) 00:01:30	LCS(db) 00:01:45	LCS(db) 00:02:00
Ηχόμετρο	68.8	59.6	61.1	67.2	64.3	59.8	62.4	60	60.4	65.3
Εφαρμογή	64.6	58.1	59.2	64.2	60.7	58.1	60.2	59.2	59.7	62.4

Δοκιμή 4:

Ηχόμετρο: Στάθμιση C, Fast (project 174)

Εφαρμογή: Στάθμιση C, Fast

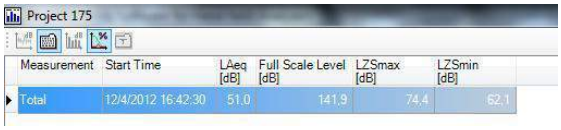


Χρόνος	LCFmax (db)	LCFmin (db)	LCF(db) 00:00:15	LCF(db) 00:00:30	LCF(db) 00:00:45	LCF(db) 00:01:00	LCF(db) 00:01:15	LCF(db) 00:01:30	LCF(db) 00:01:45	LCF(db) 00:02:00
Ηχόμετρο	99.2	57.8	58.4	59.3	61.2	58	68.9	60.1	59.2	80.4
Εφαρμογή	90.4	54.9	55.9	56.2	56.1	55.1	64.9	57	56.2	76.8

Δοκιμή 5:

Ηχόμετρο: Στάθμιση Z, slow (project 175)

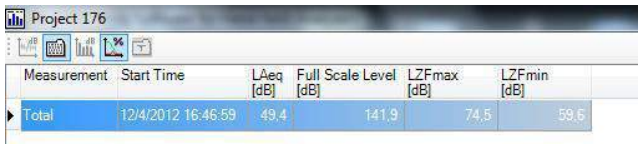
Εφαρμογή: Στάθμιση Lin, slow



Χρόνος	LZSmax (db)	LZSmin (db)	LZS(db) 00:00:15	LZS(db) 00:00:30	LZS(db) 00:00:45	LZS(db) 00:01:00	LZS(db) 00:01:15	LZS(db) 00:01:30	LZS(db) 00:01:45	LZS(db) 00:02:00
Ηχόμετρο	74.4	62.1	63.8	64.2	63.5	70.1	72.8	66.2	64.4	66.6
Εφαρμογή	71.1	61.6	62.6	63	62.3	68.1	70.4	65.2	63	65.1

Δοκιμή 6:

Ηχόμετρο: Στάθμιση Z, Fast (project 176)



Εφαρμογή: Στάθμιση Lin, Fast

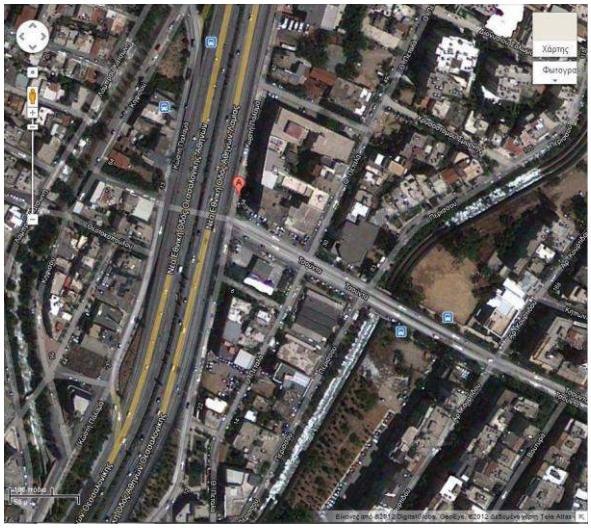
Χρόνος	LZSmax (db)	LZSmin (db)	LZS(db) 00:00:15	LZS(db) 00:00:30	LZS(db) 00:00:45	LZS(db) 00:01:00	LZS(db) 00:01:15	LZS(db) 00:01:30	LZS(db) 00:01:45	LZS(db) 00:02:00
Ηχόμετρο	74.5	59.6	60.1	62.3	61.9	64.2	68.1	72.3	61.3	60.4
Εφαρμογή	70.2	59	59.5	61.1	59.9	62.4	65.5	69.2	60.2	60

8.2 Μέτρηση Τυπικών Πηγών Θορύβου - Συγκριτικές Μετρήσεις

Έχοντας βαθμονομήσει τις συσκευές iPhone 4, με την εφαρμογή που αναπτύχθηκε και το iPhone 4S με την εξειδικευμένη εφαρμογή Audio Tools πραγματοποιήθηκαν οι παρακάτω μετρήσεις. Η εφαρμογή Audio Tools δίνει την τιμή της ηχοστάθμης σε ακέραιο αριθμό ενώ δεν καταγράφει την ελάχιστη τιμή. Κάθε μέτρηση είχε διάρκεια 2 λεπτών και οι καταγραφές γίνονταν ανά 15sec.

8.2.1 Μέτρηση Κυκλοφοριακού Θορύβου Εθνικής Οδού

Η μέτρηση πραγματοποιήθηκε Μ. Σάββατο 14 Απριλίου 2012 πρωί. Το σημείο που έγινε η μέτρηση είναι στην οδό Τσούντα & Κωστή Παλαμά (Σχ. 65). Το σημείο αυτό είναι λίγο πιο χαμηλά από την Εθνική οδό και είναι ένα μικρό κομμάτι που δεν έχει ηχοπετάσματα. Στην γύρω περιοχή υπάρχουν κυρίως επιχειρήσεις που λόγω της ημέρας δεν λειτουργούσαν, επομένως ο θόρυβος που καταγράφηκε ήταν κυρίως από την διέλευση των αυτοκινήτων. Την ημέρα των μετρήσεων υπήρχε λίγη κίνηση στην Εθνική οδό και τα διερχόμενα αυτοκίνητα περνούσαν με μεγάλες ταχύτητες ενώ δεν υπήρχε έντονος άνεμος που να επηρεάζει τις μετρήσεις. Οι μετρήσεις που έγιναν ήταν με στάθμιση Α, ενώ οι επιλογές της εφαρμογής που αναπτύχθηκε ήταν "Normal" και "slow".



Σχήμα 65: Σημείο μέτρησης κυκλοφοριακού θορύβου

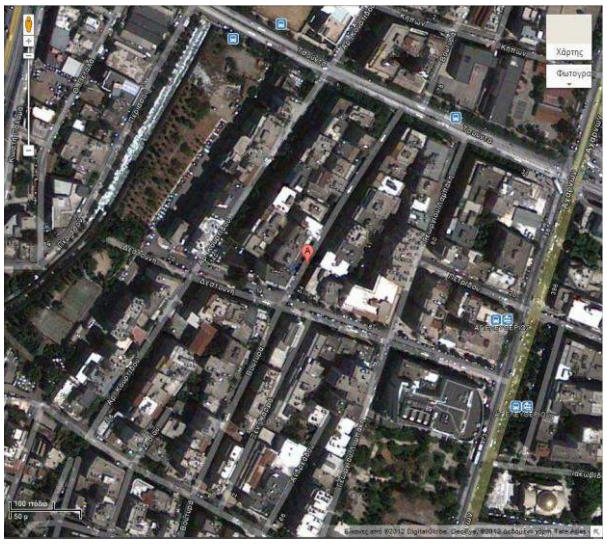
Πίνακας 6: Αποτελέσματα μέτρησης κυκλοφοριακού θορύβου στην Εθνική οδό

Χρόνος	L max db(A)	L min db(A)	L db(A) 00:15	L db(A) 00:30	L db(A) 00:45	L db(A) 01:00	L db(A) 01:15	L db(A) 01:30	L db(A) 01:45	L db(A) 02:00
Audio Tool	99	-	76	72	78	79	81	69	76	80
Εφαρμογή	97,8	68.3	75.6	73.2	77.2	78.4	81.6	70.4	75.2	79.1

Η ελάχιστη τιμή που μετρήθηκε ήταν 68,3 db(A) ενώ παρατηρούμε ότι όλες οι στιγμιαίες τιμές ήταν πάνω από 73 db(A). Η μέγιστη τιμή που καταγράφηκε ήταν 97,8 db(A). Τα αποτελέσματα των μετρήσεων είναι αναμενόμενα δεδομένου ότι η περιοχή αντιμετωπίζει πρόβλημα ηχορύπανσης και ότι για την προστασία των κατοίκων της περιοχής που βρίσκονται σε κοντινή απόσταση από την εθνική οδό έχουν τοποθετηθεί ηχοπετάσματα.

8.2.2 Μέτρηση σε Δρόμο Ήπιας Κυκλοφορίας

Η μέτρηση πραγματοποιήθηκε Μ. Σάββατο 14 Απριλίου 2012 πρωί. Το σημείο που έγινε η μέτρηση είναι στην οδό Βουτυρά (Σχ. 66). Τον δρόμο αυτό υπάρχουν μόνο κατοικίες και είναι ήπιας κυκλοφορίας. Οι πήγες θορύβου οφείλονται κυρίως από τις δραστηριότητες στις κατοικίες και από τα ελάχιστα διερχόμενα αυτοκίνητα. Επειδή ο δρόμος είναι στενός τα αυτοκίνητα διέρχονται με χαμηλή ταχύτητα. Κατά την διάρκεια της μέτρησης είχε άπνοια. Οι μετρήσεις που έγιναν ήταν με στάθμιση Α, ενώ οι επιλογές της εφαρμογής που αναπτύχθηκε ήταν "Normal" και "slow".



Σχήμα 66: Σημείο μέτρησης σε δρόμο ήπιας κυκλοφορίας

Πίνακας 7 : Αποτελέσματα μέτρησης σε δρόμο ήπιας κυκλοφορίας

Χρόνος	L max db(A)	L min db(A)	L db(A) 00:15	L db(A) 00:30	L db(A) 00:45	L db(A) 01:00	L db(A) 01:15	L db(A) 01:30	L db(A) 01:45	L db(A) 02:00
Audio Tool	87	-	58	57	57	62	62	58	72	58
Εφαρμογή	86,3	51,3	58,5	57,9	57,4	62,6	61,8	58,3	73,4	58,2

Η ελάχιστη τιμή που μετρήθηκε ήταν 51,3 db(A) ενώ παρατηρούμε ότι οι περισσότερες καταγραφές είναι κοντά στο 57 db(A). Η μέγιστη τιμή που καταγράφηκε ήταν 86,3 db(A). Η μέγιστη τιμή οφείλεται σε γαύγισμα σκύλου από ένα μπαλκόνι πολυκατοικίας. Την ώρα της μέτρησης πέρασε ένα αυτοκίνητο το οποίο όμως δεν αύξησε πολύ το θόρυβο λόγω της χαμηλής του ταχύτητας.

8.2.3 Μέτρηση σε Κατάστημα

Η μέτρηση πραγματοποιήθηκε Μ. Σάββατο 14 Απριλίου 2012 πρωί. Το κατάστημα που έγινε η μέτρηση είναι super market και λόγω της ημέρας είχε πολύ κόσμο. Το σημείο που έγινε η μέτρηση ήταν σε έναν διάδρομο του supermarket. Επιλέχτηκε διάδρομος όπου δεν περνούσε πολύς κόσμος.

Οι μετρήσεις που έγιναν ήταν με στάθμιση Α, ενώ οι επιλογές της εφαρμογής που αναπτύχθηκε ήταν "Normal" και "slow".

Πίνακας 8 : Αποτελέσματα μέτρησης σε κατάστημα

Χρόνος	L max db(A)	L min db(A)	L db(A) 00:15	L db(A) 00:30	L db(A) 00:45	L db(A) 01:00	L db(A) 01:15	L db(A) 01:30	L db(A) 01:45	L db(A) 02:00
Audio Tool	92	-	53	52	71	69	54	59	65	56
Εφαρμογή	91,6	46,5	53,3	52,1	70,8	69,2	53,7	58,9	65,0	56,3

Η ελάχιστη τιμή που μετρήθηκε ήταν 46,5 db(A)και η μέγιστη τιμή που καταγράφηκε ήταν 91,6 db(A). Η μέγιστη τιμή οφείλεται σε ανακοίνωση από τα μικρόφωνα. Στις μετρήσεις αυτές παρατηρείται ότι υπάρχουν αρκετές διακυμάνσεις οι οποίες οφείλονται κυρίως στις ομιλίες τον ανθρώπων.

8.2.4 Μέτρηση σε Ήσυχο Δωμάτιο

Η μέτρηση πραγματοποιήθηκε Μ. Σάββατο 14 Απριλίου 2012 πρωί σε ήσυχο δωμάτιο κατοικίας. Επιλέχθηκε να είναι πρωί για να μην υπάρχει έντονη δραστηριότητα στα γειτονικά διαμερίσματα. Το δωμάτιο δεν έχει ανοίγματα σε εξωτερικούς χώρους ώστε να υπάρχει θόρυβος από το δρόμο.

Οι μετρήσεις που έγιναν ήταν με στάθμιση Α, ενώ οι επιλογές της εφαρμογής που αναπτύχθηκε ήταν "Normal" και "slow".

Πίνακας 9 : Αποτελέσματα μέτρησης σε ήσυχο δωμάτιο

Χρόνος	L max db(A)	L min db(A)	L db(A) 00:15	L db(A) 00:30	L db(A) 00:45	L db(A) 01:00	L db(A) 01:15	L db(A) 01:30	L db(A) 01:45	L db(A) 02:00
Audio Tool	51	-	35	35	37	36	36	35	44	37
Εφαρμογή	50,2	34,3	35,2	35,8	37,4	35,6	35,7	35,2	44,1	37,4

Η ελάχιστη τιμή που μετρήθηκε ήταν 34,3 db(A)και η μέγιστη τιμή που καταγράφηκε ήταν 50,2 db(A), ενώ όλες οι καταγραφές είναι κοντά στα 35 db(A).

8.3 Συμπεράσματα Μετρήσεων

Από την παρουσίαση των αποτελεσμάτων της σύγκρισης της εφαρμογής που αναπτύχθηκε με το ηχόμετρο BRUEL & KJAER, TYPE 2250 Light και όπως ήταν αναμενόμενο φαίνεται ότι το ηχόμετρο ανταποκρίνεται πολύ καλύτερα στους έντονους ήχους και κυρίως στους κρότους. Η μικρότερη διαφορά που καταγράφηκε ήταν 0,4db στην μέτρηση fast χωρίς στάθμιση, ενώ η μεγαλύτερη ήταν 5,1db στην μέτρηση fast με στάθμιση C. Όμως οι διαφορές είναι μεγαλύτερες όταν εξετάζουμε την μέγιστη τιμή που κατέγραψε το ηχόμετρο και η συσκευή. Η μεγαλύτερη διαφορά παρατηρήθηκε στην μέτρηση fast με στάθμιση C όπου ήταν 8,8 db διαφορά. Κατά την διάρκεια των μετρήσεων αυτών ακούστηκε ένας έντονος ήχος από διπλανό χώρο του Εργαστηρίου όπου πραγματοποιούσαν δοκιμές. Η απόκριση του ηχομέτρου BRUEL & KJAER, TYPE 2250 Light ήταν άμεση και έφτασε τα 99,2 db ενώ η εφαρμογή κατέγραψε 90,4 db. Επίσης παρατηρήθηκε ότι οι

Σχήμα 68: Στιγμιότυπο μετρήσεων

Σχήμα 68: Εφαρμογή Audio Tool

διαφορές όταν το ηχόμετρο ήταν στη λειτουργία fast ήταν μεγαλύτερες σε σχέση με την λειτουργία slow. Τέλος με την στάθμιση C παρατηρήθηκαν λίγο μεγαλύτερες διαφορές ανάμεσα στην εφαρμογή και το ηχόμετρο. Σε κατάσταση ηρεμίας και χωρίς την ύπαρξη κάποιου εξωτερικού έντονου ήχου οι τιμές της εφαρμογής και του ηχόμετρου BRUEL & KJAER, TYPE 2250 Light ήταν πολύ κοντά. Όταν ο εξωτερικός θόρυβος είχε διάρκεια τότε οι τιμές της εφαρμογής έφταναν κοντά στις τιμές του ηχομέτρου. Οι διαφορές που παρατηρούνται οφείλονται κυρίως στις διαφορές που παρουσιάζουν στην ευαισθησία, στην

συχνότητα απόκρισης και στην στιγμιαία απόκριση τα δύο μικρόφωνα. Είναι λογικό και αναμενόμενο το μικρόφωνο του εργαστηριακού ηχόμετρου BRUEL & KJAER, TYPE 2250 Light να έχει πολύ καλύτερα χαρακτηριστικά από το μικρόφωνο μιας συσκευής κινητής τηλεφωνίας, διότι η κατασκευή του έχει γίνει για την επικοινωνία μέσω δικτύων GSM.

Από την παρουσίαση των αποτελεσμάτων της σύγκρισης της εφαρμογής που αναπτύχθηκε με την εφαρμογή Audio Tool παρατηρήθηκε ότι οι τιμές σχεδόν ταυτίζονται. Η μέγιστη διαφορά που παρατηρήθηκε ήταν 1,4 dB(A) στην μέτρηση του κυκλοφοριακού θορύβου. Στις μετρήσεις που έγιναν στο κατάστημα η μέγιστη διαφορά ήταν 0,3 dB(A) ενώ στις μετρήσεις σε ήσυχο δωμάτιο η μέγιστη διαφορά ήταν 0,8 dB(A). Οι διαφορές αυτές είναι πάρα πολύ μικρές και μπορεί να οφείλονται στο ότι η εφαρμογή Audio Tool δεν δίνει δεκαδικά ψηφία. Από τις παρατηρήσεις που έγιναν τόσο στο Εργαστήριο όσο και κατά την διάρκεια των μετρήσεων σε εξωτερικούς χώρους διαπιστώθηκε ότι το μικρόφωνο του κινητού iPhone 4S το οποίο μετρούσε με την εφαρμογή Audio Tool είναι πιο ευαίσθητο από το μικρόφωνο του iPhone 4. Για το λόγο αυτό βαθμονομήθηκαν και τα δύο κινητά στο Εργαστήριο

9 Συμπεράσματα

Βασικό αντικείμενο της συγκεκριμένης πτυχιακής εργασίας ήταν ο σχεδιασμός και η ανάπτυξη μιας εφαρμογής λογισμικού για την μέτρηση της ηχητικής στάθμης σε μια φορητή πλατφόρμα κινητής τηλεφωνίας. Η μέτρηση του θορύβου κρίνεται εξαιρετικά σημαντική καθώς οι επιπτώσεις του στον άνθρωπο είναι πολλαπλές και ποικίλες και επηρεάζοντας τις καθημερινές του δραστηριότητες. Ο θόρυβος οφείλεται στις ηχητικές συνθήκες του χώρου και προκαλείται από την συμβολή πολλών ηχογόνων παραγόντων όπως καταιγίδα, κυκλοφοριακή κίνηση, μέσα ψυχαγωγίας, ανθρώπινη δραστηριότητα κ.α. Για την περιγραφή του θορύβου είναι αναγκαία η γνώση της στάθμης ηχητικής πίεσης (dB).

