ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Ανάπτυξη Ασύρματου Συστήματος Αισθητήρων για Εφαρμογές Προγνωστικής Συντήρησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρήστος Π. Κουτσουραδής

Επιβλέπων : Παύλος-Πέτρος Σωτηριάδης

Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2021

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Ανάπτυξη Ασύρματου Συστήματος Αισθητήρων για Εφαρμογές Προγνωστικής Συντήρησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χρήστος Π. Κουτσουραδής

Επιβλέπων : Παύλος-Πέτρος Σωτηριάδης

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Μαΐου 2021.

.....	.....	.....
Παύλος-Πέτρος Σωτηριάδης Καθηγητής Ε.Μ.Π.	Συμεών Παπαβασιλείου Καθηγητής Ε.Μ.Π.	Νικόλαος Παπασπύρου Καθηγητής Ε.Μ.Π.

Αθήνα, Μάιος 2021

.....

Χρήστος Π. Κουτσουραδής

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών
Ε.Μ.Π.

Περίληψη

Στο πλαίσιο της παρούσας εργασίας σχεδιάστηκε και υλοποιήθηκε από πάνω προς τα κάτω, ξεκινώντας δηλαδή από την προδιαγραφή απαιτήσεων, μια υπηρεσία για χρήση σε εφαρμογές Condition Monitoring, με τελικό στόχο τη φυσική υλοποίηση ενός πραγματικού ασύρματου συστήματος για τη μέτρηση κρίσιμων παραμέτρων για την ορθή λειτουργία ενός συστήματος. Η υπηρεσία αποτελείται από δύο βασικά μέρη: ένα σύνολο coin-sized αισθητήρων με την αρμοδιότητα της συλλογής και της αποστολής των δεδομένων, και μία εφαρμογή πελάτη για κινητές συσκευές που είναι σε θέση να καταγράφει, να συγκεντρώνει και να αξιολογεί τα δεδομένα αυτά.

Η ασύρματη επικοινωνία μεταξύ των συσκευών γίνεται με χρήση του Bluetooth Low Energy (BLE), ενός στιβαρού και αξιόπιστου πρωτοκόλλου WPAN με εξαιρετικά χαμηλή κατανάλωση ενέργειας. Το BLE προτιμήθηκε για την ανοιχτή του προσέγγιση στα ζητήματα υλοποίησης της υπηρεσίας, καθώς η προδιαγραφή του αφήνει μεγάλα περιθώρια πρωτοβουλίας στους σχεδιαστές και τους προγραμματιστές. Η εφαρμογή πελάτη, με ένα όσο το δυνατόν πιο φιλικό γραφικό περιβάλλον, εκτελείται σε Android, τη δημοφιλέστερη πλατφόρμα κινητών συσκευών.

Λέξεις-κλειδιά

Προγνωστική συντήρηση, Βιομηχανία 4.0, Εποπτεία κατάστασης, Ασύρματα Δίκτυα Αισθητήρων, Bluetooth Low Energy, BLE, Διαδίκτυο των Αντικειμένων, IoT, IIoT

Abstract

The purpose of this diploma thesis is the development of a complete wireless Condition Monitoring Service which can be used to monitor critical parameters for the proper operation of a system. The Service is developed using a top-down approach, starting from the system specification and finally resulting in the physical implementation. The Service consists of two fundamental components: a coin-sized sensor with wireless connectivity supplying the data, and a client application running on a mobile device.

For wireless connectivity, we choose Bluetooth Low Energy (BLE), an efficient, robust, low-power WPAN technology. BLE provides an open-minded and developer-oriented approach to the use of its specification for application development, entrusting the developer with wide aspects of the service implementation. For the client device, an Android application is developed, and great care is taken in order to provide a user-friendly GUI experience.

Keywords

Predictive Maintenance, Industry 4.0, Condition Monitoring, Wireless Sensor Networks, Bluetooth Low Energy, BLE, Wireless Sensor Network, Internet of Things, IoT, IIoT

Ευχαριστίες

Ευχαριστώ τον επιβλέποντα της εργασίας, Καθηγητή ΕΜΠ κ. Παύλο-Πέτρο Σωτηριάδη, για τη στήριξη και την καθοδήγηση που μου παρείχε, καθώς για την παρότρυνσή του να βελτιώνω συνεχώς το αποτέλεσμα της δουλειάς μου. Για την αμέριστη στήριξη και βοήθεια που μου προσέφερε σε οποιαδήποτε απορία τεχνικής ή θεωρητικής φύσεως, αλλά κυρίως για τη σχεδίαση και την κατασκευή των κυκλωμάτων αισθητήρων, θέλω να ευχαριστήσω τον Υποψήφιο Διδάκτορα στο Εργαστήριο Ηλεκτρονικής, Κωνσταντίνο Παπαφώτη. Επίσης, το συνάδελφο διπλωματούχο της Σχολής μας Γιώργο Παπαδημητρίου, που με κατατόπισε στα πρώτα βήματα εκπόνησης της εργασίας. Θέλω επιπλέον να ευχαριστήσω όλα τα μέλη του Εργαστηρίου για το φιλικό και οικογενειακό κλίμα, τόσο στη δουλειά, όσο και στον ελεύθερο χρόνο που περάσαμε μαζί.

Τίποτα όμως δεν αξίζει περισσότερο από την αγάπη και την εμψύχωση που λαμβάνει κανείς καθημερινά από την οικογένειά του, τους φίλους και συμφοιτητές του, καθ’ όλη τη διάρκεια των σπουδών. Ενός ταξιδιού τόσο ξεχωριστού και αναντικατάστατου στη ζωή.

Πίνακας περιεχομένων

1 Εισαγωγή - Condition Monitoring 13
1.1 Condition Monitoring 13
1.2 Κυρίαρχες προσεγγίσεις 14
1.2.1 Στρατηγικές εποπτείας 14
1.2.2 Δομή συστήματος εποπτείας 15
1.2.3 Εξόρυξη πληροφορίας 17
1.2.4 Πολιτικές συντήρησης 22
1.2.5 Επιλογή βέλτιστης πολιτικής 23
1.3 Προσέγγιση στην παρούσα εργασία 24
2 Έξυπνα Δίκτυα 25
2.1 Έξυπνα δίκτυα – το Internet of Things (IoT) 25
2.2 Ιστορική αναδρομή 28
2.3 Σύγχρονες εφαρμογές 29
2.4 Ασύρματα Δίκτυα Αισθητήρων 29
2.4.1 Τεχνολογία αισθητήρων 30
2.4.2 Τοπολογία-δρομολόγηση 31
2.4.3 Τροφοδοσία-κατανάλωση ενέργειας 33
2.4.4 Ασφάλεια 34
2.5 Πρωτόκολλα WPAN 35
2.6 Προσέγγιση στην παρούσα εργασία 37
3 Bluetooth Low Energy 39
3.1 Εισαγωγή 39
3.1.1 Ιστορικά στοιχεία: Bluetooth BR/EDR και LE 39
3.1.2 Βασικά χαρακτηριστικά 41
3.2 Φυσικό στρώμα 42
3.3 Επίπεδο ζεύξης δεδομένων 44
3.3.1 Διευθυνσιοδότηση 45
3.3.2 Ρόλοι και καταστάσεις 46

3.3.3 Εκπομπή (advertising) 47
3.3.4 Σύνδεση (connection) 51
3.4 Στοίβα πρωτοκόλλων-προφίλ 53
3.4.1 L2CAP 54
3.4.2 Generic Access Profile (GAP) 55
3.4.3 Attribute Protocol (ATT) 59
3.4.4 Security Manager Protocol (SMP) 61
3.4.5 Generic Attribute Profile (GATT) 64
4 Προδιαγραφή του συστήματος 70
4.1 Συστατικά μέρη 70
4.2 Προδιαγραφή παρεχόμενων λειτουργιών 70
4.3 Κατάσταση λειτουργίας συσκευής 71
4.3.1 Δυνατές καταστάσεις 71
4.3.2 Ενημέρωση και κοινοποίηση κατάστασης 72
4.4 Ορισμός Condition Monitoring Profile 75
4.4.1 Γενική δομή του κύριου Condition Monitoring Service 76
4.4.2 Environmental Sensor Service 76
4.4.3 Inertial Measurement Service 79
4.5 Λειτουργία του CM Service 82
4.5.1 Είσοδος δεδομένων - Control Point 82
4.5.2 Έξοδος δεδομένων 85
4.5.3 Καταγραφή ιστορικού 85
4.5.4 Πληροφορίες συσκευής – DIS 87
4.5.5 Ρυθμίσεις συσκευής – DCS 88
4.5.6 Ειδικά ζητήματα 91
5 Σύστημα αισθητήρων 92
5.1 Εισαγωγή 92
5.2 DA14583 Bluetooth Low Energy SoC 93
5.2.1 Αρχιτεκτονική 93
5.2.2 Προγραμματισμός και αποσφαλμάτωση 96

5.3 Αισθητήρες-περιφερειακά 97
5.3.1 Διεπαφές σειριακής επικοινωνίας 97
5.3.2 Θερμόμετρο/υγρόμετρο HDC1008 102
5.3.3 Επιταχυνσιόμετρο/γυροσκόπιο BMI160 103
5.3.4 DS1347 RTC 106
5.3.5 Μετατροπέας USB σε σειριακό FT230X 107
5.4 Υλοποίηση του CM Profile 107
5.4.1 Έξοδος δεδομένων 109
5.4.2 Είσοδος δεδομένων 113
5.4.3 Καταγραφή ιστορικού 114
5.4.4 Περιοδική δειγματοληψία 114
5.4.5 Εκπομπή 116
5.4.6 Ενημέρωση κατάστασης 116
5.5 Επίπεδο αφαίρεσης υλικού 118
5.5.1 Πρόγραμμα οδήγησης Environmental Sensor 118
5.5.2 Πρόγραμμα οδήγησης Inertial Measurement Unit 119
5.5.3 Πρόγραμμα οδήγησης RTC 120
5.5.4 Υλοποίηση DIS 121
5.5.5 Υλοποίηση DCS 121
5.6 Ειδικά ζητήματα 123
5.6.1 Διάρκεια ζωής 123
5.6.2 Διέλευση δεδομένων 128
6 Το λειτουργικό σύστημα Android 130
6.1 Εισαγωγή 130
6.1.1 Βασικά χαρακτηριστικά 130
6.1.2 Ιστορικά στοιχεία 131
6.1.3 Αρχιτεκτονική 133
6.1.4 Περιβάλλον εκτέλεσης 134
6.1.5 Περιβάλλον προγραμματισμού 136
6.2 Ανάπτυξη εφαρμογών για Android 137

6.2.1 Θεμελιώδη δομικά στοιχεία 137
6.2.2 Διαχείριση γραφικού περιβάλλοντος 139
6.2.3 Υπηρεσίες 144
6.2.4 Ασύγχρονη εκτέλεση εργασιών 146
6.2.5 Διαδιεργασιακή επικοινωνία 148
6.2.6 Μόνιμη αποθήκευση δεδομένων 149
7 Εφαρμογή πελάτη 152
7.1 Δομή εφαρμογής 152
7.1.1 Συστατικά μέρη 152
7.1.2 Πακέτα-κλάσεις 153
7.2 Γραφικό περιβάλλον 155
7.2.1 Οθόνες και πλοήγηση 156
7.2.2 Παραγωγή και προβολή διαγραμμάτων 168
7.3 Υλοποίηση λειτουργιών 174
7.3.1 Αναζήτηση για συσκευές 174
7.3.2 Διαχείριση συνδέσεων 179
7.3.3 Διαχείριση Services και Characteristics 184
7.3.4 Ρυθμίσεις συσκευής 188
7.3.5 Διαχείριση συνόδου 188
7.3.6 Κατάσταση συσκευής 193
7.3.7 Ιστορικό συσκευής 194
7.3.8 Προτιμήσεις εφαρμογής 202
8 Συμπεράσματα-προτάσεις 204
8.1 Επίλογος 204
8.2 Προτάσεις για επέκταση 204

1 Εισαγωγή - Condition Monitoring

1.1 Condition Monitoring

Η *εποπτεία κατάστασης* (*condition monitoring* ή *condition-based monitoring*, CM ή CbM) περιλαμβάνει το σύνολο των τεχνικών εκείνων που επιτρέπουν την εποπτεία και την παρακολούθηση της κατάστασης καλής λειτουργίας ενός συστήματος σε πραγματικό χρόνο, με σκοπό τον εντοπισμό, την πρόληψη, ή ακόμα και την πρόβλεψη συνθηκών που μπορούν να προκαλέσουν βλάβες.

Βασίζεται στη λήψη, καταγραφή και ανάλυση δεδομένων από αισθητήρες προσαρτημένους κατάλληλα σε εξαρτήματα και μέρη βιομηχανικού εξοπλισμού. Τέτοια δεδομένα μπορούν να είναι:

- Τιμές θερμοκρασίας, υγρασίας και πίεσης, μεγεθών που συνήθως αποτελούν κρίσιμο παράγοντα για την ορθή λειτουργία κάθε συσκευής ή μηχανήματος.

- Μετρήσεις μετατοπίσεων ή περιστροφών για τον εντοπισμό δονήσεων (mechanical vibrations/shocks) ή φθοράς. Πολύτιμη πληροφορία για την κατηγοριοποίηση (*profiling*) τέτοιων φαινομένων παρέχεται με ανάλυση των μετρήσεων αυτών στο πεδίο της συχνότητας (vibration analysis).

Τα δεδομένα αυτά, κατόπιν κατάλληλης ανάλυσης, όπως με χρήση στατιστικών μεθόδων ή μηχανικής μάθησης (*machine learning*), μπορούν να αξιοποιηθούν όχι μόνο για τον εντοπισμό και την πρόληψη, αλλά και για την πρόβλεψη πιθανών προβληματικών καταστάσεων στο μέλλον. Γι΄αυτόν το λόγο οι τεχνικές CM είναι κρίσιμο συστατικό στοιχείο των τεχνολογιών *προγνωστικής* (ή *προβλεπτικής*) *συντήρησης* (predictive maintenance, PdM).

Το CM διαδραματίζει πρωταγωνιστικό ρόλο στην πρόοδο της σύγχρονης βιομηχανίας και τη διαδικασία μετάβασής της στην εποχή του Industry 4.0. Το νέο αυτό στάδιο εξέλιξης χαρακτηρίζεται από τη διασυνδεσιμότητα τόσο μεταξύ μεμονωμένων μηχανημάτων όσο και εύρυτερα μεταξύ παραγωγικών μονάδων, καθώς και την αυτοματοποίηση των διαδικασιών ελέγχου και συντήρησης.

Ταυτόχρονα , η αλματώδης πρόοδος στους τομείς των ασύρματων δικτύων και των ενσωματωμένων συστημάτων καθιστά σήμερα εφικτό ένα νέο τύπο δικτύων, τα λεγόμενα «έξυπνα δίκτυα». Είναι τα δίκτυα που σχηματίζονται από «έξυπνες συσκευές», δηλαδή ενσωματωμένα συστήματα με δυνατότητα δικτύωσης. Οι συσκευές αυτές σταδιακά μπορούν να ενσωματώνονται σε ολοένα και περισσότερα αντικείμενα της καθημερινής ζωής, σε μηχανήματα και σε στοιχεία του φυσικού περιβάλλοντος. Ειδικότερα για το CM, αυτό συνεπάγεται τη

δυνατότητα πυκνότερης ενσωμάτωσης οικονομικών, ψηφιακών συστημάτων αισθητήρων, εξοπλισμένων με δυνατότητες ασύρματης δικτύωσης, ή ακόμα και συνδεσιμότητα στο Διαδίκτυο.

Το CM ως πεδίο εφαρμογών μπορεί μόνο να ωφεληθεί από την εξέλιξη αυτή. Η εποπτεία και συντήρηση συσκευών, μηχανημάτων και μονάδων παραγωγής στο πλαίσιο ενός «έξυπνου δικτύου» καθιστά την όλη διαδικασία συλλογής, ανάλυσης δεδομένων και λήψης αποφάσεων απλούστερη, φτηνότερη, πιο αξιόπιστη και φιλική στο περιβάλλον.

1.2 Κυρίαρχες προσεγγίσεις

Προτού προχωρήσει κανείς στη σχεδίαση και την υλοποίηση μίας υπηρεσίας CM, θα πρέπει λαμβάνοντας υπόψη τις απαιτήσεις της εφαρμογής να κάνει κάποιες σχεδιαστικές επιλογές σχετικά με την πολιτική συντήρησης που θα εξυπηρετεί, καθώς και τις μεθόδους με τις οποίες το σύστημα θα συλλέγει και θα επεξεργάζεται τα δεδομένα, θα προσδιορίζει με βάση αυτά την κατάσταση λειτουργίας του συστήματος προς έλεγχο, και θα παρέχει χρήσιμη πληροφορία στο χρήστη.

1.2.1 Στρατηγικές εποπτείας

Η διαδικασία εποπτείας του συστήματος ξεκινά με την επιλογή των απαιτούμενων δεδομένων, καθώς και του τρόπου και της συχνότητας συλλογής τους. Οι στρατηγικές που ακολουθούνται συνήθως εδώ είναι δύο:

- ***Συνεχής εποπτεία (continuous/time-based)***

Οι απαραίτητες παράμετροι δειγματοληπτούνται με σταθερό ρυθμό και καταγράφονται στη μνήμη του συστήματος συνοδευόμενες από μια χρονοσφραγίδα (timestamp), ή αποστέλλονται απευθείας σε κάποιον κεντρικό σταθμό παρακολούθησης. Τα σήματα που προκύπτουν μπορούν εύκολα να μελετηθούν στο πεδίο της συχνότητας, ενώ με την αύξηση του ρυθμού δειγματοληψίας βελτιώνεται η χρονική ανάλυση (time resolution), δίνοντας ακριβέστερη εικόνα για στιγμιαία φαινόμενα ή απότομες μεταβολές.

Το αντίτιμο για την αυξημένη ακρίβεια της εποπτείας με αυτή την προσέγγιση είναι οι πρόσθετες απαιτήσεις σε επεξεργαστική ισχύ, κατανάλωση ενέργειας, αποθηκευτικό χώρο και εύρος ζώνης δικτύου. Χωρίς σταθερή πηγή τροφοδοσίας, η διάρκεια ζωής των συστημάτων συνεχούς εποπτείας αναμένεται να είναι μικρή.

- ***Διακριτή εποπτεία, ή βασισμένη σε συμβάντα (discrete/event-based)***

Εδώ δεν καταγράφονται συνεχώς οι τιμές των παραμέτρων του συστήματος, αλλά μόνο τα συμβάντα κατά τα οποία αυτές υπερβαίνουν κάποια όρια καλής λειτουργίας. Οι αισθητήρες και το σύστημα καταγραφής κανονικά παραμένουν σε

κατάσταση εξοικονόμησης ενέργειας, και ενεργοποιούνται μόνο κατά την εκδήλωση κάποιου συμβάντος. Εφόσον οι συνθήκες καλής λειτουργίας είναι σαφώς ορισμένες, η προσέγγιση αυτή μας επιτρέπει να επικεντρωθούμε στη χρήσιμη πληροφορία και να αποφύγουμε τη συμφόρηση του συστήματος ειδοποίησης.
Η απουσία όμως πρόσθετης πληροφορίας δυσχεραίνει την ανάλυση του κάθε συμβάντος ξεχωριστά, τη μελέτη της χρονικής του εξέλιξης και την εξεύρεση των πιθανών αιτίων, κι αυτό γιατί ενός χρονικού διαστήματος δυσλειτουργίας προηγούνται, και πιθανότατα επίσης έπονται, μακρά διαστήματα φαινομενικά σωστής λειτουργίας.

Εφόσον δεν είναι επιθυμητή η μετάβαση σε συνεχή εποπτεία του συστήματος, ένας τρόπος αντιστάθμισης αυτής της απώλειας ακρίβειας είναι η επιβολή περισσότερων περιοριστικών συνθηκών σε κάποιες παραμέτρους. Για παράδειγμα, μπορούν να τεθούν όρια καλής λειτουργίας όχι μόνο στην τιμή της παραμέτρου, αλλά και στο χρονικό ρυθμό μεταβολής της. Έτσι καταγράφονται πρόσθετα συμβάντα πριν την εκδήλωση της δυσλειτουργίας, και είναι δυνατό να σχηματιστεί μια πιο αξιόπιστη εικόνα του προβλήματος.

Σε κάθε περίπτωση, η διακριτή εποπτεία ενδείκνυται για την παρακολούθηση φαινομένων που εμφανίζουν έναν υψηλό βαθμό τυχαιότητας και ανεξαρτησίας από το ιστορικό της κατάστασης του συστήματος, ή προκαλούνται συνήθως από εξωγενείς παράγοντες. Τέτοια φαινόμενα είναι οι δονήσεις, οι κραδασμοί, οι αλλαγές στη θερμοκρασία του περιβάλλοντος, οι διακοπές τροφοδοσίας κλπ.

1.2.2 Δομή συστήματος εποπτείας

Από τα παραπάνω συμπεραίνουμε πως κάθε σύστημα εποπτείας κατάλληλο για εφαρμογές CM αποτελείται απαραίτητα από τα εξής συστατικά μέρη:

- ***Σύστημα μέτρησης***

Αποτελείται από έναν αναλογικό αισθητήρα ή όργανο μέτρησης συνοδευόμενο από έναν μετατροπέα από αναλογικό σε ψηφιακό (Α/Ψ). Τα δύο αυτά στοιχεία μπορούν να είναι διακριτά και να συνδέονται ενσύρματα, ή να ενσωματώνονται στο ίδιο ολοκληρωμένο κύκλωμα.

- ***Σύστημα καταγραφής***

Πρόκειται για το υπολογιστικό σύστημα που επικοινωνεί απευθείας με το σύστημα μέτρησης, μεταφέροντας εντολές και δεδομένα. Από τις μετρήσεις των αισθητήρων εξάγονται οι τιμές των εποπτευόμενων παραμέτρων, και εφόσον είναι απαραίτητο ενσωματώνονται σε κατάλληλα δομημένες εγγραφές ιστορικού. Τα δεδομένα εξόδου αποθηκεύονται σε μόνιμη (πχ Flash, EEPROM) ή προσωρινή μνήμη, ανάλογα με τις απαιτήσεις της εφαρμογής.

Ο μικροεπεξεργαστής (μΕ) του συστήματος μέσω κατάλληλων εντολών επιτρέπει τον έλεγχο της διαδικασίας δειγματοληψίας και την τροποποίηση

διάφορων ρυθμίσεων των αισθητήρων, όπως του ρυθμού και της ακρίβειας δειγματοληψίας, του εύρους τιμών κλπ.

Η ανταλλαγή δεδομένων μεταξύ του μΕ και του συστήματος μέτρησης γίνεται πάνω από κάποιο πρωτόκολλο επικοινωνίας – κατά κανόνα σειριακής – συμβατό και με τις δύο συσκευές. Το πρωτόκολλο αυτό μπορεί να είναι γενικού σκοπού (πχ I^2C, SPI, UART) ή εξειδικευμένο και ορισμένο από τον κατασκευαστή.

- ***Σύστημα ειδοποίησης***

Είναι υπεύθυνο για τη συγκέντρωση των δεδομένων από όλες τις πηγές του συστήματος και την αξιοποίησή τους για την εξαγωγή χρήσιμης πληροφορίας και τη λήψη αποφάσεων. Περιλαμβάνει τόσο το απαραίτητο λογισμικό (ευφυΐα) για το σκοπό αυτό, όσο και την υποδομή του δικτύου διασύνδεσης των κόμβων και των σημείων συλλογής (ενσύρματου ή ασύρματου).

Η συλλογή των δεδομένων και η λήψη των αποφάσεων μπορούν να γίνονται αποκεντρωμένα στους κόμβους του δικτύου, ή συγκεντρωτικά. Στην πρώτη περίπτωση, τα σημεία συλλογής καταναμένονται στους κόμβους, οι κόμβοι επικοινωνούν ανεξάρτητα μεταξύ τους και οι αποφάσεις λαμβάνονται συνεργατικά. Διαφορετικά, οι κόμβοι συνδέονται αποκλειστικά με έναν κεντρικό σταθμό παρακολούθησης και ελέγχου, στον οποίο ενσωματώνεται το σύνολο της λογικής της εφαρμογής.

Είναι φυσικά δυνατό να υιοθετήσουμε και συνδυασμό των δύο προσεγγίσεων, προσαρμόζοντας την κατανομή των σημείων συλλογής και το επίπεδο ευφυΐας που ενσωματώνουμε σε καθένα από αυτά.

Σε βιομηχανικές εφαρμογές, μέχρι τις μέρες μας τα συστήματα CM ενσωματώνονται συνήθως σε συστήματα παρακολούθησης και ελέγχου συνολικά της παραγωγικής διαδικασίας, τα λεγόμενα συστήματα SCADA (*Supervisory Control and Data Acquisition*). Ένα σύστημα SCADA ακολουθεί μία ιεραρχική δομή της παρακάτω μορφής:

- *Επίπεδο 0:* Περιέχει το σύνολο του συστήματος μέτρησης (αισθητήρες), καθώς και ένα σύνολο στοιχείων ελέγχου (διακόπτες, βαλβίδες, PLCs κλπ).
- *Επίπεδο 1:* Περιέχει το σύνολο του συστήματος συλλογής (μΕ και μονάδες εισόδου/εξόδου).
- *Επίπεδο 2:* Ολοκληρώνει τη λειτουργικότητα του CM, προσθέτοντας το σύστημα ειδοποίησης. Αποτελείται από υπολογιστικά συστήματα επιφορτισμένα με την επίβλεψη ενός συνόλου από συσκευές και εξοπλισμένα με κατάλληλες διεπαφές χειρισμού από τους αρμόδιους μηχανικούς.
- *Επίπεδο 3:* Χρησιμοποιεί την πληροφορία του επιπέδου 2 για την επίβλεψη της παραγωγικής διαδικασίας σε έναν *τομέα* ευθύνης.

- *Επίπεδο 4:* Χρησιμοποιεί την πληροφορία όλων των τομέων του επιπέδου 3 για τον σχεδιασμό και τον προγραμματισμό της παραγωγής.

Στην πλειοψηφία των συστημάτων SCADA, η διασύνδεση των δομικών μονάδων γίνεται συνήθως ενσύρματα, πάνω από ιδιοταγή πρωτόκολλα ορισμένα από τον εκάστοτε κατασκευαστή. Ταυτόχρονα, η εξάπλωση των ασύρματων δικτύων αισθητήρων και η ευρεία διαθεσιμότητα στην αγορά χαμηλού κόστους ολοκληρωμένων λύσεων CM βασισμένων σε τέτοια δίκτυα, τείνει να διαχωρίσει τις λειτουργίες CM από τις λειτουργίες επιτήρησης της παραγωγής, παραμένει όμως ανοικτό ερώτημα κατά πόσο τα WSN μπορούν μελλοντικά στις τελευταίες να υποκαταστήσουν τα συστήματα SCADA.

1.2.3 Εξόρυξη πληροφορίας

Πολύ συχνά, τα δεδομένα που παράγονται κατά τη διαδικασία εποπτείας (χρονοσειρές, εγγραφές συμβάντων) από μόνα τους δεν αρκούν για την εξαγωγή συμπερασμάτων σχετικά με την κατάσταση του συστήματος και τη λήψη αποφάσεων για τον έλεγχο και τη συντήρησή του, απαιτείται λοιπόν να αξιολογούνται από τους αρμόδιους μηχανικούς - είτε εμπειρικά είτε κατόπιν κατάλληλης επεξεργασίας - βάσει κάποιων κριτηρίων, καθολικών ή εξειδικευμένων για το σύστημα.