Στις τεχνικές δυνατότητες της εφαρμογής συμπεριλαμβάνονται τα χαρακτηριστικά ενός τυπικού συστήματος μέτρησης ηχοστάθμης. Συγκεκριμένα, η εφαρμογή υποστηρίζει την δυνατότητα καταγραφής fast / slow ανάλογα με την επιλεγμένη τιμή του time resolution και μέτρηση της στιγμιαίας ηχοστάθμης την χρήση των καμπυλών A, B , C αλλά και χωρίς καμπύλη(Lin). Επιπλέον υπολογίζονται οι τιμές της ελάχιστης, μέγιστης και μέσης τιμής ηχοστάθμης. Επιπλέον ο χρήστης έχει τη δυνατότητα αλλαγής σταθεράς βαθμονόμηση και αποστολής των αποτελεσμάτων με email. Τέλος, υπάρχει η δυνατότητα επιλογής του τρόπου εμφάνισης των αποτελεσμάτων ανάμεσα σε τρεις γραφικές παραστάσεις ή τη κλασική βελόνα.

Η ανάπτυξη της εφαρμογής κατέδειξε την σπουδαιότητα των βασικών εννοιών σημάτων και συστημάτων καθώς ο υπολογισμός του επιπέδου της ηχητικής πίεσης αποτελεί μια σύνθετη μαθηματική διαδικασία που προϋποθέτει θεμελιώδεις γνώσεις της ψηφιακής επεξεργασίας σήματος όπως η δειγματοληψία, ο ταχύς μετασχηματισμός Fourier και η αναπαράσταση στο πεδίο των συχνοτήτων.

Η παρούσα εργασία εκμεταλλεύτηκε τις εξελίξεις στον τομέα ανάπτυξης φορητών συστημάτων η οποία επιτρέπει την δημιουργία και υποστήριξη νέων μορφών εφαρμογών που ξεφεύγουν από τα στενά όρια μιας απλής επικοινωνίας. Συγκεκριμένα στηρίχτηκε στη χρήση ενός αναπτυξιακού περιβάλλοντος (SDK), το οποίο παρέχει την δυνατότητα πλήρους ενσωμάτωσης των λειτουργικών υπομονάδων μιας συσκευής. Στη συγκεκριμένη εργασία απαιτήθηκε η εκμετάλλευση του μικροφώνου για την καταγραφή και διαχείριση των ηχητικών σημάτων.

Για την επιλογή της πλατφόρμας και κατ' επέκταση του λειτουργικού συστήματος πραγματοποιήθηκε διερεύνηση και αξιολόγηση όλων των χαρακτηριστικών, των διαθέσιμων λειτουργικών συστημάτων ως προς την καταλληλότητά τους για τις ανάγκες της συγκεκριμένης εφαρμογής. Η συγκεκριμένη ανάλυση περιόρισε το πλήθος των δυνατών επιλογών μεταξύ των δύο πιο διαδεδομένα λειτουργικών συστημάτων που κατέχουν και το

μεγαλύτερο μέρος της αγοράς το iOS και το Android OS. Η εφαρμογή υλοποιήθηκε για το λειτουργικό σύστημα iOS, και κατά συνέπεια χρησιμοποιήθηκε το αναπτυξιακού περιβάλλον Xcode Developer Tools μέσω του συνδυασμού των αντικειμενοστραφών γλωσσών προγραμματισμού Objective C και C++. Η γνώση που κατακτήθηκε από την χρήση του Xcode και των επιμέρους εργαλείων Builder, LLVM compiler, iOS simulator και Instruments, επιτρέπει την ανάπτυξη εφαρμογών για όλες τις φορητές πλατφόρμες της Apple όπως iPod Touch, iPhone και iPad.

Η χρήση της μεθοδολογίας ICONIX αποδείχθηκε σημαντική για τον σωστό σχεδιασμό και την επιτυχή ολοκλήρωση του έργου. Στα πλαίσια αυτής της μεθοδολογίας αξιοποιήθηκε ένα υποσύνολο της UML για την δημιουργία διαγραμμάτων ως ενδιάμεσα προϊόντα κατά την εξέλιξη του δυναμικού και στατικού μοντέλου του συστήματος που αναπτύσσεται. Συγκεκριμένα κατά τη φάση της ανάλυσης χρησιμοποιήθηκαν τα διαγράμματα περιπτώσεων χρήσης, κλάσεων, ευρωστίας και τα διαγράμματα ακολουθίας.

Η εφαρμογή που αναπτύχθηκε για την μέτρηση της ηχοστάθμης χρησιμοποιήθηκε για να γίνουν μετρήσεις τυπικών μορφών θορύβου και σύγκριση των αποτελεσμάτων με εξειδικευμένες μετρητικές συσκευές. Αρχικά για να εξασφαλιστούν οι ορθές μετρήσεις της εφαρμογής που αναπτύχθηκε απαιτήθηκε η βαθμονόμηση του μετρητικού συστήματος. Η βαθμονόμηση αυτή έγινε με τη μέθοδο της σύγκρισης με ηχόμετρο που είχε σαφώς πολύ καλύτερα μετρολογικά χαρακτηριστικά (ακρίβεια, επαναληψημότητα, διακριτική ικανότητα). Αφού προσδιορίστηκε η σταθερά βαθμονόμησης πραγματοποιήθηκαν συγκριτικές μετρήσεις με αντίστοιχη εφαρμογή σε πλατφόρμας κινητής τηλεφωνίας. Οι μετρήσεις πραγματοποιήθηκαν δίπλα στην εθνική οδό, σε δρόμο ήπιας κυκλοφορίας, σε κατάστημα και σε ήσυχο δωμάτιο. Τα δεδομένα ηχητικής πίεσης που συλλέχθηκαν από τις παραπάνω μετρήσεις ήταν σε αντιστοιχία με τις αναμενόμενες τιμές του θορύβου για τις συγκεκριμένες ηχητικές πηγές. Η σύγκριση των αποτελεσμάτων ανέδειξε την σπουδαιότητα της απόκρισης του μικροφώνου. Τα χαρακτηριστικά του μικροφώνου διαφοροποιούνται μεταξύ των μοντέλων και για το λόγο αυτό απαιτείται η διαδικασία της βαθμονόμησης να πραγματοποιείται για κάθε μοντέλο.

Ολοκληρώνοντας, διαπιστώνουμε την επίτευξη των εκπαιδευτικών στόχων που διατυπώθηκαν στην αρχή της ανάλυσης. Επιπλέον, το τελικό προϊόν αποτελεί πράγματι μια ανταγωνιστική εφαρμογή που πετυχαίνει μετρήσεις ικανοποιητικής ακρίβειας χωρίς την χρήση εξειδικευμένων συσκευών υψηλού κόστους. Το υψηλό επίπεδο του λογισμικού που υλοποιήθηκε καταμαρτυρείται από το γεγονός ότι η εφαρμογή εγκρίθηκε από την Ομάδα Προγραμματιστών-Κριτών Ελέγχου τόσο για την άρτια λειτουργία, όσο και για την ποιότητα του κώδικα, προκειμένου να προκριθεί στο App Store. Αξίζει να σημειωθεί ότι το

αποτέλεσμα του ελέγχου, δεν είχε καμία παρατήρηση. Η ανταγωνιστικότητα της εφαρμογής δικαιολογείται από το γεγονός ότι ενσωματώνει ένα εύρος τεχνικών δυνατοτήτων, που δεν μπορεί κανείς να τις συναντήσει συγκεντρωμένες σε καμία άλλη εφαρμογή του App Store αλλά και επιπλέον πρωτοποριακές δυνατότητες. Συγκεκριμένα, εκτός από τις βασικές λειτουργίες που υπάρχουν διάσπαρτα στις ανταγωνιστικές εφαρμογές, ενσωματώνονται οι επιπλέον δυνατότητες του καθορισμού του Time Resolution, της εναλλαγής μεταξύ των γραφικών απεικονίσεων της μέτρησης και της ιδιαίτερα φιλικής επαφής με τον χρήστη.

Στις μελλοντικές επεκτάσεις του συστήματος θα μπορούσε να είναι η φασματική ανάλυση του θορυβικού σήματος προκειμένου να αναδειχθούν τα ιδιαίτερα χαρακτηριστικά του. Όπως ήδη έχει αναφερθεί, το αυτί είναι περισσότερο ευαίσθητο σε ορισμένες συχνότητες συγκριτικά με τις άλλες. Κατά συνέπεια, είναι επιθυμητό να γνωρίζουμε τον τρόπο κατανομής της ηχητικής ενέργειας σε όλο το ακουστό ηχητικό φάσμα. Αυτό επιτυγχάνεται υποδιαιρώντας τον θόρυβο σε οκτάβες και μετρώντας την στάθμη ηχητικής πίεσης κάθε οκταβικής ζώνης συχνοτήτων. Μια καταγραφή των αποτελεσμάτων της μέτρησης σε μία βάση δεδομένων για την περεταίρω επεξεργασία των μετρήσεων. Τέλος, θα μπορούσε να υπάρχει η δυνατότητα καταγραφής της γεωγραφικής θέσης της μέτρησης με σκοπό την δημιουργία ενός χάρτη θορύβου.

Αναφορές

1. J. R. Hassall "Acoustic Noise Measurments" (Bruel & Kjaer, 1979)
2. J. Broch "Mechanical Vibration and shock Measurments" (Bruel & Kjaer, 1980)
3. "Noise control principle and proctice" (Bruel & Kjaer, 1982)
4. Α. Χατζηγεωργίου "Ανάπτυξη συστήματος λογισμικού βάσει της μεθόδου ICONIX" (ΕΑΠ, 2008)
5. Γ. Φούσκας "Ψηφιακές επικοινωνίες" (ΕΑΠ, 2002)
6. Δ. Σκαρλάτος "Εφαρμοσμένη ακουστική" (Gotsis, Πάτρα, 2008)
7. ΕΛ.ΙΝ.Υ.Α.Ε "Ο θόρυβος στην Εργασία: Φύση, κίνδυνοι και προστασία" (ΕΛ.ΙΝ.Υ.Α.Ε, 2006)
8. Μ. Σηφάκης. Χ. Κουρσοδημάκης "Εφαρμοσμένη Ακουστική Ι" (ΑΤΕΙ Κρήτης, 2012)
9. Χ. Δουληγέρης, Γ.Α. Τσιχριντής "Αρχές και Εφαρμογές σημάτων και συστημάτων" (Πανεπιστήμιο Πειραιώς, 2003)
10. ΕΛΟΤ 263.1 Ακουστική Ορολογία «Ταλαντώσεις, Δονήσεις, Ήχος, Μηχανικό σοκ»
11. IEC 61672-1 Electroacoustics - Sound level meters - Part 1: Specifications
12. IEC 61672-2 Electroacoustics - Sound level meters - Part 2: Pattern evaluation tests
13. IEC 61672-3 Electroacoustics - Sound level meters - Part 3: Periodic tests
14. IEC 60942 Electroacoustics - Sound calibrators
15. http://en.wikipedia.org/wiki/Martin_Cooper_(inventor) (ημερομηνία προσπέλασης: 01/07/2012)
16. http://en.wikipedia.org/wiki/Personal_digital_assistant (ημερομηνία προσπέλασης: 01/07/2012)
17. http://en.wikipedia.org/wiki/IBM_Simon (ημερομηνία προσπέλασης: 01/07/2012)
18. http://blog.nielsen.com/nielsenwire/?p=28790 (ημερομηνία προσπέλασης: 01/07/2012)
19. http://www.research2guidance.com/shop/index.php/downloadable/download/sample/sample_id/150/ (ημερομηνία προσπέλασης: 01/07/2012)
20. http://en.wikipedia.org/wiki/Consumer_Electronics_Show (ημερομηνία προσπέλασης: 01/07/2012)
21. http://eprl.korinthos.uop.gr/BlogsPortal/pake04/ (ημερομηνία προσπέλασης: 01/07/2012)
22. http://en.wikipedia.org/wiki/Open_Handset_Alliance (ημερομηνία προσπέλασης: 01/07/2012)
23. http://www.who.int/en/ (ημερομηνία προσπέλασης: 01/07/2012)
24. http://en.wikipedia.org/wiki/Bada (ημερομηνία προσπέλασης: 01/07/2012)
25. http://en.wikipedia.org/wiki/HP_TouchPad (ημερομηνία προσπέλασης: 01/07/2012)

26. http://en.wikipedia.org/wiki/Windows_mobile (ημερομηνία προσπέλασης: 01/07/2012)
27. http://www.idc.com/getdoc.jsp?containerId=prUS22762811 (ημερομηνία προσπέλασης: 01/07/2012)
28. http://gs.statcounter.com/#mobile_os-eu-yearly-2009-2012 (ημερομηνία προσπέλασης: 01/07/2012)
29. http://www.mcafee.com/us/resources/reports/rp-securing-mobile-devices.pdf (ημερομηνία προσπέλασης: 01/07/2012)
30. http://www.idc.com/getdoc.jsp?containerId=prUS23228211 (ημερομηνία προσπέλασης: 01/07/2012)
31. http://en.wikipedia.org/wiki/Objective-c (ημερομηνία προσπέλασης: 01/07/2012)
32. https://developer.apple.com/library/ios/navigation/index.html (ημερομηνία προσπέλασης: 01/07/2012)
33. http://osha.europa.eu/el/topics/noise/index_html/problems_noise_cause_html (ημερομηνία προσπέλασης: 01/07/2012)
34. http://www.bksv.com (ημερομηνία προσπέλασης: 01/07/2012)
35. http://www.ionio.gr/~floros/ (ημερομηνία προσπέλασης: 01/07/2012)
36. http://en.wikipedia.org/wiki/BlackBerry (ημερομηνία προσπέλασης: 01/07/2012)
37. http://en.wikipedia.org/wiki/ios (ημερομηνία προσπέλασης: 01/07/2012)
38. http://en.wikipedia.org/wiki/Symbian (ημερομηνία προσπέλασης: 01/07/2012)

Παράρτημα Α: Πηγαίος κώδικας

AppDelegate.h

```
//
//  AppDelegate.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import <UIKit/UIKit.h>

@class MainViewController;

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (strong, nonatomic) MainViewController *mainViewController;

@end
```

AppDelegate.mm

```
//
//  AppDelegate.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "AppDelegate.h"

#import "MainViewController.h"
```

```
@implementation AppDelegate

@synthesize window = _window;
@synthesize mainViewController = _mainViewController;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    // Override point for customization after application launch.
    self.mainViewController = [[MainViewController alloc] initWithNibName:@"MainViewController" bundle:nil];
    self.window.rootViewController = self.mainViewController;
    [self.window makeKeyAndVisible];
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
{
    // Sent when the application is about to move from active to inactive state. This can occur for certain types of temporary interruptions (such as an incoming phone call or SMS message) or when the user quits the application and it begins the transition to the background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down OpenGL ES frame rates. Games should use this method to pause the game.
}

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
```

```
}

- (void)applicationWillEnterForeground:(UIApplication *)application
{
    // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application
{
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
}

- (void)applicationWillTerminate:(UIApplication *)application
{
    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
}

@end
```

MainViewController.h

```
//
//  MainViewController.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "FlipsideViewController.h"
#import "SoundManager.h"
```

```
#import "RecordManager.h"
#import <MessageUI/MessageUI.h>
#import "GraphVisualizer.h"
#import "GraphBar.h"
#import "NeedleView.h"
#import "GraphPulse.h"
#import "InfoViewController.h"


@interface MainViewController : UIViewController <InfoViewControllerDelegate, FlipsideViewControllerDelegate,SoundManagerDelegate,UIScrollViewDelegate,MFMailComposeViewControllerDelegate> {


    int                 speedType;
    int32_t             filteringType;
    int32_t             blockSize;
    int32_t             maxBlockSize;
     BOOL               started;
    BOOL                stopped;
    BOOL                pageControlUsed;
    NSDictionary        *firstValues;
    float               samplingRate;
    float               calibrationConstant;
    int                 operationMode;
    RecordManager       *recordManager;
    SoundManager        *sm;

//  UIImage                         *playBtnBG;
    FFT                 *fft;
    WeightedFiltering   *wf;
    SPL                 *spl;
    NSString            *cur;
    NSString            *avg;
```

```
    NSString            *max;
    NSString            *min;

    IBOutlet UIScrollView *maskScroll;
    IBOutlet UIScrollView *scroll;
    IBOutlet GraphBar *gb;
    IBOutlet GraphVisualizer *gv;
    IBOutlet GraphPulse *gp;
    IBOutlet NeedleView *nv;
    IBOutlet UIPageControl *pageControl;
    IBOutlet UIButton *startPause;
    IBOutlet UILabel *ftLabel;
    IBOutlet UILabel *omLabel;

    IBOutlet UILabel *srLabel;
    IBOutlet UILabel *bsLabel;
    IBOutlet UILabel *ccLabel;

    IBOutlet UILabel *typicalSPLLabel;
    IBOutlet UILabel *maxBackLabel;
    IBOutlet UILabel *avgBackLabel;

    IBOutlet UILabel *minBackLabel;
  //  IBOutlet UILabel *onOff;

    IBOutlet UILabel *curLabel;

    IBOutlet UILabel *avgLabel;

    IBOutlet UILabel *minLabel;
    IBOutlet UILabel *maxLabel;
    IBOutlet UIButton *switchButton;
  //  IBOutlet UIView *maskView;

}
```

```
@property(nonatomic,retain) UILabel *avgBackLabel;
@property(nonatomic,retain) UILabel *minBackLabel;
@property(nonatomic,retain) UILabel *maxBackLabel;
@property(nonatomic,retain) UIScrollView *maskScroll;
//@property(nonatomic,retain) UIView *maskView;
@property(nonatomic,retain) UIButton *switchButton;
@property(nonatomic,retain) NSDictionary *firstValues;
@property(nonatomic,retain) NSString *cur;
@property(nonatomic,retain) NSString *avg;
@property(nonatomic,retain) NSString *max;
@property(nonatomic,retain) NSString *min;
//@property(nonatomic,retain) IBOutlet UILabel *onOff;

@property(readonly) RecordManager *recordManager;
@property(nonatomic,retain) FFT *fft;
@property(nonatomic,retain) WeightedFiltering *wf;
@property(nonatomic,retain) SPL *spl;
@property(nonatomic,retain) GraphVisualizer *gv;
@property(nonatomic,retain) GraphPulse *gp;
@property(nonatomic,retain) IBOutlet UILabel *curLabel;
@property(nonatomic,retain) SoundManager *sm;
@property(retain, nonatomic) IBOutlet UILabel *avgLabel;
@property(retain, nonatomic) IBOutlet UILabel *maxLabel;
@property(retain, nonatomic) IBOutlet UILabel *minLabel;
@property(nonatomic,retain) GraphBar * gb;
@property(nonatomic,retain) UIScrollView *scroll;
@property(nonatomic,retain) IBOutlet UIButton *startPause;
@property(nonatomic,retain) IBOutlet UILabel *ftLabel;
@property(nonatomic,retain) IBOutlet UILabel *omLabel;
@property(nonatomic,retain) IBOutlet UILabel *srLabel;
@property(nonatomic,retain) IBOutlet UILabel *bsLabel;
@property(nonatomic,retain) IBOutlet UILabel *ccLabel;
@property(nonatomic,retain)  UIPageControl *pageControl;
@property(nonatomic,retain) IBOutlet NeedleView *nv;
@property(nonatomic,retain) IBOutlet UILabel *typicalSPLLabel;
```

```
- (IBAction)showInfo:(id)sender;
- (IBAction)recordPause:(id)sender;
- (IBAction)stop:(id)sender;
- (IBAction)emailResults:(id)sender;
- (IBAction)changePage:(id)sender;
- (IBAction)help:(id)sender;

-(void)stopRecord;
-(void)loadSystemDefaults;
-(void)scrollViewDidScroll:(UIScrollView *)sender ;

//- (void)infoViewControllerDidFinish:(InfoViewController *)controller;


@end
```

MainViewController.mm

```
//
//  MainViewController.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "MainViewController.h"

@interface MainViewController ()
```

```
@end

@implementation MainViewController

@synthesize recordManager;
@synthesize fft;
@synthesize wf;
@synthesize spl;
@synthesize curLabel;
@synthesize avgLabel;
@synthesize maxLabel;
@synthesize minLabel;
@synthesize sm;
@synthesize gv;
@synthesize gb;
@synthesize scroll;
@synthesize startPause;
@synthesize ftLabel;
@synthesize omLabel;
@synthesize srLabel;
@synthesize bsLabel;
@synthesize pageControl;
@synthesize nv;
@synthesize ccLabel;
@synthesize typicalSPLLabel;
@synthesize gp;
//@synthesize onOff;
@synthesize cur;
@synthesize avg;
@synthesize max;
@synthesize min;
@synthesize firstValues;
@synthesize switchButton;
//@synthesize maskView;
@synthesize maskScroll;
```

```
    @synthesize maxBackLabel;
    @synthesize minBackLabel;
    @synthesize avgBackLabel;

    - (void)viewDidLoad
    {

      // self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"newMitraRot.png"]];
        maskScroll.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"newMitraBack4.png"]];
        maxBackLabel.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"labelMax4.png"]];
        minBackLabel.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"labelMin4.png"]];
        avgBackLabel.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"labelAvg.png"]];

        started = FALSE;
        stopped = FALSE;

        firstValues = [NSDictionary dictionaryWithObjectsAndKeys:[NSNumber
numberWithFloat:67],@"calibrationConstant",
                       [NSNumber numberWithInteger:1],@"filteringType",[NSNumber
numberWithInteger:1],@"speedType",
                       [NSNumber numberWithInteger:1],@"operationMode",[NSNumber
numberWithFloat:0], @"sliderValue",nil];

        [[NSUserDefaults standardUserDefaults] registerDefaults:firstValues];
        [self loadSystemDefaults];
        [self scrollLoad];
        [self showLabels];
        [super viewDidLoad];

          // Do any additional setup after loading the view, typically from a nib.
```

```
}


- (void)viewWillAppear:(BOOL)animated
{
    [self loadSystemDefaults];
    [self showLabels];
    [super viewWillAppear:animated];

}


-(void)reloadDefaults{
    [self loadSystemDefaults];
    [self showLabels];
}

- (void)viewDidUnload
{
    scroll = nil;
    gb = nil;
    gv = nil;
    gp = nil;
    nv = nil;
    pageControl = nil;
    startPause = nil;
    ftLabel = nil;
    omLabel = nil;
    srLabel = nil;
    bsLabel = nil;
    ccLabel = nil;
    typicalSPLLabel = nil;
  //   onOff = nil;
    curLabel = nil;
    avgLabel = nil;
```

```
        minLabel = nil;
        maxLabel = nil;
        switchButton = nil;
       // maskView = nil;
        maskScroll = nil;
        maxBackLabel = nil;
        minBackLabel = nil;
        avgBackLabel = nil;
        [super viewDidUnload];
        // Release any retained subviews of the main view.
    }
    /*
    -
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientatio
n
    {
        return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
    }
    */
    -(void) showLabels {
        switch (filteringType) {
            case 0:
                ftLabel.text = @"dB(Lin)";
                break;
            case 1:
                ftLabel.text = @"dB(A)";
                break;
            case 2:
                ftLabel.text = @"dB(B)";
                break;
            case 3:
                ftLabel.text = @"dB(C)";
                break;

            default:
```

```
            break;
        }

        if (operationMode==0) {
            omLabel.text =@"Fast";

        }
        else {
            omLabel.text = @"Slow";
        }
        bsLabel.text = [NSString stringWithFormat:@"%d",blockSize];
        srLabel.text = [NSString stringWithFormat:@"%.0f",samplingRate];
      //  ccLabel.text = [NSString stringWithFormat:@"%.1f",calibrationConstant];
    }

    -(void) scrollLoad{

        scroll.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"BlackboardPlegma.png"]];
        scroll.pagingEnabled = YES;
        NSInteger numberOfViews = 3;

        CGFloat yOrigin = 0 * self.scroll.frame.size.width;
        [gv setFrame:CGRectMake(yOrigin, 0, self.scroll.frame.size.width, 225)];
        [scroll addSubview:gv];

        yOrigin = 1 * self.scroll.frame.size.width;
        [gb setFrame:CGRectMake(yOrigin, 0, self.scroll.frame.size.width, 225)];
        [scroll addSubview:gb];

        yOrigin = 2 * self.scroll.frame.size.width;
        [gp setFrame:CGRectMake(yOrigin, 0, self.scroll.frame.size.width, 225)];
        [scroll addSubview:gp];

        scroll.contentSize = CGSizeMake(self.scroll.frame.size.width * numberOfViews,
```

```
self.scroll.frame.size.height);
        // [self.view addSubview:scroll];
        pageControl.numberOfPages = numberOfViews;
        pageControl.currentPage = 0;
        scroll.delegate = self;

    }

    - (void)scrollViewDidScroll:(UIScrollView *)sender {
        // We don't want a "feedback loop" between the UIPageControl and the scroll delegate in
        // which a scroll event generated from the user hitting the page control triggers updates from
        // the delegate method. We use a boolean to disable the delegate logic when the page control is used.
        if (pageControlUsed) {
            // do nothing - the scroll was initiated from the page control, not the user dragging
            return;
        }
        // Switch the indicator when more than 50% of the previous/next page is visible
        CGFloat pageWidth = scroll.frame.size.width;
        int page = floor((scroll.contentOffset.x - pageWidth / 2) / pageWidth) + 1;
        pageControl.currentPage = page;

    }

    - (void)scrollViewDidEndDecelerating:(UIScrollView *)scrollView {
        pageControlUsed = NO;
    }

    -(void) loadSystemDefaults{

        NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];

        int st;
```

```
float cc;
int32_t ft;
int32_t om;
float sv;

st = [defaults integerForKey:@"speedType"];
cc = [defaults floatForKey:@"calibrationConstant"];
ft = [defaults integerForKey:@"filteringType"];
om = [defaults integerForKey:@"operationMode"];
sv = [defaults floatForKey:@"sliderValue"];

speedType = st;
calibrationConstant = cc + sv;
filteringType = ft;
operationMode = om;



   switch (speedType) {
   case 0:
      if (operationMode==0) {
         blockSize = 512;
      }
      else {
         blockSize = 4096;
      }
      break;
   case 1:
      if (operationMode==0) {
         blockSize = 1024;
      }
      else {
         blockSize = 8192;
      }
      break;
   case 2:
```

```
            if (operationMode==0) {
               blockSize = 2048;
            }
            else {
               blockSize = 16384;
            }
            break;
         default: NSLog(@"SWITCH ERROR: error blocksize!!!");
            break;
      }

      maxBlockSize = 16384;
      samplingRate      = 44100.0;

      if(!stopped){

      recordManager = [[RecordManager alloc] initWithBlockSize:blockSize];
      fft = [[FFT alloc] initWithSize:blockSize withMaxBlockSize:maxBlockSize];
      wf = [[WeightedFiltering alloc] initWithFilteringType:filteringType
ofBufferSize:blockSize ofSamplingFrequency:samplingRate];
      spl = [[SPL alloc] initWithBufferSize:blockSize ofSamplingFrequency:samplingRate
withCalibrationConstant:calibrationConstant withOperationMode:operationMode
withFilterType:filteringType];
      sm = [[SoundManager alloc] initWithWeightedFiltering:wf withFFT:fft withSPL:spl
withBlockSize:blockSize withGraphVisualizer:gv withGraphBar:gb withGraphPulse:gp
withNeedleView:nv withCurLabel:curLabel];
      sm.delegate = self;
      [recordManager setSoundManager:sm];

      }
      else {
         [recordManager Reset:blockSize];
         [fft Reset:blockSize];
         [wf ResetWithFilteringType:filteringType ofBufferSize:blockSize
ofSamplingFrequency:samplingRate];
```

```
        [spl ResetWithBufferSize:blockSize ofSamplingFrequency:samplingRate
withCalibrationConstant:calibrationConstant withOperationMode:operationMode
withFilterType:filteringType];
        [sm ResetWithWeightedFiltering:wf withFFT:fft withSPL:spl
withBlockSize:blockSize];
        [recordManager setSoundManager:sm];
    }

}


-(void)EmailSPLMeasurements{
    // create an NSData object with the selected recording
    NSData *data = [NSData dataWithContentsOfFile:[spl dataFilePath]];

    // create a MFMailComposeViewController for sending an e-mail
    MFMailComposeViewController *controller =
    [[MFMailComposeViewController alloc] init];

    // add the recording as an attachment
    [controller addAttachmentData:data mimeType:@"xml" fileName:
    [spl dataFilePath]];

    // set controller's delegate to this object
    controller.mailComposeDelegate = self;

    // show the MFMailComposeViewController
    [self presentModalViewController:controller animated:YES];

}

- (void)mailComposeController:(MFMailComposeViewController*)controller
        didFinishWithResult:(MFMailComposeResult)result error:(NSError*)error {

    [self dismissModalViewControllerAnimated:YES];
```

```
}

- (void)stopRecord
{

    recordManager.recorder->StopRecord();

}

#pragma mark - Flipside View

- (void)flipsideViewControllerDidFinish:(FlipsideViewController *)controller
{
    [self dismissModalViewControllerAnimated:YES];
}

- (IBAction)showInfo:(id)sender
{
    [self stop:sender];
    FlipsideViewController *controller = [[FlipsideViewController alloc]
initWithNibName:@"FlipsideViewController" bundle:nil];
    controller.delegate = self;
    controller.modalTransitionStyle = UIModalTransitionStyleFlipHorizontal;
    [self presentModalViewController:controller animated:YES];
}

- (void)infoViewControllerDidFinish:(InfoViewController *)controller
{
    [self dismissModalViewControllerAnimated:YES];
}




- (IBAction)recordPause:(id)sender {
```

```
        if (recordManager.recorder->IsRunning()) // If we are currently recording, stop and save the file.
            {
                    //pause the timer from generating events
            [self stopRecord];
            [sender setImage:[UIImage imageNamed:@"btPlay110.png"] forState:UIControlStateNormal];

            }
            else // If we're not recording, start.
            {
            // Start the recorder
            started = TRUE;
            recordManager.recorder->StartRecord();
            [omLabel setHidden:NO];
            [srLabel setHidden:NO];
            [ccLabel setHidden:NO];
            [bsLabel setHidden:NO];
            [ftLabel setHidden:NO];
            [sender setImage:[UIImage imageNamed:@"btPause110.png"] forState:UIControlStateNormal];
            [switchButton setImage:[UIImage imageNamed:@"btOn110.png"] forState:UIControlStateNormal];
            }
    }

    - (IBAction)stop:(id)sender {

        if (started == TRUE) {

            [switchButton setImage:[UIImage imageNamed:@"btOff110.png"] forState:UIControlStateNormal];
            [startPause setImage:[UIImage imageNamed:@"btPlay110.png"] forState:UIControlStateNormal];
```

```
            //Before doing anything save spl measurements to file.
            [spl saveMeasurementsToFile];
            if (recordManager.recorder->IsRunning()) {
               [self stopRecord];
            }
        //                  [startPause setImage:[UIImage imageNamed:@"playM64.png"]
forState:UIControlStateNormal];
           //      self.onOff.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:@"off.png"]];

            [gv clear];
            [gb clear];
            [gp clear];

            [gv setNeedsDisplay];
            [gb setNeedsDisplay];
            [gp setNeedsDisplay];
            [nv moveToStart];
            [curLabel setText:nil];
            [avgLabel setText:nil];
            [maxLabel setText:nil];
            [minLabel setText:nil];
            [typicalSPLLabel setText:nil];
            [typicalSPLLabel setBackgroundColor:nil];
            [omLabel setHidden:YES];
            [srLabel setHidden:YES];
            [ccLabel setHidden:YES];
            [bsLabel setHidden:YES];
            [ftLabel setHidden:YES];
            started = FALSE;
            stopped = TRUE;
            [self reloadDefaults];
        }
    }
```

```
- (IBAction)emailResults:(id)sender {
    [self stop:sender];
    [self EmailSPLMeasurements];
}

- (IBAction)changePage:(id)sender {

    int page = pageControl.currentPage; // load the visible page and the page on either side of it (to avoid flashes when the user starts scrolling)

    // update the scroll view to the appropriate page
    CGRect frame = scroll.frame;
    frame.origin.x = frame.size.width * page;
    frame.origin.y = 0;
    [scroll scrollRectToVisible:frame animated:YES];

    // Set the boolean used when scrolls originate from the UIPageControl. See scrollViewDidScroll: above.
    pageControlUsed = YES;
}

- (IBAction)help:(id)sender {

    InfoViewController *controller = [[InfoViewController alloc] initWithNibName:@"InfoViewController" bundle:nil];
    controller.delegate = self;
    controller.modalTransitionStyle = UIModalTransitionStylePartialCurl;
    [self presentModalViewController:controller animated:YES];
}

-(void) UpdateCurrentSPLLabel:(SoundManager *)sm withCurrSPL: (float)currSPL
            withAvgSPL: (float) avgSPL withMaxSPL:(float)maxSPL withMinSPL:(float) minSPL{
    @autoreleasepool {
```

```
        //Current SPL Label
        cur =  [NSString stringWithFormat:@"%.1f",currSPL];
        [curLabel  setText:cur];

        //Average Label
        avg =  [NSString stringWithFormat:@"%.1f",avgSPL];
        [avgLabel setText:avg];



        //Max Label
        max =  [NSString stringWithFormat:@"%.1f",maxSPL];
        [maxLabel setText:max];

        //Min Label
        min =  [NSString stringWithFormat:@"%.1f",minSPL];
        [minLabel setText:min];

    }

}

-(void) UpdateTypicalSPLLabel: (SoundManager *)sm withCurrSPL: (float)currSPL withFilteringType:(int) ft{

    int splInterval = currSPL/10;
    if (ft==0) {
        switch (splInterval) {
            case 0:
                [typicalSPLLabel setBackgroundColor:[UIColor whiteColor]];
                typicalSPLLabel.text = @"Auditory Threshold";
                break;
            case 1:
                [typicalSPLLabel setBackgroundColor:[UIColor greenColor]];
                typicalSPLLabel.text = @"Recording Studio";
                break;
```

```
            case 2:
                [typicalSPLLabel setBackgroundColor:[UIColor greenColor]];
                typicalSPLLabel.text = @"Rustling Leaves";
                break;
            case 3:
                [typicalSPLLabel setBackgroundColor:[UIColor colorWithRed:0 green:0.8
blue:0 alpha:1.0]];
                typicalSPLLabel.text = @"Quiet Library";
                break;
            case 4:
                [typicalSPLLabel setBackgroundColor:[UIColor colorWithRed:0 green:0.8
blue:0 alpha:1.0]];
                typicalSPLLabel.text = @"Quiet Room";
                break;
            case 5:
                [typicalSPLLabel setBackgroundColor:[UIColor colorWithRed:0 green:0.8
blue:0 alpha:1.0]];
                typicalSPLLabel.text = @"Quiet Office";
                break;
            case 6:
                [typicalSPLLabel setBackgroundColor:[UIColor colorWithRed:0 green:0.8
blue:0 alpha:1.0]];
                typicalSPLLabel.text = @"Normal Conversation";
                break;
            case 7:
                [typicalSPLLabel setBackgroundColor:[UIColor yellowColor]];
                [typicalSPLLabel setTextColor:[UIColor blackColor]];
                typicalSPLLabel.text = @"Noisy Office";
                break;
            case 8:
                [typicalSPLLabel setBackgroundColor:[UIColor yellowColor]];
                [typicalSPLLabel setTextColor:[UIColor blackColor]];
                typicalSPLLabel.text = @"Traffic Noise";
                break;
            case 9:
```

```
            [typicalSPLLabel setBackgroundColor:[UIColor orangeColor]];
            typicalSPLLabel.text = @"Truck";
            break;
        case 10:
            [typicalSPLLabel setBackgroundColor:[UIColor orangeColor]];
            typicalSPLLabel.text = @"Close to Train";
            break;
        case 11:
            [typicalSPLLabel setBackgroundColor:[UIColor redColor]];
            typicalSPLLabel.text = @"Chainsaw";
            break;
        case 12:
            [typicalSPLLabel setBackgroundColor:[UIColor redColor]];
            typicalSPLLabel.text = @"Thunder";
            break;
        case 13:
            [typicalSPLLabel setBackgroundColor:[UIColor redColor]];
            typicalSPLLabel.text = @"Immediate Ear Damage";
            break;
        default:
            break;
    }

  }



}



@end
```

FlipsideViewController.h

```
//
//  FlipsideViewController.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import <UIKit/UIKit.h>

@class FlipsideViewController;

@protocol FlipsideViewControllerDelegate
- (void)flipsideViewControllerDidFinish:(FlipsideViewController *)controller;
@end

@interface FlipsideViewController : UIViewController <UITextFieldDelegate> {

    int32_t              filteringType;
    float                calibrationConstant;
    int                  operationMode;
    float                sliderValue;
    int                  speedType;
    NSString             * str;


    IBOutlet UISegmentedControl *segmentFW;
    IBOutlet UITextField *textFieldC;
    IBOutlet UISegmentedControl *segmentSF;
    IBOutlet UIButton *setDefaults;
    IBOutlet UISlider *calibrationSlider;
    IBOutlet UILabel *calibrationLabel;
    IBOutlet UISegmentedControl *segmentST;
```

```
    IBOutlet UIScrollView *scrollSet;




}

@property (weak, nonatomic) id <FlipsideViewControllerDelegate> delegate;
@property (nonatomic,retain) UISegmentedControl *segmentFW;
@property (nonatomic,retain) UITextField *textFieldC;
@property (nonatomic,retain) UISegmentedControl *segmentSF;
@property(nonatomic, retain) UISlider *calibrationSlider;
@property(nonatomic,retain) UILabel *calibrationLabel;
@property(nonatomic,retain) UISegmentedControl *segmentST;
@property(nonatomic,retain) UIScrollView *scrollSet;



- (IBAction)done:(id)sender;
- (IBAction)setDefault:(id)sender;
- (IBAction)dragOutside:(id)sender;
- (IBAction)cancel:(id)sender;



@end
```