Στις επόμενες παραγράφους αναφερόμαστε σε κάποιες γενικές αρχές για τα κριτήρια αυτά, καθώς και στις πιο διαδεδομένες μεθόδους επεξεργασίας δεδομένων και εξόρυξης πληροφορίας.

1.2.3.1 Κύριοι παράγοντες

Ερμηνεύοντας τις τιμές των εποπτευόμενων παραμέτρων ενός συστήματος, πρέπει κανείς να λαμβάνει υπόψη τα εξής:

α) Από τη σκοπιά των *πηγών δεδομένων*:

- Το *πλήθος* των πηγών: με τοποθέτηση περισσότερων συσκευών στο ίδιο σημείο μέτρησης, ή σε διαφορετικά σημεία κοντά στο σημείο που μας ενδιαφέρει, ο συνδυασμός (aggregation) των τιμών - π.χ. με στάθμιση (weighing), λήψη της μέσης τιμής (averaging) κλπ. - μπορεί να βελτιώσει την ακρίβεια της μέτρησης.

- Τη *σχετική θέση* των πηγών ως προς το σύστημα: η αύξηση της απόστασης της πηγής από το σύστημα, γενικά η απουσία απευθείας φυσικής σύνδεσης με αυτό, μπορεί να μειώνει την αξιοπιστία της μέτρησης, ταυτόχρονα όμως και να παρέχει χρήσιμη πληφορορία για τις συνθήκες του περιβάλλοντος.

- Την *τεχνολογία* των πηγών: αποτελεί τον κρίσιμο παράγοντα αξιοπιστίας στην περίπτωση που οι δύο παραπάνω παράγοντες παραμένουν αμετάβλητοι. Αναλυτικά εξετάζουμε αυτό το ζήτημα στην υποενότητα 2.4.1.

Θα δώσουμε ένα παράδειγμα αβεβαιότητας που υπεισέρχεται στη μέτρηση μιας παραμέτρου εξαιτίας της θέσης των πηγών. Ας θεωρήσουμε πως προσπαθούμε να εκτιμήσουμε την κατάσταση λειτουργίας ενός μηχανήματος. Υποθέτουμε πως η θερμοκρασία πυρήνα t_c σε ένα συγκεκριμένο σημείο C στο εσωτερικό του μηχανήματος είναι παράμετρος κρίσιμης σημασίας για τη λειτουργία του, αλλά ταυτόχρονα είναι αδύνατη η προσάρτηση αισθητήρων απευθείας στο C. Εφόσον δεν μπορούμε να χρησιμοποιήσουμε την t_c για να εκτιμήσουμε την κατάσταση λειτουργίας, τοποθετούμε έναν αισθητήρα σε κάποιο άλλο εσωτερικό σημείο Α, και έναν ακόμα σε κάποιο σημείο Β πάνω στο περίβλημα του μηχανήματος. Αν t_i (θερμοκρασία εσωτερικού) και t_s (επιφανειακή θερμοκρασία) είναι οι μετρήσεις της θερμοκρασίας στα Α και Β αντίστοιχα, κάθε συμπέρασμα που εξάγεται για την κατάσταση λειτουργίας της συσκευής εμπεριέχει ένα βαθμό αβεβαιότητας.

Γνωρίζοντας τα όρια ορθής λειτουργίας και για τις τρεις αυτές τιμές θερμοκρασίας, είναι επόμενο να προσδοκούμε υψηλότερη αξιοπιστία από την t_i παρά από την t_s, αφενός όμως ο βαθμός αξιοπιστίας εξαρτάται από τη συσχέτιση μεταξύ των t_i και t_c, αφετέρου η επιφανειακή θερμοκρασία t_s (και ευρύτερα η θερμοκρασία περιβάλλοντος) μπορεί να επιδρά στη μελλοντική εξέλιξη της t_c. Συνδυάζοντας λοιπόν τις τιμές των t_i και t_s καταλήγουμε στα συμπεράσματα του παρακάτω πίνακα, όπου με Ο συμβολίζουμε την ορθή λειτουργία και με Π την προβληματική λειτουργία:

t_i	t_s	**Εκτίμηση κατάστασης**
Ο	Ο	Μάλλον Ο
Ο	Π	Πιθανά Π
Π	Ο	Μάλλον Π
Π	Π	Π

Πίνακας 1.1: Πίνακας αληθείας για εκτίμηση κατάστασης

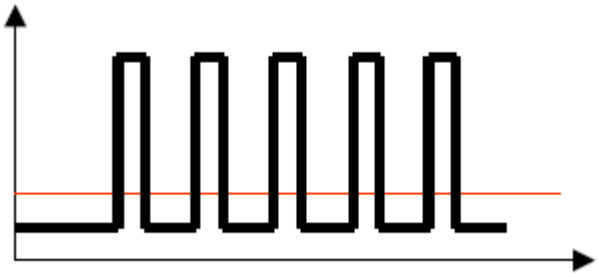
β) Από τη σκοπιά των *δεδομένων*:

- Τη *συχνότητα* των συμβάντων: πόσες φορές εντός ενός συγκεκριμένου χρονικού διαστήματος η παράμετρος παραβαίνει τα όρια καλής λειτουργίας, δηλαδή πόσα συμβάντα προκύπτουν στη μονάδα του χρόνου.

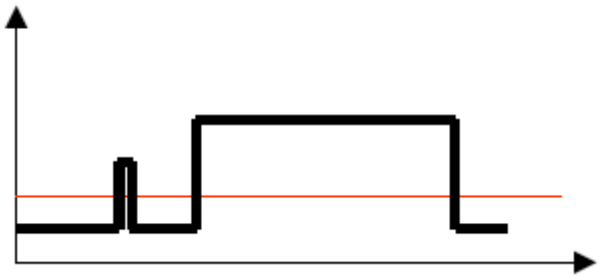
- Τη *χρονική διάρκεια* των συμβάντων: για πόσο χρόνο η παράμετρος παραβαίνει τα όρια κάθε φορά.

- Το *μέγεθος* της απόκλισης: πόσο αποκλίνει η τιμή της παραμέτρου από τα όρια κάθε στιγμή, και πώς εξελίσσεται αυτή η απόκλιση κατά τη διάρκεια ενός συμβάντος.

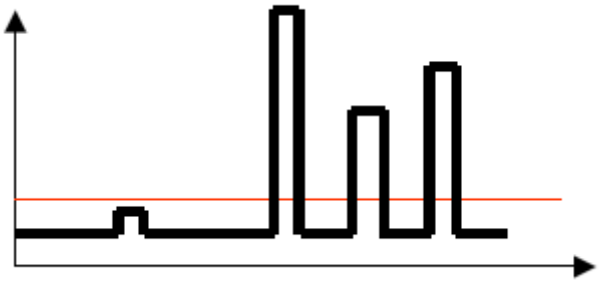
Οι παρακάτω τρεις γραφικές παραστάσεις παρουσιάζουν τρεις διαφορετικές περιπτώσεις απόκλισης μιας παραμέτρου X από ένα όριο. Με μαύρο χρώμα απεικονίζεται η χρονική εξέλιξη X(t) της παραμέτρου, ενώ με κόκκινο η τιμή του ορίου. Κάθε φορά απεικονίζεται η επίδραση ενός από τους παράγοντες που αναφέραμε παραπάνω, ενώ οι υπόλοιποι υποτίθενται σταθεροί:



Σχήμα 1.2: Συχνότητα συμβάντων



Σχήμα 1.3: Χρονική διάρκεια συμβάντων



Σχήμα 1.4: Μέγεθος απόκλισης

1.2.3.2 Ανάλυση στο πεδίο της συχνότητας

Η ανάλυση στο πεδίο της συχνότητας με χρήση του μετασχηματισμού Fourier μπορεί να μας δώσει περισσότερη πληροφορία σχετικά με τη μορφή των συμβάντων που περιγράψαμε στην προηγούμενη παράγραφο, εκφράζοντας μια συνάρτηση του χρόνου ως άπειρο άθροισμα συχνοτικών συνιστωσών, δηλαδή ημιτονικών συναρτήσεων στις αντίστοιχες συχνότητες. Για τον υπολογισμό του μετασχηματισμού Fourier ψηφιακών σημάτων χρησιμοποιείται ο αλγόριθμος FFT (Fast Fourier Transform).

Κατανοούμε φυσικά πως σε τέτοια ανάλυση μπορούν να υποβληθούν μόνο αποθηκευμένες χρονοσειρές και όχι καταγραφές μεμονωμένων συμβάντων. Σύμφωνα με το θεώρημα του Nyquist, η δειγματοληψία πρέπει να γίνεται με ρυθμό τουλάχιστον διπλάσιο από την υψηλότερη συχνοτική συνιστώσα που αναμένεται να εμφανιστεί.

Η ανάλυση συχνότητας βρίσκει το σπουδαιότερο πεδίο εφαρμογών της στη *μελέτη δονήσεων (vibration analysis)*, ακριβώς λόγω των απότομων μεταβολών και της μικρής χρονικής διάρκειας που χαρακτηρίζουν τέτοια φαινόμενα. Για τον εντοπισμό της μετατόπισης ενός αντικειμένου μετράται η δύναμη επιτάχυνσης (g-force) με χρήση ενός επιταχυνσιομέτρου (το μέγεθος αυτό ορίζεται αναλυτικά στην παράγραφο 4.4.3.1). Έτσι η κίνηση του αντικειμένου μπορεί να αναλυθεί σε μία σειρά από αρμονικές ταλαντώσεις. Αν f_m είναι η μέγιστη συχνότητα ταλάντωσης, έχει διαπιστωθεί πως για τον εντοπισμό της απαιτείται η δειγματοληψία να γίνεται με ρυθμό $F_s > 2.56f_m$.

1.2.3.3 Στατιστικές μέθοδοι

Υποθέτοντας πως είναι διαθέσιμες καταγεγραμμένες χρονοσειρές από διαφορετικά συμβάντα, η στατιστική ανάλυση των χρονοσειρών παρέχει τη δυνατότητα εκτίμησης μιας πιθανής συσχέτισης μεταξύ των συμβάντων. Με επιπλέον διαθέσιμο ένα ικανό πλήθος ενδεικτικών χρονοσειρών καλής λειτουργίας, είναι δυνατός άνα πάσα στιγμή ο εντοπισμός μικρών αποκλίσεων της τρέχουσας κατάστασης από τις καταστάσεις καλής λειτουργίας.

Θα αναφερθούμε τώρα σύντομα στις κυριότερες μεθόδους στατιστικής ανάλυσης.

- ***Ανάλυση παλινδρόμησης (regression analysis):*** Δοθέντος ενός συνόλου παρατηρήσεων (ή σημείων), εκτιμά τον τύπο μίας συνάρτησης η οποία το προσεγγίζει βέλτιστα με βάση κάποιο κριτήριο. Για γραμμικές συναρτήσεις, το ευρύτερα χρησιμοποιούμενο κριτήριο είναι η ελαχιστοποίηση του αθροίσματος των τετραγώνων των σφαλμάτων (μέθοδος ελαχίστων τετραγώνων).
- ***Παρεμβολή (interpolation):*** Εκτιμά τη συναρτησιακή σχέση τμηματικά, κάθε φορά μεταξύ διαδοχικών σημείων, ώστε το γράφημα της συνάρτησης που τελικά προκύπτει να περιλαμβάνει όλες τις παρατηρήσεις. Η συνάρτηση αυτή μπορεί να είναι συνεχής ή τμηματικά συνεχής, και να αποτελείται από σταθερές, γραμμικές ή πολυωνυμικές συναρτήσεις.
- ***Συσχέτιση (cross-correlation):*** Εκτιμά το βαθμό ομοιότητας δύο σημάτων ως συνάρτηση της χρονικής μετατόπισης μεταξύ τους. Ένα συχνά χρησιμοποιούμενο κριτήριο είναι η μεγιστοποίηση - κατ' απόλυτη τιμή - του λόγου της συνδιακύμανσης δύο σημάτων για συγκεκριμένη μετατόπιση, προς το γινόμενο των τυπικών τους αποκλίσεων. Ο λόγος αυτός καλείται *συντελεστής συσχέτισης Pearson* και λαμβάνει τιμές από -1 έως 1. Τιμές του συντελεστή πολύ κοντά στα όρια αυτά υποδηλώνουν ότι δύο σήματα - αγνοώντας τη χρονική μετατόπιση - παρουσιάζουν γραμμική σχέση μεταξύ τους.

- ***Εκτίμηση κατανομής πιθανότητας (density estimation):*** Γίνεται με λήψη διαδοχικών ομάδων παρατηρήσεων, από τις οποίες κατασκευάζονται ομάδες κανονικών κατανομών ή ιστογράμματα. Η επανάληψη της διαδικασίας με κάποιον τρόπο συνδυασμού και στάθμισης των αποτελεσμάτων όλων των φάσεων (πχ λήψη του μέσου όρου των κατανομών), βελτιώνει τη σύγκλιση της μεθόδου.

1.2.3.4 Μηχανική μάθηση

Η μηχανική μάθηση, σε στενή σύνδεση με την εξόρυξη δεδομένων (data mining), αποτελεί ένα από τα πιο σημαίνοντα και πολλά υποσχόμενα πεδία εφαρμογών της τεχνητής νοημοσύνης.

Δίνοντας έναν αυστηρά μαθηματικό ορισμό, μπορούμε να ισχυριστούμε πως περιλαμβάνει όλες τις αλγοριθμικές τεχνικές που αποσκοπούν στην εκτίμηση της συναρτησιακής σχέσης μεταξύ μιας εξαρτημένης μεταβλητής και ενός συνόλου ανεξάρτητων μεταβλητών, βάσει ενός συνόλου παρατηρήσεων.

Η διαφορά μεταξύ των τεχνικών αυτών και των στατιστικών μεθόδων είναι πως οι αλγόριθμοι μηχανικής μάθησης είναι σε θέση να αναπροσαρμόζονται και να βελτιώνονται στις προβλέψεις και τις εκτιμήσεις τους όσο τροφοδοτούνται με νέα αντιπροσωπευτικά δεδομένα, δηλαδή να «αυτοεκπαιδεύονται» και σταδιακά μόνοι τους να «μαθαίνουν» να αναγνωρίζουν την προσδιοριστέα σχέση.

Αν θεωρήσουμε πως Y είναι η εξαρτημένη μεταβλητή, X το διάνυσμα των ανεξάρτητων μεταβλητών και f η προσδιοριστέα συνάρτηση Y = f(X), τότε η διαδικασία εκτίμησης της f από έναν αλγόριθμο A βάσει ενός συνόλου παρατηρήσεων S αποτελούμενου από ζεύγη τιμών (X, Y) καλείται *επιβλεπόμενη εκμάθηση* (*supervised learning*) της *f* από τον *A* με *εκπαίδευση* (*training*) από το *S*. Το *S* καλείται *σύνολο δεδομένων εκπαίδευσης* (*training data set*).

Οι αλγόριθμοι μηχανικής μάθησης αποτελούν το κυριότερο εργαλείο ανάλυσης δεδομένων για πληθώρα εφαρμογών, όπως είναι η αναγνώριση προτύπων (pattern recognition), η όραση υπολογιστών (computer vision) και η προγνωστική ανάλυση (predictive analytics), στην οποία εντάσσεται και το πεδίο εφαρμογών του CM για σκοπούς PdM. Στο τελευταίο, η χρήση ενός αλγορίθμου μηχανικής μάθησης επιτρέπει την ταχεία αναγνώριση καταστάσεων που μπορούν να οδηγήσουν μελλοντικά σε δυσλειτουργία το σύστημα, εφόσον ο αλγόριθμος έχει εκπαιδευτεί με κατάλληλα σύνολα δεδομένων.

Τα κυριότερα μοντέλα μηχανικής μάθησης είναι τα εξής:

- ***Νευρωνικά δίκτυα:*** Τα *τεχνητά νευρωνικά δίκτυα (artificial neural networks, ANN)*, ή απλούστερα *νευρωνικά δίκτυα*, οφείλουν την ονομασία τους στη δομή τους, η οποία παραπέμπει ευθέως στη διασύνδεση των νευρώνων του εγκεφάλου. Αποτελούνται από υπολογιστικούς κόμβους *(νευρώνες)* οργανωμένους σε επίπεδα: ένα επίπεδο εισόδου *(input layer)*, στο οποίο εισάγονται οι τιμές των ανεξάρτητων μεταβλητών, ένα επίπεδο εξόδου *(output layer)*, από το οποίο εξάγεται η τιμή της εξαρτημένης μεταβλητής, και ένα ή περισσότερα ενδιάμεσα

επίπεδα, τα λεγόμενα «κρυφά» επίπεδα *(hidden layers)*. Κάθε κόμβος δέχεται ως είσοδο ένα γραμμικό συνδυασμό των εξόδων των κόμβων του προηγούμενου επιπέδου, και αντιστοιχίζει την είσοδο σε μία τιμή εξόδου μέσω μίας ορισμένης *συνάρτησης ενεργοποίησης (activation function)*. Η ικανότητα αυτοεκπαίδευσης του αλγορίθμου συνίσταται ακριβώς στην αναπροσαρμογή των συντελεστών (βαρών) των συνιστωσών των εισόδων στους κόμβους, ώστε με κάθε επανάληψη να προσεγγίζεται καλύτερα η επιθυμητή τιμή εξόδου για κάθε είσοδο. Η αναπροσαρμογή αυτή γίνεται με ανάδραση του σφάλματος από το επίπεδο εξόδου προς τα πίσω *(πίσω-διάδοση, back-propagation)*.

- ***Δένδρα αποφάσεων:*** Ένα *δένδρο αποφάσεων (decision tree)* απεικονίζει τις τιμές της ανεξάρτητης μεταβλητής σε αντικείμενα ενός πεπερασμένου διακριτού συνόλου, τα οποία αναπαρίστανται από τα φύλλα, βάσει μίας ακολουθίας αποφάσεων, η οποία συνιστά ουσιαστικά μία διαδρομή από τη ρίζα στο εκάστοτε σωστό φύλλο.
- ***Μπεϋζιανά δίκτυα:*** Αποτελούν ισχυρά εργαλεία μελέτης φαινομένων κατά τα οποία οι μεταβλητές εισόδου αναπαριστούν τυχαία γεγονότα, και ζητούμενο είναι η εξέταση των πιθανοτικών (μη ντετερμινιστικών) αιτιακών σχέσεων μεταξύ τους. Ένα *μπεϋζιανό δίκτυο* (Bayesian network) αποτελεί έναν κατευθυνόμενο ακυκλικό γράφο (directed acyclic graph, DAG) με κόμβους τις μεταβλητές-γεγονότα. Η αιτιακή σχέση μεταξύ δύο γεγονότων αναπαρίσταται από μία ακμή με κατεύθυνση από το αίτιο προς το αποτέλεσμα, και βάρος την αντίστοιχη δεσμευμένη πιθανότητα. Διασχίζοντας το γράφο, είναι δυνατή η εύρεση της πιθανότητας συμβάντων που εξαρτώνται από πολλά συμβάντα των ανώτερων επιπέδων, η ακόμα και της από κοινού κατανομής πιθανότητας περισσότερων συμβάντων.

1.2.4 Πολιτικές συντήρησης

Για σκοπούς συντήρησης βιομηχανικών συστημάτων ακολουθείται συνήθως μία από τις ακόλουθες *πολιτικές*, ή συνδυασμός τους:

- ***Λειτουργία μέχρι την εκδήλωση βλάβης (Reactive/Run-to-failure maintenance, RTF):*** Το σύστημα αφήνεται να λειτουργεί ακατάπαυστα χωρίς συντήρηση μέχρι να παρουσιάσει βλάβη. Ενδείκνυται μεταξύ άλλων αν η επισκευή του συστήματος είναι ασύμφορη σε σύγκριση με την αντικατάστασή του.

- ***Προληπτική συντήρηση (preventive maintenance, PvM)*:** Πραγματοποιούνται εργασίες συντήρησης σε τακτά χρονικά διαστήματα με στόχο την πρόληψη μελλοντικών βλαβών. Συνεπάγεται ένα σχετικά υψηλό – αλλά πάγιο, συνεπώς γνωστό – κόστος σε έξοδα συντήρησης, ελαχιστοποιεί όμως την απώλεια παραγωγικότητας σε περίπτωση βλάβης.

- ***Προγνωστική ή προβλεπτική συντήρηση (predictive maintenance, PdM)*:** Αξιοποιεί την τρέχουσα κατάσταση και το καταγεγραμμένο ιστορικό του

συστήματος για την εκτίμηση της μελλοντικής του κατάστασης, εφαρμόζοντας τεχνικές ανάλυσης δεδομένων. Μια αρκετά αξιόπιστη εκτίμηση επιτρέπει στους μηχανικούς συντήρησης να βελτιστοποιήσουν τον προγραμματισμό της συντήρησης, πραγματοποιώντας εργασίες μόνον εφόσον αυτές είναι απαραίτητες για την πρόληψη μίας επικείμενης δυσλειτουργίας. Έτσι τόσο το κόστος συντήρησης όσο και η απώλεια παραγωγικότητας ελαχιστοποιούνται.

1.2.5 Επιλογή βέλτιστης πολιτικής

Η επιλογή της ενδεδειγμένης πολιτικής συντήρησης προκύπτει από μία ανάλυση κόστους-οφέλους βασισμένη στα χαρακτηριστικά του συστήματος και το ιστορικό λειτουργίας του. Μια ενδεικτική σχέση κόστους-οφέλους που εφαρμόζεται σε ένα ευρύ φάσμα συστημάτων φαίνεται στον παρακάτω πίνακα:

Συχνότητα βλάβης	Απώλεια παραγωγικότητας		
	Χαμηλή	Μέτρια	Υψηλή
Χαμηλή	RTF	PvM	PdM
Μέτρια	PvM	PvM	PvM
Υψηλή	SLU	PvM	DOM

Πίνακας 1.5: Επιλογή πολιτικής συντήρησης

Στη σχέση αυτή μετέχουν δύο παράγοντες, το κόστος συντήρησης (άμεσα εξαρτώμενο από τη συχνότητα των βλαβών) και η απώλεια παραγωγικότητας εξαιτίας της απουσίας του συστήματος από την παραγωγή. Ο προγραμματισμός της συντήρησης σε τακτά χρονικά διαστήματα (πράσινο χρώμα) αποτελεί την πιο συμφέρουσα επιλογή στις περιπτώσεις όπου ένας τουλάχιστον από τους δύο παράγοντες διατηρείται σε προβλέψιμα και εύκολα διαχειρίσιμα επίπεδα, καθώς το κόστος συντήρησης και η απώλεια παραγωγικότητας βρίσκονται σε ισορροπία.

Σε μία παραγωγική μονάδα όμως πιθανότατα υπάρχουν και συστήματα κρίσιμης σημασίας, των οποίων η διακοπή λειτουργίας μπορεί να σταματήσει ολόκληρη την παραγωγική διαδικασία. Το ρίσκο μάλιστα μεγιστοποιείται αν οι σοβαρές βλάβες σε τέτοια συστήματα παρουσιάζουν μεγάλο βαθμό τυχαιότητας, δηλαδή χαμηλή και δύσκολα προσδιορίσιμη συχνότητα. Τότε το κόστος της εγκατάστασης μιας λύσης CM, αν και αρκετά υψηλό, καταβάλλεται εφάπαξ και αντισταθμίζεται από το όφελος της δυνατότητας πρόβλεψης και αποτροπής βλαβών δυσθεώρητα υψηλότερου κόστους. Γι΄αυτόν ακριβώς το λόγο τέτοιες περιπτώσεις αποτελούν το κυριότερο πεδίο εφαρμογής πολιτικών PdM (κίτρινο χρώμα).

Βέβαια, κάποιες κατηγορίες βλαβών ενδέχεται να εξακολουθούν να παρουσιάζονται με υψηλή συχνότητα, ανεξάρτητα από τις εργασίες συντήρησης που πραγματοποιούνται. Εάν τέτοιες βλάβες συνδυάζονται με πολύ μικρούς ή πολύ μεγάλους χρόνους επιδιόρθωσης, η επίπτωση στην παραγωγικότητα θα είναι

αντίστοιχα είτε αμελητέα, είτε εξαιρετικά σοβαρή. Στην πρώτη περίπτωση, θα πρέπει να εξεταστεί το ενδεχόμενο οι βλάβες να οφείλονται σε συστηματικά ανθρώπινα σφάλματα (πχ χειρισμού του εξοπλισμού), κάτι που αντιμετωπίζεται με επανεκπαίδευση ενός τμήματος του εργατικού δυναμικού (*skill level upgrade*, *SLU*). Διαφορετικά, είναι πολύ πιθανό ο συγκεκριμένος εξοπλισμός να είναι εγγενώς ανεπαρκής για το φορτίο της παραγωγής που του έχει ανατεθεί, επομένως να απαιτείται η μόνιμη απομάκρυνσή του από τη διαδικασία (*design out machine*, *DOM*).

1.3 Προσέγγιση στην παρούσα εργασία

Για την υλοποίηση του συστήματος της παρούσας εργασίας κάνουμε τις εξής σχεδιαστικές επιλογές με βάση όσα περιγράφηκαν στις προηγούμενες παραγράφους:

- ***Στρατηγική εποπτείας*:** Λόγω των περιορισμών ενός ενσωματωμένου συστήματος τροφοδοτούμενου από μπαταρία τόσο σε κατανάλωση ισχύος όσο και σε διαθέσιμη μνήμη, επιλέγουμε τη στρατηγική *διακριτής* εποπτείας.

- ***Δομή συστήματος εποπτείας*:** Το σύστημα μέτρησης, το σύστημα καταγραφής και το ένα άκρο του συστήματος ειδοποίησης συνυπάρχουν ως ξεχωριστά κυκλώματα στην ίδια πλακέτα και συνιστούν έναν κόμβο του δικτύου CM. Το άλλο άκρο του συστήματος ειδοποίησης βρίσκεται στην εφαρμογή πελάτη που εκτελείται στην κινητή συσκευή. Κάθε κόμβος του δικτύου συνδέεται ανά πάσα στιγμή με μία μόνο εφαρμογή πελάτη. Έτσι, ο πελάτης μαζί με τους συνδεδεμένους με αυτόν κόμβους σχηματίζει μία τοπολογία αστέρα.

2 Έξυπνα Δίκτυα

2.1 Έξυπνα δίκτυα – το Internet of Things (IoT)

Η πρόοδος που συντελείται τα τελευταία χρόνια – σε επίπεδο έρευνας και παραγωγής – στους κλάδους της Πληροφορικής και των Τηλεπικοινωνιών, και ιδιαίτερα στους τομείς των ενσωματωμένων συστημάτων (embedded systems) και των ασύρματων τοπικών δικτύων (WLAN), επιτρέπει την εισαγωγή επεξεργαστικής ισχύος σε αντικείμενα της καθημερινής ζωής και τη δικτύωση αυτών σε ολοένα και μεγαλύτερη κλίμακα παγκοσμίως, αποτελώντας την κινητήριο δύναμη στην ανάπτυξη μιας νέας τεχνολογικής τάσης, του *Internet of Things* (IoT).