FlipsideViewController.m

```
//
//  FlipsideViewController.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "FlipsideViewController.h"
```

```
@interface FlipsideViewController ()

@end

@implementation FlipsideViewController



@synthesize delegate = _delegate;
@synthesize segmentFW;
@synthesize textFieldC;
@synthesize segmentSF;
@synthesize calibrationSlider;
@synthesize calibrationLabel;
@synthesize segmentST;
@synthesize scrollSet;



- (void)viewDidLoad
{
    [self updateGUI];
    [self obtainSliderValue];
    textFieldC.returnKeyType = UIReturnKeyDone;
    textFieldC.delegate = self;
  //  scrollSet.delegate = self;
   // scrollSet.bounces = YES;
    //scrollSet.pagingEnabled=YES;
    //[scrollSet addSubview:textFieldC];
    [super viewDidLoad];
        // Do any additional setup after loading the view, typically from a nib.
}

- (void)viewDidUnload
{
    segmentFW = nil;
```

```
    textFieldC = nil;
    segmentSF = nil;
    setDefaults = nil;
    calibrationSlider = nil;
    calibrationLabel = nil;
    segmentST = nil;
    scrollSet = nil;
    [super viewDidUnload];
    // Release any retained subviews of the main view.
}
/*
-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientatio
n
{
    return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
}
*/

-(void) obtainCalibrationConstant{
    str = textFieldC.text;
    float f = [str floatValue];
    calibrationConstant = f;
}



-(void) obtainSliderValue{
    sliderValue = calibrationSlider.value;
    calibrationLabel.text = [NSString stringWithFormat:@"%.1f",sliderValue];


}

-(void) obtainFilteringType{
    NSInteger  nsi = segmentFW.selectedSegmentIndex;
    int32_t type =  (int32_t) nsi;
```

```
    filteringType = type;
}

-(void) obtainOperationMode{
    NSInteger  nsi = segmentSF.selectedSegmentIndex;
    int32_t type =  (int32_t) nsi;
    operationMode = type;
}

-(void) obtainSpeedType{
    NSInteger nsi = segmentST.selectedSegmentIndex;
    int type = (int)nsi;
    speedType = type;
}

-(void) updateGUI {
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    int st = [defaults integerForKey:@"speedType"];
    segmentST.selectedSegmentIndex = st;
    float cc = [defaults floatForKey:@"calibrationConstant"];
    textFieldC.text =  [NSString stringWithFormat:@"%.1f",cc];
    int32_t ft = [defaults integerForKey:@"filteringType"];
    segmentFW.selectedSegmentIndex = ft;
    int32_t om = [defaults integerForKey:@"operationMode"];
    segmentSF.selectedSegmentIndex = om;
    float sv = [defaults floatForKey:@"sliderValue"];
    calibrationSlider.value = sv;
    calibrationLabel.text = [NSString stringWithFormat:@"%.1f",sliderValue];
}

-(BOOL) textFieldShouldReturn:(UITextField *)field{
    [textFieldC resignFirstResponder];
    return YES;
}
```

```
#pragma mark - Actions

- (IBAction)done:(id)sender
{
    [self obtainSpeedType];
    [self obtainFilteringType];
    [self obtainOperationMode];
    [self obtainCalibrationConstant];
    [self obtainSliderValue];

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults setInteger:speedType forKey:@"speedType"];
    [defaults setFloat:calibrationConstant forKey:@"calibrationConstant"];
    [defaults setInteger:filteringType forKey:@"filteringType"];
    [defaults setInteger:operationMode forKey:@"operationMode"];
    [defaults setFloat:sliderValue forKey:@"sliderValue"];
    [defaults synchronize];

    [self.delegate flipsideViewControllerDidFinish:self];
}

- (IBAction)setDefault:(id)sender {
    speedType = 1;
    calibrationConstant = 67;
    filteringType = 1;
    operationMode = 1;
    sliderValue = 0;
    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];
    [defaults setInteger:speedType forKey:@"speedType"];
    [defaults setFloat:calibrationConstant forKey:@"calibrationConstant"];
    [defaults setInteger:filteringType forKey:@"filteringType"];
    [defaults setInteger:operationMode forKey:@"operationMode"];
    [defaults setFloat:sliderValue forKey:@"sliderValue"];
    [defaults synchronize];
```

```
    [self updateGUI];
}

- (IBAction)dragOutside:(id)sender {
    [self obtainSliderValue];
}

- (IBAction)cancel:(id)sender {
    [self.delegate flipsideViewControllerDidFinish:self];
}



@end
```

RecordManager.h

```
//
//  RecordManager.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "Recorder.h"



@interface RecordManager : NSObject
{
    Recorder         *recorder;
  //  IBOutlet UILabel  *fileDescription;      // File description
    SoundManager      *sm;
    int32_t          blockSize;
    NSString* description;
}
```

```
@property(nonatomic) Recorder *recorder;
//@property(nonatomic,retain) UILabel *fileDescription;
@property(nonatomic,retain) SoundManager *sm;
@property(nonatomic) int32_t blockSize;
@property(nonatomic,retain) NSString *description;



char *OSTypeToStr(char *buf, OSType t);  // Sets path for file
void interruptionListener(void *inClientData,UInt32 inInterruptionState);
void propListener(void                  *inClientData,
                  AudioSessionPropertyID   inID,
                  UInt32                   inDataSize,
                  const void               *inData);

-(void)InitializeAudioSession;                          // Initialized Audio Session
-(void)stopRecord;                                      // Stops AudioQueue Recording for the
Recorder Object
-(void)setSoundManager:(SoundManager*)newSM;
-(id)initWithBlockSize:(int32_t)newBlockSize;
-(void)Reset:(int32_t)newBlockSize;

@end
```

RecordManager.m

```
//
//  RecordManager.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "RecordManager.h"
```

```
@implementation RecordManager

@synthesize sm;
@synthesize recorder;
//@synthesize fileDescription;
@synthesize blockSize;
@synthesize description;



// Constructor
-(id)initWithBlockSize:(int32_t)newBlockSize {
        if (self = [super init]) {
        self.blockSize = newBlockSize;
        [self InitializeAudioSession];
        }
        return self;
}

-(void)Reset:(int32_t)newBlockSize{
    self.blockSize = newBlockSize;
    recorder->Reset(self.blockSize);
}

// Sets SoundManager Object for the Recorder
-(void) setSoundManager:(SoundManager*)newSM
{
    sm = newSM;
    recorder->setSoundManager(sm);
}

// Initialized Audio Session
-(void) InitializeAudioSession {
```

```
        // Allocate our singleton instance for the recorder object
        recorder = new Recorder(blockSize);

        void * THIS = (__bridge void *)self;

        OSStatus error = AudioSessionInitialize(NULL, NULL, interruptionListener, THIS);
        if (error) printf("ERROR INITIALIZING AUDIO SESSION! %ld\n", error);
        else
        {
            UInt32 category = kAudioSessionCategory_PlayAndRecord;
            error = AudioSessionSetProperty(kAudioSessionProperty_AudioCategory,
sizeof(category), &category);
            if (error) printf("couldn't set audio category!");

            error =
AudioSessionAddPropertyListener(kAudioSessionProperty_AudioRouteChange,
propListener, THIS);
            if (error) printf("ERROR ADDING AUDIO SESSION PROP LISTENER! %ld\n",
error);
            UInt32 inputAvailable = 0;
            UInt32 size = sizeof(inputAvailable);

            // we do not want to allow recording if input is not available
            error = AudioSessionGetProperty(kAudioSessionProperty_AudioInputAvailable,
&size, &inputAvailable);
            if (error) printf("ERROR GETTING INPUT AVAILABILITY! %ld\n", error);


            // we also need to listen to see if input availability changes
            error =
AudioSessionAddPropertyListener(kAudioSessionProperty_AudioInputAvailable,
propListener, THIS);
            if (error) printf("ERROR ADDING AUDIO SESSION PROP LISTENER! %ld\n",
error);
```

```
        error = AudioSessionSetActive(true);
        if (error) printf("AudioSessionSetActive (true) failed");
    }
}



// Stops AudioQueue Recording for the Recorder Object
-(void)stopRecord
{
        recorder->StopRecord();
}

// Handles unexpected events
void interruptionListener(      void *   inClientData,
                        UInt32          inInterruptionState)
{
        RecordManager *THIS = (__bridge  RecordManager*)inClientData;
        if (inInterruptionState == kAudioSessionBeginInterruption)
        {
                if (THIS->recorder->IsRunning()) {
                        [THIS stopRecord];
                }
    }

}

// Handle unexpected events
void propListener(    void *                inClientData,
                AudioSessionPropertyID  inID,
                UInt32                  inDataSize,
                const void *            inData)
{
        RecordManager *THIS = (__bridge  RecordManager*)inClientData;
        if (inID == kAudioSessionProperty_AudioRouteChange)
        {
```

```
                CFDictionaryRef routeDictionary = (CFDictionaryRef)inData;

                CFNumberRef reason =
(CFNumberRef)CFDictionaryGetValue(routeDictionary,
CFSTR(kAudioSession_AudioRouteChangeKey_Reason));
                SInt32 reasonVal;
                CFNumberGetValue(reason, kCFNumberSInt32Type, &reasonVal);
                if (reasonVal != kAudioSessionRouteChangeReason_CategoryChange)
                {
                        // stop the queue if we had a non-policy route change
                        if (THIS->recorder->IsRunning()) {
                                [THIS stopRecord];
                        }
                }
        }

    }

    @end
```

Recorder.h

```
    //
    //  Recorder.h
    //  iSPL PRO
    //
    //  Created by Σαρρής Αντώνιος on 11/4/12.
    //  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
    //

    #include <AudioToolbox/AudioToolbox.h>
    #include <Foundation/Foundation.h>
    #include <libkern/OSAtomic.h>
    #import "SoundManager.h"
    #include "CAStreamBasicDescription.h"
```

```
#include "CAXException.h"
#include "string.h"
#include <algorithm>
#include <Accelerate/Accelerate.h>
#include <time.h>


#ifndef iSPL_PRO_Recorder_h
#define iSPL_PRO_Recorder_h



#endif


#define kNumberRecordBuffers	5	// number of buffers used by AudioQueue
#define SampleRate 44100.0


class Recorder
{
public:
    Recorder(int32_t blockSize);   // constructor
    void Reset(int32_t blockSize);
    ~Recorder();  // destructor

    SoundManager *sm;

    UInt32	GetNumberChannels()	const{	return mRecordFormat.NumberChannels(); } // returns number of channels
    CFStringRef	GetFileName()	const{ return mFileName; }	// returns file name
    AudioQueueRef Queue()	const{ return mQueue; }	// returns AudioQueue reference
    CAStreamBasicDescription	DataFormat()	const{	return	mRecordFormat;	}
```

```
// returns record format structure
        Boolean    IsRunning()              const{ return mIsRunning; }              // returns
recording status

        //void StartRecord(CFStringRef inRecordFile);
        void StartRecord();
        // starts AudioQueue Recording
        void StopRecord();                                                  // stops AudioQueue
Recording
        void setSoundManager(SoundManager *newSM);
        NSString *recordFile;



    private:
        CFStringRef                             mFileName;
        AudioQueueRef                       mQueue;
        AudioQueueBufferRef                 mBuffers[kNumberRecordBuffers];
        AudioFileID                             mRecordFile;
        SInt64                                  mRecordPacket;
// current packet number in record file
        CAStreamBasicDescription        mRecordFormat;
        Boolean                                 mIsRunning;
        int32_t             currentFrame;
        int64_t             totalSamples;
        int32_t             blockSize;
        double               kBufferDurationSeconds;

        void                    CopyEncoderCookieToFile();                          //
used for compressed audio format
        void                    SetupAudioFormat(UInt32 inFormatID);
        int                             ComputeRecordBufferSize(const
AudioStreamBasicDescription *format, float seconds);
```

```
        static void MyInputBufferHandler(void                                *inUserData,    // Main
Callback Function
                                         AudioQueueRef
inAQ,
                                         AudioQueueBufferRef                                 inBuffer,
                                         const AudioTimeStamp            *inStartTime,
                                         UInt32
            inNumPackets,
                                         const AudioStreamPacketDescription *inPacketDesc);


        static void UpdateSoundArray(void * inUserData, AudioQueueBufferRef inBuffer);
// Obtain normalized audio samples
        static void PerformComputation(void * inUserData,float * samples);                  //
performs SPL computations


    };
```

Recorder.mm

```
    //
    //  Recorder.cpp
    //  iSPL PRO
    //
    //  Created by Σαρρής Αντώνιος on 11/4/12.
    //  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
    //

    #include <iostream>
    #include "Recorder.h"



    // Estimate the requared size, in bytes, of a buffer in order to represent the given number
of seconds of audio data.
    int  Recorder::ComputeRecordBufferSize(const  AudioStreamBasicDescription  *format,
```

```
float seconds)
    {
        int packets, frames, bytes = 0;
        try {
            frames = (int)ceil(seconds * format->mSampleRate);

            if (format->mBytesPerFrame > 0)
                bytes = frames * format->mBytesPerFrame;
            else {
                UInt32 maxPacketSize;
                if (format->mBytesPerPacket > 0)
                    maxPacketSize = format->mBytesPerPacket;    // constant
packet size
                else {
                    UInt32 propertySize = sizeof(maxPacketSize);
                    XThrowIfError(AudioQueueGetProperty(mQueue,
kAudioQueueProperty_MaximumOutputPacketSize, &maxPacketSize,
                        &propertySize), "couldn't get queue's maximum output
packet size");
                }
                if (format->mFramesPerPacket > 0)
                    packets = frames / format->mFramesPerPacket;
                else
                    packets = frames;        // 1 frame in a packet
                if (packets == 0)
                    packets = 1;
                bytes = packets * maxPacketSize;
            }
        } catch (CAXException e) {
            char buf[256];
            fprintf(stderr, "Error: %s (%s)\n", e.mOperation, e.FormatError(buf));
            return 0;
        }
        return bytes;
    }
```

```
// Main Callback Function, called when an input buffers has been filled.
void Recorder::MyInputBufferHandler(	void *
                        inUserData,
                        AudioQueueRef
        inAQ,
                        AudioQueueBufferRef                                     inBuffer,
                        const AudioTimeStamp *
        inStartTime,
                        UInt32
        inNumPackets,
                        const AudioStreamPacketDescription* inPacketDesc)
{

    Recorder *aqr = (Recorder *)inUserData;  // pointer to  Recorder object
        try {
                if (inNumPackets > 0) {

                        aqr->mRecordPacket += inNumPackets;    // update current record
packet
                }
        UpdateSoundArray(inUserData, inBuffer);      // obtain normalized audio data -
perform main computation

        // if we're not stopping, re-enqueue the buffe so that it gets filled again
                if (aqr->IsRunning())
                        XThrowIfError(AudioQueueEnqueueBuffer(inAQ,     inBuffer,     0,
NULL), "AudioQueueEnqueueBuffer failed");

        } catch (CAXException e) {
                char buf[256];
                fprintf(stderr, "Error: %s (%s)\n", e.mOperation, e.FormatError(buf));
        }
}
```

```
// Obtains Normalized audio data from input buffer
void Recorder::UpdateSoundArray(void * inUserData, AudioQueueBufferRef inBuffer){

    Recorder *aqr = (Recorder *)inUserData;                    // pointer to  Recorder object
    int sampleCount = inBuffer->mAudioDataBytesCapacity / sizeof(SInt16);    // samples number in audio buffer

    // cast audio samples to float
    SInt16 * data = (SInt16 *) inBuffer->mAudioData;
    float *samples  = (float *)malloc(aqr->blockSize * sizeof(float));
    std::copy(data,data+sampleCount,samples);


    //Normalization from [-32768...32767] to [-1...+1]
    float num1 = 32768.0;
    vDSP_vsadd(samples, 1, &num1, samples, 1, aqr->blockSize);
    float num2 = 2.0 / 65535.0;
    vDSP_vsmul(samples, 1, &num2, samples, 1, aqr->blockSize);
    float num3 = -1.0;
    vDSP_vsadd(samples, 1, &num3, samples, 1, aqr->blockSize);

    PerformComputation(aqr,samples); // perfrm SPL computations
    aqr->currentFrame++;            // update current frame
    free(samples);

}

// Performs SPL computations
void Recorder::PerformComputation(void * inUserData,float * samples){
    Recorder *aqr = (Recorder *)inUserData;
    SoundManager *sm = (SoundManager *)aqr->sm;
    [sm PerformMainComputation:samples];
}
```

```
// Constructor
Recorder::Recorder(int32_t blockSize)
{
        mIsRunning = false;
        mRecordPacket = 0;
    currentFrame = 0;
    totalSamples = 0;
    this->blockSize = blockSize;
    this->kBufferDurationSeconds = (double)(blockSize / (double)SampleRate);
}

//Resetor
void Recorder::Reset(int32_t blockSize){
    mIsRunning = false;
        mRecordPacket = 0;
    currentFrame = 0;
    totalSamples = 0;
    this->blockSize = blockSize;
    this->kBufferDurationSeconds = (double)(blockSize / (double)SampleRate);
}

// Destructor
Recorder::~Recorder()
{
        AudioQueueDispose(mQueue, TRUE);
}

// Copy a queue's encoder's magic cookie to an audio file.
void Recorder::CopyEncoderCookieToFile()
{
    UInt32 propertySize;
        // get the magic cookie, if any, from the converter
        OSStatus err = AudioQueueGetPropertySize(mQueue,
kAudioQueueProperty_MagicCookie, &propertySize);
```

```
        // we can get a noErr result and also a propertySize == 0
        // -- if the file format does support magic cookies, but this file doesn't have one.
        if (err == noErr && propertySize > 0) {
                Byte *magicCookie = new Byte[propertySize];
                UInt32 magicCookieSize;
                XThrowIfError(AudioQueueGetProperty(mQueue,
kAudioQueueProperty_MagicCookie, magicCookie, &propertySize), "get audio converter's
magic cookie");
                magicCookieSize = propertySize;          // the converter lies and tell us the
wrong size

                // now set the magic cookie on the output file
                UInt32 willEatTheCookie = false;
                // the converter wants to give us one; will the file take it?
                err                =                AudioFileGetPropertyInfo(mRecordFile,
kAudioFilePropertyMagicCookieData, NULL, &willEatTheCookie);
                if (err == noErr && willEatTheCookie) {
                        err                =                AudioFileSetProperty(mRecordFile,
kAudioFilePropertyMagicCookieData, magicCookieSize, magicCookie);
                        XThrowIfError(err, "set audio file's magic cookie");
                }
                delete[] magicCookie;
        }
    }

    //  Sets the audio format for the audio Queue
    void Recorder::SetupAudioFormat(UInt32 inFormatID)
    {
        memset(&mRecordFormat, 0, sizeof(mRecordFormat));

        UInt32 size = sizeof(mRecordFormat.mSampleRate);
        XThrowIfError(AudioSessionGetProperty(
        kAudioSessionProperty_CurrentHardwareSampleRate,
                        &size,
```

```
                        &mRecordFormat.mSampleRate), "couldn't get hardware
sample rate");

        size = sizeof(mRecordFormat.mChannelsPerFrame);
        XThrowIfError(AudioSessionGetProperty(
        kAudioSessionProperty_CurrentHardwareInputNumberChannels,
                        &size,
                        &mRecordFormat.mChannelsPerFrame), "couldn't get input
channel count");

        mRecordFormat.mFormatID = inFormatID;
        if (inFormatID == kAudioFormatLinearPCM)
        {
                // if we want pcm, default to signed 16-bit little-endian
                mRecordFormat.mFormatFlags = kLinearPCMFormatFlagIsSignedInteger |
kLinearPCMFormatFlagIsPacked;
                mRecordFormat.mBitsPerChannel = 16;
        mRecordFormat.mChannelsPerFrame=1;
                mRecordFormat.mBytesPerPacket = mRecordFormat.mBytesPerFrame =
(mRecordFormat.mBitsPerChannel / 8) * mRecordFormat.mChannelsPerFrame;
                mRecordFormat.mFramesPerPacket = 1;
        }
      printf("SampleRate:%f\n",mRecordFormat.mSampleRate);
      printf("Channels:%lu\n",mRecordFormat.mChannelsPerFrame);
   }


   // Starts Audioqueue Recording
   void Recorder::StartRecord()
   {
        int i, bufferByteSize;
        UInt32 size;


        try {
```

```
		// specify the recording format
		SetupAudioFormat(kAudioFormatLinearPCM);

		// create the queue


		XThrowIfError(AudioQueueNewInput(
				&mRecordFormat,
				MyInputBufferHandler,
				this, // userData
				NULL, // run loop
				NULL, // run loop mode
				0,// flags
				&mQueue), "AudioQueueNewInput failed");

		// get the record format back from the queue's audio converter --
		// the file may require a more specific stream description than was necessary to create the encoder.
		mRecordPacket = 0;

		size = sizeof(mRecordFormat);
		XThrowIfError(AudioQueueGetProperty(mQueue, kAudioQueueProperty_StreamDescription,
					&mRecordFormat, &size), "couldn't get queue's format");









		// copy the cookie first to give the file object as much info as we can about
```

```
the data going in
                // not necessary for pcm, but required for some compressed audio
                //CopyEncoderCookieToFile();

                // allocate and enqueue buffers
                bufferByteSize = ComputeRecordBufferSize(&mRecordFormat, this->kBufferDurationSeconds);       // enough bytes for half a second
                for (i = 0; i < kNumberRecordBuffers; ++i) {
                        XThrowIfError(AudioQueueAllocateBuffer(mQueue, bufferByteSize, &mBuffers[i]),
                    "AudioQueueAllocateBuffer failed");
                        XThrowIfError(AudioQueueEnqueueBuffer(mQueue, mBuffers[i], 0, NULL),
                    "AudioQueueEnqueueBuffer failed");
                }
                // start the queue
                mIsRunning = true;
                XThrowIfError(AudioQueueStart(mQueue, NULL), "AudioQueueStart failed");
        }
        catch (CAXException &e) {
    //              char buf[256];
    //              fprintf(stderr, "Error: %s (%s)\n", e.mOperation, e.FormatError(buf));
        }
        catch (...) {
                fprintf(stderr, "An unknown error occurred\n");
        }

    }



    // Stops AudioQueue Recording
    void Recorder::StopRecord()
    {
```

```
        // end recording
        mIsRunning = false;
        XThrowIfError(AudioQueueStop(mQueue, true), "AudioQueueStop failed");
        AudioQueueDispose(mQueue, true);

}

// Sets SoundManager Object
void Recorder::setSoundManager(SoundManager *newSM){
   sm = newSM;
}
```

SoundManager.h

```
//
//  SoundManager.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

//#import <Foundation/Foundation.h>

#import "FFT.h"
#import "WeightedFiltering.h"
#import "SPL.h"
#import "GraphVisualizer.h"
#import "GraphBar.h"
#import "NeedleView.h"
#import "GraphPulse.h"
```

```
#define FrameSize 44100



@class SoundManager;

@protocol SoundManagerDelegate

- (void)UpdateCurrentSPLLabel:(SoundManager *)sm withCurrSPL: (float)currSPL
                withAvgSPL:  (float)  avgSPL  withMaxSPL:(float)maxSPL
withMinSPL:(float) minSPL;
-(void) UpdateTypicalSPLLabel: (SoundManager *)sm withCurrSPL: (float)currSPL
withFilteringType:(int) ft;

@end



@interface SoundManager : NSObject {

    FFT             *fft;
    WeightedFiltering *wf;
    SPL             *spl;

    NSMutableData       *squaredFilterResponse;     // squared filter responce for current
audio buffer
    NSMutableData      *squaredFFTData;            // squared FFT Components for current
audio buffer
    int             blockSize;              // size of current audio buffer

    int32_t          totalFrames;            // number of total frames processed
    GraphVisualizer   *gv;
    GraphBar         *gb;
    NeedleView        *nv;
    GraphPulse        *gp;
```

```
    UILabel         *cl;

    NSArray         *paths;
    NSString *documentsDirectory;
    NSString *audioFile;
    NSString *audioblocksFile;
    NSString *fftFile;
    NSString *filterFile;
    NSString *meanEnergyFile;
    NSString *currentSPLFile;
    NSString *avgSPLFile;
    NSString *maxSPLFile;
    NSString *minSPLFile;


    int32_t inframe_blocks;
    int32_t current_inframe_block;
    NSMutableData *currentBlockData;
}


@property (nonatomic,retain) id <SoundManagerDelegate> delegate;
@property(nonatomic,retain) NSMutableData *currentBlockData;
@property(nonatomic,retain) NSArray *paths;
@property(nonatomic,retain) NSString *documentsDirectory;
@property(nonatomic,retain) NSString *audioFile;
@property(nonatomic,retain) NSString *audioblocksFile;
@property(nonatomic,retain) NSString *fftFile;
@property(nonatomic,retain) NSString *filterFile;
@property(nonatomic,retain) NSString *meanEnergyFile;
@property(nonatomic,retain) NSString *currentSPLFile;
@property(nonatomic,retain) NSString *avgSPLFile;
@property(nonatomic,retain) NSString *maxSPLFile;
@property(nonatomic,retain) NSString *minSPLFile;
@property(nonatomic,retain) GraphPulse *gp;
```

```
    @property(nonatomic) int32_t totalFrames;
    @property(nonatomic) int32_t blockSize;

    @property(nonatomic,retain) FFT *fft;
    @property(nonatomic,retain) WeightedFiltering *wf;
    @property(nonatomic,retain) SPL *spl;

    @property(nonatomic,retain) NSMutableData *squaredFilterResponse;
    @property(nonatomic,retain) NSMutableData *squaredFFTData;
    @property(nonatomic,retain) GraphVisualizer *gv;
    @property(nonatomic,retain) GraphBar *gb;
    @property(nonatomic,retain) NeedleView *nv;
    @property(nonatomic,retain) UILabel *cl;

    @property(nonatomic) int32_t inframe_blocks;
    @property(nonatomic) int32_t current_inframe_block;



    -(void) PerformMainComputation:(float *) samples;                // Main SPL Computation
called from Main Audioqueue Callback


    -(id)initWithWeightedFiltering:(WeightedFiltering *)newWF        // Constructor
                        withFFT:(FFT * ) newFFT
                        withSPL:(SPL * ) newSPL
                  withBlockSize:(int) newBlockSize
              withGraphVisualizer: (GraphVisualizer *) newGV
                   withGraphBar:(GraphBar *)newGB
                 withGraphPulse: (GraphPulse *) newGP
                 withNeedleView:(NeedleView *)newNV
                   withCurLabel:(UILabel *)newCL;
    -(void)ResetWithWeightedFiltering:(WeightedFiltering *)newWF
                        withFFT:(FFT *) newFFT
```

```
                withSPL:(SPL *) newSPL
           withBlockSize:(int) newBlockSize;



@end
```

SoundManager.m

```
//
//  SoundManager.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "SoundManager.h"

@implementation SoundManager



@synthesize delegate = _delegate;
@synthesize paths;
@synthesize documentsDirectory;
@synthesize audioFile;
@synthesize audioblocksFile;
@synthesize fftFile;
@synthesize filterFile;
@synthesize meanEnergyFile;
@synthesize currentSPLFile;
@synthesize avgSPLFile;
@synthesize maxSPLFile;
@synthesize minSPLFile;

@synthesize fft;
```

```
@synthesize wf;
@synthesize spl;

@synthesize squaredFFTData;
@synthesize squaredFilterResponse;

@synthesize totalFrames;
@synthesize gv;
@synthesize gb;
@synthesize nv;
@synthesize cl;
@synthesize gp;
@synthesize currentBlockData;

@synthesize inframe_blocks;
@synthesize current_inframe_block;
@synthesize blockSize;



// Constructor
-(id)initWithWeightedFiltering:(WeightedFiltering *)newWF
                     withFFT:(FFT *) newFFT
                     withSPL:(SPL *) newSPL
               withBlockSize:(int) newBlockSize
          withGraphVisualizer:(GraphVisualizer *) newGV
                withGraphBar:(GraphBar *)newGB
              withGraphPulse:(GraphPulse *)newGP
              withNeedleView:(NeedleView *)newNV
                withCurLabel:(UILabel *)newCL
{
    if (self = [super init]) {
        wf = newWF;
        fft = newFFT;
        spl = newSPL;
        blockSize = newBlockSize;
```

```
        gv = newGV;
        gb = newGB;
        gp = newGP;
        nv = newNV;
        cl= newCL;
        totalFrames = 0;
        [gv clear];
        [gb clear];
        [gp clear];

        //Initialization for tracking the current_inframe_block within the current audio second(frame).
        inframe_blocks = ceil(FrameSize / blockSize);
        current_inframe_block = 1;



        }
        return self;
    }


    -(void)ResetWithWeightedFiltering:(WeightedFiltering *)newWF
                        withFFT:(FFT *) newFFT
                        withSPL:(SPL *) newSPL
                  withBlockSize:(int) newBlockSize{

        wf = newWF;
        fft = newFFT;
        spl = newSPL;
        blockSize = newBlockSize;
        totalFrames = 0;


      //Initialization for tracking the current_inframe_block within the current audio
```

```
second(frame).
        inframe_blocks = ceil(FrameSize / blockSize);
        current_inframe_block = 1;



    }



    // Main SPL Computation called from Main Audioqueue Callback
    -(void) PerformMainComputation:(float *) audioSamples {



        float * _squaredFFTData;
        float * _squaredFilterResponse;

        // Inframe block counter is used in order to locate the position of the current block
        // whithin the audio frame of AugmentedFrameSize. This is critical in order to estimate
        // correctly the corresponing filterting responses.

        currentBlockData = nil;
        currentBlockData = [[NSMutableData alloc] init];
        [currentBlockData appendBytes:audioSamples length:blockSize*sizeof(float)];




        // obtain squared filter response for current audio block
        // Mind that the inframe position of the audio block should ne used!!!!
        _squaredFilterResponse = [wf outputSquaredFilterResponse:current_inframe_block];

        squaredFilterResponse = nil;
        squaredFilterResponse = [[NSMutableData alloc] init];

        [squaredFilterResponse                    appendBytes:_squaredFilterResponse
length:blockSize*sizeof(float)];
        [fft setSoundBuffer:currentBlockData];
```

```
        // obtain squared FFT data for current audio block
        _squaredFFTData = [fft outputSquaredFFTData];

        squaredFFTData = nil;
        squaredFFTData = [[NSMutableData alloc] init ];

        [squaredFFTData appendBytes:_squaredFFTData length:blockSize*sizeof(float)];

        // compute SPL measurements for current audio block
        [spl setSquaredFilteredResponse:squaredFilterResponse
withSquaredFFTData:squaredFFTData];
        [spl computeSPLMeasurements];

        [gv setPower:spl.currentSPL];
        [gv setNeedsDisplay];
        [gb setPower:spl.currentSPL];
        [gb setNeedsDisplay];
        [gp setPower:spl.currentSPL withMin:spl.minSPL];
        [gp setNeedsDisplay];
        [nv setPower:spl.currentSPL];
        [nv setNeedsDisplay];

        [self.delegate UpdateCurrentSPLLabel:self withCurrSPL: spl.currentSPL
withAvgSPL: spl.avgSPL withMaxSPL:spl.maxSPL withMinSPL:spl.minSPL];
        [self.delegate UpdateTypicalSPLLabel:self withCurrSPL:spl.currentSPL
withFilteringType:spl.FilteringType];



        // Update inframe block counter by 1 and if it happens to be grater than inframe blocks
        // set it again equal to 1.
        current_inframe_block++;

        if(current_inframe_block > inframe_blocks){
           current_inframe_block = 1;
```

```
        totalFrames++;

    }


}



@end
```

WeightedFiltering.h

```
//
//  WeightedFiltering.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

//#import <Foundation/Foundation.h>
#import <Accelerate/Accelerate.h>



@interface WeightedFiltering : NSObject {


    int32_t        filteringType;
    int32_t        N;
    float          Fs;
    float   * _squaredFilterResponce;
    float        * frequencies;
    float        * filterResponse;
```

```
}

@property  int32_t   filteringType;
@property(nonatomic) int32_t   N;
@property(nonatomic)  float     Fs;
@property float * _squaredFilterResponse;
@property  float   * frequencies;
@property  float   * filterResponse;




-(id)initWithFilteringType:(int32_t ) newFilteringType
            ofBufferSize:(int32_t) bufferSize
      ofSamplingFrequency:(float)samplingFreaquency;


-(void)createFrequenciesVector;
-(void)createFrequenciesVector:(int32_t)currentBlock;
-(void)getAWeightingResponse;
-(void)getBWeightingResponse;
-(void)getCWeightingResponse;
-(void)applyFiltering;
-(float *)outputSquaredFilterResponse:(int32_t)currentBlock;
-(void)ResetWithFilteringType:(int32_t) newFilteringType
ofBufferSize:(int32_t) bufferSize
        ofSamplingFrequency:(float)samplingFrequency;
```

```
@end
```

WeightedFiltering.m

```
//
//  WeightedFiltering.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "WeightedFiltering.h"

@implementation WeightedFiltering

@synthesize    filteringType;
@synthesize    N;
@synthesize    Fs;
@synthesize   _squaredFilterResponse;
@synthesize    frequencies;
@synthesize    filterResponse;

-(id)initWithFilteringType:(int32_t) newFilteringType
              ofBufferSize:(int32_t) bufferSize
      ofSamplingFrequency:(float)samplingFreaquency
{
   if(self=[super init]){
       self.filteringType = newFilteringType;
       self.N = bufferSize;
       self.Fs = samplingFreaquency;

       self._squaredFilterResponse = (float *)malloc(N*sizeof(float));
```

```
        self.frequencies = (float *) malloc(N*sizeof(float));
        self.filterResponse = (float *) malloc(N*sizeof(float));
        [self createFrequenciesVector];
    }
    return self;
}


-(void)ResetWithFilteringType:(int32_t) newFilteringType
ofBufferSize:(int32_t) bufferSize
          ofSamplingFrequency:(float)samplingFrequency{
    self.filteringType = newFilteringType;
    self.N = bufferSize;
    self.Fs = samplingFrequency;

    free(self._squaredFilterResponse);
    free(self.frequencies);
    free(self.filterResponse);

    self._squaredFilterResponse = (float *)malloc(N*sizeof(float));
    self.frequencies = (float *) malloc(N*sizeof(float));
    self.filterResponse = (float *) malloc(N*sizeof(float));



}


-(void)createFrequenciesVector{
    float start = 0.0;
    float end = Fs * (N-1) / N;
    vDSP_vgen(&start, &end, frequencies, 1, N);
}

-(void)createFrequenciesVector:(int32_t)currentBlock{
```

```
    int32_t frame_size = self.Fs;
    int32_t block_size = self.N;
    float start = (self.Fs / frame_size)*block_size*(currentBlock-1);
    float end = (self.Fs / frame_size)*(currentBlock*block_size-1);
    vDSP_vgen(&start, &end, frequencies, 1, N);
}

-(void)getAWeightingResponse{

    float c1 = 12200 * 12200;
    float c2 = 20.6 * 20.6;
    float c3 = 107.7 * 107.7;
    float c4 = 737.9 * 739.7;
    float *x1, *x2, *x3, *x4, *x5, *x6;

    x1 = (float *)malloc(N*sizeof(float));
    x2 = (float *)malloc(N*sizeof(float));
    x3 = (float *)malloc(N*sizeof(float));
    x4 = (float *)malloc(N*sizeof(float));
    x5 = (float *)malloc(N*sizeof(float));
    x6 = (float *)malloc(N*sizeof(float));

    vDSP_vsq(frequencies, 1, x1, 1, N); // x1 = f^2
    vDSP_vsq(x1, 1, x2, 1, N);          // x2 = f^4
    vDSP_vsmul(x2, 1, &c1, x2, 1, N);   // x2 = c1 * f^4
    vDSP_vsadd(x1, 1, &c2, x3, 1, N);   // x3 = f^2 + c2
    vDSP_vsadd(x1, 1, &c1, x4, 1, N);   // x4 = f^2 + c1
    vDSP_vsadd(x1, 1, &c3, x5, 1, N);   // x5 = f^2 + c3
    vvsqrtf(x5, x5, &N);                // x5 = (f^2 + c3)^0.5
    vDSP_vsadd(x1, 1, &c4, x6, 1, N);   // x6 = f^2 + c4
    vvsqrtf(x6, x6, &N);                // x6 = (f^2 + c4)^0.5
    vDSP_vmul(x3, 1, x4, 1, x3, 1, N);  // x3 = (f^2 + c2) * (f^2 + c1)
    vDSP_vmul(x3, 1, x5, 1, x3, 1, N);  // x3 = (f^2 + c2) * (f^2 + c1) * (f^2 + c3)
    vDSP_vmul(x3, 1, x6, 1, x3, 1, N);  // x3 = (f^2 + c2) * (f^2 + c1) * (f^2 + c3) * (f^2 +
c4)
```

```
        vvdivf(filterResponse, x2, x3, &N); // RA(f) = (c1 * f^4)/(f^2 + c2) * (f^2 + c1) * (f^2
+ c3) * (f^2 + c4)
        free(x1);
        free(x2);
        free(x3);
        free(x4);
        free(x5);
        free(x6);


    }

    -(void)getBWeightingResponse{

        float c1 = 12200 * 12200;
        float c2 = 20.6 * 20.6;
        float c3 = 158.5 * 158.5;

        float *y1, *y2, *y3, *y4, *y5;

        y1 = (float *)malloc(N*sizeof(float));
        y2 = (float *)malloc(N*sizeof(float));
        y3 = (float *)malloc(N*sizeof(float));
        y4 = (float *)malloc(N*sizeof(float));
        y5 = (float *)malloc(N*sizeof(float));


        vDSP_vsq(frequencies, 1, y1, 1, N);         // x1 = f^2
        vDSP_vmul(frequencies, 1, y1, 1, y2, 1, N);  // x2 = f^3
        vDSP_vsmul(y2, 1, &c1, y2, 1, N);           // x2 = c1 * f^3
        vDSP_vsadd(y1, 1, &c2, y3, 1, N);   // x3 = f^2 + c2
        vDSP_vsadd(y1, 1, &c1, y4, 1, N);   // x4 = f^2 + c1
        vDSP_vsadd(y1, 1, &c3, y5, 1, N);   // x5 = f^2 + c3
        vvsqrtf(y5, y5, &N);                // x5 = (f^2 + c3)^0.5
        vDSP_vmul(y3, 1, y4, 1, y3, 1, N);  // x3 =  (f^2 + c2) * (f^2 + c1)
```

```
        vDSP_vmul(y3, 1, y5, 1, y3, 1, N);  // x3 = (f^2 + c2) * (f^2 + c1) * (f^2 + c3)^0.5
        vvdivf(filterResponse, y2, y3, &N); // RB(f)  = (c1 * f^3)/(f^2 + c2) * (f^2 + c1) * (f^2
+ c3)^0.5
        free(y1);
        free(y2);
        free(y3);
        free(y4);
        free(y5);



    }

    -(void)getCWeightingResponse{

        float c1 = 12200 * 12200;
        float c2 = 20.6 * 20.6;

        float *z1, *z2, *z3, *z4;

        z1 = (float *)malloc(N*sizeof(float));
        z2 = (float *)malloc(N*sizeof(float));
        z3 = (float *)malloc(N*sizeof(float));
        z4 = (float *)malloc(N*sizeof(float));



        vDSP_vsq(frequencies, 1, z1, 1, N); // x1 = f^2

        vDSP_vsmul(z1, 1, &c1, z2, 1, N);

        vDSP_vsadd(z1, 1, &c2, z3, 1, N);   // x3 = f^2 + c2
        vDSP_vsadd(z1, 1, &c1, z4, 1, N);   // x4 = f^2 + c1
        vDSP_vmul(z3, 1, z4, 1, z3, 1, N);  // x3 = (f^2 + c2) * (f^2 + c1)
        vvdivf(filterResponse, z2, z3, &N); // RC(f) = (c1 * f^2)/(f^2 + c2) * (f^2 + c1)
        free(z1);
        free(z2);
```

```
    free(z3);
    free(z4);

}

-(void)applyFiltering{

    float one = 1;

    switch (filteringType) {
        case 0:
            vDSP_vfill(&one, _squaredFilterResponse, 1, N);
            break;
        case 1:
            [self getAWeightingResponse];
            vDSP_vsq(filterResponse, 1, _squaredFilterResponse, 1, N);
            break;
        case 2:
            [self getBWeightingResponse];
            vDSP_vsq(filterResponse, 1, _squaredFilterResponse, 1, N);
            break;
        case 3:
            [self getCWeightingResponse];
            vDSP_vsq(filterResponse, 1, _squaredFilterResponse, 1, N);
            break;
        default:
            break;
    }
}



-(float *)outputSquaredFilterResponse:(int32_t)currentBlock{
    [self createFrequenciesVector:currentBlock];
    [self applyFiltering];
```

```
    return _squaredFilterResponse;
}


@end
```