Ο όρος *Internet of Things* αποδίδεται επί λέξει ως «διαδίκτυο των αντικειμένων» (ΔΔΑ), μπορεί συνεπώς να ερμηνευθεί κυριολεκτικά ως ένα παγκόσμιο δίκτυο από αντικείμενα εξοπλισμένα με ενσωματωμένα συστήματα, τα οποία – εκτός από την απαιτούμενη επεξεργαστική ισχύ και δικτυακές διεπαφές – περιλαμβάνουν περιφερειακά όπως αισθητήρες, συστήματα ελέγχου κ.ά. Αυτή η λειτουργικότητα είναι που καθιστά τα συστήματα αυτά «έξυπνες συσκευές» (smart devices), ικανές να αλληλεπιδρούν και να συνεργάζονται για την επίτευξη ενός στόχου.

Οι συσκευές-κόμβοι του δικτύου αυτού όμως αναμένει κανείς να αξιοποιούνται σε διαφορετικές εφαρμογές, να προσφέρουν διαφορετικές υπηρεσίες και να ανήκουν σε τοπικά δίκτυα που χρησιμοποιούν διαφορετικά πρωτόκολλα, γεγονός το οποίο καθιστά δύσκολο – πιθανώς και ασύμφορο – εγχείρημα την προτυποποίησή του. Γενικά είναι αποδεκτό πως το IoT δεν αποτελεί μια συγκεκριμένη τεχνολογία με ενιαίο πρότυπο, ή μια αυστηρά ορισμένη έννοια, αλλά σε μεγάλο βαθμό μια νέα φιλοσοφία, ένα όραμα ενοποίησης του φυσικού και του ψηφιακού κόσμου μέσω των «έξυπνων συσκευών», που σήμερα έχει ήδη περάσει από το στάδιο της ιδέας στη φάση πραγματοποίησής του. Παρ' όλα αυτά, μέσω του ορισμού που δόθηκε παραπάνω αντικατοπτρίζεται επαρκώς το IoT – ως πεδίο εφαρμογών πλέον – στις τεχνολογίες που το υλοποιούν.

Οι κυριότεροι φορείς που εμπλέκονται στην ανάπτυξη των παραπάνω τεχνολογιών και εφαρμογών προσεγγίζουν από διαφορετικές πλευρές, τις οποίες αναλύουμε παρακάτω, τη φύση του IoT και τις δυνατότητες που προσφέρει:

- ***Προσανατολισμός στα πράγματα (things-oriented)***

Η προσέγγιση αυτή - από την οποία άλλωστε προκύπτει ιστορικά για πρώτη φορά και ο όρος IoT - αντιμετωπίζει τα «πράγματα», δηλαδή τους κόμβους του δικτύου, ως «έξυπνα αντικείμενα» (smart objects) - αντικείμενα εξοπλισμένα με

«έξυπνες συσκευές» συνεχώς διευρυνόμενης «ευφυΐας», σε κάθε περίπτωση με δυνατότητα ταυτοποίησης και εντοπισμού σε πραγματικό χρόνο, ανεξάρτητα από την τεχνολογία διασύνδεσής τους.

Η ανάπτυξη ασύρματων τεχνολογιών ταυτοποίησης, όπως είναι η ταυτοποίηση μέσω ραδιοσυχνοτήτων (radio frequency identification, RFID) και η επικοινωνία κοντινού πεδίου (near field communication, NFC), άνοιξε στην πραγματικότητα το δρόμο για τις πρώτες εφαρμογές IoT, ενώ η ευρεία αξιοποίησή τους - ήδη στις μέρες μας - σε πληθώρα προϊόντων διατηρεί τον «προσανατολισμό στα πράγματα» στη θέση του «οδηγού» των εξελίξεων στον τομέα αυτό.

Ταυτόχρονα, η ενσωμάτωση υπολογιστικής ισχύος, αισθητήρων και συστημάτων ελέγχου προσδίδει στα «έξυπνα αντικείμενα» επιπλέον την ικανότητα παρακολούθησης και ανάλυσης των συνθηκών του περιβάλλοντος και της κατάστασής τους, καθώς και ανάληψης αυτόνομης δράσης στη βάση των δεδομένων αυτών, με χειρισμό συνδεδεμένων σε αυτά μηχανικών μερών, μηχανών κλπ. Τα δίκτυα από τέτοια αντικείμενα καλούνται συχνά *ασύρματα δίκτυα αισθητήρων-ενεργοποιητών* (*wireless sensor-actuator networks*, WSAN), και τροφοδοτούν ένα πολλά υποσχόμενο πεδίο εφαρμογών.

Ένα από τα πιο φιλόδοξα οράματα για το μέλλον των smart objects συνδέεται με την έννοια του *spime* (από τη συγκόλληση των λέξεων "*space*"-"χώρος" και "*time*"-"χρόνος"). Ο νεολογισμός αυτός περιγράφει θεωρητικά ένα αντικείμενο με μία εικονική-λογική υπόσταση και το πολύ μία ανά πάσα χρονική στιγμή φυσική υπόσταση, το οποίο μπορεί να ταυτοποιείται και να παρακολουθείται στο χώρο και το χρόνο καθ’ όλη τη διάρκεια της ζωής του. Ένα σημαντικό μέρος από τα smart objects σήμερα ήδη εμπίπτουν εν μέρει στον ορισμό αυτό.

- ***Προσανατολισμός στο Διαδίκτυο (Internet-oriented)***

Η προσέγγιση αυτή επικεντρώνεται στη συνδεσιμότητα των κόμβων του IoT στο Διαδίκτυο, απευθείας ή μέσω μίας πύλης (gateway) σε επίπεδο δικτύου, ανεξάρτητα από τις υπηρεσίες που παρέχονται από κάθε κόμβο.

Το κύριο πρόβλημα προς επίλυση από αυτή τη σκοπιά είναι η υλοποίηση της στοίβας πρωτοκόλλων του Internet (ή TCP/IP) - από το επίπεδο δικτύου, δηλαδή το πρωτόκολλο διαδικτύου (*Internet Protocol*, IP), και πάνω - στο σύνολο των smart devices, εξέλιξη που θα επιτρέψει τη μετάβαση από τη σύγχρονη εποχή του «διαδικτύου των συσκευών» (*Internet of Devices*) στη νέα εποχή του IoT. Γι’ αυτόν το σκοπό έχουν συγκροτηθεί ομάδες εργασίας από επιχειρήσεις και φορείς, όπως η IP for Smart Objects Alliance (IPSO).

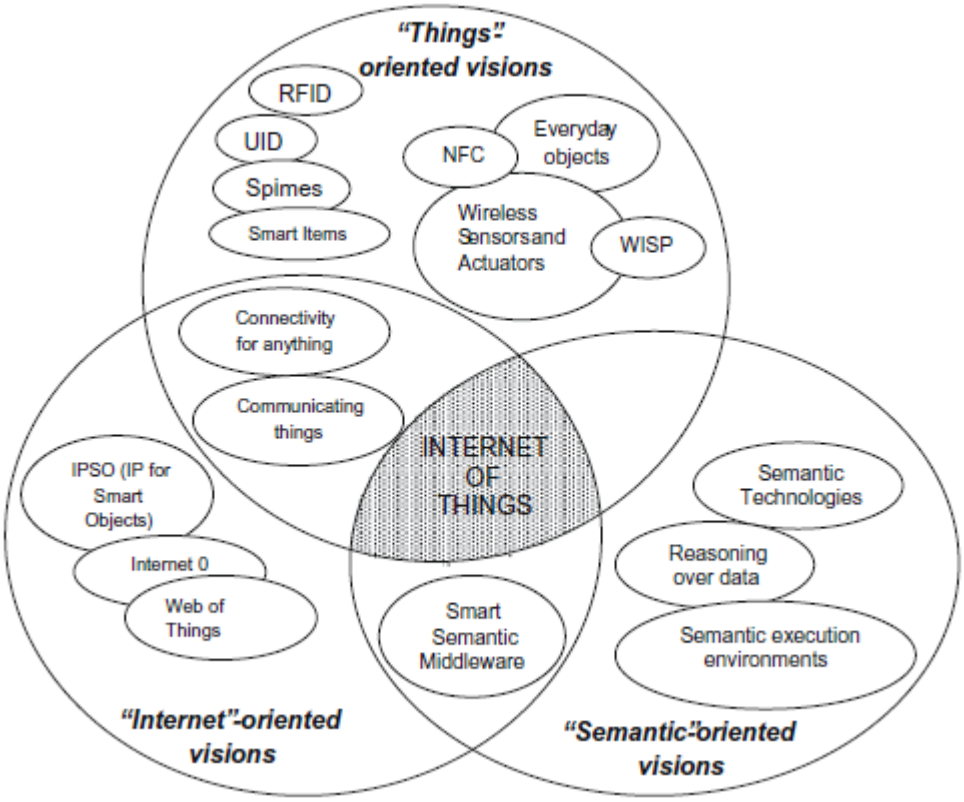
Σύμφωνα με την IPSO, αρκεί η προσαρμογή του IP ώστε να μπορεί να χρησιμοποιεί τα οριζόμενα από το πρότυπο IEEE 802.15.4 πρωτόκολλα ασύρματων δικτύων προσωπικού χώρου (wireless personal area network, WPAN), όπως τα Zigbee, Thread και SNAP, στο φυσικό στρώμα και στο υποστρώμα ελέγχου πρόσβασης μέσου (MAC, medium access control) του επιπέδου ζεύξης δεδομένων (link layer).

Για απευθείας συνδεσιμότητα στο Διαδίκτυο του συνόλου των smart devices - που αποτελεί άλλωστε και τον τελικό στόχο του IoT - απαιτείται η χρήση ενός δυσθεώρητα μεγάλου αριθμού δημόσιων (public) διευθύνσεων IP. Το IPv4, μέσω του οποίου δρομολογείται ακόμα το μεγαλύτερο μέρος της διαδικτυακής κίνησης, ορίζει διευθύνσεις μήκους 32 bit, συνεπώς το μέγιστο πλήθος διευθύνσεων περιορίζεται στις $2^{32} \sim 4 \cdot 10^9$.

Σε κάθε περίπτωση, η εξάντληση των διαθέσιμων δημόσιων διευθύνσεων IPv4 επιβάλλει τη χρήση του IPv6 σε οποιαδήποτε τέτοια λύση. Το μήκος των διευθύνσεων IPv6 (128 bit) είναι επαρκές ώστε να δίνει έναν πρακτικά απεριόριστο - σε σύγκριση με το προβλεπόμενο πλήθος αντικειμένων - αριθμό διευθύνσεων.

- ***Σημασιολογικός προσανατολισμός (semantic-oriented)***

Η προσέγγιση αυτή εξετάζει το IoT ως ένα δίκτυο από πηγές και σημεία συλλογής δεδομένων, αναζητώντας μεθόδους ανάλυσης και επεξεργασίας των δεδομένων αυτών για ένα συγκεκριμένο σκοπό. Εδώ εντάσσονται το Web of Things, καθώς και οι τεχνικές χρονικής και χωρικής συσχέτισης των δεδομένων που προέρχονται από τις smart devices.



Σχήμα 2.1: Τρόποι εννοιολογικής προσέγγισης του IoT

2.2 Ιστορική αναδρομή

Παρακάτω παρουσιάζουμε, σε χρονολογική σειρά, κάποια γεγονότα που θα μπορούσαν να θεωρηθούν «σταθμοί» στην ανάπτυξη του IoT και των «έξυπνων δικτύων»:

1982 - Καταγράφεται η πρώτη ιστορικά δικτυωμένη «έξυπνη συσκευή». Πρόκειται για έναν αυτόματο πωλητή αναψυκτικών στο Πανεπιστήμιο Carnegie Mellon των ΗΠΑ, ο οποίος ήταν συνδεδεμένος στο Διαδίκτυο και μπορούσε ανά πάσα στιγμή να αναφέρει τη διαθεσιμότητα και την κατάσταση των προϊόντων στο εσωτερικό του.
1991 - Ο Mark Weiser, επικεφαλής τότε του εργαστηρίου επιστήμης υπολογιστών της Xerox, στο paper του με τίτλο "*The Computer of the 21st Century*" εισάγει την έννοια της «*πανταχού παρούσας υπολογιστικής*» (*Ubiquitous Computing*) για να περιγράψει τη διείσδυση της πληροφορικής σε ολοένα και περισσότερους τομείς της καθημερινής ζωής μέσω της ενσωμάτωσης υπολογιστικής ισχύος σε αντικείμενα.
1994 - Στην έκδοση του Reza Haji "*Smart Networks for Control*" για το περιοδικό IEEE Spectrum σκιαγραφούνται για πρώτη φορά οι δυνατότητες των «έξυπνων δικτύων» (*smart networks*) και η εμβέλεια των πιθανών εφαρμογών τους στον έλεγχο αντικειμένων, ξεκινώντας από απλές οικιακές συσκευές και καταλήγοντας σε πολύπλοκες βιομηχανικές εγκαταστάσεις και συστήματα αυτοματισμού.
1999 - Ο Kevin Ashton, τότε στέλεχος της Procter & Gamble, χρησιμοποιεί πρώτος τον όρο *Internet of Things* ως τίτλο μιας παρουσίασής του και δίνει έναν things-oriented ορισμό. Στο όραμά του για το IoT σημαντικό ρόλο διαδραματίζει η χρήση του RFID. Το ίδιο έτος συνεργάζεται με το MIT για την ίδρυση του *Auto-ID Center*, μιας ερευνητικής κοινοπραξίας για αυτόν το σκοπό. Στα επιτεύγματα του Auto-ID Center συγκαταλέγεται η ανάπτυξη του *ηλεκτρονικού κωδικού προϊόντος* (*Electronic Product Code*, EPC).
2003 - Ιδρύονται οι κοινοπραξίες *Auto-ID Labs* και *EPCGlobal*, διαδεχόμενες το Auto-ID Center και συνεχίζοντας τις ερευνητικές του δραστηριότητες. Ταυτόχρονα, η ομάδα εργασίας 802.15 της IEEE ορίζει το πρότυπο IEEE 802.15.4 για πρωτόκολλα WPAN.
2005 - Σε σχετική σύνοδο του ΟΗΕ προβλέπεται η μετάβαση στην εποχή του IoT, όπου κύριες πηγές της δικτυακής κίνησης - αντί των χρηστών του Internet - θα είναι τα smart objects. Παράλληλα, η Διεθνής Ένωση Τηλεπικοινωνιών (*International Telecommunications Union*, ITU) διατυπώνει τη δική της αντίληψη για το IoT, βασιζόμενη στην αρχή «συνδεσιμότητα για τα πάντα».
2006 - Η Ευρωπαϊκή Επιτροπή οραματίζεται το IoT ως ένα δίκτυο από «ταυτοποιήσιμα αντικείμενα με εικονικές προσωπικότητες που λειτουργούν σε έξυπνους χώρους και χρησιμοποιούν ευφυείς διεπαφές για τη μεταξύ τους διασύνδεση και επικοινωνία σε εφαρμογές κοινωνικές, περιβαλλοντικές ή/και οριζόμενες από το χρήστη».
2007 - Δημοσιεύεται η προδιαγραφή του 6LoWPAN και ιδρύεται ομάδα εργασίας της IETF (*Internet Engineering Task Force*) για την ανάπτυξή του.

2008 - Ιδρύεται η IP for Smart Objects Alliance (IPSO).
2018 - Δημοσιεύεται η προδιαγραφή της τεχνολογίας *Thread*, αποτέλεσμα της συνεργασίας 50 επιχειρήσεων για την ανάπτυξη ενός ενιαίου δικτυακού πρωτοκόλλου για smart devices, συμβατού με το 6LoWPAN.

2.3 Σύγχρονες εφαρμογές

Η ένταξη των smart networks στην καθημερινή ζωή είναι πλέον γεγονός. Στις πιο γνωστές και διαδεδομένες εφαρμογές συγκαταλέγονται:

- **Εφαρμογές υγείας:** Η παρακολούθηση σε πραγματικό χρόνο ζωτικών δεικτών λειτουργίας του ανθρώπινου οργανισμού από φορητές συσκευές (*wearables*), όπως heart rate monitors και fitness trackers, δεν αποφέρει μόνο προσωπικό όφελος στο χρήστη, αλλά καθιστά δυνατή και την καταγραφή, ανάλυση και συσχέτιση μεγάλου όγκου τέτοιου είδους δεδομένων για την πρόληψη και τη διάγνωση ασθενειών, προάγοντας παράλληλα και την ανάπτυξη κλάδων όπως η τηλεϊατρική.

- **Κτίρια:** Εφαρμογές «έξυπνων» σπιτιών και κτιρίων (smart home/building) βρίσκονται σε ευρεία χρήση εδώ και αρκετά χρόνια, με συνεχώς διευρυνόμενες δυνατότητες. Η λειτουργικότητα που προσφέρει ο ασύρματος έλεγχος αντικειμένων - όπως θυρών και συστημάτων φωτισμού - μέσω των smart networks δεν αφορά μόνο στη διευκόλυνση της καθημερινής ζωής, αλλά και σε ζητήματα ασφάλειας, αντιπυρικής θωράκισης κλπ.

- **Πόλεις:** Απομακρυνόμενοι από τα κτίρια και κινούμενοι σε πιο μακροσκοπικό επίπεδο, είναι εύκολο να αντιληφθούμε το ρόλο που τα smart networks μπορούν να διαδραματίσουν στην πρόληψη και αντιμετώπιση καταστροφών, την προστασία του περιβάλλοντος, της ζωής και τη βελτίωση της ποιότητας ζωής, ακριβώς λόγω της δυνατότητας εύκολης και ταχείας καταγραφής δεικτών τόσο του φυσικού (ατμόσφαιρα, γη, υδροφόρος ορίζοντας), όσο και του ανθρωπογενούς περιβάλλοντος (τηλεματική, εποπτεία οδικής κίνησης).

2.4 Ασύρματα Δίκτυα Αισθητήρων

Στην απλούστερη μορφή του, ένα *ασύρματο δίκτυο αισθητήρων (wireless sensor network, WSN)* απαρτίζεται από ένα σύνολο *κόμβων-αισθητήρων* (*sensor nodes*) στο χώρο, οι οποίοι συνδέονται σε ένα κοινό ασύρματο δίκτυο, και έναν *κόμβο-πύλη (gateway node)* με ρόλο γέφυρας μεταξύ του εν λόγω δικτύου και του Διαδικτύου. Σε ένα WSN είναι δυνατό να συνδέονται επιπλέον, είτε άμεσα ως κόμβοι, είτε διαμέσου της πύλης:

- *Κόμβοι ενεργοποιητών (actuator nodes)* συνδεδεμένοι με συστήματα ελέγχου όπως διακόπτες, αντλίες, βαλβίδες, PLCs κλπ. Τότε το δίκτυο συνιστά ένα WSAN, όπως περιγράφηκε στην ενότητα 2.1.

- *Συσκευές-πελάτες (clients)*, όπως υπολογιστές και κινητές συσκευές, για συλλογή δεδομένων από το δίκτυο και την εκτέλεση εντολών και υπηρεσιών πάνω σε αυτό.

2.4.1 Τεχνολογία αισθητήρων

Η χρήση σύγχρονων τεχνολογιών κατασκευής – με βάση την τεχνολογία των *μικροηλεκτρομηχανικών συστημάτων* (MEMS) – επιτρέπει την ενσωμάτωση αισθητήρων σε μικροηλεκτρονικά κυκλώματα διαστάσεων της τάξης των mm, ευνοώντας την εξάπλωσή τους σε πληθώρα αντικειμένων.

Κάθε ηλεκτρονικός αισθητήρας, ανταποκρινόμενος στις μεταβολές ενός φυσικού μεγέθους, μετατρέπει το συγκεκριμένο μέγεθος σε ηλεκτρικό (συνήθως τάση). Η σχέση $V = f(K)$ μεταξύ της πραγματικής τιμής K και της μετρώμενης τιμής V του μεγέθους ονομάζεται *χαρακτηριστική* ή *συνάρτηση μεταφοράς* του αισθητήρα.

Μία συνάρτηση μεταφοράς μπορεί να είναι γραμμική, προσεγγιστικά γραμμική, αυστηρά μη γραμμική, ή να αποτελείται από γραμμικές και μη γραμμικές περιοχές. Για τις ανάγκες ακριβών μετρήσεων, είναι επιθυμητό η γραμμική περιοχή της συνάρτησης μεταφοράς να εκτείνεται στο μεγαλύτερο δυνατό εύρος τιμών.

Η συνάρτηση μεταφοράς με την πάροδο του χρόνου μπορεί να μετατοπίζεται από την αρχική της μορφή, μόνιμα ή προσωρινά. Τα κυριότερα φαινόμενα μετατόπισης είναι τα εξής:

- *Υστέρηση*: Η συνάρτηση μεταφοράς μετά από μία μέτρηση μετατοπίζεται προσωρινά οριζόντια, και επανέρχεται στην αρχική της θέση με την παρέλευση ενός μικρού χρονικού διαστήματος. Μέσα στο διάστημα αυτό, διαδοχικές μετρήσεις θα διαγράφουν ένα *βρόχο υστέρησης*, δίνοντας εσφαλμένες τιμές.

- *Ολίσθηση:* Επέρχεται με την μακροχρόνια χρήση του αισθήτηρα. Η συνάρτηση μεταφοράς μετατοπίζεται μόνιμα, τόσο οριζόντια όσο και κατακόρυφα. Η κατάσταση αυτή αίρεται μόνον με εκ νέου βαθνονόμηση του αισθητήρα.

Για την αποθήκευση και την επεξεργασία της πληροφορίας με χρήση κάποιου ψηφιακού συστήματος, είναι απαραίτητη η μετατροπή του αναλογικού σήματος του αισθητήρα σε ψηφιακό (*ψηφιοποίηση*). Σε σταθερά χρονικά διαστήματα λαμβάνονται δείγματα του αναλογικού σήματος, και σε καθένα από τα δείγματα αντιστοιχίζεται μία δυαδική αριθμητική τιμή.

Το σύνολο των διαθέσιμων προς αντιστοίχιση τιμών προκύπτει από μία διαμέριση του συνεχούς συνόλου τιμών $[K_{min}, K_{max}]$ του αισθητήρα σε $N = 2^b$ διακριτές στάθμες, όπου b το μήκος της δυαδικής αναπαράστασης των μετρήσεων.

Η αντιστοίχιση μίας τιμής K στην κατάλληλη στάθμη $Q = 0…N\text{-}1$ καλείται *κβαντισμός*. Αντίστοιχα, η σχέση $Q = g(K)$ καλείται *συνάρτηση κβαντισμού*. Είναι εμφανές πως η g είναι μη αντιστρέψιμη, αφού κάθε στάθμη καλύπτει ένα εύρος τιμών από το αρχικό σύνολο.

Για την ανακατασκευή του αρχικού σήματος από τα ψηφιακά δείγματα, από κάθε στάθμη Q επιλέγεται μία αντιπροσωπευτική τιμή K_d του μεγέθους, σύμφωνα με μία σχέση $K_d = r(Q)$. Η διαφορά $e_d = K_d – K$ της αντιπροσωπευτικής τιμής K_d από την αρχικά μετρηθείσα τιμή K του μεγέθους καλείται *σφάλμα κβαντισμού*.

Στην απλή περίπτωση του ομοιόμορφου κβαντισμού, το σήμα θεωρείται ότι ακολουθεί ομοιόμορφη κατανομή πιθανότητας, δηλαδή οι δυνατές τιμές του κατανέμονται ισοπίθανα σε ισομήκεις στάθμες κβαντισμού. Παρακάτω δίνουμε τις σχέσεις που περιγράφουν έναν ομοιόμορφο κβαντιστή με στρογγυλοποίηση του K_d στο κατώφλι της πλησιέστερης στάθμης (*mid-tread*):

Μήκος στάθμης (βήμα κβαντισμού):
$$L = \left\lfloor \frac{K_{max} - K_{min}}{N} \right\rfloor \qquad (2.2)$$

Επιλογή στάθμης (συνάρτηση κβαντισμού):
$$Q = \left\lfloor \frac{K - K_{min}}{L} + \frac{1}{2} \right\rfloor \qquad (2.3)$$

Συνάρτηση ανακατασκευής:
$$K_d = K_{min} + LQ \qquad (2.4)$$

Μέσο τετραγωνικό σφάλμα κβαντισμού:
$$\overline{e_d}^2 = \frac{L^2}{12} \qquad (2.5)$$

Ανεξάρτητα από την ακολουθούμενη διαδικασία ψηφιοποίησης, η τεχνολογία κατασκευής, το επίπεδο αντοχής σε ακραίες συνθήκες περιβάλλοντος, καθώς και η διαδικασία μετατροπής από αναλογικό σε ψηφιακό επηρεάζουν δύο επιπλέον καθοριστικούς παράγοντες για την αξιολόγηση της ποιότητας ενός αισθητήρα:

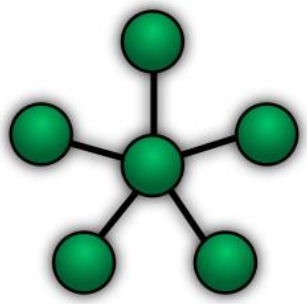
- *Πιστότητα (accuracy):* Αποτελεί μέτρο της απόκλισης της μετρώμενης τιμής από την πραγματική τιμή του μεγέθους.
- *Ακρίβεια (precision):* Αποτελεί μέτρο της διασποράς των τιμών διαδοχικών μετρήσεων υπό τις ίδιες συνθήκες.

2.4.2 Τοπολογία-δρομολόγηση

Η τοπολογία στην οποία οργανώνεται ένα WSN επηρεάζει τις λειτουργίες λήψης αποφάσεων και δρομολόγησης στο εσωτερικό του. Στο σημείο αυτό θα παρουσιάσουμε τις ευρύτερα χρησιμοποιούμενες τοπολογίες, δίχως να επεξηγήσουμε στοιχειώδεις έννοιες της Θεωρίας Γραφημάτων.

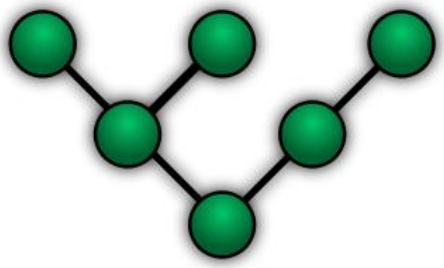
- ***Αστέρας (star):*** Κάθε κόμβος διαθέτει ακριβώς μία ζεύξη, και αυτή μόνο απευθείας προς την πύλη (σύνδεση ενός προς πολλούς), συνεπώς η επικοινωνία

μεταξύ των κόμβων είναι δυνατή μόνο διαμέσου της πύλης. Η τοπολογία αυτή είναι ισοδύναμη με ένα δένδρο μοναδιαίου ύψους, με ρίζα την πύλη και φύλλα το σύνολο των κόμβων-αισθήτηρων.



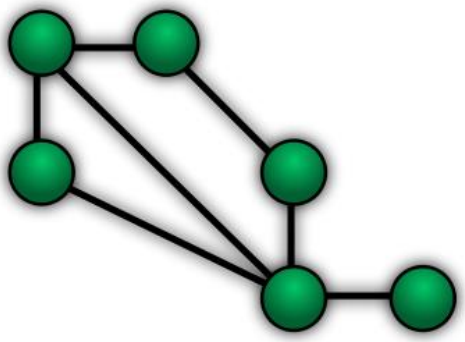
Σχήμα 2.6: Τοπολογία αστέρα

- ***Δένδρο (tree):*** Αποτελεί γενίκευση της τοπολογίας αστέρα. Κάθε κόμβος συνδέεται με κανέναν ή περισσότερους θυγατρικούς κόμβους (παιδιά), και ακριβώς έναν γονικό κόμβο (γονέα), ώστε στο γράφο του δικτύου να μην σχηματίζονται κυκλικές διαδρομές. Η πύλη δεν έχει γονέα και αποτελεί τη *ρίζα* του δένδρου, ενώ οι κόμβοι χωρίς παιδιά συνιστούν τα *φύλλα* του. Το μήκος (σε κόμβους) του μακρύτερου μονοπατιού από τη ρίζα στα φύλλα καλείται *ύψος* του δένδρου. Η δρομολόγηση στο εσωτερικό του δικτύου γίνεται είτε κεντρικά από την πύλη, είτε ιεραρχικά από τον εκάστοτε κόμβο στο γονικό ή το θυγατρικό του.



Σχήμα 2.7: Τοπολογία δένδρου

- ***Κατανεμημένο δίκτυο (mesh):*** Κάθε κόμβος-αισθητήρας μπορεί να διαθέτει ζεύξεις προς οσουσδήποτε επιθυμητούς κόμβους (σύνδεση πολλών προς πολλούς). Οι ζεύξεις μπορούν να εκφράζουν χωρική εγγύτητα ή κάποιας μορφής συσχέτιση των μετρήσεων, διευκολύνοντας την ανάλυση των παραγόμενων δεδομένων. Η κατανεμημένη τοπολογία ευνοεί την αυτορρύθμιση και τη συνεργασία στο πλαίσιο του δικτύου, διαμοιράζοντας στους ίδιους τους κόμβους τις αρμοδιότητες λήψης αποφάσεων, συλλογής δεδομένων και δρομολόγησης, με την πύλη να ενεργεί μόνο ως σημείο σύνδεσης του δικτύου με το Διαδίκτυο.



Σχήμα 2.8: Τοπολογία κατανεμημένου δικτύου

2.4.3 Τροφοδοσία-κατανάλωση ενέργειας

Η ανάγκη για την ελαχιστοποίηση του μεγέθους των κόμβων-αισθητήρων και τη δυνατότητα εγκατάστασής τους σε δύσκολα προσβάσιμα σημεία, επιβάλλει την τροφοδοσία τους από φορητές πηγές, κυρίως μπαταρίες. Για την ενίσχυση της αυτονομίας της κύριας πηγής τροφοδοσίας συχνά χρησιμοποιούνται επικουρικά προς αυτήν φορτιστές που αντλούν ισχύ από το περιβάλλον του αισθητήρα (*ambient power*). Το σύνολο των τεχνικών που χρησιμοποιούνται για το σκοπό αυτό συνιστά το πεδίο εφαρμογών του *energy harvesting (συλλογή ενέργειας)*. Οι ευρύτερα αξιοποιούμενες πηγές ενέργειας περιλαμβάνουν:

- *Ηλιακή ενέργεια*: Πρόκειται για μία από τις δημοφιλέστερες και πιο αποδοτικές επιλογές στην περίπτωση που οι κόμβοι τοποθετούνται σε υπαίθριο χώρο, επιτυγχάνοντας πυκνότητες ισχύος της τάξης μεγέθους μερικών mW/cm^3. Στην μπαταρία συνδέεται μία διάταξη ηλιακού φορτιστή (solar charger), βασισμένη σε ένα μικρών διαστάσεων φωτοβολταϊκό πλαίσιο (panel).

- *Η/Μ ακτινοβολία*: Μία εξίσου φιλική στο περιβάλλον και αποδοτική λύση - που μπορεί να χρησιμοποιηθεί επιπλέον σε κλειστούς χώρους - είναι η άντληση ενέργειας από την εκπεμπόμενη στην ατμόσφαιρα ηλεκτρομαγνητική ακτινοβολία από παρακείμενες συσκευές, τηλεπικοινωνιακούς διαύλους κλπ.

- *Κινητική ενέργεια*: Επιτυγχάνεται με προσάρτηση πιεζοηλεκτρικών στοιχείων, μαγνητικών διατάξεων και μεταβλητών πυκνωτών σε μηχανικά μέρη συσκευών ή κινούμενα σώματα. Η χαμηλότερη απόδοσή της συγκριτικά με τις δύο προηγούμενες πηγές - της τάξης μερικών εκατοντάδων $\mu W/cm^3$ - την καθιστά κατάλληλη για λιγότερο απαιτητικές εφαρμογές.

- *Θερμότητα:* Με ενσωμάτωση θερμοηλεκτρικών γεννητριών (TEG/Seebeck generators) και πυροηλεκτρικών στοιχείων είναι δυνατή η αξιοποίηση των μεταβολών της θερμοκρασίας στο χώρο για παραγωγή διαφορών δυναμικού. Η απόδοση όμως τέτοιων διατάξεων παραμένει εξαιρετικά μικρή, της τάξης μερικών μόνο $\mu W/cm^3$.

Σε κάθε περίπτωση, είναι εμφανές πως δεδομένης της πηγής τροφοδοσίας και του ασύρματου πρωτοκόλλου, ο κύριος περιορισμός που εισάγει ένα WSN εντοπίζεται στην κατανάλωση ενέργειας. Στην κατανάλωση αυτή κυρίαρχο ρόλο διαδραματίζει η μετάδοση δεδομένων στο δίκτυο, και όχι η επεξεργασία δεδομένων ή η εκτέλεση υπολογισμών. Ισοδύναμα, για ελαχιστοποίηση της κατανάλωσης ενέργειας βαρύτητα δίνεται κυρίως στον περιορισμό της χρήσης του πομποδέκτη, και δευτερευόντως .

2.4.4 Ασφάλεια

Όπως σε κάθε δίκτυο υπολογιστών, έτσι και στα WSN ισχύουν οι γνωστές θεμελιώδεις απαιτήσεις για την ασφάλεια πληροφοριακών συστημάτων:

- *Έλεγχος ταυτότητας (Authentication):* Ο αποστολέας οφείλει να είναι σε θέση να εξακριβώσει την ταυτότητα του παραλήπτη, και αντίστροφα. Ο έλεγχος γίνεται με χρήση ασύμμετρης κρυπτογραφίας (δημόσιου κλειδιού), ή ψηφιακών ταυτοτήτων/πιστοποιητικών.

- *Εμπιστευτικότητα (Confidentiality):* Η αποστελλόμενη πληροφορία οφείλει να είναι ορατή μόνο στους παραλήπτες για τους οποίους προορίζεται, και σε κανέναν άλλο. Αυτό εξασφαλίζεται με την κρυπτογράφηση των μηνυμάτων. Τα κλειδιά κρυπτογράφησης μπορούν να προκύψουν από τα κλειδιά που χρησιμοποιήθηκαν κατά τον έλεγχο ταυτότητας, ή να παραχθούν εκ νέου, ειδικά για τη συγκεκριμένη σύνοδο επικοινωνίας.

- *Ακεραιότητα (Integrity):* Το μήνυμα οφείλει να φθάνει αναλλοίωτο στον παραλήπτη. Για τον εντοπισμό αλλαγών χρησιμοποιούνται απλούστερες (πχ υπολογισμός αθροίσματος ελέγχου CRC) ή πιο σύνθετες τεχνικές (πχ ενσωμάτωση περίληψης με χρήση κρυπτογραφικών συναρτήσεων κατακερματισμού - *hash functions*).

- *Εξουσιοδότηση (Authorization):* Μετά την ταυτοποίηση των μετεχόντων στην επικοινωνία μερών, πρέπει να καθορίζονται σαφώς τα δικαιώματα πρόσβασης καθενός από αυτά σε υπηρεσίες και πόρους του δικτύου.

- *Μη αποποίηση (Non-repudiation):* Κανένα από τα μετέχοντα μέρη δεν πρέπει να είναι σε θέση να αρνηθεί την πραγματοποίηση μίας ενέργειας που έχει αναγνωρισθεί από τα υπόλοιπα ως δική του.

Όλα τα παραπάνω απαιτείται να παρέχονται τόσο στα κατώτερα επίπεδα της στοίβας πρωτοκόλλων του δικτύου (φυσικό, ζεύξης δεδομένων), όσο και στα ανώτερα, μέχρι το επίπεδο εφαρμογής. Από τη φύση τους, τα WSN είναι ευάλωτα στους εξής τύπους επιθέσεων:

- *Παθητική παρακολούθηση (passive eavesdropping/sniffing):* Έχοντας υποκλέψει το κλειδί κρυπτογράφησης για τη σύνδεση μεταξύ δύο συσκευών που βρίσκονται στην εμβέλειά του, ο επιτιθέμενος παρακολουθεί το σύνολο της δικτυακής κίνησης της σύνδεσης. Προστασία απέναντι σε τέτοιες επιθέσεις παρέχεται με χρήση ασύμμετρης κρυπτογραφίας, ή κάποιας ασφαλούς μεθόδου διανομής του κλειδιού.

- *Επίθεση ενδιάμεσου (Man-in-the-middle, MITM):* Αναφέρεται και ως *ενεργή παρακολούθηση*. Έχοντας υποκλέψει τα κλειδιά ελέγχου ταυτότητας, ο επιτιθέμενος έχει τη δυνατότητα να παρουσιάζεται σε καθεμία από τις δύο συσκευές υποδυόμενος την άλλη. Έτσι, προτού οι συσκευές εγκαταστήσουν σύνδεση μεταξύ τους, ο επιτιθέμενος παρεμβάλλεται στην επικοινωνία και εγκαθιστά σύνδεση ξεχωριστά με καθεμία από αυτές. Εφόσον αυτό που οι συσκευές αντιλαμβάνονται είναι ότι συνδέονται απευθείας μεταξύ τους, ο επιτιθέμενος είναι ελεύθερος όχι μόνο να παρακολουθεί, αλλά και να τροποποιεί τα δικτυακή κίνηση που ανταλλάσσουν. Προστασία απέναντι σε τέτοιες επιθέσεις παρέχεται με χρήση κάποιας ασφαλούς μεθόδου για έλεγχο ταυτότητας, πιθανώς δίχως μετάδοση των κλειδιών στον αέρα.

- *Αποστέρηση ύπνου (Sleep deprivation):* Ο επιτιθέμενος επιχειρεί επανειλημμένα να επικοινωνήσει με μία συσκευή, αποστέλλοντας διαδοχικά ανεπιτυχή αιτήματα. Ακόμα και αν τα πακέτα απορρίπτονται, η συσκευή-στόχος υποχρεούται κάθε φορά να εξέρχεται από την κατάσταση εξοικονόμησης ενέργειας και να ενεργοποιεί τον πομποδέκτη της. Πρόκειται λοιπόν για μία επίθεση *άρνησης υπηρεσίας* (*Denial-of-Service*, DoS) που αποσκοπεί περισσότερο στην εξάντληση της πηγής τροφοδοσίας της συσκευής, παρά στην απασχόλησή της ή τη συμφόρηση του δικτύου, γεγονός που την καθιστά πιο επικίνδυνη για τα WSN συγκριτικά με άλλους τύπους ασύρματων δικτύων. Μπορεί να αντιμετωπιστεί χρησιμοποιώντας ξεχωριστό πομποδέκτη χαμηλής κατανάλωσης για την αφύπνιση της συσκευής (wakeup radio). Σε αυτή την περίπτωση, ο κύριος πομποδέκτης επιφορτίζεται μόνο με τη μετάδοση και λήψη δεδομένων πάνω από εγκατεστημένες συνδέσεις.

2.5 Πρωτόκολλα WPAN

Ως *ασύρματα δίκτυα προσωπικού χώρου (wireless personal networks, WPAN)* αναφέρονται τα ασύρματα δίκτυα με εμβέλεια στον προσωπικό χώρο εργασίας ενός ατόμου, δηλαδή σε αποστάσεις της τάξης των 10 m σε εσωτερικούς χώρους, και χρησιμοποιούνται κυρίως για σύνδεση κοντινών περιφερειακών συσκευών. Παραδείγματα ενσύρματων ζεύξεων ανάλογης εμβέλειας αποτελούν οι κλασικές σειριακές θύρες, καθώς και οι δίαυλοι USB, Thunderbolt και FireWire (IEEE 1394).

Οι δημοφιλέστερες τεχνολογίες WPAN για εφαρμογές δικτύων αισθητήρων περιλαμβάνουν:

- *Bluetooth/Bluetooth Low Energy:* Παρουσιάζεται λεπτομερώς στο κεφάλαιο 3.

- *Zigbee:* Αναπτύχθηκε το 1998 και η πρώτη του προδιαγραφή δημοσιεύθηκε το 2003. Προσφέρει χαμηλότερους ρυθμούς μετάδοσης σε σύγκριση με τα υπόλοιπα πρωτόκολλα που εξετάζουμε, το στοιχείο όμως που πραγματικά το διακρίνει είναι η συμβατότητά του με το IEEE 802.15.4, γεγονός που το καθιστά ικανό από κατασκευής για συνεργασία με άλλες καινοτόμες τεχνολογίες του προτύπου αυτού (6LoWPAN, Thread κλπ), καθώς και με το IPv6.

- *ANT/ANT+:* Το ANT έκανε την πρώτη του εμφάνιση το 2003, προοριζόμενο αρχικά για αθλητικές εφαρμογές (fitness sensors κλπ), με την ειδική του εκδοχή ANT+ για αυξημένη εξοικονόμηση ενέργειας να ακολουθεί την επόμενη ακριβώς χρονιά. Το πεδίο εφαρμογών του έχει πλέον επεκταθεί στη βιομηχανία, τους αυτοματισμούς και τα ασύρματα δίκτυα αισθητήρων. Παρουσιάζει αρκετές ομοιότητες με το Bluetooth Low Energy στη σχεδιαστική φιλοσοφία και τη λειτουργία του, δίχως να είναι συμβατό με αυτό. Οι κόμβοι του δύνανται να λειτουργούν τόσο ως πηγές ή καταβόθρες δεδομένων (παρόμοια με τους ρόλους *slave* και *master* αντίστοιχα σε ένα Bluetooth piconet), όσο και ως κόμβοι προώθησης (*relay nodes*) ή δρομολόγησης (*routing nodes*).

- *Z-Wave:* Σχεδιάστηκε το 1999 από την δανική Zensys, και αποτέλεσε ένα από τα πρώτα δοκιμασμένα και αξιόπιστα πρωτόκολλα που βρήκαν χρήση σε εφαρμογές έξυπνου ελέγχου φωτισμού και οικιακών αυτοματισμών. Η επιτυχία του Z-Wave στη φάση γέννησης κιόλας αυτών των νέων πεδίων εφαρμογών, προσέλκυσε το ενδιαφέρον μεγάλων κατασκευαστών, όπως των Ingersoll-Rand, Panasonic και Cisco, και οδήγησε μεταξύ άλλων στην ίδρυση της Z-Wave Alliance, μίας κοινοπραξίας με σκοπό την περαιτέρω ανάπτυξη και προώθησή του (αντίστοιχης του Bluetooth SIG). Αποτελεί μία πολλά υποσχόμενη τεχνολογία, η οποία κερδίζει συνεχώς έδαφος στην αγορά, εμπλουτίζεται με νέες δυνατότητες (όπως υποστήριξη IPv6) και επεκτείνεται στον τομέα των WSN.

Στον παρακάτω πίνακα δίνουμε μία συνοπτική σύγκριση των κυριότερων τεχνικών χαρακτηριστικών τους:

	Bluetooth Low Energy	*Κλασικό Bluetooth*	*ZigBee*	*ANT/ ANT+*	*Z-Wave*
Φέρουσα συχνότητα (MHz)	2400	2400	868 (Z1), 915 (Z2), 2400 (Z3)	2400	868 (Z1), 908 (Z2), 2400 (Z3)
Ρυθμός μετάδοσης (Kbps)	1000 (v4.2), 2000 (v5.0)	721 (v1.x) 2000 (v2.x), 24000 (v3.x)	20, 40, 250	1000	9.6/40 (Z1/Z2), 200 (Z3)
Ισχύς μετάδοσης (dBm)	20, 4, 0	-20 έως -10	-32 έως 0	-20 έως 0	-20 έως 0
Εμβέλεια (m)	10-100	10	10-100	100	100
Μήκος πακέτου (bytes)	47 (v4.1), 265 (v4.2)	358	127	24	64
Τοπολογία δικτύου	Star/Mesh	P2P/ Star/ Scatternet	Star/Mesh	P2P/ Star/ Mesh	Mesh
Υποστήριξη IPv6	Όχι (v4.1), Ναι (v4.2)	Όχι	Ναι	Όχι	Ναι

Πίνακας 2.9: Σύγκριση πρωτοκόλλων WPAN

2.6 Προσέγγιση στην παρούσα εργασία

Για το σύστημά μας κάνουμε τις εξής σχεδιαστικές επιλογές:

- ***Πρωτόκολλο WPAN***

Επιλέγουμε το Bluetooth Low Energy για μια σειρά από λόγους. Από την πλευρά των τελικού χρήστη, το BLE – αν και μη συμβατό με το IEEE 802.15.4 – υποστηρίζεται από το σύνολο σχεδόν των κινητών συσκευών που βρίσκονται σήμερα σε χρήση και κυκλοφορία στην αγορά. Ταυτόχρονα, παρά την περιορισμένη εμβέλεια μετάδοσης, προσφέρει τη χαμηλότερη κατανάλωση ισχύος για το πεδίο εφαρμογών που μας απασχολεί.

Από την πλευρά του σχεδιαστή του συστήματος, για το BLE παρέχονται ελεγμένες και δοκιμασμένες βιβλιοθήκες λογισμικού από το σύνολο των σύγχρονων λειτουργικών συστημάτων. Επιπλέον, στην αγορά διατίθεται πληθώρα οικονομικών ICs που υλοποιούν τη στοίβα πρωτοκόλλων του BLE και υποστηρίζονται από αναπτυξιακές πλακέτες και εργαλεία ταχείας ανάπτυξης λογισμικού.

- ***Τοπολογία δικτύου***

Το δίκτυο οργανώνεται σε τοπολογία αστέρα, με την κινητή συσκευή να συνδέεται με τους κόμβους, επέχοντας ρόλο πύλης. Αυτή είναι άλλωστε η μοναδική τοπολογία που υποστηρίζεται από εκδόσεις του Bluetooth παλαιότερες της 5.x.

- ***Τροφοδοσία αισθητήρων***

Κάτω από συνθήκες τις οποίες θα περιγράψουμε αναλυτικά στην υποενότητα 5.6.1, μία μπαταρία CR2032 (coin cell) αρκεί για διάρκεια ζωής του sensor node τουλάχιστον ίση με μερικούς μήνες.

3 Bluetooth Low Energy

3.1 Εισαγωγή

Η ενσωμάτωση του Bluetooth Low Energy (Bluetooth LE, ή απλούστερα BLE) σε πληθώρα συσκευών, τόσο για τελικούς καταναλωτές όσο και για βιομηχανικές εφαρμογές, αποτελεί τον καρπό πολύχρονης προσπάθειας από τα μέλη του Bluetooth *Special Interest Group* (SIG) για την ανάπτυξη ενός πρωτοκόλλου WPAN με χαμηλή κατανάλωση ενέργειας, για επικοινωνία σε μικρές αποστάσεις. Παρότι το BLE εντάσσεται στα WPAN και ενσωματώνει δυνατότητες IoT, δεν αποτελεί μέρος του IEEE 802.15.4, αλλά αυτοτελές τμήμα της γνωστής οικογένειας τεχνολογιών Bluetooth.

3.1.1 Ιστορικά στοιχεία: Bluetooth BR/EDR και LE

Ο σκοπός της επικοινωνίας σε μικρές αποστάσεις είναι το στοιχείο που συνδέει την πορεία ανάπτυξης του BLE με το κλασικό Bluetooth, το πρωτόκολλο που αναπτύχθηκε το 1989 από τους Nils Rydbeck, Johan Ullman, Tord Wingren, Jaap Haartsen και Sven Mattisson για λογαριασμό της Ericsson, και ακόμα βρίσκεται σε χρήση στις μέρες μας σε πληθώρα συσκευών, όπως handsfree, ασύρματα headsets κλπ. Οι δημιουργοί του κλασικού Bluetooth το οραματίστηκαν ως μια ασύρματη τεχνολογία που θα ήταν σε θέση για μικρές αποστάσεις να υποκαταστήσει τη χρήση καλωδίων, ή διαφορετικά, ως μια ασύρματη εναλλακτική της σειριακής θύρας RS-232.

Το κλασικό Bluetooth λειτουργεί στις συχνότητες 2402-2480, ή 2400-2483.5 MHz της ζώνης ISM (industrial, scientific and medical). Για την αποφυγή παρεμβολών από είδωλα που προέρχονται από εκπομπές στην ίδια ζώνη, κάνει χρήση της τεχνικής διασποράς φάσματος *frequency-hopping spread spectrum* (FHSS), με περιοδικές μεταπηδήσεις (*hops*) της φέρουσας συχνότητας. Η επιλογή συχνότητας ισοδυναμεί με την επιλογή ενός από τα 79 κανάλια μετάδοσης στα οποία το κλασικό Bluetooth διαμερίζει τη ζώνη συχνοτήτων του. Το εύρος ζώνης καθενός από αυτά, όπως και η απόσταση μεταξύ δύο διαδοχικών καναλιών, είναι 1 MHz.

Στην πρώτη του φάση, το κλασικό Bluetooth χρησιμοποιούσε διαμόρφωση GFSK (*Gaussian Frequency-Shift Keying*), υποστηρίζοντας στιγμαίους ρυθμούς μετάδοσης έως 1 Mbps. Με την εισαγωγή του Bluetooth 2.0, οι συμβατές συσκευές είχαν πλέον τη δυνατότητα χρήσης των τεχνικών διαμόρφωσης π/4-DQPSK (*Differential Quadrature Phase-Shift Keying*) και 8-DPSK (Differential Phase-Shift Keying), αυξάνοντας το μέγιστο ρυθμό μετάδοσης στα 2 και 3 Mbps αντίστοιχα. Η

χρήση της παλιάς τεχνικής διαμόρφωσης έμεινε γνωστή ως λειτουργία BR (*Basic Rate Mode*), ενώ της νέας τεχνικής ως EDR (*Extended Data Rate Mode*). Σήμερα είναι δυνατή η χρήση και των δύο λειτουργιών από συσκευές που υποστηρίζουν το κλασικό Bluetooth. Γι' αυτόν ακριβώς το λόγο, το κλασικό Bluetooth είναι γνωστό στις μέρες μας και ως Bluetooth BR/EDR.

Όπως θα αναλύσουμε στο επόμενο κεφάλαιο, το Bluetooth Low Energy διατηρεί πολλά κοινά στοιχεία στο φυσικό στρώμα με το BR/EDR, με κάποιες όμως σημαντικές διαφορές που το καθιστούν μη συμβατό προς τα πίσω (backwards-compatible) με το τελευταίο. Έγινε μέλος της οικογένειας του Bluetooth με την εισαγωγή του στην έκδοση 4.0 του Bluetooth Core Specification το 2010, η ανάπτυξή του όμως ξεκίνησε ανεξάρτητα πολύ πριν την υιοθέτησή του από το Bluetooth SIG.

Η πρώτη εκδοχή του πρωτοκόλλου που είναι σήμερα γνωστό ως BLE, δημοσιεύτηκε το 2004 με το όνομα *Bluetooth Low End Extension*. Ήταν αποτέλεσμα τρίχρονης ερευνητικής εργασίας μηχανικών της Nokia για την ανάπτυξη ενός νέου πρωτοκόλλου, βασισμένου στην ήδη υπάρχουσα προδιαγραφή του Bluetooth και κατάλληλου για νέα σενάρια χρήσης που δεν είχαν προβλεφθεί από αυτήν. Ήταν από τότε εμφανές πως η συμβατότητα προς τα πίσω του νέου πρωτοκόλλου ήταν ανέφικτη, παρ' όλα αυτά δόθηκε έμφαση στην ελαχιστοποίηση της απόκλισής του από το κλασικό Bluetooth.

Η ανάπτυξη του νέου πρωτοκόλλου συνέχισε την πορεία της υπό την ονομασία *Wibree*, αυτήν τη φορά σε συνεργασία με την Logitech, την STMicroelectronics, καθώς και το ευρωπαϊκό ερευνητικό πρόγραμμα MIMOSA ("*Making Innovation in MObility and Sustainable Actions*"). Τελικά, το 2007 το Wibree υιοθετήθηκε από το Bluetooth SIG, με στόχο να συμπεριληφθεί σε κάποιο μελλοντικό Bluetooth Specification, όπως και έγινε. Παρακάτω συνοψίζουμε κάποιους από τους σημαντικότερους ιστορικούς σταθμούς της οικογένειας τεχνολογιών του Bluetooth:

1989 - Ανάπτυξη της αρχικής έκδοσης του κλασικού Bluetooth στις εγκαταστάσεις της Ericsson, στο Lund της Σουηδίας.
1998 - Ιδρύεται το Bluetooth SIG από τις Ericsson, Nokia, Intel, IBM και Toshiba.
1999 - Το Bluetooth προτυποποιείται με τη δημοσίευση της πρώτης έκδοσης του Bluetooth Core Specification (1.0).
2000 - Κυκλοφορεί το πρώτο κινητό τηλέφωνο με υποστήριξη Bluetooth. Ταυτόχρονα, κάνουν την εμφάνισή τους κάρτες επέκτασης και USB dongles για προσωπικούς υπολογιστές.
2001 - Η υποστήριξη Bluetooth επεκτείνεται σε περιφερειακά υπολογιστών όπως πληκτρολόγια, ποντίκια, εκτυπωτές κλπ.
2003 - Δημοσιεύεται η έκδοση 1.2 του Bluetooth Core Specification. Το Bluetooth υποστηρίζεται πλέον από πληθώρα φορητών συσκευών, όπως MP3 players.
2004 - Δημοσιεύεται η έκδοση 2.0 του Bluetooth Core Specification, με την οποία εισάγεται η λειτουργία EDR.
2005 - Το Bluetooth SIG αριθμεί πλέον 4000 μέλη.

2009 - Δημοσιεύεται η έκδοση 3.0 του Bluetooth Core Specification, με την οποία εισάγεται η λειτουργία High-Speed (HS). Η λειτουργία HS επιτυγχάνει ρυθμούς μετάδοσης έως 24 Mbps, με μεταφορά δεδομένων πάνω από ζεύξεις IEEE 802.11 και χρήση του Bluetooth μόνο στη φάση εγκατάστασης της σύνδεσης.
2010 - Το Bluetooth Low Energy κάνει την εμφάνισή του στην έκδοση 4.0 του Bluetooth Core Specification.
2012 - Η υποστήριξη του BLE επεκτείνεται σε φορητές συσκευές αναπαραγωγής μουσικής, tablets κλπ.
2014 - Δημοσιεύεται η έκδοση 4.2 του Bluetooth Core Specification, με την οποία το BLE εμπλουτίζεται με δυνατότητες συνδεσιμότητας IPv6 και ουσιαστικά εντάσσεται στις τεχνολογίες IoT.
2016 - Δημοσιεύεται η έκδοση 5.0 του Bluetooth Core Specification.
2017 - Οι συσκευές BLE αποκτούν τη δυνατότητα χρήσης των νέων Mesh Profiles για την υποστήριξη επικοινωνίας πολλών-προς-πολλούς.

3.1.2 Βασικά χαρακτηριστικά

Όμοια με το κλασικό Bluetooth, οι συσκευές ενός δικτύου BLE οργανώνονται σε στοιχειώδεις *ομάδες* με τοπολογία αστέρα, οι οποίες ονομάζονται *piconets*. Ο ριζικός κόμβος του piconet επέχει ρόλο *οδηγού* (*master*), ενώ οι συνδεδεμένες σε αυτόν συσκευές ενεργούν ως *ακόλουθοι* (*slaves*). Στους ρόλους αυτούς θα αναφερθούμε αναλυτικά στην υποενότητα 3.3.2.