FFT.h

```
//
//  FFT.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

//#import <Foundation/Foundation.h>
//#import "FFTCore.h"
#include <Accelerate/Accelerate.h>

@interface FFT : NSObject {

    //FFTCore        * fftc;
    float          * _soundArray;
    NSMutableData   * soundArray;
    //NSMutableData   * squaredFFTData;
    float          * fftSoundArray;
    int32_t          soundArraySize;
    float          * realFFTComponents;
    float          * imagFFTComponents;
    //float          * complexMeasures;
    //DSPSplitComplex   stride;
    float          * realSquare;
    float          * imagSquare;
    float          * realImagSquareAdd;
```

```
    //These variables are needed in order to bypass the FFTCore class.

    COMPLEX_SPLIT   A;
    FFTSetup        setupReal;
    uint32_t        log2n;
    uint32_t        n, nOver2;
    uint32_t        n_max,nOver2_max;
    int32_t         stride;
    //uint32_t        i;
    float           *realComponents;
    float           *imagComponents;
    float           *obtainedReal;
    float           *originalReal;
    float           scale;



    // Bypass mallocs by using NSMutableData.
    NSMutableData * _realp;
    NSMutableData * _imagp;
    NSMutableData * _obtainedReal;
    NSMutableData * _realComponents;
    NSMutableData * _imagComponents;
    NSMutableData * _realSquare;
    NSMutableData * _imagSquare;
    NSMutableData * _realImagSquareAdd;



}

//@property(nonatomic) FFTCore * fftc;
@property(nonatomic) float * _soundArray;
@property(nonatomic,retain) NSMutableData *soundArray;//
```

```
//@property(nonatomic,retain) NSMutableData *squaredFFTData;//
@property(nonatomic) float * fftSoundArray;
@property(nonatomic) int32_t soundArraySize;
@property(nonatomic) float * realFFTComponents;
@property(nonatomic) float * imagFFTComponents;
//@property(nonatomic) float * complexMeasures;
//@property(nonatomic) DSPSplitComplex stride;
@property(nonatomic) float * realSquare;
@property(nonatomic) float * imagSquare;
@property(nonatomic) float * realImagSquareAdd;


// Properties for bypassing the FFTCore class.
@property(nonatomic) float *realComponents;
@property(nonatomic) float *imagComponents;
@property(nonatomic) float *obtainedReal;
@property(nonatomic) float *originalReal;


//Properties for bypassing malloc.
@property(nonatomic,retain) NSMutableData * _realp;
@property(nonatomic,retain) NSMutableData * _imagp;
@property(nonatomic,retain) NSMutableData * _obtainedReal;
@property(nonatomic,retain) NSMutableData * _realComponents;
@property(nonatomic,retain) NSMutableData * _imagComponents;
@property(nonatomic,retain) NSMutableData * _realSquare;
@property(nonatomic,retain) NSMutableData * _imagSquare;
@property(nonatomic,retain) NSMutableData * _realImagSquareAdd;


-(id)initWithSize:(int32_t) newSize withMaxBlockSize:(int32_t)maxBlockSize;
-(void)setSoundBuffer:(NSMutableData *)newSoundArray;
-(float *) outputSquaredFFTData;
-(void)computeFFT;
-(void)getRealImagFFTComponents;
```

```
//-(void)getComplexMeasures;
//-(void)getDSPSplitStride;
-(void)getRealSquare;
-(void)getImagSquare;
-(void)getRealImagSquareAdd;
-(void)Reset:(int32_t) newSize;



@end
```

FFT.m

```
//
//  FFT.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "FFT.h"

@implementation FFT

//@synthesize fftc;
@synthesize _soundArray;
@synthesize  soundArray;

@synthesize fftSoundArray;
@synthesize soundArraySize;
@synthesize realFFTComponents;
@synthesize imagFFTComponents;
//@synthesize complexMeasures;
//@synthesize stride;
@synthesize realSquare;
```

```
@synthesize imagSquare;
@synthesize realImagSquareAdd;

//Additional synthesizes for bypassing the FFTCore class.
@synthesize realComponents;
@synthesize imagComponents;
@synthesize obtainedReal;
@synthesize originalReal;



//Additional synthesizes for bypassing mallocs.
@synthesize _realp;
@synthesize _imagp;
@synthesize _obtainedReal;
@synthesize _realComponents;
@synthesize _imagComponents;
@synthesize _realSquare;
@synthesize _imagSquare;
@synthesize _realImagSquareAdd;




//CHANGED!!!
-(id)initWithSize:(int32_t) newSize withMaxBlockSize:(int32_t)maxBlockSize{

    if(self=[super init]){
        self.soundArraySize = newSize;
        //self.fftc = new FFTCore(soundArraySize);

        //Here is the FFTCore() constructor code.
        n = newSize;
        stride = 1;
        nOver2 = n/2;
        log2n = (uint32_t) log2(n);
```

```
n_max = maxBlockSize;
nOver2_max = maxBlockSize/2;



/*
//Initialize NSMutableData objects.
_realp = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_imagp = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_obtainedReal = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realComponents = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_imagComponents = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realSquare = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_imagSquare = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realImagSquareAdd = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];



self->A.realp = (float *)[_realp mutableBytes];
self->A.imagp = (float *)[_imagp mutableBytes];
self.obtainedReal = (float *)[_obtainedReal mutableBytes];
self.realComponents = (float *)[_realComponents mutableBytes];
self.imagComponents = (float *)[_imagComponents mutableBytes];
self.realSquare = (float *)[_realSquare mutableBytes];
self.imagSquare = (float *)[_imagSquare mutableBytes];
self.realImagSquareAdd = (float *)[_realImagSquareAdd mutableBytes];
*/



//Allocate memory for the input operands and check its availability,
 // use the vector version to get 16-byte alignment.
self->A.realp = (float *) calloc(nOver2_max , sizeof(float));
self->A.imagp = (float *) calloc(nOver2_max , sizeof(float));
self.obtainedReal = (float *) calloc(n_max , sizeof(float));
self.realComponents = (float *) calloc(n_max , sizeof(float));
```

```
        self.imagComponents = (float *) calloc(n_max , sizeof(float));
        //self.complexMeasures = (float *) malloc(n * sizeof(float));
        self.realSquare = (float *) calloc(n_max , sizeof(float));
        self.imagSquare = (float *) calloc(n_max , sizeof(float));
        self.realImagSquareAdd = (float *) calloc(n_max , sizeof(float));

        //This object is passed as an argument. Check whether it should be allocated.
        //soundArray = [[NSMutableData alloc] init];

    }
    return self;
}


-(void)Reset:(int32_t) newSize {

    //self.fftc->Reset(newSize);
    self.soundArraySize = newSize;
    if(n!=newSize){


    n = newSize;
    stride = 1;
    nOver2 = n/2;
    log2n = (uint32_t) log2(n);
    //@autoreleasepool {

     /*
        _realp = nil;
        _imagp = nil;
        _obtainedReal = nil;
        _realComponents = nil;
        _imagComponents = nil;
        _realSquare = nil;
        _imagSquare = nil;
```

```
    _realImagSquareAdd = nil;
 */



/*
   free(self->A.realp);
   free(self->A.imagp);
   free(self.obtainedReal);
   free(self.realComponents);
   free(self.imagComponents);
   ////////////////////////////free(self.complexMeasures);
   free(self.realSquare);
   free(self.imagSquare);
   free(self.realImagSquareAdd);



   self->A.realp = NULL;
   self->A.imagp = NULL;
   self.obtainedReal = NULL;
   self.realComponents = NULL;
   self.imagComponents = NULL;
   self.realSquare = NULL;
   self.imagSquare = NULL;
   self.realImagSquareAdd = NULL;

*/



/*
//Initialize NSMutableData objects.
_realp = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_imagp = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_obtainedReal = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realComponents = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
```

```
_imagComponents = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realSquare = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_imagSquare = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];
_realImagSquareAdd = [[NSMutableData alloc] initWithCapacity:n*sizeof(float)];



self->A.realp = (float *)[_realp mutableBytes];
self->A.imagp = (float *)[_imagp mutableBytes];
self.obtainedReal = (float *)[_obtainedReal mutableBytes];
self.realComponents = (float *)[_realComponents mutableBytes];
self.imagComponents = (float *)[_imagComponents mutableBytes];
self.realSquare = (float *)[_realSquare mutableBytes];
self.imagSquare = (float *)[_imagSquare mutableBytes];
self.realImagSquareAdd = (float *)[_realImagSquareAdd mutableBytes];
//}
*/



//delete self.fftc;
// Here is the ~FFTCore() destructor code.









/*
self->A.realp = NULL;
self->A.imagp = NULL;
self.obtainedReal = NULL;
self.realComponents = NULL;
self.imagComponents = NULL;
self.realSquare = NULL;
self.imagSquare = NULL;
```

```
    self.realImagSquareAdd = NULL;
    */

    //self.fftc = new FFTCore(newSize);
    //Here is the FFTCore() constructor code.

    // Allocate memory for the input operands and check its availability,
    // use the vector version to get 16-byte alignment.

  /*
    self->A.realp = (float *) calloc(nOver2 , sizeof(float));
    self->A.imagp = (float *) calloc(nOver2 , sizeof(float));
    self.obtainedReal = (float *) calloc(n , sizeof(float));
    self.realComponents = (float *) calloc(n , sizeof(float));
    self.imagComponents = (float *) calloc(n , sizeof(float));
    //self.complexMeasures = (float *) malloc(n * sizeof(float));
    self.realSquare = (float *) calloc(n , sizeof(float));
    self.imagSquare = (float *) calloc(n , sizeof(float));
    self.realImagSquareAdd = (float *) calloc(n , sizeof(float));
  */
    }

}

-(float *) outputSquaredFFTData {

    [self computeFFT];
    [self getRealImagFFTComponents];
    //[self getComplexMeasures];
    //[self getDSPSplitStride];
    //[self getRealSquare];
    //[self getImagSquare];
    [self getRealImagSquareAdd];

    return realImagSquareAdd;
```

```
}


//CHANGED!!!
-(void)setSoundBuffer:(NSMutableData *)newSoundArray{

    self.soundArray = newSoundArray;
    self._soundArray = (float *) [soundArray mutableBytes];
    //fftc->setSoundArray(_soundArray);


    // Here is the FFTCore setSoundArray code.
    originalReal = _soundArray;
}


//CHANGED!!!
-(void)computeFFT {

    //fftc->computeFFT();

    //Here is the FFTCore computeFFT() code.

    /* Look at the real signal as an interleaved complex vector  by
     * casting it.  Then call the transformation function vDSP_ctoz to
     * get a split complex vector, which for a real signal, divides into
     * an even-odd configuration. */
    vDSP_ctoz((COMPLEX *) originalReal, 2, &A, 1, nOver2);

    /* Set up the required memory for the FFT routines and check  its
     * availability. */
    setupReal = vDSP_create_fftsetup(log2n, FFT_RADIX2);

    /* Carry out a Forward FFT transform. */
    vDSP_fft_zrip(setupReal, &A, stride, log2n, FFT_FORWARD);
```

```
    /* Verify correctness of the results, but first scale it by  2. */
    scale = (float) 1.0 / 2;
    vDSP_vsmul(A.realp, 1, &scale, A.realp, 1, nOver2);
    vDSP_vsmul(A.imagp, 1, &scale, A.imagp, 1, nOver2);



    /* The output signal is now in a split real form.  Use the  function
     * vDSP_ztoc to get a split real vector. */
    vDSP_ztoc(&A, 1, (COMPLEX *) obtainedReal, 2, n);

    vDSP_destroy_fftsetup(setupReal);




    //fftSoundArray = fftc->getFFTData();
    // The FFTCore getFFTData() code is no longer needed.

}

//CHANGED!!!
-(void)getRealImagFFTComponents {

    //fftc->getRealImagComponents();

    // Here is the FFTCore getRealImagComponents() code.

    realComponents[0] = obtainedReal[0];
    realComponents[nOver2] = obtainedReal[1];
    imagComponents[0] = 0;
    imagComponents[nOver2] = 0;
    for (int i=1; i<nOver2; i++) {
        realComponents[i] = obtainedReal[2*i];
        imagComponents[i] = obtainedReal[2*i+1];
    }
```

```
    for (int i=1; i<nOver2; i++) {
        realComponents[nOver2+i] = realComponents[nOver2-i];
        imagComponents[nOver2+i] = -imagComponents[nOver2-i];
    }


    //realFFTComponents = fftc->getRealComponents();
    // The FFTCore getRealComponents() is no longer needed.


    //imagFFTComponents = fftc->getImagComponents();
    // The FFTCore getImagComponents() is no longer needed.

}


//CHANGED!!!
/*
-(void)getComplexMeasures{

    //fftc->computeComplexMeasures();
    // Here is the FFTCore computeComplexMeasures() code.
    vDSP_zvmags(&A, 1, complexMeasures, 1, n);

    //complexMeasures = fftc->getComplexMeasures();
    // The FFTCore getComplexMeasures() code is no longer needed.
}
*/


/*
-(void)getDSPSplitStride{
    stride = fftc->getStride();
}
*/
```

```
//CHANGED!!!
-(void)getRealSquare{
    //fftc->computeComplexMesures2();
    // Here is the FFTCore computeComplexMesures2() code.
    vDSP_vsq(realComponents, 1, realSquare, 1, n);
    vDSP_vsq(imagComponents, 1, imagSquare, 1, n);
    vDSP_vadd(realSquare, 1, imagSquare, 1, realImagSquareAdd, 1, n);

    //realSquare = fftc->getRealSquare();
    // The FFTCore getRealSquare() code is no longer needed.
}


//CHANGED!!!
-(void)getImagSquare{


    //fftc->computeComplexMesures2();
    // Here is the FFTCore computeComplexMesures2() code.
    vDSP_vsq(realComponents, 1, realSquare, 1, n);
    vDSP_vsq(imagComponents, 1, imagSquare, 1, n);
    vDSP_vadd(realSquare, 1, imagSquare, 1, realImagSquareAdd, 1, n);

    //imagSquare = fftc->getImagSquare();
    // The FFTCore getImagSquare() code is no longer needed.

}

//CHANGED!!!
-(void)getRealImagSquareAdd {

    //fftc->computeComplexMesures2();
```

```
    // Here is the FFTCore computeComplexMesures2() code.
    vDSP_vsq(realComponents, 1, realSquare, 1, n);
    vDSP_vsq(imagComponents, 1, imagSquare, 1, n);
    vDSP_vadd(realSquare, 1, imagSquare, 1, realImagSquareAdd, 1, n);

    //realImagSquareAdd = fftc->getRealImagSquareAdd();
    // The FFTCore getRealImagSquareAdd() is no longer needed.


}


@end
```

FFTCore.h

```
//
//  FFTCore.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#include <stdio.h>
#include <stdlib.h>
//#import "cmath"
#include <Accelerate/Accelerate.h>

#ifndef iSPL_PRO_FFTCore_h
#define iSPL_PRO_FFTCore_h

#endif
```

```
class FFTCore {



public:

    FFTCore(int32_t fftSize);
    void Reset(int32_t fftSize);
    ~FFTCore();
    void setSoundArray(float * inputArray);
    void     computeFFT();
    float *  getFFTData();
    void    getRealImagComponents();
    float * getRealComponents();
    float * getImagComponents();
    void    computeComplexMeasures();
    float * getComplexMeasures();
    DSPSplitComplex  getStride();

    void computeComplexMesures2();
    float *getRealSquare();
    float *getImagSquare();
    float *getRealImagSquareAdd();

private:

    COMPLEX_SPLIT   A;
    FFTSetup        setupReal;
    uint32_t        log2n;
    uint32_t        n, nOver2;
    int32_t         stride;
    uint32_t        i;
    float          *originalReal, *obtainedReal;
    float           scale;
    float          *realComponents, *imagComponents;
    float          *complexMeasures;
```

```
    float       *realSquare, *imagSquare;
    float       *realImagSquareAdd;

};
```