Τα piconets συνενώνονται σε ευρύτερα *δίκτυα* δενδρικής τοπολογίας, τα λεγόμενα *scatternets*. Οι συσκευές του δικτύου μπορούν να αλληλεπιδρούν με τους εξής τρόπους:

- Να εκπέμπουν πακέτα προς ολόκληρο το υπόλοιπο δίκτυο (broadcast), ώστε να κοινοποιούν («διαφημίζουν») την ύπαρξή τους και να επιτρέπουν σε άλλες συσκευές να τις εντοπίσουν. Ο μηχανισμός που προσφέρεται από το BLE για αυτόν το σκοπό είναι τα *πακέτα διαφήμισης (advertising)*.
- Να εντοπίζουν άλλες συσκευές που διαφημίζονται *(device discovery)*. Αυτή η διαδικασία αναζήτησης στο BLE καλείται *σάρωση (scanning)*.
- Να εγκαθιστούν μόνιμες ζεύξεις σημείου-προς-σημείο, συνδεόμενες με συσκευές τις οποίες έχουν εντοπίσει μέσω της διαδικασίας σάρωσης.
- Να επιτρέπουν τον τερματισμό των συνδέσεών τους με πρωτοβουλία οποιουδήποτε από τα συμμετέχοντα μέρη.

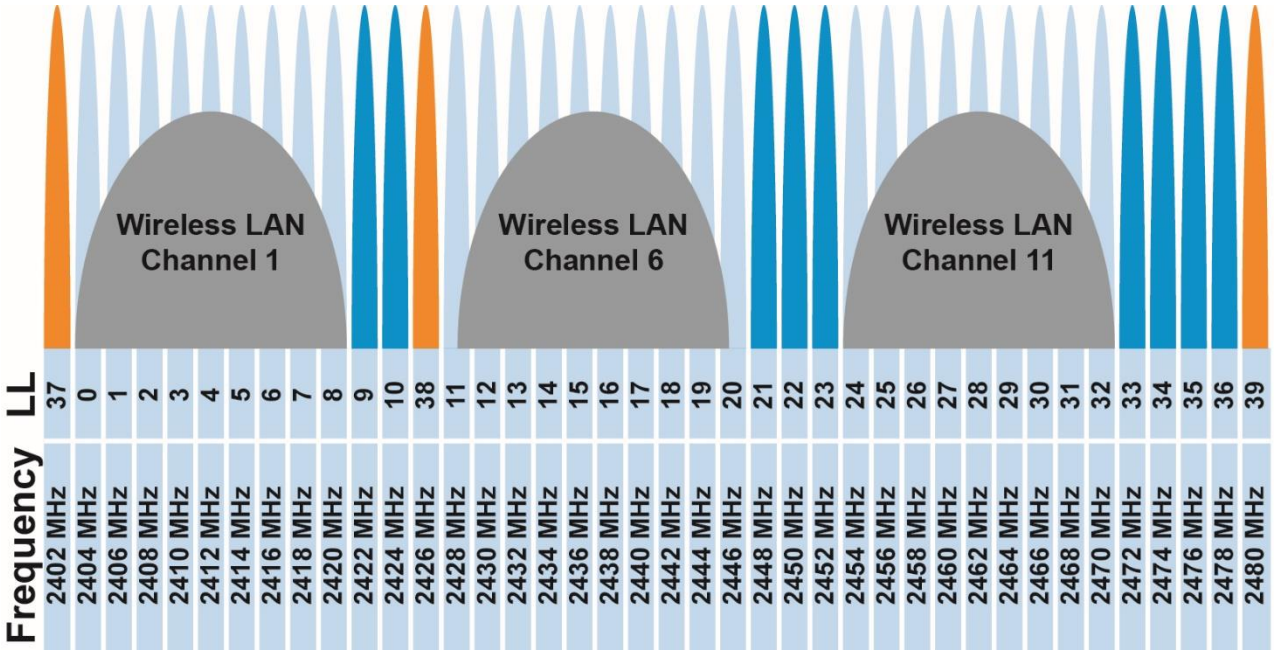
Οι συνδέσεις σε ένα δίκτυο BLE είναι σχεδιασμένες ώστε να εγκαθίστανται γρήγορα, και – σε αντίθεση με το κλασικό Bluetooth – να διαρκούν όσο το δυνατό λιγότερο, είναι κατάλληλες λοιπόν για ταχύρρυθμες και σύντομες συνόδους ανταλλαγής δεδομένων, συνήθως μικρού μεγέθους. Εδώ ακριβώς βρίσκεται και η θεμελιώδης διαφορά του BLE από το BR/EDR, η οποία επιτρέπει τη δραστική μείωση της κατανάλωσης ισχύος με διατήρηση των γνωστών ρυθμών και εμβελειών μετάδοσης. Βέβαια, για τον ίδιο ακριβώς λόγο το κλασικό Bluetooth εξακολουθεί

να κυριαρχεί σε εφαρμογές που απαιτούν αξιόπιστες μακροχρόνιες συνδέσεις και συνεχείς ροές δεδομένων (streaming), πχ για σύνδεση περιφερειακών και συσκευών ήχου.

Μία σύνδεση BLE επιτρέπει την πραγματοποίηση δοσοληψιών εξυπηρετητή-πελάτη στα ανώτερα επίπεδα, με τα δύο άκρα να είναι σε θέση να εναλλάσσουν τους ρόλους τους κάθε φορά. Η λειτουργικότητα που παρέχει ο εξυπηρετητής στον πελάτη εκτίθεται ως ένα σύνολο από *υπηρεσίες (services)*. Όμοια με το κλασικό Bluetooth, μία καλά ορισμένη προδιαγραφή της λειτουργικότητας μιας συσκευής, σε σχέση με τις υπηρεσίες που οφείλει να υλοποιεί, ονομάζεται *προφίλ (profile)*. Για παράδειγμα, το προτυποποιημένο από το Bluetooth SIG *Proximity Profile* απαιτεί την υλοποίηση των εξής services: *Link Loss*, *Immediate Alert* και *TX Power*. Στη δομή των υπηρεσιών BLE και στα απαιτούμενα πρωτόκολλα για το χειρισμό τους αναφερόμαστε στην ενότητα 3.4.

Κατά τον υπόλοιπο χρόνο ζωής της, μία συσκευή BLE μπορεί να εκπέμπει πακέτα advertising σε αραιά χρονικά διαστήματα, παραμένοντας ανενεργή μεταξύ των μεταδόσεων αυτών. Εάν μάλιστα αυτό είναι αρκετό για τις απαιτήσεις μίας εφαρμογής, για παράδειγμα ενός WSN με αισθητήρες χαμηλών ρυθμών δειγματοληψίας, τότε μπορεί η συσκευή, μη αποδεχόμενη συνδέσεις, να περιοριστεί σε αυτή τη λειτουργία, ενσωματώνοντας την απαραίτητη πληροφορία αποκλειστικά στα πακέτα advertising. Τέτοιες συσκευές καλούνται συχνά *φάροι (beacons)*, και επιδεικνύουν εξαιρετικά υψηλή αυτονομία με χρήση απλών μπαταριών.

3.2 Φυσικό στρώμα



Σχήμα 3.1: Ζώνη συχνοτήτων και κανάλια μετάδοσης του BLE

Το Bluetooth Low Energy χρησιμοποιεί την ίδια ακριβώς ζώνη συχνοτήτων με αυτήν του BR/EDR, με τη διαφορά ότι τα κανάλια μετάδοσης έχουν 2 MHz

εύρος ζώνης και απόσταση μεταξύ τους. Η διαμέριση αυτή δίνει 40 κανάλια, όπως φαίνεται στο παραπάνω σχήμα. Η κεντρική συχνότητα κάθε καναλιού υπολογίζεται από τη σχέση:

$$f_c = 2402 + 2k\ (MHz), k = 0 \ldots 39$$

Τα κανάλια 37, 38 και 39, με δείκτες k = 0, 12 και 39 αντίστοιχα, επιλέγονται ως κανάλια *εκπομπής* (ή *διαφήμισης* - *advertising*), καθώς στις συχνότητες που ορίζουν ελαχιστοποιείται η επικάλυψη με τα κανάλια 1, 6 και 11 του Wi-Fi (IEEE 802.11). Τα υπόλοιπα 37 κανάλια είναι διαθέσιμα για γενικού σκοπού μετάδοση δεδομένων. Αν και με τη διαδικασία εκπομπής θα ασχοληθούμε στην επόμενη ενότητα, αυτό που αξίζει να σημειωθεί εδώ είναι ότι δεν είναι αναγκαία η ταυτόχρονη χρήση και των τριών καναλιών διαφήμισης. Η ενεργοποίηση των καναλιών advertising ορίζεται με μία δυαδική τιμή-μάσκα (bitmask) μήκους 3 bit η οποία καλείται *channel map*. Τα τρία κανάλια advertising αντιστοιχίζονται ένα προς ένα στα bits του channel map.

Όμοια με το κλασικό Bluetooth, για την αποφυγή παρεμβολών από γειτονικές μεταδόσεις χρησιμοποιείται η τεχνική μεταπήδησης συχνότητας *adaptive frequency hopping*, με επιλογή ενός από τα 37 κανάλια δεδομένων πριν από κάθε μετάδοση πακέτου. Σε αντίθεση όμως με το BR/EDR, εδώ όλα τα κανάλια χρησιμοποιούν διαμόρφωση GFSK. Το φυσικό στρώμα του BLE που ορίζεται από τις εκδόσεις 4.x του Core Specification προβλέπει μέγιστο ρυθμό μετάδοσης 1 Mbps, και είναι γνωστό ως *LE 1M PHY*. Η έκδοση 5.0 προσθέτει τις εξής επεκτάσεις:

- *LE 2M PHY:* Προσφέρει μέγιστο ρυθμό μετάδοσης 2 Mbps, χωρίς δραστική μείωση της εμβέλειας του δικτύου.

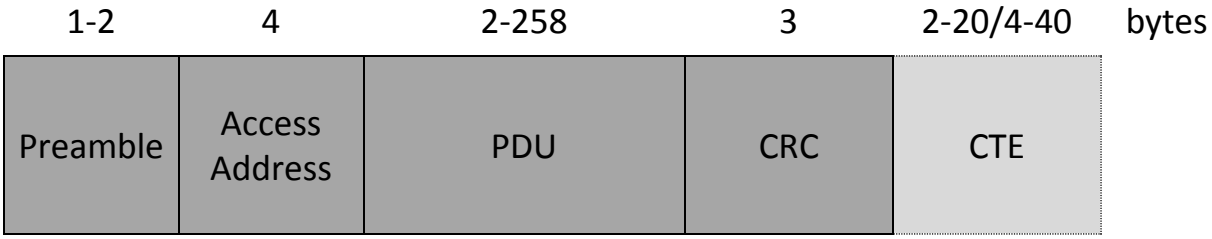
- *LE Coded PHY:* Προσθέτει στο 1M PHY λειτουργία διόρθωσης σφαλμάτων με χρήση κωδικοποίησης FEC. Έτσι, στην ίδια απόσταση είναι δυνατή η μείωση του ρυθμού σφαλμάτων bit (bit error rate, BER). Ένα bit πληροφορίας κωδικοποιείται σε σύμβολα μήκους S = 2 ή 8 bit. Αν δεχθούμε πως η εμβέλεια του δικτύου ισοδυναμεί με τη μέγιστη απόσταση για την οποία ο BER διατηρείται σε τιμές μικρότερες του 0.1% , τότε με χρήση του Coded PHY η εμβέλεια πρακτικά πολλαπλασιάζεται επί S, ενώ ο πραγματικός ρυθμός μετάδοσης διαιρείται με τον ίδιο παράγοντα.

- *Ισόχρονες μεταδόσεις*: Επιτρέπει τη χρήση των καναλιών διαφήμισης ή δεδομένων ως *ισόχρονων* (*isochronous*) καναλιών για τη μετάδοση, με ή χωρίς σύνδεση αντίστοιχα, συνεχών ροών δεδομένων προς έναν ή περισσότερους παραλήπτες.

3.3 Επίπεδο ζεύξης δεδομένων

Το επίπεδο ζεύξης δεδομένων (link layer) ορίζει τους τρόπους με τους οποίους μπορούν οι συσκευές στο δίκτυο να χρησιμοποιήσουν το διαθέσιμο εύρος ζώνης για να ανταλλάξουν δεδομένα με τη μορφή πακέτων. Οι ανταλλαγές αυτές γίνονται περιοδικά, σε συγκεκριμένα χρονικά διαστήματα (*intervals*), πολλαπλάσια της στοιχειώδους *χρονοσχισμής* BLE (*BLE timeslot*, 625 μs), και καλούνται *συμβάντα* BLE (*BLE events*). Μια συσκευή μπορεί να επικοινωνήσει στο δίκτυο με δύο τρόπους: την *εκπομπή* (*advertising*) και τη *σύνδεση* (*connection*). Τότε, τα συμβάντα αυτά καλούνται αντίστοιχα *advertising events* ή *connection events*.

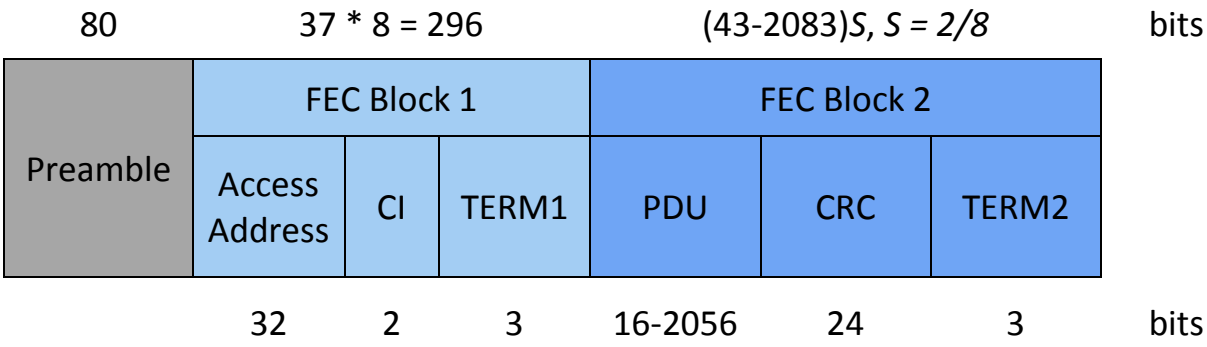
Η μετάδοση του πακέτου γίνεται από το λιγότερο στο περισσότερο σημαντικό byte (*little endian*), και από το λιγότερο στο περισσότερο σημαντικό bit εντός του byte. Για μετάδοση πάνω από τα 1/2M PHY, το πακέτο συντίθεται από τα εξής πεδία:



Σχήμα 3.2: Γενική μορφή πακέτου BLE (1/2M PHY)

- ***Preamble:*** Δυαδική ακολουθία έναρξης από εναλλασσόμενα 0 και 1, μήκους 1 byte στο 1M ή 2 bytes στο 2M.
- ***Access Address:*** Διεύθυνση συσκευής κατά την εκπομπή ή τη σύνδεση.
- ***PDU:*** Το ωφέλιμο φορτίο (payload) του πακέτου. Μπορεί να είναι πακέτο εκπομπής ή πακέτο δεδομένων, όπως αυτά ορίζονται στις επόμενες υποενότητες.
- ***CRC:*** Άθροισμα ελέγχου με 24-bit έλεγχο κυκλικού πλεονασμού (CRC) για εντοπισμό σφαλμάτων.
- ***Constant Tone Extension (CTE):*** Προαιρετικό πεδίο, διαθέσιμο μόνο στο Bluetooth 5.2 και σε νεότερες εκδόσεις. Ισοδυναμεί με μία χρονοσχισμή μετάδοσης ενός απλού τόνου στη φέρουσα συχνότητα, διάρκειας από 16 έως 160 μs. Ο υπολογισμός των ορθογωνικών συνιστωσών I και Q από το περιεχόμενο του πεδίου επιτρέπει την εκτίμηση της κατεύθυνσης μετάδοσης, με προσέγγιση των γωνιών άφιξης (*angle of arrival*, AoA) και αναχώρησης (*angle of departure*, AoD) του σήματος.

Για μετάδοση πάνω από το Coded PHY, η σύνθεση του πακέτου είναι η εξής:



Σχήμα 3.3: Γενική μορφή πακέτου BLE (Coded PHY)

- ***Preamble:*** Αποτελείται από 10 επαναλήψεις της ακολουθίας έναρξης 00111100.
- ***FEC Block 1:*** Σύνθετο πεδίο, κωδικοποιούμενο με 8 bits/σύμβολο και αποτελούμενο από τα εξής υποπεδία:
  - ***Access Address:*** Ορίζεται όμοια με το αντίστοιχο πεδίο του πακέτου 1/2M.
  - ***Coding Indicator (CI):*** Ορίζει την κωδικοποίηση του *FEC Block 2*.
  - ***TERM1:*** Σύμβολο λήξης του *FEC Block 1*.
- ***FEC Block 2:*** Σύνθετο πεδίο, κωδικοποιούμενο με *S* = 2 ή 8 bits/σύμβολο και αποτελούμενο από τα εξής υποπεδία:
  - ***PDU:*** Ορίζεται όμοια με το αντίστοιχο πεδίο του πακέτου 1/2M.
  - ***CRC:*** Ορίζεται όμοια με το αντίστοιχο πεδίο του πακέτου 1/2M.
  - ***TERM2:*** Σύμβολο λήξης του *FEC Block 2*.

3.3.1 Διευθυνσιοδότηση

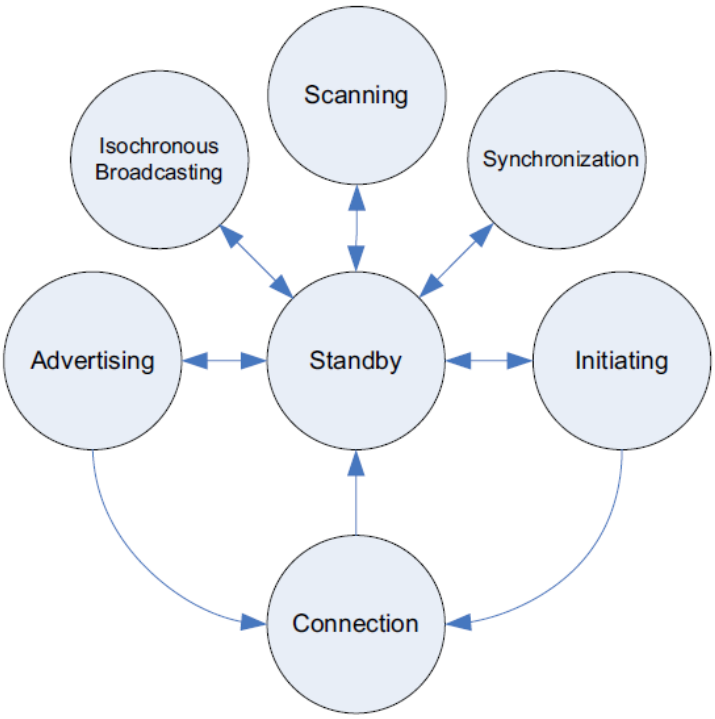
Η *διεύθυνση* μίας συσκευής BLE, μήκους 48 bit, αποτελεί το αναγνωριστικό με το οποίο άλλες συσκευές στο ίδιο δίκτυο μπορούν να αναφέρονται σε αυτήν, και οφείλει να είναι μοναδική εντός αυτής της εμβέλειας. Με βάση του τρόπο απόδοσής τους, οι διευθύνσεις BLE διακρίνονται στους εξής *τύπους*:

- ***Δημόσια (public):*** Χρησιμοποιείται η 24-bit σταθερή διεύθυνση υλικού που έχει προεπιλεγεί από τον κατασκευαστή, με τα υπόλοιπα 24 (περισσότερο σημαντικά) bits να καταλαμβάνονται από το αναγνωριστικό IEEE του τελευταίου.
- ***Τυχαία (random):*** Η συσκευή μεταβάλλει τη διευθύνσή της, χρησιμοποιώντας τυχαίες τιμές. Η διεύθυνση μπορεί να είναι:
  - ***Στατική (static):*** Αποδίδεται νέα διεύθυνση με κάθε νέα επανεκκίνηση της συσκευής.
  - ***Ιδιωτική (private):*** Προς διαφύλαξη της ιδιωτικότητάς της, η συσκευή μεταβάλλει περιοδικά τη διεύθυνσή της όσο παραμένει ενεργή στο δίκτυο. Εάν είναι επιθυμητή η δυνατότητα εντοπισμού της συσκευής από άλλες *έμπιστες* συσκευές, η διεύθυνση μπορεί να επανυπολογίζεται από αυτές με χρήση ενός κλειδιού κρυπτογράφησης, του IRK (*identity resolving key*),

την παραγωγή του οποίου πραγματευόμαστε στην υποενότητα 3.4.4. Μια τέτοια διεύθυνση καλείται *επανυπολογίσιμη* (*resolvable*). *Μη επανυπολογίσιμες* (*non-resolvable*) διευθύνσεις μπορούν να γίνουν αντιληπτές μόνον εφόσον αποσταλούν χωρίς κρυπτογράφηση, κάτι που από τη σκοπιά της ασφάλειας είναι θεμιτό μόνο για *στιγμιαία* επικοινωνία με *ακριβώς έναν* έμπιστο παραλήπτη (*unicast*), πχ για την αυτόματη επανεγκατάσταση μίας απολεσθείσας σύνδεσης.

3.3.2 Ρόλοι και καταστάσεις

Ανά πάσα χρονική στιγμή, το link layer μιας συσκευής μπορεί να βρίσκεται σε μία από τις *καταστάσεις* που απεικονίζονται στην παρακάτω μηχανή καταστάσεων (*finite state machine*, FSM):



Σχήμα 3.4: Η FSM του link layer

Η FSM παρουσιάζει επιπλέον τις δυνατές μεταβάσεις μεταξύ των καταστάσεων αυτών. Μία συσκευή σε κατάσταση σύνδεσης έχει τη δυνατότητα να ενεργεί ως master του piconet στο οποίο ανήκει, ή ως slave. Ο ρόλος του master σε ένα piconet μπορεί βέβαια να μεταβιβάζεται από τη μία συσκευή στην άλλη. Εάν η υλοποίηση του link layer επιτρέπει την παράλληλη εκτέλεση πολλών FSM, καθεμία από αυτές ενεργεί σαν ένα αυτόνομο link layer, διατηρώντας τοπικά τη δική της κατάσταση. Έτσι, σε κάθε αντίγραφο μία συσκευή μπορεί να ενεργεί με διαφορετικό ρόλο. Παρακάτω αναλύουμε κάθε κατάσταση ξεχωριστά:

- **Scanning:** Η συσκευή διατηρεί ενεργό τον πομποδέκτη της, «ακούγουντας» για εκπομπές συσκευών που βρίσκονται στην κατάσταση *Advertising*. Μπορεί να αλλάξει κατάσταση μόνο περνώντας σε *Standby*.

- **Standby:** Η συσκευή απενεργοποιεί τον πομποδέκτη της και παραμένει σε κατάσταση εξοικονόμησης ενέργειας.

- **Advertising:** Η συσκευή εκπέμπει πακέτα στο δίκτυο. Τα πακέτα αυτά ονομάζονται πακέτα εκπομπής (advertising packets).

- **Initiating:** Η συσκευή έχει αποστείλει αίτημα σύνδεσης προς μία συσκευή που βρίσκεται σε κατάσταση *Advertising*, με σκοπό την εγκατάσταση μόνιμης ζεύξης μεταξύ τους.

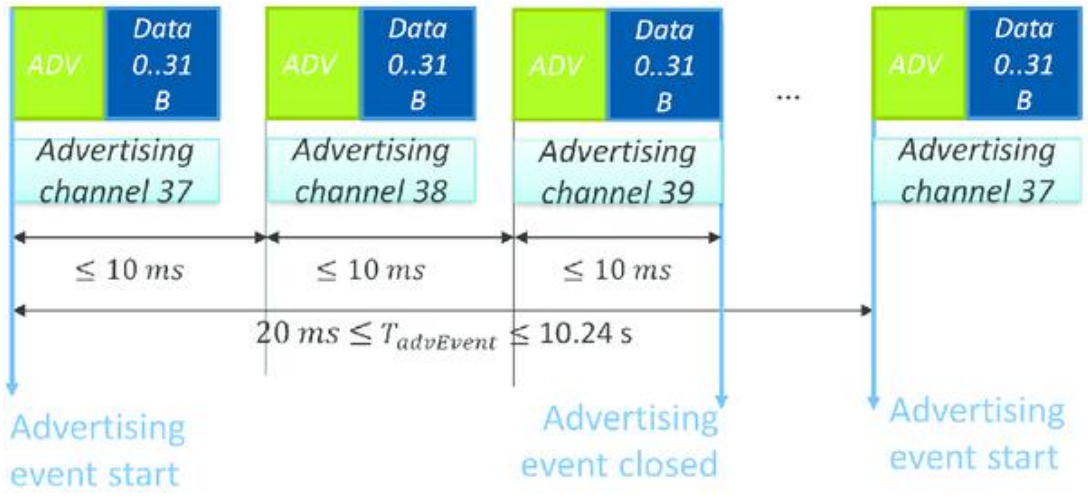
- **Connection:** Σε αυτήν την κατάσταση βρίσκεται ένα ζεύγος συσκευών μετά την εγκατάσταση σύνδεσης μεταξύ τους. Σύνδεση μπορεί να δημιουργηθεί μόνον μεταξύ μιας συσκευής σε κατάσταση *Initiating* και μιας άλλης σε κατάσταση *Advertising*. Όσο διαρκεί η σύνδεση, η πρώτη συσκευή ενεργεί ως master, ενώ η δεύτερη ως slave.

- **Isochronous Broadcasting:** Η συσκευή εκπέμπει πακέτα δεδομένων σε σταθερά χρονικά διαστήματα, πάνω από ένα ισόχρονο κανάλι (μόνο για Bluetooth 5.0 ή νεότερο).

- **Synchronization:** Η συσκευή «ακούει» την περιοδική εκπομπή πακέτων μίας άλλης συσκευής, συγχρονίζοντας το δέκτη της (μόνο για Bluetooth 5.0 ή νεότερο).

3.3.3 Εκπομπή (advertising)

Πρόκειται για μονόδρομη μετάδοση πακέτων διαφήμισης, κατά κανόνα προς το σύνολο του δικτύου. Στην κατάσταση αυτή, η συσκευή μπορεί να στείλει δεδομένα και να δεχθεί αιτήματα, όχι όμως να λάβει δεδομένα. Η μετάδοση των πακέτων λαμβάνει χώρα σε *συμβάντα διαφήμισης (advertising events)* πάνω από τα *κανάλια διαφήμισης (advertising channels)* 37, 38 και 39 του 1M PHY, τα οποία είναι γνωστά ως *πρωτεύοντα (primary)* κανάλια. Η χρονική απόσταση μεταξύ διαδοχικών συμβάντων καλείται *διάστημα διαφήμισης* (*advertising interval*, 20 ms – 10.24 s). Κατά τη διάρκεια ενός συμβάντος διαφήμισης εκπέμπεται ένα αντίγραφο του πακέτου ανά κανάλι, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.5: Παράδειγμα advertising event με χρήση τριών καναλιών

Η έκδοση 5.0 του Bluetooth επιτρέπει επιπλέον τη χρήση ενός ή περισσότερων από τα ελεύθερα κανάλια δεδομένων του 2M PHY ως *δευτερευόντων (secondary)* καναλιών διαφήμισης. Η αξιοποίησή τους αυξάνει το μέγιστο μήκος του πακέτου εκπομπής από τα 37 bytes του Bluetooth 4.x στα 255 bytes, και είναι γνωστή ως *εκπομπή με επέκταση πακέτου (extended advertising)*. Αποστέλλοντας μάλιστα μία *ακολουθία* τέτοιων πακέτων κατά τη διάρκεια *ενός* advertising event *(αλυσιδωτή εκπομπή - chained advertising)*, είναι δυνατή η ταχεία μετάδοση μπλοκ δεδομένων συνολικού μήκους έως 1650 bytes.

3.3.3.1 Τρόποι εκπομπής

Μία συσκευή μπορεί να πραγματοποιήσει εκπομπή με τους παρακάτω τρόπους:

- **Γενικού σκοπού εκπομπή (General advertising):** Η συσκευή μεταδίδει τα advertising data στο σύνολο του δικτύου, και επιπλέον δέχεται αιτήματα σύνδεσης ή αιτήματα αναζήτησης. Πρόκειται συνεπώς για *ακατεύθυντη* (*undirected*), *συνδέσιμη* (*connectable*) και *αναζητήσιμη* (*scannable*) εκπομπή.

- **Απευθείας εκπομπή (Directed advertising):** Πρόκειται για *κατευθυνόμενη* (*directed*) και *συνδέσιμη* (*connectable*) εκπομπή. Τα πακέτα αποστέλλονται με ταχύ ρυθμό (κάθε 3.75 ms) προς *μία* συγκεκριμένη συσκευή προορισμού, με σκοπό τη γρήγορη εγκατάσταση σύνδεσης. Γι' αυτόν ακριβώς το λόγο τα πακέτα αυτά δεν περιέχουν άλλο φορτίο, παρά μόνο τις διευθύνσεις αποστολέα και παραλήπτη.

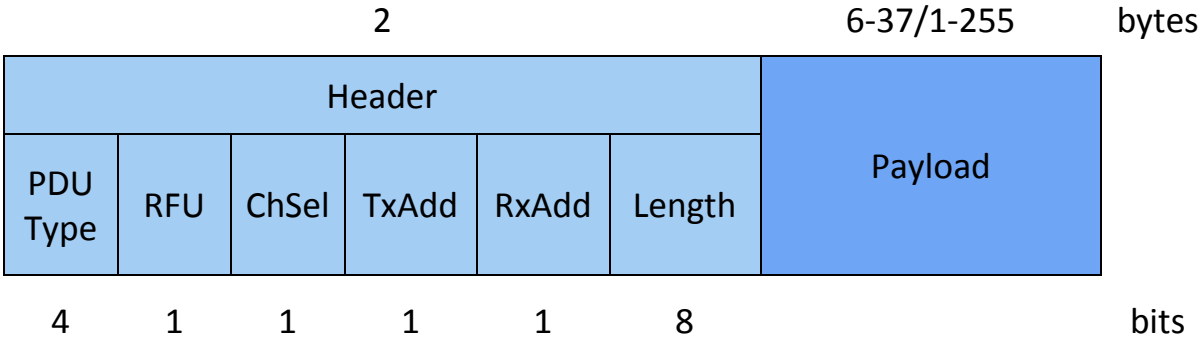
- **Απλή εκπομπή (Nonconnectable advertising):** Η συσκευή μεταδίδει τα advertising data στο δίκτυο, δε δέχεται όμως αιτήματα σύνδεσης. Πρόκειται λοιπόν για *ακατεύθυντη* (*undirected*), *μη συνδέσιμη* (*non-connectable*) και *μη αναζητήσιμη* (*non-scannable*) εκπομπή.

- **Εκπομπή με εύρεση (Discoverable advertising):** Πρόκειται για υποπερίπτωση της απλής εκπομπής, με τη διαφορά ότι η συσκευή δέχεται αιτήματα αναζήτησης απαντώντας με κατάλληλα πακέτα, τα οποία περιέχουν δεδομένα συμπληρωματικά των advertising data. Πρόκειται συνεπώς για *ακατεύθυντη* (*undirected*), *μη συνδέσιμη* (*connectable*) και *αναζητήσιμη* (*scannable*) εκπομπή.

- **Περιοδική εκπομπή (Periodic advertising):** Χρησιμοποιεί τα παρεχόμενα από το Bluetooth 5.0 δευτερεύοντα κανάλια για πολλαπλή αποστολή (*multicast*). Η συσκευή αποστέλλει πακέτα σε μία ομάδα (*group*) ενδιαφερόμενων παραληπτών (συνδρομητών), ακολουθώντας το πρότυπο *δημοσίευσης-συνδρομής* (*publish-subscribe*). Με τους ενδιαφερόμενους έχει συμφωνηθεί η περίοδος ενημέρωσης και αποστολής των δεδομένων. Κατά συνέπεια, οι συσκευές μπορούν να αφυπνίζονται στις κατάλληλες χρονικές στιγμές και να διατηρούν τους πομποδέκτες τους ενεργούς μόνο για όσο χρόνο απαιτεί η διάρκεια κάθε μετάδοσης, επιτρέποντας περαιτέρω μείωση της κατανάλωσης ισχύος.

3.3.3.2 Δομή πακέτου

Η PDU εκπομπής ενθυλακώνεται στο payload του πακέτου του link layer όπως φαίνεται στο παρακάτω σχήμα, και αναλύεται στα πεδία που παρουσιάζονται στη συνέχεια:



Σχήμα 3.6: Πακέτο εκπομπής

- ***Header:*** Η επικεφαλίδα του πακέτου, αποτελούμενη από τα ακόλουθα υποπεδία:
  - ***PDU Type:*** Το πεδίο PDU Type προσδιορίζει το είδος του πακέτου εκπομπής, άρα και του αντίστοιχου συμβάντος. Δύναται να λάβει μία από τις εξής τιμές:

    • **ADV_IND**: Πακέτο εκπομπής γενικού σκοπού.
    • **ADV_DIRECT_IND**: Πακέτο απευθείας εκπομπής για εγκατάσταση σύνδεσης.
    • **ADV_NONCONN_IND**: Πακέτο εκπομπής για συσκευές που δεν αποδέχονται συνδέσεις.

• **ADV_SCAN_IND**: Πακέτο εκπομπής με εύρεση.
• **ADV_EXT_IND** (μόνο για Bluetooth 5.0 ή νεότερο): Εναρκτήριο πακέτο για εκπομπή με επέκταση πακέτου. Μεταδίδεται πάντα στα πρωτεύοντα κανάλια.
• **AUX_ADV_IND** (μόνο για Bluetooth 5.0 ή νεότερο): Πακέτο εκπομπής με επέκταση πακέτου. Έπεται οπωσδήποτε ενός πακέτου *ADV_EXT_IND*, και χρησιμοποιείται επιπλέον ως εναρκτήριο πακέτο για περιοδική ή αλυσιδωτή εκπομπή.
• **AUX_SYNC_IND** (μόνο για Bluetooth 5.0 ή νεότερο): Πακέτο έναρξης περιοδικής εκπομπής σε δευτερεύον κανάλι (μόνο για Bluetooth 5.0 ή νεότερο). Έπεται οπωσδήποτε ενός πακέτου *AUX_ADV_IND*.
• **AUX_CHAIN_IND** (μόνο για Bluetooth 5.0 ή νεότερο): Πακέτο αλυσιδωτής εκπομπής. Περιέχει ένα τμήμα των δεδομένων εκπομπής.

- ***RFU:*** Δεσμευμένος χώρος για μελλοντική χρήση.
- ***ChSel:*** Για συγκεκριμένους τύπους πακέτου, επιτρέπει τη χρήση ενός εναλλακτικού αλγόριθμου επιλογής των καναλιών μετάδοσης.
- ***TxAdd/RxAdd:*** Για συγκεκριμένους τύπους πακέτου, περιέχει πρόσθετη πληροφορία για τον αποστολέα/παραλήπτη.
- ***Length:*** Το μήκος του ωφέλιμου φορτίου του πακέτου.
- ***Payload:*** Το ωφέλιμο φορτίο του πακέτου advertising.

3.3.3.3 Δεδομένα εκπομπής

Στο ωφέλιμο φορτίο του πακέτου advertising περιέχεται οπωσδήποτε η διεύθυνση της συσκευής. Επιπλέον μπορεί να ενσωματωθεί μία σειρά από εγγραφές *(advertising structures)* της μορφής [*AD Length*, *AD Type*, *AD Data*], η οποία συνιστά το συνόλο των *δεδομένων διαφήμισης (advertising data)*. Το πεδίο *AD Length* (μήκους 1 byte) προσδιορίζει το μήκος της εγγραφής (1-255 bytes για Bluetooth 5.x, 1-31 bytes για Bluetooth 4.x), ενώ το *AD Type* (μήκους 1 byte) περιγράφει τον τύπο των δεδομένων που περιέχονται στο πεδίο *AD Data*.

Οι συνηθέστερα χρησιμοποιούμενοι τύποι δεδομένων διαφήμισης περιλαμβάνουν:

- *Flags AD:* Ένα σύνολο από δυαδικές τιμές αληθείας που ορίζουν την ανευρεσιμότητα της συσκευής όπως ορίζεται στην παράγραφο 3.4.2.2, την υποστήριξη ή όχι του BR/EDR κλπ.
- *Service AD:* Μερική ή πλήρης απαρίθμηση των 16/32/128-bit αναγνωριστικών UUID των υπηρεσιών που υποστηρίζονται από αυτή τη συσκευή. Αναλυτικά στη δομή των υπηρεσιών BLE αναφερόμαστε στην υποενότητα 3.4.5.
- *Local Name:* Το όνομα με το οποίο εμφανίζεται η συσκευή όσο διαφημίζεται, μέγιστου μήκους 29 χαρακτήρων.
- *TX Level:* Η τρέχουσα ισχύς εκπομπής σε dBm.

- *Slave Connection Interval Range:* Το προτιμώμενο εύρος τιμών για το connection interval.
- *Service Solicitation AD:* Ορίζει πως σε περίπτωση σύνδεσης, ο εξυπηρετητής ζητά από την πλευρά του πελάτη την υλοποίηση συγκεκριμένων υπηρεσιών.
- *Service Data AD:* Εκθέτει πρόσθετη πληροφορία σχετικά με μία υπηρεσία.
- *Manufacturer-specific Data AD:* Αποτελείται από το 16-bit αναγνωριστικό του κατασκευαστή (με προεπιλεγμένη τιμή 0xFFFF για απροσδιόριστο κατασκευαστή), και ένα πεδίο γενικού σκοπού για ενσωμάτωση πρόσθετων δεδομένων της εφαρμογής.

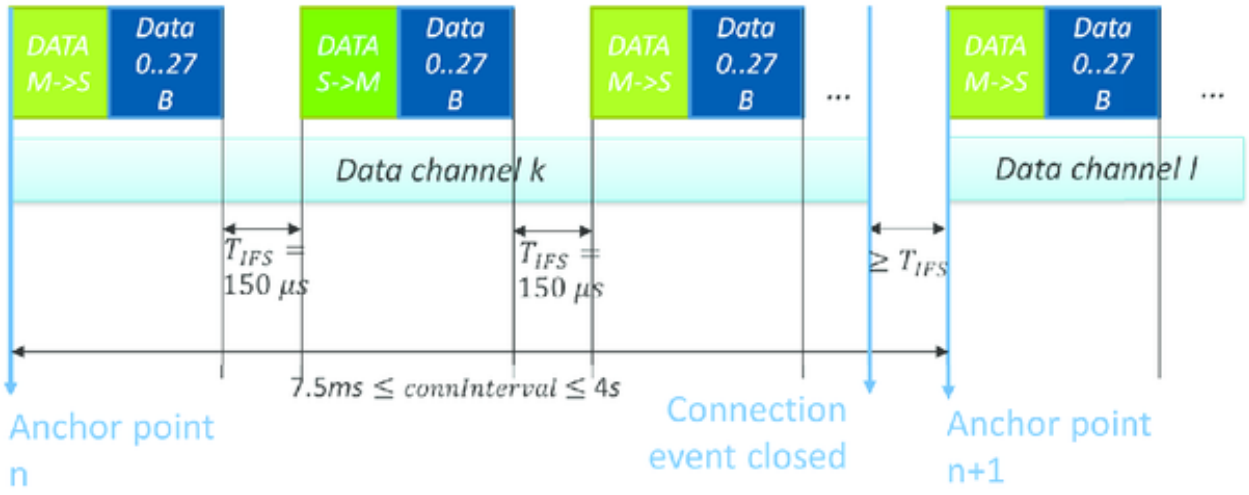
Στις λειτουργίες σύνδεσης (connection) και αναζήτησης (scanning) αναφερόμαστε αναλυτικά στις υποενότητες που ακολουθούν.

3.3.4 Σύνδεση (connection)

Η σύνδεση επιτρέπει την αμφίδρομη μεταφορά δεδομένων, και μπορεί να εγκατασταθεί μόνο μεταξύ ενός scanner και ενός advertiser, με αποστολή αιτήματος σύνδεσης (*CONNECT_IND*) από τον scanner στον advertiser. Τότε ο scanner αναλαμβάνει το ρόλο του *initiator*, και από τη στιγμή της επιτυχούς εγκατάστασης της σύνδεσης λέμε πως ο initiator έχει συνδεθεί με τον advertiser. Ο πρώτος αναλαμβάνει το ρόλο του master, ορίζοντας τις παραμέτρους σύνδεσης (connection parameters), και ο δεύτερος το ρόλο του slave.

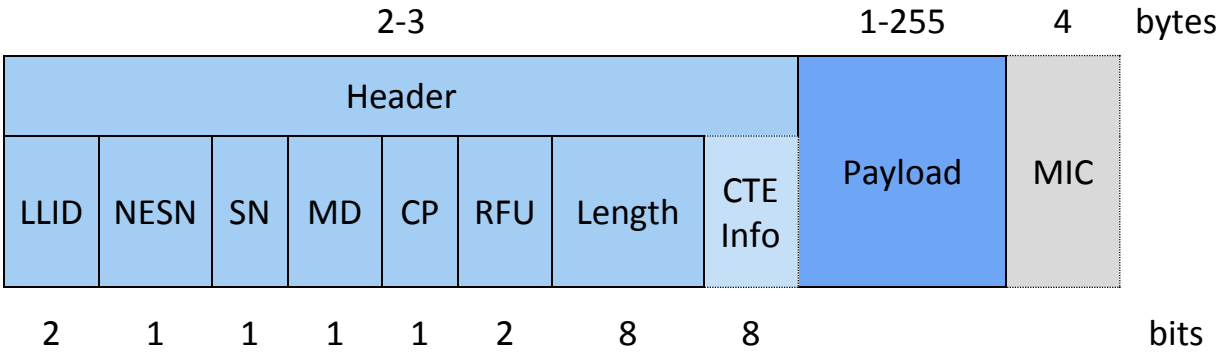
Η σύνδεση μπορεί να τερματιστεί με πρωτοβουλία τόσο του master, όσο και του slave. Όσο όμως διαρκεί η σύνδεση, είναι απαραίτητη η μεταξύ τους περιοδική ανταλλαγή δεδομένων σε χρονικά διαστήματα ίσα με την *περίοδο σύνδεσης* (*connection interval*, 7.5 ms – 4 s). Μεταξύ των παραμέτρων σύνδεσης ορίζεται και ένα *όριο επίβλεψης* (*supervision timeout*), με την πάροδο του οποίου η απουσία ανταλλαγής δεδομένων σηματοδοτεί την απώλεια της σύνδεσης.

Ένα μεμονωμένο connection event δεσμεύει ένα συγκεκριμένο κανάλι δεδομένων, και περιλαμβάνει ένα ή περισσότερα πακέτα. Κάθε μεταδιδόμενο πακέτο απαιτείται να ακολουθείται από ένα κενό πακέτο επιβεβαίωσης από το άλλο άκρο επικοινωνίας. Μεταξύ διαδοχικών πακέτων μεσολαβεί απαραίτητα ένα κενό (inter-frame space, IFS), όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 3.7: Παράδειγμα connection event

Παρακάτω παρουσιάζουμε ένα πακέτο δεδομένων, όπως ενθυλακώνεται σε ένα πλαίσιο του επιπέδου ζεύξης, και στη συνέχεια αναλύουμε τα πεδία τα οποία το απαρτίζουν:



Σχήμα 3.8: Πακέτο δεδομένων

- **Header:** Η επικεφαλίδα του πακέτου, μήκους 16 ή 24 bit, αποτελούμενη από τα εξής υποπεδία:
  - **LLID:** Το αναγνωριστικό της λογικής ζεύξης που μεταδίδεται πάνω από το φυσικό κανάλι δεδομένων, όπως ορίζεται στην υποενότητα 3.4.1.
  - **NESN:** Ο αναμενόμενος αύξων αριθμός του επόμενου πακέτου προς αποστολή/λήψη.
  - **SN:** Ο αύξων αριθμός του τρέχοντος πακέτου.
  - **MD:** Συνθήκη αληθείας για αποστολή περισσότερων πακέτων στο τρέχον συμβάν σύνδεσης.
  - **CP:** Συνθήκη αληθείας για την ενσωμάτωση του πεδίου CTE στο link layer πακέτο.
  - **RFU:** Δεσμευμένος χώρος για μελλοντική χρήση.

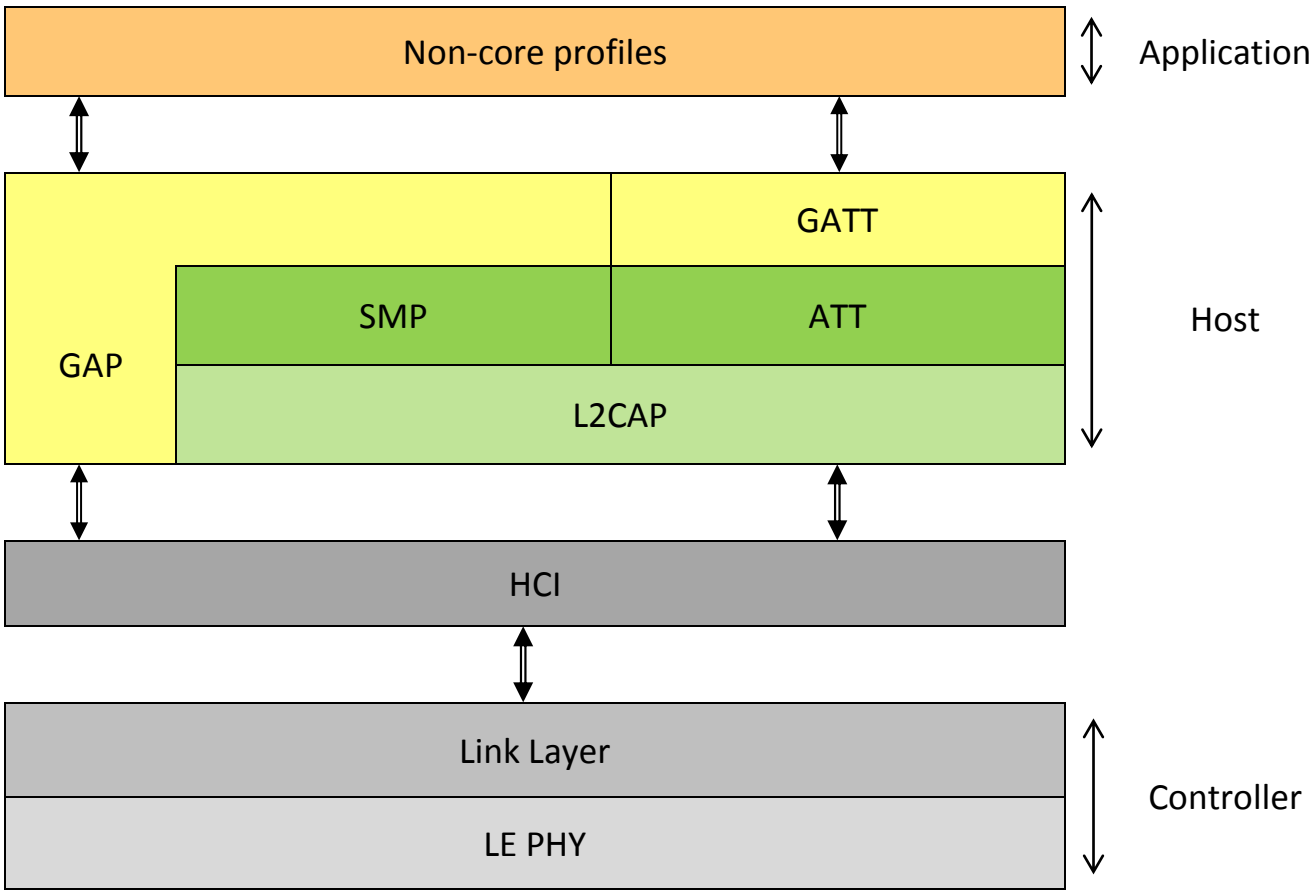
  - **Length:** Το μήκος του πεδίου *Payload*, συμπεριλαμβανομένου του MIC αν αυτό υπάρχει.
  - **CTE Info:** Περιλαμβάνει ρυθμίσεις για την λειτουργία CTE, και ενσωματώνεται στην επικεφαλίδα μόνο αν η *CP* είναι αληθής.
- **Payload:** Το ωφέλιμο φορτίο του πακέτου, προερχόμενο από ανώτερο πρωτόκολλο.
- **MIC:** Άθροισμα ελέγχου από συνάρτηση κατακερματισμού για έλεγχο ακεραιότητας. Ενσωματώνεται στο payload μόνο πάνω από *ασφαλείς* συνδέσεις.

3.4 Στοίβα πρωτοκόλλων-προφίλ

Οι λειτουργίες του φυσικού στρώματος και του επιπέδου ζεύξης δεδομένων επιτελούνται από το τμήμα του συστήματος Bluetooth που καλείται *Controller* (*ελεγκτής*). Ο *Host* (*οικοδεσπότης*) αποτελεί το αμέσως ανώτερο τμήμα του συστήματος και μεσολαβεί ανάμεσα στον Controller και την εφαρμογή χρήστη, αναλαμβάνοντας τη διαχείριση σε λογικό επίπεδο των συνδέσεων με άλλες συσκευές, των δεδομένων που λαμβάνονται και αποστέλλονται μέσω του Controller κλπ. Ο καταμερισμός αρμοδιοτήτων μεταξύ Host και Controller είναι κοινός στο κλασικό Bluetooth και στο BLE, διαφέρει όμως η υλοποίησή τους για κάθε πρωτόκολλο. Έτσι, μια συσκευή συμβατή με Bluetooth 4.0 ή νεότερο μπορεί να υποστηρίζει ένα από τα δύο πρωτόκολλα (*single mode*) ή αμφότερα (*dual mode*), ενσωματώνοντας στη στοίβα Bluetooth της τα απαιτούμενα ζεύγη Host/Controller.

Η διεπαφή μεταξύ του Host και του Controller καλείται *Host/Controller Interface (HCI)* και ορίζει τις εντολές που δέχεται ο Controller από τον Host, καθώς και τον τρόπο προώθησης και επεξεργασίας των πακέτων τόσο από τα κατώτερα στα ανώτερα στρώματα, όσο και αντίστροφα. Αν ο Host και ο Controller φιλοξενούνται σε διαφορετικά συστήματα, για την επικοινωνία μεταξύ τους απαιτείται το HCI να υλοποιείται πάνω σε φυσική διεπαφή, η οποία μπορεί να είναι UART, USB ή SDIO (Secure Digital Input Output).

Για την εξασφάλιση της διαλειτουργικότητας μεταξύ των εφαρμογών σε διαφορετικές συσκευές και την αποφυγή της πολυπλοκότητας της χρήσης των ενδιάμεσων πρωτοκόλλων απευθείας από την εφαρμογή, ο Host περιλαμβάνει οπωσδήποτε μία σειρά από *θεμελιώδη προφίλ (core profiles)*, τα οποία αποτελούν και τη διεπαφή του προς την εφαρμογή. Κάθε συσκευή Bluetooth οφείλει να υλοποιεί το θεμελιώδες προφίλ GAP (*Generic Access Profile*), και επιπρόσθετα το GATT (*Generic Attribute Profile*) εάν υποστηρίζει το BLE. Έτσι, για το BLE ο Host και ο Controller σχηματίζουν την παρακάτω *στοίβα πρωτοκόλλων-προφίλ*:



Σχήμα 3.9: Στοίβα πρωτοκόλλων-προφίλ του BLE

3.4.1 L2CAP

Το *Logical Link Control and Adaptation Protocol (L2CAP)* αναλαμβάνει όλες τις ενέργειες που απαιτούνται για την τροφοδοσία του Controller με SDU των ανώτερων πρωτοκόλλων του Host, και αντίστροφα. Τέτοιες ενέργειες είναι αντίστοιχα η κατάτμηση και η ανασύνθεση των πακέτων, η πολυπλεξία των πρωτοκόλλων, καθώς και η ανταλλαγή μηνυμάτων σηματοδοσίας και ελέγχου.

Κάθε ροή δεδομένων μεταξύ του L2CAP και του Controller αντιμετωπίζεται ως μία *λογική ζεύξη (logical link).* Καθώς το L2CAP μπορεί να επικοινωνεί με ελεγκτές τόσο του κλασικού Bluetooth όσο και του BLE, ορίζει τους τύπους *AMP-U* και *ACL-U* για τις λογικές ζεύξεις του BR/EDR, και τον τύπο *LE-U* για τις λογικές ζεύξεις του BLE. Η διάκριση και η πολυπλεξία των πρωτοκόλλων των παραπάνω επιπέδων επιτυγχάνεται με την υποδιαίρεση των λογικών ζεύξεων σε *λογικά κανάλια (logical channels)* διευσυνσιοδοτούμενα με 16-bit αναγνωριστικά *CID (channel ID).* Για λογικές ζεύξεις LE-U, αυτά είναι:

• *CID= 0x0004:* Attribute Protocol (ATT), που αναλύεται στην παράγραφο.
• *CID = 0x0005:* Σηματοδοσία και ελέγχος του L2CAP.
• *CID = 0x0006:* Security Manager Protocol (SMP), που αναλύεται στην υποενότητα 3.4.4.

Στις υπόλοιπες δυνατές τιμές του CID περιλαμβάνονται αναγνωριστικά λογικών καναλιών των ACL-M, ACL-U, καναλιών διαθέσιμων προς δυναμική δέσμευση, καθώς και δεσμευμένα αναγνωριστικά του Bluetooth SIG, ενώ οι περισσότερες εξ αυτών προβλέπονται για μελλοντική χρήση.

3.4.2 Generic Access Profile (GAP)

Το Generic Access Profile (GAP) προτυποποιεί σε υψηλό επίπεδο τη συμπεριφορά και την αλληλεπίδραση των συσκευών, ορίζοντας σε πρώτο επίπεδο τους *ρόλους (roles)* τους στο δίκτυο, και σε δεύτερο επίπεδο *λειτουργίες (modes)* και *διαδικασίες (procedures)* για τον εντοπισμό συσκευών και υπηρεσιών, καθώς και τη διαχείριση της εγκατάστασης και της ασφάλειας των συνδέσεων.