FFTCore.mm

```
//
//  FFTCore.cpp
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#include "FFTCore.h"
//#include <iostream>









FFTCore::FFTCore(int32_t fftSize) {



    n = fftSize;
    stride = 1;
    nOver2 = n/2;
    log2n = (uint32_t) log2(n);
    /* Allocate memory for the input operands and check its availability,
     * use the vector version to get 16-byte alignment. */
   A.realp = (float *) malloc(nOver2 * sizeof(float));
   A.imagp = (float *) malloc(nOver2 * sizeof(float));
```

```
    obtainedReal = (float *) malloc(n * sizeof(float));
    realComponents = (float *) malloc(n * sizeof(float));
    imagComponents = (float *) malloc(n * sizeof(float));
    complexMeasures = (float *) malloc(n * sizeof(float));
    realSquare = (float *) malloc(n * sizeof(float));
    imagSquare = (float *) malloc(n * sizeof(float));
    realImagSquareAdd = (float *) malloc(n * sizeof(float));

}


void FFTCore::Reset(int32_t fftSize){


    A.realp = NULL;
    A.imagp = NULL;
    obtainedReal = NULL;
    realComponents = NULL;
    imagComponents = NULL;
    complexMeasures = NULL;
    realSquare = NULL;
    imagSquare = NULL;
    realImagSquareAdd = NULL;


    free(A.realp);
    free(A.imagp);
    free(obtainedReal);
    free(realComponents);
    free(imagComponents);
    free(complexMeasures);
    free(realSquare);
    free(imagSquare);
    free(realImagSquareAdd);
```

```
n = fftSize;
stride = 1;
nOver2 = n/2;
log2n = (uint32_t) log2(n);
/* Allocate memory for the input operands and check its availability,
 * use the vector version to get 16-byte alignment. */
/*
  A.realp = new float[nOver2];
A.imagp = new float[nOver2];
obtainedReal = new float[n];

realComponents = new float[n];
imagComponents = new float[n];
complexMeasures = new float[n];
realSquare = new float[n];
imagSquare = new float[n];
realImagSquareAdd = new float[n];
*/

A.realp = (float *) malloc(nOver2 * sizeof(float));
A.imagp = (float *) malloc(nOver2 * sizeof(float));
obtainedReal = (float *) malloc(n * sizeof(float));
realComponents = (float *) malloc(n * sizeof(float));
imagComponents = (float *) malloc(n * sizeof(float));
complexMeasures = (float *) malloc(n * sizeof(float));
realSquare = (float *) malloc(n * sizeof(float));
imagSquare = (float *) malloc(n * sizeof(float));
realImagSquareAdd = (float *) malloc(n * sizeof(float));
```

```
}


void FFTCore::setSoundArray(float * inputArray){
    originalReal = inputArray;
}

FFTCore::~FFTCore() {

    /* Free the allocated memory. */

    free(A.realp);
    free(A.imagp);
    free(obtainedReal);
    free(realComponents);
    free(imagComponents);
    free(complexMeasures);
    free(realSquare);
    free(imagSquare);
    free(realImagSquareAdd);
}

void FFTCore::computeFFT(){

    /* Look at the real signal as an interleaved complex vector  by
     * casting it.  Then call the transformation function vDSP_ctoz to
     * get a split complex vector, which for a real signal, divides into
     * an even-odd configuration. */
    vDSP_ctoz((COMPLEX *) originalReal, 2, &A, 1, nOver2);

    /* Set up the required memory for the FFT routines and check  its
     * availability. */
    setupReal = vDSP_create_fftsetup(log2n, FFT_RADIX2);

    /* Carry out a Forward FFT transform. */
```

```
    vDSP_fft_zrip(setupReal, &A, stride, log2n, FFT_FORWARD);

    /* Verify correctness of the results, but first scale it by  2. */
    scale = (float) 1.0 / 2;
    vDSP_vsmul(A.realp, 1, &scale, A.realp, 1, nOver2);
    vDSP_vsmul(A.imagp, 1, &scale, A.imagp, 1, nOver2);



    /* The output signal is now in a split real form.  Use the  function
     * vDSP_ztoc to get a split real vector. */
    vDSP_ztoc(&A, 1, (COMPLEX *) obtainedReal, 2, n);

    vDSP_destroy_fftsetup(setupReal);
}

void FFTCore::getRealImagComponents(){

    realComponents[0] = obtainedReal[0];
    realComponents[nOver2] = obtainedReal[1];
    imagComponents[0] = 0;
    imagComponents[nOver2] = 0;
    for (int i=1; i<nOver2; i++) {
        realComponents[i] = obtainedReal[2*i];
        imagComponents[i] = obtainedReal[2*i+1];
    }
    for (int i=1; i<nOver2; i++) {
        realComponents[nOver2+i] = realComponents[nOver2-i];
        imagComponents[nOver2+i] = -imagComponents[nOver2-i];
    }
}

void FFTCore::computeComplexMesures2(){
    vDSP_vsq(realComponents, 1, realSquare, 1, n);
    vDSP_vsq(imagComponents, 1, imagSquare, 1, n);
    vDSP_vadd(realSquare, 1, imagSquare, 1, realImagSquareAdd, 1, n);
```

```
}

float * FFTCore::getRealSquare(){
   return realSquare;
}

float * FFTCore::getImagSquare(){
   return imagSquare;
}

float * FFTCore::getRealImagSquareAdd(){
   return realImagSquareAdd;
}

void FFTCore::computeComplexMeasures(){
   vDSP_zvmags(&A, 1, complexMeasures, 1, n);
}

float * FFTCore::getComplexMeasures(){
   return complexMeasures;
}

float * FFTCore::getRealComponents(){
   return realComponents;
}

float * FFTCore::getImagComponents(){
   return imagComponents;
}

DSPSplitComplex  FFTCore::getStride(){
   return A;
}

float* FFTCore::getFFTData(){
```

```
    return obtainedReal;
}
```

SPL.h

```
//
//  SPL.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//



#define kFilename @"measurements.txt"
#import <Accelerate/Accelerate.h>

#include <algorithm>

@interface SPL : NSObject {

    NSMutableData  *squaredFilterResponce;
    NSMutableData  *squaredFFTData;
    NSMutableData  *splMeasurements;
    NSMutableData  *avgSplMeasurements;
    NSMutableData  *minSplMeasurements;
    NSMutableData  *maxSplMeasurements;
    NSMutableData  *inframeinstantSPLMeasurements;
    int32_t        N;
    int32_t        inframeBlocks;
    int32_t        currentBlock;
    int64_t        measurementsNumber;
    float          instantSPL;
    float          currentSPL;
    float          currentAverage;
```

```
float         avgSPL;
float         maxSPL;
float         minSPL;
float         Fs;
float         calibrationConstant;
float         MeanEnergy;
int           OperationMode;
int           FilteringType;
bool          ShouldUpdate;
bool          FirstFramePassed;
NSMutableArray *displayMeasurements;
float *inwindowMeasurements;
NSDate *currentDateTime;
NSString *currentSPLText;
NSString *currentSPLValue;
NSString *currentSPLLine;

NSString *avgSPLText;
NSString *avgSPLValue;
NSString *avgSPLLine;

NSString *maxSPLText;
NSString *maxSPLValue;
NSString *maxSPLLine;

NSString *minSPLText;
NSString *minSPLValue;
NSString *minSPLLine;

NSDateFormatter *dateFormatter;
NSString *dateInStringFormated;

NSString *filterTypeValue;
NSString *operationModeValue;
```

```
    NSString *sampleRateText;
    NSString *sampleRateValue;
    NSString *sampleRateLine;

    NSString *blockSizeText;
    NSString *blockSizeValue;
    NSString *blockSizeLine;

    NSString *filterTypeText;



    NSString *filterTypeLine;
    NSString *operationModeText;
    NSString *operationModeLine;
    NSString *documentsDirectory;

}

@property(nonatomic,retain) NSString *documentsDirectory;
@property(nonatomic,retain)NSString *filterTypeLine;
@property(nonatomic,retain)NSString *operationModeText;
@property(nonatomic,retain)NSString *operationModeLine;

@property(nonatomic,retain)NSString *currentSPLText;
@property(nonatomic,retain)NSString *currentSPLValue;
@property(nonatomic,retain)NSString *currentSPLLine;

@property(nonatomic,retain)NSString *avgSPLText;
@property(nonatomic,retain)NSString *avgSPLValue;
@property(nonatomic,retain)NSString *avgSPLLine;

@property(nonatomic,retain)NSString *maxSPLText;
@property(nonatomic,retain)NSString *maxSPLValue;
@property(nonatomic,retain)NSString *maxSPLLine;
```

```
@property(nonatomic,retain)NSString *minSPLText;
@property(nonatomic,retain)NSString *minSPLValue;
@property(nonatomic,retain)NSString *minSPLLine;

@property(nonatomic,retain)NSDateFormatter *dateFormatter;
@property(nonatomic,retain)NSString *dateInStringFormated;



@property(nonatomic,retain)NSString *filterTypeValue;
@property(nonatomic,retain)NSString *operationModeValue;

@property(nonatomic,retain)NSString *sampleRateText;
@property(nonatomic,retain)NSString *sampleRateValue;
@property(nonatomic,retain)NSString *sampleRateLine;

@property(nonatomic,retain)NSString *blockSizeText;
@property(nonatomic,retain)NSString *blockSizeValue;
@property(nonatomic,retain)NSString *blockSizeLine;

@property(nonatomic,retain)NSString *filterTypeText;



@property(nonatomic,retain)  NSMutableData *squaredFilterResponce;
@property(nonatomic,retain)  NSMutableData *squaredFFTData;
@property(nonatomic,retain)  NSMutableData *splMeasurements;
@property(nonatomic,retain)  NSMutableData *avgSplMeasurements;
@property(nonatomic,retain)  NSMutableData *minSplMeasurements;
@property(nonatomic,retain)  NSMutableData *maxSplMeasurements;
@property(nonatomic,retain)  NSMutableData *inframeinstantSPLMeasurements;
@property(nonatomic)        int32_t       N;
@property(nonatomic)        int32_t       inframeBlocks;
@property(nonatomic)        int32_t       currentBlock;
@property(nonatomic)        float         instantSPL;
@property(nonatomic)        float         currentSPL;
```

```
@property(nonatomic)        float       currentAverage;
@property(nonatomic)        float       avgSPL;
@property(nonatomic)        float       minSPL;
@property(nonatomic)        float       maxSPL;
@property(nonatomic)        float       Fs;
@property(nonatomic)        float       calibrationConstant;
@property(nonatomic)        int64_t      measurementsNumber;
@property(nonatomic)        float       MeanEnergy;
@property(nonatomic)        int         OperationMode;
@property(nonatomic)        bool         ShouldUpdate;
@property(nonatomic)        bool         FirstFramePassed;
@property(nonatomic,retain) NSMutableArray *displayMeasurements;
@property(nonatomic) float *inwindowMeasurements;
@property(nonatomic,retain) NSDate *currentDateTime;
@property(nonatomic)       int FilteringType;

-(id)initWithSquaredFilterResponce:(NSMutableData *) newSquaredFilterResponce
            withSquaredFFTData:(NSMutableData *) newsquaredFFTData
                ofBufferSize:(int32_t) newBufferSize
            ofSamplingFrequency:(float) samplingFrequency
          withCalibrationConstant:(float) newCalibrationConstant;


-(id)initWithBufferSize:(int32_t)newBufferSize
   ofSamplingFrequency:(float) samplingFrequency
withCalibrationConstant:(float) newCalibrationConstant
     withOperationMode:(int) newOperationMode
       withFilterType:(int)newFilteringType;

-(void)setSquaredFilteredResponse:(NSMutableData *) newSquaredFilterResponse
            withSquaredFFTData:(NSMutableData *) newSquaredFFTData;

-(void)computeinstantSPL;
-(void)computeAverageSPL;
-(void) computeSPLMeasurements;
```

```
-(void) updateInframeInstantSPLMeasurements;
-(void)saveMeasurements;
- (NSString *)dataFilePath;
-(void)saveMeasurementsToFile;
-(void)printInWindowMeasurements;
-(float)average:(float *)values valuesNumber:(int)num;
-(void)ResetWithBufferSize:(int32_t)BufferSize
ofSamplingFrequency:(float) samplingFrequency
withCalibrationConstant:(float) newCalibrationConstant
withOperationMode:(int) newOperationMode
withFilterType:(int)newFilteringType;
```

```
@end
```