3.4.2.1 Ρόλοι

Σύμφωνα με το GAP, μια συσκευή μπορεί άνα πάσα στιγμή να έχει έναν από τους εξής *ρόλους*, καθένας από τους οποίους επιβάλλει συγκεκριμένες απαιτήσεις στον Controller και αντιστοιχεί σε συγκεκριμένες επιτρεπτές ακολουθίες καταστάσεων στο link layer:

- **Broadcaster:** Είναι μία συσκευή με μοναδική λειτουργία τη συνεχή εκπομπή δεδομένων σε όλο το δίκτυο, δηλαδή δεν είναι διαθέσιμη για σύνδεση. Αυτό σημαίνει πως στέλνει πακέτα διαφήμισης χωρίς δυνατότητα σύνδεσης (*non-connectable advertising*). Το σύνολο επιτρεπτών καταστάσεων του link layer είναι {*Standby, Advertising, Isochronous Broadcasting*}.

- **Observer:** Είναι μία συσκευή με μοναδική λειτουργία τη συνεχή αναζήτηση για broadcasters στο δίκτυο και τη λήψη των δεδομένων που αυτοί εκπέμπουν, συνεπώς δεν εκκινεί συνδέσεις. Οι επιτρεπτές καταστάσεις του link layer είναι {*Standby, Scanning, Synchronization*}.

- **Peripheral:** Είναι μία συσκευή που εκπέμπει πακέτα διαφήμισης με δυνατότητα σύνδεσης (connectable advertising), συνεπώς μπορεί να συνδεθεί σε κάποιο central. Μετά την εγκατάσταση της σύνδεσης, το peripheral ενεργεί ως slave του central. Οι επιτρεπτές καταστάσεις του link layer είναι {*Standby, Advertising, Connection*}.

- **Central:** Είναι μία συσκευή με αρμοδιότητα την σάρωση του δικτύου για peripherals και την εκκίνηση συνδέσεων προς αυτά. Μετά την εγκατάσταση μίας σύνδεσης, το central ενεργεί ως master του peripheral. Το link layer μπορεί να βρίσκεται στις καταστάσεις {*Standby, Scanning, Initiating, Connection*}.

Από τη συσχέτιση των παραπάνω ρόλων προκύπτουν τα δύο κυρίαρχα σενάρια χρήσης συσκευών εντός ενός δικτύου BLE:

- *Σύνδεση central με peripherals*: Ένα central μπορεί να συνδεθεί με περισσότερα από ένα peripherals, το αντίστροφο όμως δεν ισχύει. Εφόσον το central ενεργεί ως master, ορίζει εκείνο τις παραμέτρους της σύνδεσης. Στη συνέχεια, το central και το peripheral μπορούν να επικοινωνούν αμφίδρομα.

- *Συνεχής λήψη δεδομένων από beacons:* Ένας observer «ακούει» τις εκπομπές ενός ή περισσότερων broadcasters, δίχως να είναι δυνατή η σύνδεση με οποιονδήποτε από αυτούς.

Αξίζει να σημειώσουμε πως ο στιγμιαίος ρόλος της συσκευής αναφέρεται κάθε φορά σε *συγκεκριμένη* FSM του link layer. Αυτό σημαίνει πως αν μία συσκευή υποστηρίζει περισσότερες από μία FSM, μπορεί να ενεργεί ταυτόχρονα με περισσότερους από έναν ρόλους στο GAP.

3.4.2.2 Εντοπισμός συσκευών

Μία συσκευή που διαφημίζεται μπορεί να είναι ορατή σε μία ή περισσότερες συσκευές, όχι απαραίτητα όμως και στο σύνολο του δικτύου. Ειδικά αν πρόκειται για Broadcaster, αναμένεται να είναι ορατή *μόνο* σε συσκευές τύπου Observer. Σε αυτή την περίπτωση, η συσκευή χρησιμοποιεί τη *λειτουργία εκπομπής (broadcast mode)* του GAP. Αν όμως είναι επιθυμητό η συσκευή να είναι ορατή στο σύνολο του δικτύου και να έχει τη δυνατότητα να αλληλεπιδρά με άλλες συσκευές, τότε απαιτείται να είναι *ανευρέσιμη (discoverable)*, δηλαδή να έχει ρόλο Peripheral και να επικοινωνεί με όλες τις συσκευές Central που εκτελούν σάρωση. Η ιδιότητα αυτή ελέγχεται από τη συσκευή με επιλογή μίας από τις παρακάτω λειτουργίες, μεταβάλλοντας κατάλληλα το πεδίο *Flags AD* των δεδομένων εκπομπής:

- ***Non-discoverable mode (NDM)***: Η συσκευή δεν είναι ορατή σε κανένα central, ακόμα και αν διαφημίζεται. Αυτή είναι πάντοτε η προεπιλεγμένη λειτουργία.
- ***Limited discoverable mode (LDM):*** Η συσκευή είναι ορατή για χρονικό διάστημα μικρότερο ή ίσο των 30 s, συνήθως μετά την εκκίνησή της ή την απόλυση της τελευταίας σύνδεσης, με ταχύρρυθμη εκπομπή πακέτων διαφήμισης. Χρησιμοποιείται κατά κανόνα για εντοπισμό από centrals τα οποία έχουν πρόσφατα αλληλεπιδράσει με τη συσκευή ή - με την ευρεία έννοια - την γνωρίζουν (πχ με το όνομα ή τη δημόσια διεύθυνσή της).
- ***General discoverable mode (GDM):*** Η συσκευή παραμένει ορατή στο σύνολο του δικτύου όσο διαφημίζεται, εκπέμποντας πακέτα σε αραιά χρονικά διαστήματα (τάξης μεγέθους μερικών δευτερόλεπτων).

Βάσει των παραπάνω, ένα Peripheral είναι ανευρέσιμο αν χρησιμοποιεί την GDM ή την LDM. Αντίστοιχα, ένα Central κάνει χρήση των διαδικασιών *General Discovery* και *Limited Discovery* για τον εντοπισμό όλων των ανευρέσιμων ή των περιορισμένα ανευρέσιμων συσκευών.

3.4.2.3 Εγκατάσταση συνδέσεων

Μια συσκευή BLE, ανεξάρτητα από την ανευρεσιμότητά της, είναι σε θέση να ελέγχει πότε και με ποιες συσκευές είναι *συνδέσιμη (connectable),* ενεργοποιώντας μία από τις παρακάτω λειτουργίες:

- ***Non-connectable mode****:* Η συσκευή μπορεί να εκπέμπει πακέτα διαφήμισης *ADV_NONCONN_IND*, αλλά δεν αποδέχεται αιτήματα σύνδεσης, ακόμα και αν είναι ανευρέσιμη.
- ***Directed connectable mode****:* Η συσκευή αποστέλλει πακέτα απευθείας διαφήμισης (*ADV_DIRECT_IND*) προς ένα συγκεκριμένο central, αναμένοντας αίτημα σύνδεσης από αυτό.
- ***Undirected connectable mode****:* Η συσκευή εκπέμπει στο δίκτυο πακέτα διαφήμισης γενικού σκοπού (*ADV_IND* ή *ADV_AUX_IND*) και είναι σε θέση να αποδεχθεί αιτήματα σύνδεσης.

Αντίστοιχα, ένα Central μπορεί να συνδεθεί με ένα ή περισσότερα Peripherals εκτελώντας τις ακόλουθες διαδικασίες του GAP:

- ***Auto connection establishment****:* Η συσκευή δε σαρώνει το δίκτυο, παρά μόνο δηλώνει μία λίστα από έμπιστα Peripherals *(white list),* και αποπειράται αυτόματα να συνδεθεί με καθένα από αυτά μόλις το εντοπίσει στο δίκτυο. Οι παράμετροι σύνδεσης είναι κοινές για όλες τις συσκευές της λίστας.
- ***General connection establishment****:* Η συσκευή πραγματοποιεί απαραίτητα σάρωση για τον εντοπισμό όλων των διαθέσιμων συσκευών που χρησιμοποιούν UCM, και στη συνέχεια είναι δυνατή η επιλογή μίας ή περισσότερων συσκευών προς σύνδεση.
- ***Selective connection establishment****:* Επιτελεί την ίδια λειτουργία με την auto connection establishment, με ορισμό όμως διαφορετικών παραμέτρων σύνδεσης για κάθε συσκευή.
- ***Direct connection establishment****:* Η συσκευή αποπειράται να συνδεθεί με ένα Peripheral που χρησιμοποιεί DCM, δίχως να απαιτείται ο εντοπισμός του.

3.4.2.4 Ασφάλεια

Μία συσκευή ή μία υπηρεσία BLE ορίζει τις απαιτήσεις ασφαλείας της επιλέγοντας μία από τις διαθέσιμες *λειτουργίες ασφαλείας (security modes)* του GAP. Η λειτουργία 1 παρέχει κρυπτογράφηση στο επίπεδο ζεύξης, ενώ η λειτουργία 2 επιτρέπει την ψηφιακή υπογραφή των δεδομένων στα ανώτερα από το L2CAP επιπέδα. Η έκδοση 5.0 προβλέπει επιπλέον μία τρίτη λειτουργία, ισοδύναμη με την πρώτη, μόνο για ισόχρονες εκπομπές. Όλες οι λειτουργίες διαβαθμίζουν σε *επίπεδα (levels)* την παροχή των απαιτήσεων που υποστηρίζουν, όπως φαίνεται στον παρακάτω πίνακα:

		Έλεγχος ταυτότητας	Κρυπτογράφηση ζεύξης	Υπογραφή δεδομένων
Λειτουργία Ασφαλείας 1	Επίπεδο 1	Όχι	Όχι	Όχι
	Επίπεδο 2	Όχι	Ναι	Όχι
	Επίπεδο 3	Ναι	Ναι	Όχι
	Επίπεδο 4	Ναι	Ναι	Όχι
Λειτουργία Ασφαλείας 2	Επίπεδο 1	Όχι	Όχι	Ναι
	Επίπεδο 2	Ναι	Όχι	Ναι
Λειτουργία Ασφαλείας 3	Επίπεδο 1	Όχι	Όχι	Όχι
	Επίπεδο 2	Όχι	Ναι	Όχι
	Επίπεδο 3	Ναι	Ναι	Όχι

Πίνακας 3.10: Λειτουργίες και επίπεδα ασφαλείας

Κατά την εγκατάσταση της σύνδεσης, καθένα από τα δύο μέρη γνωστοποιεί στο άλλο το δικό του επιθυμητό επίπεδο ασφαλείας, και επιλέγεται το ισχυρότερο από τα δύο για χρήση καθ' όλη τη διάρκεια της σύνδεσης. Καθώς η κρυπτογράφηση αντιμετωπίζεται ως ισχυρότερη απαίτηση συγκριτικά με την ψηφιακή υπογραφή, τα επίπεδα 2 και 3 της λειτουργίας 1 υπερκαλύπτουν αντίστοιχα τα επίπεδα 1 και 2 της λειτουργίας 2, ενώ η απουσία ασφάλειας ισοδυναμεί με το επίπεδο 1 της λειτουργίας 1. Όμοια, με δεδομένο το επίπεδο ασφαλείας της σύνδεσης, είναι δυνατή η χρήση αποκλειστικά υπηρεσιών που ορίζουν επίπεδα ασφαλείας ισοδύναμα ή χαμηλότερα από αυτό. Για παράδειγμα, μία υπηρεσία που απαιτεί ψηφιακή υπογραφή, δηλαδή χρησιμοποιεί τη λειτουργία 2, μπορεί να μεταδώσει δεδομένα πάνω από μια κρυπτογραφημένη ζεύξη που χρησιμοποιεί τα επίπεδα 2, 3 της λειτουργίας 1.

Από τις απαιτήσεις που ικανοποιούνται σε κάθε επίπεδο προκύπτει ένα συνόλο κλειδιών κρυπτογράφησης, το οποίο συνιστά και το κοινό μυστικό της επικοινωνίας. Η διαδικασία παραγωγής και ανταλλαγής των κλειδιών καλείται *αντιστοίχιση (pairing)*, και πραγματοποιείται μόνο κατά την εγκατάσταση της σύνδεσης. Όπως φαίνεται στην τρίτη στήλη του πίνακα, η αντιστοίχιση μπορεί να γίνεται με ή δίχως έλεγχο ταυτότητας των δύο μερών, ανάλογα με τη μέθοδο αντιστοίχισης που χρησιμοποιείται, κάτι που θα αναλύσουμε περαιτέρω στην υποενότητα 3.4.4.

Με την μόνιμη αποθήκευση των κλειδιών της πρώτης επιτυχούς αντιστοίχισης μεταξύ δύο συσκευών, καθίσταται περιττός ο επανυπολογισμός τους

σε κάθε νέα σύνδεση. Σε αυτή την περίπτωση, η αντιστοίχιση μεταξύ των δύο συσκευών είναι μόνιμη και καλείται *σύζευξη (bonding)*. Μία συσκευή επιλέγει να κάνει χρήση ή όχι αυτής της δυνατότητας χρησιμοποιώντας αντίστοιχα τις λειτουργίες *Bondable* και *Non-bondable* του GAP.

3.4.3 Attribute Protocol (ATT)

Το *Attribute Protocol* (ATT) ορίζει το *attribute (ιδιότητα)* ως την στοιχειώδη μονάδα δεδομένων που είναι διαθέσιμη στα θεμελιώδη προφίλ (GAP, GATT) και στα προφίλ της εφαρμογής χρήστη. Το attribute αποτελεί ουσιαστικά μια εγγραφή της μορφής του παρακάτω σχήματος:

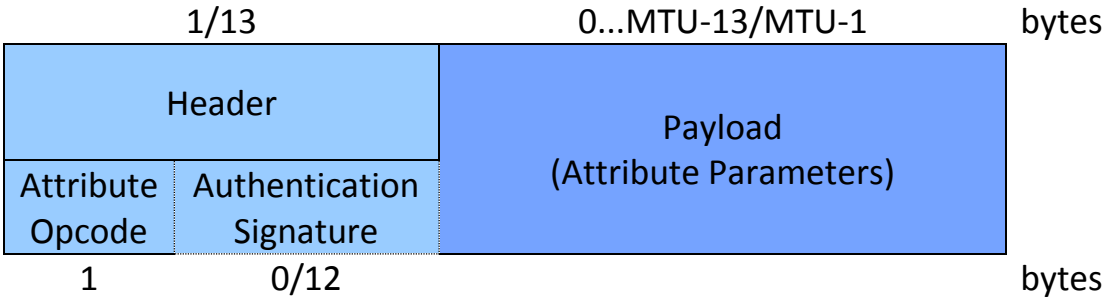
Handle (2 bytes)	UUID (2/16 bytes)	Value (1-512 bytes)	Permissions (μεταβλητό)

Σχήμα 3.11: Δομή ενός attribute

Τα δεδομένα προς αποθήκευση ενσωματώνονται κάθε φορά στο πεδίο *Value* του attribute, ενώ το σύνολο των attributes διατηρείται σε έναν πίνακα που καλείται *Attribute Table*. Το κλειδί αναζήτησης στον Attribute Table είναι η τιμή του πεδίου *Handle*, η οποία είναι μοναδική για κάθε attribute και προσδιορίζει τη θέση του στον πίνακα. Επιπλέον, για κάθε attribute ορίζεται ένα αναγνωριστικό UUID (universal unique identifier) μήκους 16 ή 128 bit, το οποίο προσδιορίζει τον τύπο δεδομένων που αντιπροσωπεύει (*Attribute Type*). Η απόκτηση δικαιωμάτων πρόσβασης σε ένα attribute από απομακρυμένους clients ενδέχεται να προϋποθέτει κρυπτογράφηση της ζεύξης, έλεγχο ταυτότητας, εξουσιοδότηση, ή οποιονδήποτε συνδυασμό αυτών. Οι απαιτήσεις αυτές ορίζονται ξεχωριστά για κάθε ενέργεια πάνω στο attribute, και συγκεντρώνονται στο πεδίο *Permissions*.

Στο επίπεδο του ATT, μία από τις συνδεδεμένες συσκευές ενεργεί ως πελάτης, και η άλλη ως εξυπηρετητής. Ο πελάτης αποκτά άμεσα πρόσβαση στα attributes του server για ανάγνωση ή εγγραφή τους, ή ενημερώνεται ασύγχρονα από τον server για αλλαγές στις τιμές τους με χρήση των μηχανισμών *ειδοποίησης (notification)* και *ένδειξης (indication)* του ATT. Οι ειδοποιήσεις αποστέλλονται χωρίς να ακολουθεί επιβεβαίωση από τον πελάτη, επιτρέποντας υψηλότερους ρυθμούς μετάδοσης, ενδείκνυνται συνεπώς για ταχείες ροές δεδομένων με υψηλό βαθμό ανοχής σε σφάλματα (πχ πολυμέσα). Αντίστοιχα, οι ενδείξεις απαιτούν επιβεβαίωση, γεγονός που τις καθιστά καταλληλότερες για ροές που απαιτούν εγγύηση παράδοσης (πχ αρχεία), περιορίζοντας όμως την ταχύτητα.

Η ανταλλαγή των μηνυμάτων του πρωτοκόλλου γίνεται πάνω από πακέτα ATT, των οποίων τη δομή παρουσιάζουμε στο παρακάτω σχήμα. Κάθε μέρος της επικοινωνίας ορίζει το μέγιστο μήκος μηνύματος (ATT MTU) που είναι σε θέση να λάβει στο επίπεδο του ATT, με προεπιλεγμένη τιμή τα 23 bytes, και για την επικοινωνία επιλέγεται το μικρότερο από αυτά.



Σχήμα 3.12: Πακέτο ATT

Η επικεφαλίδα του πακέτου ATT περιλαμβάνει απαραίτητα τον κωδικό (opcode) της ενέργειας προς εκτέλεση. Οι ενέργειες του πελάτη διακρίνονται σε *αιτήματα (requests)* και *εντολές (commands)*. Τα αιτήματα ολοκληρώνονται με απόκριση (response) από τον server, σε αντίθεση με τις εντολές που εκτελούνται αμέσως. Παρακάτω δίνουμε τις συνηθέστερα χρησιμοποιούμενες ενέργειες πελάτη:

- *Exchange MTU(size)*: Ο client ζητά μέγιστο ωφέλιμο φορτίο (ATT MTU) ίσο με *size*. Το αίτημα επιτυγχάνει εάν ο server υποστηρίζει αυτό το μήκος πακέτου, διαφορετικά επιστρέφεται η προεπιλεγμένη τιμή των 23 bytes.
- *Find Information Request(h_1, h_2)*: Επιστρέφει τα handles και τους τύπους όλων των attributes μεταξύ των handles h_1, h_2.
- *Find by Type Request(type, h_1, h_2)*: Επιστρέφει τα handles όλων των attributes με τύπο δεδομένων (UUID) *type* μεταξύ των handles h_1, h_2.
- *Read by Type/Group Type Request(type, h_1, h_2)*: Αιτείται ανάγνωση των τιμών όλων των attributes με τύπο δεδομένων (UUID) *type* μεταξύ των handles h_1, h_2.
- *Read Request(h)*: Αιτείται ανάγνωση της τιμής του attribute με handle *h*.
- *Read Blob Request(h, offset)*: Εάν το μήκος του attribute ξεπερνά την ATT MTU, η τιμή του διαβάζεται τμηματικά σε πακέτα του μέγιστου δυνατού ωφέλιμου φορτίου. Χρησιμοποιούνται αιτήματα ισάριθμα των τμημάτων που προκύπτουν, με την παράμετρο *offset* να δείχνει κάθε φορά τη θέση (σε bytes) του επόμενου τμήματος προς ανάγνωση.
- *Read Multiple Request(h_1, h_2)*: Ενσωματώνει σειριακά τις τιμές όσο το δυνατόν περισσότερων attributes μεταξύ των handles h_1, h_2 σε ένα πακέτο απόκρισης.
- *Write Request(h, value)*: Γράφει την τιμή *value* στο attribute με handle *h*.
- *Prepare/Execute Write Request(h, value, offset)*: Κατ' αναλογία με το *Read Blob Request*, χρησιμοποιείται για εγγραφή της τιμής ενός attribute που υπερβαίνει σε μήκος την ATT MTU.
- *Write Command(h, value):* Η εντολή αυτή γράφει άμεσα την τιμή *value* στο χαρακτηριστικό με handle *h*, δίχως απόκριση από τον server.

Οι κυριότερες ενέργειες που εκτελούνται με πρωτοβουλία του εξυπηρετητή είναι οι εξής:

- *Handle Value Notification(h, value)*: Ενημερώνει τον client για αλλαγή της τιμής του χαρακτηριστικού με handle *h*, αποστέλλοντας τη νέα τιμή *value* με χρήση ειδοποίησης (notification).
- *Handle Value Indication(h, value)*: Ενημερώνει τον client για αλλαγή της τιμής του χαρακτηριστικού με handle *h*, αποστέλλοντας τη νέα τιμή *value* με χρήση ένδειξης (indication).

3.4.4 Security Manager Protocol (SMP)

Το *Security Manager Protocol (SMP)* επιφορτίζεται μόνο με την ανταλλαγή των απαραίτητων μηνυμάτων για την παραγωγή των κλειδιών αντιστοίχισης. Μπορούμε έτσι χονδρικά να ισχυριστούμε πως η διαδικασία αντιστοίχισης αποτελεί αρμοδιότητα του SMP, η σύζευξη όμως πραγματοποιείται εξ ολοκλήρου στο GAP.

Η αντιστοίχιση ξεκινά με αίτημα *Security Request* από την πλευρά του slave (responder), ή *Pairing Request* από την πλευρά του master (initiator). Για εκδόσεις του BLE παλαιότερες της 4.2, από προεπιλογή ακολουθείται η *κλασική* διαδικασία αντίστοιχισης (*LE Legacy Pairing*), η οποία εξελίσσεται σε τρεις φάσεις:

(α) Επιλογή μεθόδου

Το προσωρινό κλειδί (*temporary key*, TK) χρησιμοποιείται για την παραγωγή των κλειδιών στις επόμενες φάσεις. Για προστασία από επιθέσεις ενδιάμεσου (MITM), θα πρέπει το TK να μην είναι ορατό στον αέρα, ή τουλάχιστον στη ζώνη συχνοτήτων μίας συσκευής παρακολούθησης. Η ανταλλαγή μπορεί να γίνει με μία από τις παρακάτω *μεθόδους*:

- *Just Works (JW):* Το TK παίρνει την προεπιλεγμένη τιμή 0. Ένας επιτιθέμενος που γνωρίζει ότι χρησιμοποιείται αυτή η μέθοδος μπορεί να υπολογίσει τα προκύπτοντα κλειδιά των επόμενων φάσεων, συνεπώς η μέθοδος αυτή δεν παρέχει προστασία από MITM. Αποτελεί τη μοναδική μέθοδο αντιστοίχισης χωρίς έλεγχο ταυτότητας (unauthenticated pairing), και χρησιμοποιείται μόνο εφόσον δεν απαιτείται τέτοιος έλεγχος ή οι επόμενες μέθοδοι είναι αδύνατο να εφαρμοστούν.

- *Passkey Entry (PKE):* Για την αρχικοποίηση του TK, στην οθόνη ενός εκ των δύο μερών τυπώνεται ένα εξαψήφιο PIN, το οποίο πρέπει να εισαχθεί από το πληκτρολόγιο ή άλλη συσκευή εισόδου του άλλου μέρους. Οι τιμές του TK που αποθηκεύονται τοπικά στις συσκευές ταυτίζονται αν και μόνο αν ταυτίζονται τα δύο PIN. Αφού το TK δε μεταδίδεται στον αέρα, κάθε απόπειρα MITM έχει πιθανότητα επιτυχίας μόλις 10^{-6}, συνεπώς η μέθοδος αυτή παρέχει ικανοποιητικό επίπεδο προστασίας.

- *Numeric Comparison (NC):* Αποτελεί παραλλαγή της μεθόδου PKE και επιλέγεται εάν στη φάση 2 πρόκειται να χρησιμοποιηθεί η διαδικασία *LE Secure*

Connections, και μόνον τότε. Στις οθόνες και των δύο συσκευών τυπώνεται το ίδιο εξαψήφιο PIN, και η ταύτιση των εμφανιζόμενων αριθμών επιβεβαιώνεται από το χρήστη επιλέγοντας NAI ή OXI.

- *Out-of-Band (OOB):* Το TK μεταδίδεται στον αέρα, σε διαφορετική όμως ζώνη συχνοτήτων από αυτήν του BLE και συνήθως σε μικρή απόσταση, για παράδειγμα μέσω NFC. Συνιστά ακόμα ασφαλέστερη επιλογή σε σύγκριση με την PKE/NC.

Για την επιλογή της κατάλληλης μεθόδου απαιτείται προηγουμένως οι συσκευές να έχουν κοινοποιήσει τις δυνατότητες εισόδου και εξόδου (E/E) που διαθέτουν. Αυτές μπορούν να είναι οθόνη για έξοδο, και πληκτρολόγιο δυαδικής επιλογής NAI/OXI (Y/N) ή εισαγωγής αριθμών για είσοδο. Οι επιτρεπτοί συνδυασμοί και οι επιλογές που προκύπτουν (εξαιρουμένης της OOB) παρουσιάζονται στον ακόλουθο πίνακα:

Initiator	Responder				
	Μόνο οθόνη	Οθόνη + Y/N	Μόνο πληκτρολόγιο	Καμία E/E	Οθόνη + πληκτρολόγιο
Μόνο οθόνη	JW	JW	PKE	JW	PKE
Οθόνη + Y/N	JW	JW/NC	PKE	JW	PKE/NC
Μόνο πληκτρολόγιο	PKE	PKE	PKE	JW	PKE
Καμία E/E	JW	JW	JW	JW	JW
Οθόνη + πληκτρολόγιο	PKE	JW/NC	PKE	JW	PKE/NC

Πίνακας 3.13: Μέθοδοι αντιστοίχισης

(β) Κρυπτογράφηση ζεύξης

Για την κρυπτογράφηση της ζεύξης μέχρι την παραγωγή των τελικών κλειδιών χρησιμοποιείται το 128-bit *κλειδί βραχείας διάρκειας* (*short-term key*, STK), το οποίο προκύπτει από κρυπτογράφηση AES-128 της συνένωσης (*SRand* | *MRand*) δύο 64-bit τυχαίων αριθμών *SRand* και *MRand* με κλειδί το TK, όπου ο *SRand* παράγεται από τον slave και ο *MRand* από τον master:

$$STK = \mathbf{E}_{TK}(SRand \mid MRand)$$

(γ) Παραγωγή τελικών κλειδιών

Με βάση το STK της προηγούμενης φάσης παράγονται - ανάλογα με τις απαιτήσεις ασφαλείας - μέχρι τρία κλειδιά μήκους 128-bit για οριστική χρήση κατά την τρέχουσα σύνδεση, καθένα από τα οποία επιτελεί διαφορετική λειτουργία. Αυτά είναι:

- **Long-term key (LTK)**: Παρέχει κρυπτογράφηση των δεδομένων, εξασφαλίζοντας την εμπιστευτικότητα στο επίπεδο ζεύξης δεδομένων. Σε περίπτωση σύζευξης, το LTK δεν αποθηκεύεται στον slave λόγω έλλειψης πόρων. Αντ' αυτού ο slave γνωστοποιεί στον master δύο τιμές *Ediv* και *Rand* από τις οποίες με τη βοήθεια του STK μπορεί το LTK κάθε φορά να επανυπολογίζεται.