SPL.m

```
//
//  SPL.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "SPL.h"



@implementation SPL

@synthesize squaredFilterResponce;
@synthesize squaredFFTData;
@synthesize splMeasurements;
@synthesize N;
@synthesize inframeBlocks;
@synthesize currentBlock;
@synthesize instantSPL;
@synthesize currentSPL;
@synthesize currentAverage;
@synthesize avgSPL;
@synthesize maxSPL;
@synthesize minSPL;
@synthesize Fs;
@synthesize calibrationConstant;
@synthesize avgSplMeasurements;
@synthesize minSplMeasurements;
@synthesize maxSplMeasurements;
@synthesize measurementsNumber;
@synthesize inframeinstantSPLMeasurements;
```

```
@synthesize MeanEnergy;
@synthesize OperationMode;
@synthesize ShouldUpdate;
@synthesize FirstFramePassed;
@synthesize displayMeasurements;
@synthesize inwindowMeasurements;
@synthesize currentDateTime;
@synthesize FilteringType;
@synthesize documentsDirectory;
@synthesize currentSPLText;
@synthesize currentSPLValue;
@synthesize currentSPLLine;
@synthesize avgSPLText;
@synthesize avgSPLValue;
@synthesize avgSPLLine;
@synthesize maxSPLText;
@synthesize maxSPLValue;
@synthesize maxSPLLine;
@synthesize minSPLText;
@synthesize minSPLValue;
@synthesize minSPLLine;
@synthesize dateFormatter;
@synthesize dateInStringFormated;
@synthesize filterTypeValue;
@synthesize operationModeValue;
@synthesize sampleRateText;
@synthesize sampleRateValue;
@synthesize sampleRateLine;
@synthesize blockSizeText;
@synthesize blockSizeValue;
@synthesize blockSizeLine;
@synthesize filterTypeText;
@synthesize filterTypeLine;
@synthesize operationModeText;
@synthesize operationModeLine;
```

```
-(id)initWithSquaredFilterResponce:(NSMutableData *) newSquaredFilterResponce
              withSquaredFFTData:(NSMutableData *) newsquaredFFTData
                  ofBufferSize:(int32_t) BufferSize
             ofSamplingFrequency:(float) samplingFrequency
          withCalibrationConstant:(float) newCalibrationConstant
{
   if(self=[super init]){
      splMeasurements = [[NSMutableData alloc] init];
      avgSplMeasurements = [[NSMutableData alloc] init];
      minSplMeasurements = [[NSMutableData alloc] init];
      maxSplMeasurements = [[NSMutableData alloc] init];
      squaredFilterResponce = newsquaredFFTData;
      squaredFFTData = newsquaredFFTData;
      N = BufferSize;
      Fs = samplingFrequency;
      calibrationConstant = newCalibrationConstant;
      measurementsNumber = 0;

   }
   return self;
}


-(id)initWithBufferSize:(int32_t)BufferSize
   ofSamplingFrequency:(float) samplingFrequency
withCalibrationConstant:(float) newCalibrationConstant
     withOperationMode:(int) newOperationMode
       withFilterType:(int)newFilteringType
{
```

```
    if(self = [super init]){
        self.splMeasurements = [[NSMutableData alloc] init];
        self.avgSplMeasurements = [[NSMutableData alloc] init];
        self.minSplMeasurements = [[NSMutableData alloc] init];
        self.maxSplMeasurements = [[NSMutableData alloc] init];
        self.N = BufferSize;
        self.Fs = samplingFrequency;
        self.calibrationConstant = newCalibrationConstant;
        self.OperationMode = newOperationMode;
        self.measurementsNumber = 0;
        self.currentSPL = 0;
        self.currentAverage = 0;
        self.inframeBlocks = self.Fs / self.N;
        self.currentBlock = 0;
        self.inframeinstantSPLMeasurements = [[NSMutableData alloc] init];
        self.ShouldUpdate = NO;
        self.FirstFramePassed = NO;
        self.displayMeasurements = [[NSMutableArray alloc] init];
        self.inwindowMeasurements = (float *)malloc(self.inframeBlocks*sizeof(float));
        self.FilteringType = newFilteringType;
        [self InitializeDisplayMeasurements];

    }
    return  self;

}


-(void)ResetWithBufferSize:(int32_t)BufferSize
       ofSamplingFrequency:(float) samplingFrequency
  withCalibrationConstant:(float) newCalibrationConstant
        withOperationMode:(int) newOperationMode
          withFilterType:(int)newFilteringType{
```

```
// Set to nil before reallocating.

self.splMeasurements = nil;
self.avgSplMeasurements = nil;
self.minSplMeasurements = nil;
self.maxSplMeasurements = nil;



self.splMeasurements = [[NSMutableData alloc] init];
self.avgSplMeasurements = [[NSMutableData alloc] init];
self.minSplMeasurements = [[NSMutableData alloc] init];
self.maxSplMeasurements = [[NSMutableData alloc] init];

self.N = BufferSize;
self.Fs = samplingFrequency;
self.calibrationConstant = newCalibrationConstant;
self.OperationMode = newOperationMode;
self.measurementsNumber = 0;
self.currentSPL = 0;
self.currentAverage = 0;
self.inframeBlocks = self.Fs / self.N;
self.currentBlock = 0;
self.inframeinstantSPLMeasurements = nil;
self.inframeinstantSPLMeasurements = [[NSMutableData alloc] init];

self.ShouldUpdate = NO;
self.FirstFramePassed = NO;

self.displayMeasurements = nil;
self.displayMeasurements = [[NSMutableArray alloc] init];

free(self.inwindowMeasurements);
self.inwindowMeasurements = (float *)malloc(self.inframeBlocks*sizeof(float));
```

```
    self.FilteringType = newFilteringType;
    [self InitializeDisplayMeasurements];



}



-(void)setSquaredFilteredResponse:(NSMutableData *) newSquaredFilterResponse
            withSquaredFFTData:(NSMutableData *) newSquaredFFTData{
    self.squaredFilterResponce = newSquaredFilterResponse;
    self.squaredFFTData = newSquaredFFTData;
}

-(void)computeinstantSPL{

    float meanEnergy;
    float * _squaredFilterResponce = (float *)[self.squaredFilterResponce mutableBytes];
    float * _squaredFFTData = (float *)[self.squaredFFTData mutableBytes];
    float totalEnergy = cblas_sdot(N, _squaredFilterResponce, 1, _squaredFFTData, 1);
    if(totalEnergy == 0){
        meanEnergy = 0.0000000001;
        NSLog(@"Total Energy found to be 0");
    }
    else{
        totalEnergy /= N;
        if(totalEnergy == 0){
            NSLog(@"Mean Energy found to be 0");
            meanEnergy = 0.0000000001;
        }
        else{
            meanEnergy = totalEnergy / ((1/Fs)*N);
        }
    }
    self.MeanEnergy = meanEnergy;
    self.instantSPL = 10*log10f(meanEnergy)+calibrationConstant;
```

```
    }


    -(void)updateInframeInstantSPLMeasurements{

        //Computation of the current spl value.
        //Current spl is computed as the moving average of the past inframe_block spl values.
        //Therefore, at least inframe_block spl values should be obtained.
        float *_currentSPL = (float *)malloc(sizeof(float));

        if(!self.FirstFramePassed){
            //Save instant spl measurements in inwindowMeasurements array.
            //Current spl value cannot be computed yet.
            self.inwindowMeasurements[currentBlock] = self.instantSPL;
            self.currentBlock++;


            self.currentAverage = ((self.currentBlock - 1)*self.currentAverage +
self.instantSPL) / self.currentBlock;
            @autoreleasepool {
                currentDateTime = [NSDate date];
            }
            self.currentSPL = self.currentAverage;

            // Save currentSPL measurement in splMeasurements array.
            * _currentSPL = currentSPL;
            [self.splMeasurements appendBytes:_currentSPL length:sizeof(float)];
            measurementsNumber++;




            self.ShouldUpdate = YES;
          }
```

```
        else {
          // Get current date time
          @autoreleasepool {
             currentDateTime = [NSDate date];
          }
          self.currentSPL = [self average:self.inwindowMeasurements
valuesNumber:inframeBlocks];
          // Save currentSPL measurement in splMeasurements array.
          * _currentSPL = self.currentSPL;
          [self.splMeasurements appendBytes:_currentSPL length:sizeof(float)];
          measurementsNumber++;

          self.inwindowMeasurements[self.currentBlock] = self.instantSPL;
          self.currentBlock++;

       }
       if(self.currentBlock == self.inframeBlocks){
          self.FirstFramePassed = YES;
          self.currentBlock = 0;
          self.ShouldUpdate = YES;
       }
       //free(ones);
       free(_currentSPL);
    }



    -(void)computeAverageSPL{
       if(ShouldUpdate){
          float * _splMeasurements = (float *)[self.splMeasurements mutableBytes];
          self.avgSPL = cblas_sasum(measurementsNumber, _splMeasurements, 1);
          self.avgSPL = avgSPL / measurementsNumber;
          float * _avgSPL = (float *) malloc(sizeof(float));
          * _avgSPL = avgSPL;
          [self.avgSplMeasurements appendBytes:_avgSPL length:sizeof(float)];
```

```
        free(_avgSPL);
    }
}

-(void)computeMaxSPL{
    if(ShouldUpdate){
        float * _splMeasurements = (float *)[self.splMeasurements mutableBytes];
        int maxSPLPosition = cblas_isamax(measurementsNumber, _splMeasurements, 1);
        self.maxSPL = _splMeasurements[maxSPLPosition];
        float * _maxSPL = (float *) malloc(sizeof(float));
        * _maxSPL = maxSPL;
        [self.maxSplMeasurements appendBytes:_maxSPL length:sizeof(float)];
        free(_maxSPL);
    }
}

-(void)computeMinSPL{
    if(ShouldUpdate){
        float * _splMeasurements = (float *)[self.splMeasurements mutableBytes];
        float * _minSPL = (float *) malloc(sizeof(float));
        vDSP_minv(_splMeasurements,1,&minSPL,measurementsNumber);
        * _minSPL = minSPL;
        [self.minSplMeasurements appendBytes:_minSPL length:sizeof(float)];
        free(_minSPL);
    }
}

-(void)saveMeasurements{
    if(ShouldUpdate){

        currentSPLText  = @"Current SPL: ";
        currentSPLValue =  [NSString stringWithFormat:@"%.1f",self.currentSPL];
        currentSPLLine = [currentSPLText stringByAppendingString:currentSPLValue];

        avgSPLText = @"Average SPL: ";
```

```
        avgSPLValue = [NSString stringWithFormat:@"%.1f",self.avgSPL];
        avgSPLLine = [avgSPLText stringByAppendingString:avgSPLValue];

        maxSPLText = @"Max SPL: ";
        maxSPLValue = [NSString stringWithFormat:@"%.1f",self.maxSPL];
        maxSPLLine = [maxSPLText stringByAppendingString:maxSPLValue];

        minSPLText = @"Min SPL: ";
        minSPLValue = [NSString stringWithFormat:@"%.1f",self.minSPL];
        minSPLLine = [minSPLText stringByAppendingString:minSPLValue];

        dateFormatter = [[NSDateFormatter alloc] init];
        [dateFormatter setDateFormat:@"yyyy-MM-dd HH:mm:ss.SSS"];
        dateInStringFormated = [dateFormatter stringFromDate:currentDateTime];


        [self.displayMeasurements addObject:dateInStringFormated];
        [self.displayMeasurements addObject:currentSPLLine];
        [self.displayMeasurements addObject:avgSPLLine];
        [self.displayMeasurements addObject:maxSPLLine];
        [self.displayMeasurements addObject:minSPLLine];
    }
}



-(void)InitializeDisplayMeasurements{



    sampleRateText = @"Sampling Rate: ";
    sampleRateValue = [NSString stringWithFormat:@"%.1f",self.Fs];
    sampleRateLine = [sampleRateText stringByAppendingString:sampleRateValue];

    blockSizeText = @"Block Size: ";
```

```
blockSizeValue = [NSString stringWithFormat:@"%d",self.N];
blockSizeLine = [blockSizeText stringByAppendingString:blockSizeValue];

filterTypeText = @"Filtering Type: ";
switch (self.FilteringType) {
    case 0:
        filterTypeValue = @"Lin";
        break;
    case 1:
        filterTypeValue = @"A";
        break;
    case 2:
        filterTypeValue = @"B";
        break;
    case 3:
        filterTypeValue = @"C";
        break;
    default:
        break;
}

filterTypeLine = [filterTypeText stringByAppendingString:filterTypeValue];


operationModeText = @"Operation Mode: ";
switch (self.OperationMode) {
    case 0:
        operationModeValue = @"FAST";
        break;
    case 1:
        operationModeValue = @"SLOW";
        break;
    default:
        break;
}
```

```
        operationModeLine = [operationModeText
stringByAppendingString:operationModeValue];

        [self.displayMeasurements addObject:sampleRateLine];
        [self.displayMeasurements addObject:blockSizeLine];
        [self.displayMeasurements addObject:filterTypeLine];
        [self.displayMeasurements addObject:operationModeLine];

    }



    - (NSString *)dataFilePath {
        NSArray *paths = NSSearchPathForDirectoriesInDomains(
                                        NSDocumentDirectory, NSUserDomainMask, YES);
        documentsDirectory = [paths objectAtIndex:0];
        return [documentsDirectory stringByAppendingPathComponent:kFilename];
    }

    -(void)saveMeasurementsToFile{
        NSLog(@"Display MeasurementsFile: %@",[self dataFilePath]);
        [self.displayMeasurements writeToFile:[self dataFilePath] atomically:YES];
    }



    -(void) computeSPLMeasurements{
        [self computeinstantSPL];
        [self updateInframeInstantSPLMeasurements];
        //NSLog(@"CURRENT SPL: %f",currentSPL);
        [self computeAverageSPL];
        //NSLog(@"AVG SPL: %f",avgSPL);
        [self computeMaxSPL];
        //NSLog(@"MAX SPL: %f",maxSPL);
        [self computeMinSPL];
        //NSLog(@"MIN SPL: %f",minSPL);
```

```
    [self saveMeasurements];

}

-(void)printInWindowMeasurements{
    for(int i=0;i<self.inframeBlocks;i++){
        printf("SPL_BLOCK[%d] = %f\n",i+1,self.inwindowMeasurements[i]);
    }
}



-(float)average:(float *)values valuesNumber:(int)num{
    float sum = 0;
    float avg;
    for(int i=0;i<num;i++)
        sum += values[i];
    avg = sum / num;
    return avg;
}

@end
```

GraphVisualizer.h

```
//
//  GraphVisualizer.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface GraphVisualizer : UIView
{
```

```
    NSMutableArray *powers; // past power levels in the recording
    float minPower; // the lowest recorded power level
    float offsetX;
    float offsetY;
    float newPower;
    float nextPower;
} // end instance variable declaration



@property (nonatomic) float newPower;
@property (nonatomic) float nextPower;

- (void)setPower:(float)p; // set the powerLevel
- (void)clear; // clear all the past power levels
@end
```

GraphVisualizer.mm

```
//
//  GraphVisualizer.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//



#import "GraphVisualizer.h"

@implementation GraphVisualizer

@synthesize newPower;
@synthesize nextPower;



- (id)initWithFrame:(CGRect)frame
```

```
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}



/*
 // Only override drawRect: if you perform custom drawing.
 // An empty implementation adversely affects performance during animation.
 - (void)drawRect:(CGRect)rect
 {
 // Drawing code
 }
 */



// initialize the Visualizer
- (id)initWithCoder:(NSCoder *)aDecoder
{
    // if the superclass initializes properly
    if (self = [super initWithCoder:aDecoder])
    {
        // initialize powers with an entry for every other pixel of width
        offsetX = 15.0;
        offsetY = 0.0;
        powers = [[NSMutableArray alloc]
                  initWithCapacity:(self.frame.size.width / 1)-offsetX];

    //          UIColor *background = [[UIColor alloc] initWithPatternImage:[UIImage
imageNamed:@"BlackboardPlegma.png"]];
    //    self.backgroundColor = background;
```

```
    } // end if

    return self; // return this BarVisualizer
} // end method initWithCoder:

// sets the current power in the recording
- (void)setPower:(float)p
{
    @autoreleasepool {

        [powers addObject:[NSNumber numberWithFloat:p]]; // add value to powers

        // while there are enough entries to fill the entire screen
        while (((int)(powers.count * 1))+offsetX > self.frame.size.width)
            [powers removeObjectAtIndex:0]; // remove the oldest entry

        // if the new power is less than the smallest power recorded
        if (p < minPower)
            minPower = p; // update minPower with the new power
    }
} // end method setPower:

// clears all the points from the visualizer
- (void)clear
{
    //[powers removeAllObjects]; // remove all objects from powers

    powers = nil;
    powers = [[NSMutableArray alloc]
```

```
            initWithCapacity:(self.frame.size.width / 1)-offsetX];

} // end method clear

// draws the visualizer
- (void)drawRect:(CGRect)rect
{
   // get the current graphics context
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGSize size = self.frame.size;

      // draw a line for each point in powers

         for (int i = 0; i < (int)(powers.count-1); i++)

         {

            if(powers.count>=2){
            // get next power level
            newPower = [[powers objectAtIndex:i] floatValue];
            nextPower = [[powers objectAtIndex:i+1] floatValue];

            float height = (newPower/140) * 200;
            float nextHeight = (nextPower/140) * 200;

            float x1=(i*1)+offsetX;
            float y1=(size.height - offsetY-height);
            float x2 = ((i+1)*1)+offsetX;
            float y2=(size.height - offsetY- nextHeight);
            // move to a point above the middle of the screen
            //  CGContextMoveToPoint(context, (i * 2)+offsetX, size.height - offsetY);
            CGContextMoveToPoint(context, x1 , y1);

            // add a line to a point below the middle of the screen
            //    CGContextAddLineToPoint(context, (i * 2)+offsetX, size.height - offsetY -
```

```
height);
                CGContextAddLineToPoint(context, x2, y2);

                // set the color for this line segment based on f
                CGContextSetRGBStrokeColor(context, 1, 1, 1, 1);
                CGContextStrokePath(context); // draw the line
                // CGContextDrawPath(context,kCGPathStroke);









                }//end if

            } // end for

    } // end method drawRect:


@end
```

GraphBar.h

```
    //
    //  GraphBar.h
    //  iSPL PRO
    //
    //  Created by Σαρρής Αντώνιος on 11/4/12.
    //  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
    //

    #import <UIKit/UIKit.h>

    @interface GraphBar : UIView {
```

```
    NSMutableArray *powers; // past power levels in the recording
    float minPower; // the lowest recorded power level
    float offsetX;
    float offsetY;
    float newPower;
} // end instance variable declaration


@property(nonatomic) float newPower;

- (void)setPower:(float)p; // set the powerLevel
- (void)clear; // clear all the past power levels



@end
```

GraphBar.m

```
//
//  GraphBar.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "GraphBar.h"

@implementation GraphBar

@synthesize newPower;

- (id)initWithFrame:(CGRect)frame
{
```

```
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}

/*
 // Only override drawRect: if you perform custom drawing.
 // An empty implementation adversely affects performance during animation.
 - (void)drawRect:(CGRect)rect
 {
 // Drawing code
 }
 */

// initialize the Visualizer
- (id)initWithCoder:(NSCoder *)aDecoder
{
    // if the superclass initializes properly
    if (self = [super initWithCoder:aDecoder])
    {
        // initialize powers with an entry for every other pixel of width
        offsetX = 15.0;
        offsetY = 0.0;
        powers = [[NSMutableArray alloc]
                  initWithCapacity:(self.frame.size.width / 2)-offsetX];

    ///         UIColor *background = [[UIColor alloc] initWithPatternImage:[UIImage imageNamed:@"BlackboardPlegma.png"]];
    //    self.backgroundColor = background;

        NSLog(@"HEIGHT = %f",self.frame.size.height);
```

```
    } // end if

    return self; // return this BarVisualizer
} // end method initWithCoder:

// sets the current power in the recording
- (void)setPower:(float)p
{
    @autoreleasepool {

        [powers addObject:[NSNumber numberWithFloat:p]]; // add value to powers



        // while there are enough entries to fill the entire screen
        while (((int)(powers.count * 1))+offsetX > self.frame.size.width)
            [powers removeObjectAtIndex:0]; // remove the oldest entry

        // if the new power is less than the smallest power recorded
        if (p < minPower)
            minPower = p; // update minPower with the new power
    }
} // end method setPower:

// clears all the points from the visualizer
- (void)clear
{
    //[powers removeAllObjects]; // remove all objects from powers
    powers = nil;
    powers = [[NSMutableArray alloc]
              initWithCapacity:(self.frame.size.width / 1)-offsetX];
} // end method clear

// draws the visualizer
```

```
- (void)drawRect:(CGRect)rect
{
    // get the current graphics context
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGSize size = self.frame.size;

    // draw a line for each point in powers
    @autoreleasepool {


        for (int i = 0; i < (int)(powers.count); i++)
        {
            // get next power level

            newPower = [[powers objectAtIndex:i] floatValue];

            // calculate the height for this power level
            float height  = (newPower/140)*200;

            // move to a point above the middle of the screen
            CGContextMoveToPoint(context, (i * 1)+offsetX, size.height - offsetY);


            // add a line to a point below the middle of the screen
            CGContextAddLineToPoint(context, (i * 1)+offsetX, size.height - offsetY -
height);


            // set the color for this line segment based on f
            CGContextSetRGBStrokeColor(context, 0, 1, 0, 1);
            CGContextStrokePath(context); // draw the line



        } // end for
```

```
        }

    } // end method drawRect:
@end
```

GraphPulse.h

```
//
//  GraphPulse.h
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//



#import <UIKit/UIKit.h>

@interface GraphPulse : UIView {



    NSMutableArray *powers; // past power levels in the recording
    float minPower; // the lowest recorded power level
    float offsetX;
    float offsetY;
    float minLevel;
    float newPower;
} // end instance variable declaration

@property(nonatomic) float newPower;

- (void)setPower:(float)p withMin:(float)min; // set the powerLevel
- (void)clear; // clear all the past power levels
```

```
@end
```

GraphPulse.m

```
//
//  GraphPulse.m
//  iSPL PRO
//
//  Created by Σαρρής Αντώνιος on 11/4/12.
//  Copyright (c) 2012 sarris.antonis@gmail.com. All rights reserved.
//

#import "GraphPulse.h"

@implementation GraphPulse
@synthesize newPower;
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code
    }
    return self;
}

/*
 // Only override drawRect: if you perform custom drawing.
 // An empty implementation adversely affects performance during animation.
 - (void)drawRect:(CGRect)rect
 {
 // Drawing code
 }
 */

// initialize the Visualizer
```

```
- (id)initWithCoder:(NSCoder *)aDecoder
{
    // if the superclass initializes properly
    if (self = [super initWithCoder:aDecoder])
    {
        // initialize powers with an entry for every other pixel of width
        offsetX = 15.0;
        offsetY = 0.0;
        powers = [[NSMutableArray alloc]
                  initWithCapacity:(self.frame.size.width / 1)-offsetX];

    //          UIColor *background = [[UIColor alloc] initWithPatternImage:[UIImage imageNamed:@"BlackboardPlegma.png"]];
    //    self.backgroundColor = background;









    } // end if

    return self; // return this BarVisualizer
} // end method initWithCoder:

// sets the current power in the recording
- (void)setPower:(float)p withMin:(float) min
{
    @autoreleasepool {

        minLevel = min;
        [powers addObject:[NSNumber numberWithFloat:p]]; // add value to powers

        // while there are enough entries to fill the entire screen
        while (((int)(powers.count * 1))+offsetX > self.frame.size.width)
```

```
            [powers removeObjectAtIndex:0]; // remove the oldest entry

        // if the new power is less than the smallest power recorded
        if (p < minPower)
            minPower = p; // update minPower with the new power
    }
} // end method setPower:

// clears all the points from the visualizer
- (void)clear
{
    //[powers removeAllObjects]; // remove all objects from powers
    powers = nil;
    powers = [[NSMutableArray alloc]
              initWithCapacity:(self.frame.size.width / 1)-offsetX];
} // end method clear

// draws the visualizer
- (void)drawRect:(CGRect)rect
{
    // get the current graphics context
    CGContextRef context = UIGraphicsGetCurrentContext();
    CGSize size = self.frame.size;

    // draw a line for each point in powers
    @autoreleasepool {
        for (int i = 0; i < (int)(powers.count); i++)
        {
            // get next power level
            newPower = [[powers objectAtIndex:i] floatValue];



            // calculate the height for this power level
            float height  = ((newPower-minLevel)/140)*200;
```

```
            // move to a point above the middle of the screen
            CGContextMoveToPoint(context, (i * 1)+offsetX, size.height/2  - height/2);



            // add a line to a point below the middle of the screen
            CGContextAddLineToPoint(context, (i * 1)+offsetX, size.height/2 + height/2);



            // set the color for this line segment based on f
            CGContextSetRGBStrokeColor(context, 0, 1, 1, 1);
            CGContextStrokePath(context); // draw the line

        } // end for
    }

} // end method drawRect:


@end
```


ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
HELLENIC OPEN UNIVERSITY
SCHOOL OF SCIENCES AND TECHNOLOGY
WWW.EAP.GR