- **Connection signature resolving key (CSRK)**: Χρησιμοποιείται για την υπογραφή των μεταδιδόμενων δεδομένων, εξασφαλίζοντας τον έλεγχο ταυτότητας του αποστολέα στο επίπεδο του ATT. Η ψηφιακή υπογραφή αποτελείται από το κρυπτοκείμενο του μηνύματος που παράγεται από τον αλγόριθμο CMAC με κλειδί το CSRK, και τον μετρητή *SignCounter* μήκους 32 bit, ο οποίος καταγράφει τον αύξοντα αριθμό του τελευταίου μεταδοθέντος μηνύματος. Ο *SignCounter* αρχικοποιείται στο 0 με τη σύζευξη των συσκευών, αυξάνεται με κάθε έγκυρο και ταυτοποιημένο μήνυμα, και η τιμή του διατηρείται σε κάθε επόμενη σύνδεση. Με αυτόν τον τρόπο παρέχεται προστασία απέναντι σε επιθέσεις επανάληψης.

- **Identity resolving key (IRK)**: Επιτρέπει την απόκρυψη της δημόσιας διεύθυνσης της συσκευής και την κρυπτογράφησή της για την παραγωγή - σε τακτά χρονικά διαστήματα - μίας ιδιωτικής διεύθυνσης. Αυτή προκύπτει από τη συνένωση ενός τυχαίου ακεραίου *rand* μήκους 64 bit, και της κρυπτογραφημένης τιμής του με χρήση του IRK. Τα δύο άκρα της επικοινωνίας γνωρίζουν τόσο το *rand* όσο και το IRK - σε αντίθεση με έναν πιθανό επιτιθέμενο, που μπορεί να υποκλέψει μόνο το *rand* - συνεπώς αυτά και μόνο μπορούν να κρυπτογραφούν τις δημόσεις διευθύνσεις τους:

$$private\_address = (rand \mid \mathbf{E}_{IRK}(rand))$$

Η έκδοση 4.2 του Bluetooth Core Specification προβλέπει επιπλέον τη διαδικασία αντιστοίχισης *LE Secure Connections*. Εφόσον υποστηρίζεται και από τις δύο συσκευές, η διαδικασία αυτή επιλέγεται με ενεργοποίηση του bit *SC* στο πεδίο *AuthReq* του πακέτου *Pairing Request*, και εξελίσσεται όμοια με την κλασική, με τη διαφορά ότι εδώ χρησιμοποιείται κρυπτογραφία *δημόσιου* κλειδιού: κατά τη φάση 2, για την κρυπτογράφηση της ζεύξης, αντί του STK παράγεται για κάθε συσκευή ένα ζεύγος δημόσιου-ιδιωτικού κλειδιού με χρήση του πρωτοκόλλου *ελλειπτικής καμπύλης Diffie-Hellman* (*Elliptic Curve Diffie-Hellman*, *ECDH*).

3.4.5 Generic Attribute Profile (GATT)

Το *Generic Attribute Profile* (GATT) είναι εκείνο το τμήμα της στοίβας που είναι περισσότερο ορατό στους προγραμματιστές εφαρμογών, κι αυτό γιατί ορίζει την οργάνωση σε λογικό επίπεδο των δεδομένων εφαρμογής που εκτίθενται από μια συσκευή. Κατά τη διάρκεια μίας σύνδεσης, το GATT λειτουργεί πάνω από μία διεπαφή πελάτη-εξυπηρετητή, με τον πελάτη (GATT client) να χρησιμοποιεί τις διαθέσιμες *υπηρεσίες (services)* του εξυπηρετητή (GATT server). Οι ρόλοι GATT server και GATT client είναι ανεξάρτητοι από τους ρόλους peripheral και central του GAP.

3.4.5.1 Υπηρεσίες και χαρακτηριστικά

Στην απλούστερη μορφή της, μία υπηρεσία GATT εκτίθεται από τον GATT server στον GATT client ως μία συλλογή από ειδικές δομές δεδομένων, αποτελούμενες από ένα σύνολο από attributes, οι οποίες καλούνται *χαρακτηριστικά (characteristics)*. Το σύνολο των υπηρεσιών ενός εξυπηρετητή, αποθηκευμένο σε έναν attribute table, καλείται *βάση δεδομένων GATT (GATT database)*.

Μία υπηρεσία GATT μπορεί να είναι *πρωτεύουσα (primary)*, αν είναι αυτοτελής και εκφράζει ένα μέρος της συμπεριφοράς της συσκευής, ή *δευτερεύουσα (secondary)*, αν εξαρτάται από μία πρωτεύουσα υπηρεσία. Εκτός από χαρακτηριστικά, μία υπηρεσία μπορεί να περιέχει άλλες υπηρεσίες ως θυγατρικές της, ορίζοντας αναδρομικά ένα δένδρο υπηρεσιών. Τότε όμως η γονική υπηρεσία δεν ενσωματώνει τις θυγατρικές υπηρεσίες στο δικό της ορισμό, παρά μόνο διατηρεί αναφορές στους ορισμούς τους. Οι θυγατρικές υπηρεσίες μπορούν να είναι τόσο πρωτεύουσες, η λειτουργικότητα των οποίων επεκτείνεται από τη γονική υπηρεσία, όσο και δευτερεύουσες, οι οποίες ενεργούν ως επαναχρησιμοποιήσιμες μονάδες λογισμικού για πραγματοποίηση βασικών ενεργειών.

Οι υπηρεσίες αποθηκεύονται σειριακά στον attribute table ως μπλοκ διαδοχικών attributes. Ως παράδειγμα δίνουμε - σε συντομευμένη μορφή - την GATT database μίας συσκευής με τρεις πρωτεύουσες υπηρεσίες A, B, C και μία δευτερεύουσα υπηρεσία D που υπάγεται στην B:

Handle	UUID	Description
0x0100	0x2800	Primary Service Declaration: Service A
...	...	Service A details
0x0150	0x2800	Primary Service Declaration: Service B
0x0151	0x2802	Include Declaration: Service D

Handle	UUID	Description
...	...	Service B details
0x0300	0x2800	Primary Service Declaration: Service C
...	...	Service C details
0x0350	0x2801	Secondary Service Declaration: Service D
...	...	Service D details

Πίνακας 3.14: Παράδειγμα βάσης δεδομένων GATT

Παρατηρούμε πως οι πρωτεύουσες και οι δευτερεύουσες υπηρεσίες οριοθετούνται από ειδικά attributes έναρξης με UUID 0x2800 και 0x2801 αντίστοιχα. Ο ειδικός αυτός τύπος attribute καλείται *δήλωση υπηρεσίας* (*Service Declaration*), ακριβώς επειδή κάθε τέτοιο attribute εισάγει μία συγκεκριμένη υπηρεσία στη βάση.

Η τιμή του *Service Declaration* συνιστά και αυτή ένα UUID (*Service UUID*), το οποίο προσδιορίζει μοναδικά τον *τύπο* της υπηρεσίας (*service type*). Ο τύπος μίας υπηρεσίας καθορίζει πλήρως την εσωτερική της δομή και τη σημασιολογία των χαρακτηριστικών της, ο σχεδιαστής όμως ενός συστήματος έχει την ελευθερία να ορίσει οσεσδήποτε επιθυμητές υπηρεσίες-στιγμιότυπα του ίδιου τύπου στη βάση δεδομένων GATT. Για την αναγνώριση προτυποποιημένων υπηρεσιών, το Bluetooth SIG εκχωρεί 16-bit Service UUIDs. Για όλες τις υπόλοιπες υπηρεσίες, είναι η δυνατή η ελεύθερη χρήση Service UUIDs μήκους 128 bit. Για μια υπηρεσία που χρησιμοποιεί άλλες υπηρεσίες, οι αναφορές σε αυτές αποθηκεύονται στον πίνακα ως διαδοχικά attributes τύπου *Include Declaration* (UUID 0x2802), αμέσως μετά τη δήλωση της υπηρεσίας.

Στο σημείο αυτό είναι φυσικό να αναμένουμε πως τα υπόλοιπα attributes που εσωκλείονται στον ορισμό της υπηρεσίας αναφέρονται στα χαρακτηριστικά της, και πράγματι αυτό συμβαίνει. Για παράδειγμα, παρουσιάζουμε το εσωτερικό του Service A:

Handle	UUID	Description	Value
0x0100	0x2800	Primary Service Declaration: Thermometer Service	UUID 0x1816
0x0101	0x2803	Characteristic Declaration: Temperature	Properties: Read, Notify UUID 0x2A2B Value handle: 0x0102
0x0102	0x2A2B	Characteristic Value:	20.0 degrees

Handle	UUID	Description	Value
		Temperature	
0x0103	0x2902	Characteristic Descriptor: Client Characteristic Configuration	Notifications: enabled Indications: disabled
0x0104	0x2904	Characteristic Descriptor: Characteristic Presentation Format	Format: SFLOAT (0x16) Unit: Celsius (0x272F)
0x0110	0x2803	Characteristic Declaration: Date/time	Properties: Read, Write UUID 0x2A08 Value handle: 0x0111
0x0111	0x2A08	Characteristic Value: Date/Time	"1/1/1980 12:00"
0x0112	0x2904	Characteristic Descriptor: Characteristic Presentation Format	Format: UTF-8 (0x19)

Πίνακας 3.15: Παράδειγμα δομής υπηρεσίας GATT

Για λόγους απλότητας, υποθέτουμε πως το ενδεικτικό Service που παρουσιάζουμε στο παραπάνω σχήμα έχει 16-bit Service UUID ίσο με 0x1816. Το ονοματίζουμε Thermometer Service, και του αναθέτουμε δύο χαρακτηριστικά: Temperature (UUID 0x2A2B) και Date/Time (UUID 0x2A08). Παρατηρούμε ότι τα χαρακτηριστικά αποθηκεύονται σειριακά όπως και οι υπηρεσίες, οριοθετούμενα από attributes τύπου *Characteristic Declaration (δήλωση χαρακτηριστικού)* με UUID 0x2803.

Γενικά, η καταχώρηση ενός characteristic ξεκινά υποχρεωτικά με τα attributes *Characteristic Declaration* (UUID 0x2803) και *Characteristic Value*. Το *Characteristic Value* αντιπροσωπεύει την *τιμή* του χαρακτηριστικού, όπως είναι ορατή στις εφαρμογές. Η τιμή του *Characteristic Declaration* συντίθεται από τα εξής πεδία:

- ***Properties***: Ορίζει τα γνωστά από το ATT δικαιώματα πρόσβασης στο attribute *Characteristic Value*.
- ***Value Handle***: Παρέχει έναν δείκτη προς τη θέση του *Characteristic Value* στον attribute table.
- ***Characteristic UUID***: Ορίζει το UUID του *Characteristic Value*, το οποίο προσδιορίζει μοναδικά τον *τύπο* του χαρακτηριστικού *(characteristic type)* και αποτελεί το αναγνωριστικό του εντός της εμβέλειας της υπηρεσίας στην οποία ανήκει.

Ένα χαρακτηριστικό μπορεί να συνοδεύεται από επιπλέον attributes, που ονομάζονται *περιγραφείς (descriptors)* και αφορούν σε συγκεκριμένες του ιδιότητες. Οι περιγραφείς διακρίνονται στους εξής τύπους:

- *Characteristic Extended Properties (UUID 0x2900)*: Ορίζει επιπλέον ιδιότητες του χαρακτηριστικού που δεν περιλαμβάνονται στο πεδίο *Properties* του *Characteristic Declaration*.
- *Characteristic User Description (UUID 0x2901)*: Περιέχει μία σύντομη και φιλική στο χρήστη περιγραφή του χαρακτηριστικού με τη μορφή συμβολοσειράς UTF-8.
- *Client Characteristic Configuration (UUID 0x2902)*: Ορίζει κατά πόσο επιτρέπονται οι ειδοποιήσεις και οι ενδείξεις ATT για το *Characteristic Value*.
- *Server Characteristic Configuration (UUID 0x2903)*: Ορίζει κατά πόσο επιτρέπεται η εκπομπή πληροφορίας στα πακέτα διαφήμισης σχετικά με την υπηρεσία στην οποία ανήκει το χαρακτηριστικό.
- *Characteristic Presentation Format (UUID 0x2904)*: Πλαισιώνει την τιμή του χαρακτηριστικού με βοηθητικά στοιχεία για τη σωστή της ερμηνεία από τον client. Τέτοια στοιχεία είναι ο τύπος της τιμής, οι μονάδες μέτρησης κλπ.
- *Characteristic Aggregation Format (UUID 0x2905)*: Για ένα χαρακτηριστικό με σύνθετη τιμή, επιτρέπει τη χρήση επιπλέον περιγραφέων *Characteristic Presentation Format* για καθένα από τα πεδία της τιμής.

3.4.5.2 Εύρεση υπηρεσιών

Με βάση τα παραπάνω, κατανοούμε πως είναι υπολογιστικά εύκολη η εύρεση όλων των υπηρεσιών και χαρακτηριστικών μίας συσκευής. Η λειτουργία αυτή *(service discovery)* είναι κρίσιμης σημασίας στη λειτουργία του BLE, καθώς επιτρέπει στον απομακρυσμένο client να συνεργαστεί με το υλοποιούμενο προφίλ, και συνήθως έπεται κάθε επιτυχούς εγκατάστασης σύνδεσης, με εξαίρεση τις περιπτώσεις όπου η πληροφορία αυτή βρίσκεται αποθηκευμένη στην προσωρινή μνήμη του client από προγενέστερη σύνδεση *(caching)*. Πραγματοποιείται με χρήση των ακόλουθων *διαδικασιών* του GATT, οι οποίες ανάγονται σε αιτήματα του ATT:

- ***Discover all Services***: Πραγματοποιεί εύρεση όλων των υπηρεσιών στο σύνολο του attribute table, αποστέλλοντας το αίτημα *Read by Group*("*Service*", *0x0001*, *0xFFFF*).
- ***Discover Primary Service by UUID***: Αναζητά μια πρωτεύουσα υπηρεσία στο σύνολο του πίνακα με κλειδί το Service UUID της, αποστέλλοντας το αίτημα *Read by Type*("*Primary Service*", *0x0001*, *0xFFFF*).
- ***Find Included Services***: Βρίσκει όλες τις υπηρεσίες στις οποίες αναφέρεται η υπηρεσία που περιέχεται μεταξύ των θέσεων h_1 και h_2 του πίνακα, αποστέλλοντας το αίτημα *Read by Type*("*Include*", h_1, h_2).

- ***Discover Characteristics by Service***: Βρίσκει όλα τα χαρακτηριστικά της υπηρεσίας που περιέχεται μεταξύ των θέσεων h_1 και h_2, αποστέλλοντας το αίτημα *Read by Type*("*Characteristic*", h_1, h_2).
- ***Discover Characteristic Descriptors:*** Βρίσκει όλους τους περιγραφείς του χαρακτηριστικού που περιέχεται μεταξύ των θέσεων h_1 και h_2, αποστέλλοντας το αίτημα *Find Information Request*(h_1, h_2).

3.4.5.3 Διαχείριση χαρακτηριστικών

Έχοντας στη διάθεσή του την απαιτούμενη πληροφορία από τη βάση δεδομένων GATT του server, ο client είναι πλέον σε θέση να πραγματοποιήσει ενέργειες ανάγνωσης και εγγραφής πάνω στα χαρακτηριστικά και τους περιγραφείς τους, χρησιμοποιώντας τις ακόλουθες διαδικασίες:

- ***Read Characteristic Value/Descriptor:*** Πραγματοποιεί ανάγνωση της τιμής ή ενός περιγραφέα του χαρακτηριστικού αποστέλλοντας το αίτημα *Read Request* ή *Read Blob Request*.
- ***Read Using Characteristic UUID:*** Χρησιμοποιείται για ανάγνωση της τιμής του χαρακτηριστικού με UUID *t*, στην περίπτωση που η θέση αυτής στη βάση δεν είναι γνωστή. Χρησιμοποιεί το αίτημα *Read by Type*(*t, 0x0001, 0xFFFF*).
- ***Read Multiple Characteristic Values:*** Πραγματοποιεί ανάγνωση των τιμών περισσότερων χαρακτηριστικών αποστέλλοντας το αίτημα *Read Multiple Request*. Επειδή όλες οι τιμές τοποθετούνται σε ένα πακέτο απόκρισης, απαιτείται να είναι εκ των προτέρων γνωστό το μήκος καθεμιάς από αυτές.
- ***Characteristic Value/Descriptor Write Request:*** Πραγματοποιεί εγγραφή της τιμής ή ενός περιγραφέα του χαρακτηριστικού αποστέλλοντας το αίτημα *Write Request*. Εάν το μήκος των δεδομένων προς εγγραφή υπερβαίνει την ATT MTU, χρησιμοποιείται το αίτημα *Prepare Write Request*, ακολουθούμενο από το *Execute Write Request*.
- ***Characteristic Write without Response:*** Μεταβάλλει την τιμή του χαρακτηριστικού αποστέλλοντας την εντολή *Write Command*. Καθώς πρόκειται για εντολή και όχι για αίτημα, δεν απαιτείται απόκριση από τον server. Τα χαρακτηριστικά που χρησιμοποιούνται αποκλειστικά κατ' αυτόν τον τρόπο, δηλαδή για μεταφορά εντολών και δεδομένων στον server, καλούνται συχνά *σημεία ελέγχου (control points)*.
- ***Characteristic Value Reliable Write:*** Πραγματοποιεί ταυτόχρονη εγγραφή των τιμών περισσότερων χαρακτηριστικών, συγκεντρώνοντας μία ακολουθία αιτημάτων *Prepare Write Request* και *Execute Write Request* σε μία ατομική δοσοληψία.

Επικοινωνία όμως μπορεί να υπάρξει και με πρωτοβουλία του server. Αυτό είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου ο server επιθυμούμε να ενημερώνει περιοδικά τον client για την τιμή ενός μεγέθους, ή να τον ειδοποιεί ασύγχρονα για διάφορα συμβάντα, και επιτυγχάνεται με χρήση των εξής διαδικασιών:

- ***Characteristic Value Notification*:** Ενημερώνει την τιμή του χαρακτηριστικού, και την αποστέλλει στον client δίχως αναμονή για απόκριση, με χρήση της εντολής *Handle Value Notification* του ATT.
- ***Characteristic Value Indication*:** Ενημερώνει την τιμή του χαρακτηριστικού και την αποστέλλει στον client, αναμένοντας την απόκρισή του. Χρησιμοποιεί την εντολή *Handle Value Indication* του ATT.

4 Προδιαγραφή του συστήματος

4.1 Συστατικά μέρη

Το σύστημα που υλοποιείται στην παρούσα εργασία βασίζεται σε μια αρχιτεκτονική πελάτη-εξυπηρετητή (client-server). Απαρτίζεται από την εφαρμογή πελάτη (client), που εκτελείται σε μία κινητή συσκευή, και την εφαρμογή εξυπηρετητή (server), που εκτελείται σε ειδικό ενσωματωμένο σύστημα (smart sensor) τοποθετημένο στο αντικείμενο προς εποπτεία και αποκρίνεται στα αιτήματα του client. Οι ρόλοι των δύο εφαρμογών αντιστοιχούν ευθέως στους ρόλους GATT των συσκευών που τις φιλοξενούν. Έτσι, η κινητή συσκευή ενεργεί ως GATT client, και ταυτόχρονα ως GAP central, ενώ ο smart sensor ως GATT server, και ταυτόχρονα ως GAP peripheral.

4.2 Προδιαγραφή παρεχόμενων λειτουργιών

Το σύστημα παρέχει τις παρακάτω βασικές λειτουργίες:

- ***Εποπτεία σε πραγματικό χρόνο (real-time monitoring):*** Ο client μπορεί να παρακολουθεί την εξέλιξη του εποπτευόμενου μεγέθους (π.χ. θερμοκρασία, υγρασία, επιτάχυνση, γωνιακή ταχύτητα κλπ) σε πραγματικό χρόνο, μόνον όσο είναι συνδεδεμένος με τον server.
- ***Εποπτεία κατάστασης της συσκευής:*** Ο client μπορεί να ενημερώνεται για την κατάσταση της συσκευής προς εποπτεία τόσο σε σύνδεση, όσο και εκτός σύνδεσης. Όσο το peripheral δεν είναι συνδεδεμένο με κάποιο central, περιοδικά ενημερώνει την κατάστασή του, και στη συνέχεια την κοινοποιεί στο δίκτυο για χρονικό διάστημα οριζόμενο από το χρήστη.
- ***Ορισμός συνθηκών καλής λειτουργίας:*** Ο χρήστης μπορεί μέσω του client να ορίσει τις συνθήκες καλής λειτουργίας της συσκευής προς εποπτεία, για παράδειγμα σε συνάρτηση με την τιμή κάποιου μεγέθους ή τη χρονική μεταβολή του.
- ***Ρύθμιση παραμέτρων αισθητήρων:*** Ο χρήστης μπορεί μέσω του client να ενεργοποιήσει ή να απενεργοποιήσει κάποιον αισθητήρα, καθώς και να μεταβάλει κάποιες παραμέτρους του (π.χ. ακρίβεια μέτρησης, εύρος τιμών κλπ).
- ***Ειδοποίηση σε περίπτωση δυσλειτουργίας:*** Με επιλογή του χρήστη, ο client μπορεί ακόμα και στο παρασκήνιο να συνεχίζει να «ακούει» για διαθέσιμες συσκευές στο δίκτυο. Αν κάποια από αυτές δηλώνει κατάσταση δυσλειτουργίας, ο client αποστέλλει στο χρήστη κατάλληλη ειδοποίηση.

- ***Καταγραφή και διατήρηση ιστορικού:*** Τόσο οι ειδοποιήσεις που περιγράφονται παραπάνω, όσο και οι μετρήσεις που λήφθηκαν από τον client σε πραγματικό χρόνο κατά τη διάρκεια μιας συνόδου εποπτείας, μπορούν με επιλογή του χρήστη να αποθηκεύονται και να ανακαλούνται για περαιτέρω ανάλυση.

Η υλοποίηση των λειτουργιών αυτών αναλύεται διεξοδικά στη συνέχεια του παρόντος κεφαλαίου, καθώς και στα κεφάλαια 5 και 7. Σημειώνουμε ότι στο εξής για περισσότερη ευκολία ο όρος *συσκευή*, απουσία επεξήγησης, θα αναφέρεται στο GAP Peripheral/GATT Server που ενσωματώνεται στη συσκευή προς εποπτεία.

4.3 Κατάσταση λειτουργίας συσκευής

Η κατάσταση λειτουργίας (status) της συσκευής προς εποπτεία κατέχει κεντρική θέση στο σύστημα, όχι μόνο επειδή είναι η πιο σημαντική πληροφορία, αλλά επιπρόσθετα γιατί είναι και η μοναδική που παρέχεται στον client τόσο σε σύνδεση, όσο και εκτός σύνδεσης. Γι' αυτόν ακριβώς το λόγο, το status προσδιορίζεται και ενημερώνεται από το ίδιο το peripheral και όχι από το central, πάντα όμως βάσει των τελευταίων ρυθμίσεων που του έχουν δοθεί από το central.

4.3.1 Δυνατές καταστάσεις

Ορίζουμε τρεις καταστάσεις, μία για κανονική λειτουργία και δύο για προβληματική λειτουργία, με διαφορετικά *επίπεδα ειδοποίησης* (*alert levels*). Ανά πάσα χρονική στιγμή η κατάσταση λειτουργίας της συσκευής προς εποπτεία μπορεί να είναι μία μόνο από αυτές. Το επίπεδο ειδοποίησης, οριζόμενο από το χρήστη, αναφέρεται σε μία μεμονωμένη παράμετρο, και ισοδυναμεί ακριβώς με την κατάσταση που εκείνη δηλώνει σε περίπτωση προβληματικής λειτουργίας. Αναλυτικά οι καταστάσεις και τα επίπεδα ειδοποίησης είναι:

- ***Καλή/κανονική λειτουργία (OK, 0x00):*** Όλα τα εποπτευόμενα μεγέθη της συσκευής βρίσκονται στις φυσιολογικές τους τιμές. Ενδέχεται να υπάρχουν μεγέθη που δε λαμβάνονται υπόψη και παραβιάζουν τις συνθήκες καλής λειτουργίας. Ισοδυναμεί με το επίπεδο ειδοποίησης *MONITOR_NONE*, δηλαδή την απενεργοποίηση της εποπτείας.

- ***Προειδοποίηση (Warning, 0x01):*** Πρόκειται για το χαμηλότερο επίπεδο ειδοποίησης για προβληματική λειτουργία. Μία ή περισσότερες από τις εποπτευόμενες παραμέτρους παραβιάζουν τις συνθήκες καλής λειτουργίας, και για κανένα από αυτά δεν έχει επιλεγεί το επίπεδο ειδοποίησης *Critical* (*MONITOR_CRITICAL*).

- ***Κρίσιμη (Critical, 0x02):*** Πρόκειται για το υψηλότερο alert level για προβληματική λειτουργία. Ένα τουλάχιστον από τα εποπτευόμενα μεγέθη για το οποίο έχει επιλεγεί το *Critical* (*MONITOR_CRITICAL*) alert level, παραβιάζει τις συνθήκες καλής λειτουργίας.

4.3.2 Ενημέρωση και κοινοποίηση κατάστασης

Οι υπηρεσίες που υλοποιούνται στην παρούσα εργασία χρησιμοποιούν δυαδική λογική (binary logic) για τον προσδιορισμό της κατάστασης λειτουργίας. Θα ήταν όμως δυνατό μια άλλη υπηρεσία που επεκτείνει το σύστημα να χρησιμοποιεί τεχνικές ασαφούς λογικής (fuzzy logic).

4.3.2.1 Ορισμός παραμέτρων εποπτείας

Σε κάθε περίπτωση, η κατάσταση λειτουργίας εξαρτάται από μία σειρά οριζόμενων από την υπηρεσία *παραμέτρων* προς εποπτεία. Καθεμία από αυτές αναπαρίσταται σε λογικό επίπεδο από μία μονάδα λογισμικού την οποία καλούμε *παρακολουθητή (monitor)*. Ένας παρακολουθητής συστεγάζει σε μία δομή την τρέχουσα κατάσταση της παραμέτρου (*OK, Warning* ή *Critical*) συρνατήσει ενός ή περισσότερων μετρώμενων μεγεθών, και τη λογική προσδιορισμού της κατάστασης αυτής βάσει της τρέχουσας τιμής της παραμέτρου και ενός συνόλου *τιμών ελέγχου (control values)*. Για παράδειγμα, μπορούμε να έχουμε μία μόνο τιμή ελέγχου ως κατώφλι, δύο τιμές ως πάνω και κάτω όρια, ή περισσότερες τιμές για πολυπλοκότερες λογικές ελέγχου.

Δίνοντας έναν τυπικό ορισμό όσων μόλις περιγράψαμε, μπορούμε να πούμε πως αν μία υπηρεσία περιλαμβάνει ένα σύνολο παρακολουθητών $M = \{m_1, m_2, \ldots, m_k\}$ και ένα σύνολο μετρώμενων μεγεθών $X = \{x_1, x_2, \ldots, x_n\}$, τότε άμεσα έπονται τα παρακάτω:

- Η τρέχουσα τιμή v_i της παραμέτρου του παρακολουθητή m_i υπολογίζεται από μία συνάρτηση $v_i = f(X)$, i = 1…k.

- Ο έλεγχος καλής λειτουργίας για μία παράμετρο με βάση την τρέχουσα τιμή της *v*, ένα διάνυσμα τιμών ελέγχου *C* και ένα επίπεδο ειδοποίησης *l*, πραγματοποιείται από μία *συνάρτηση ελέγχου* c, η οποία σε περίπτωση απόκλισης επιστρέφει το κατάλληλο επίπεδο ειδοποίησης:

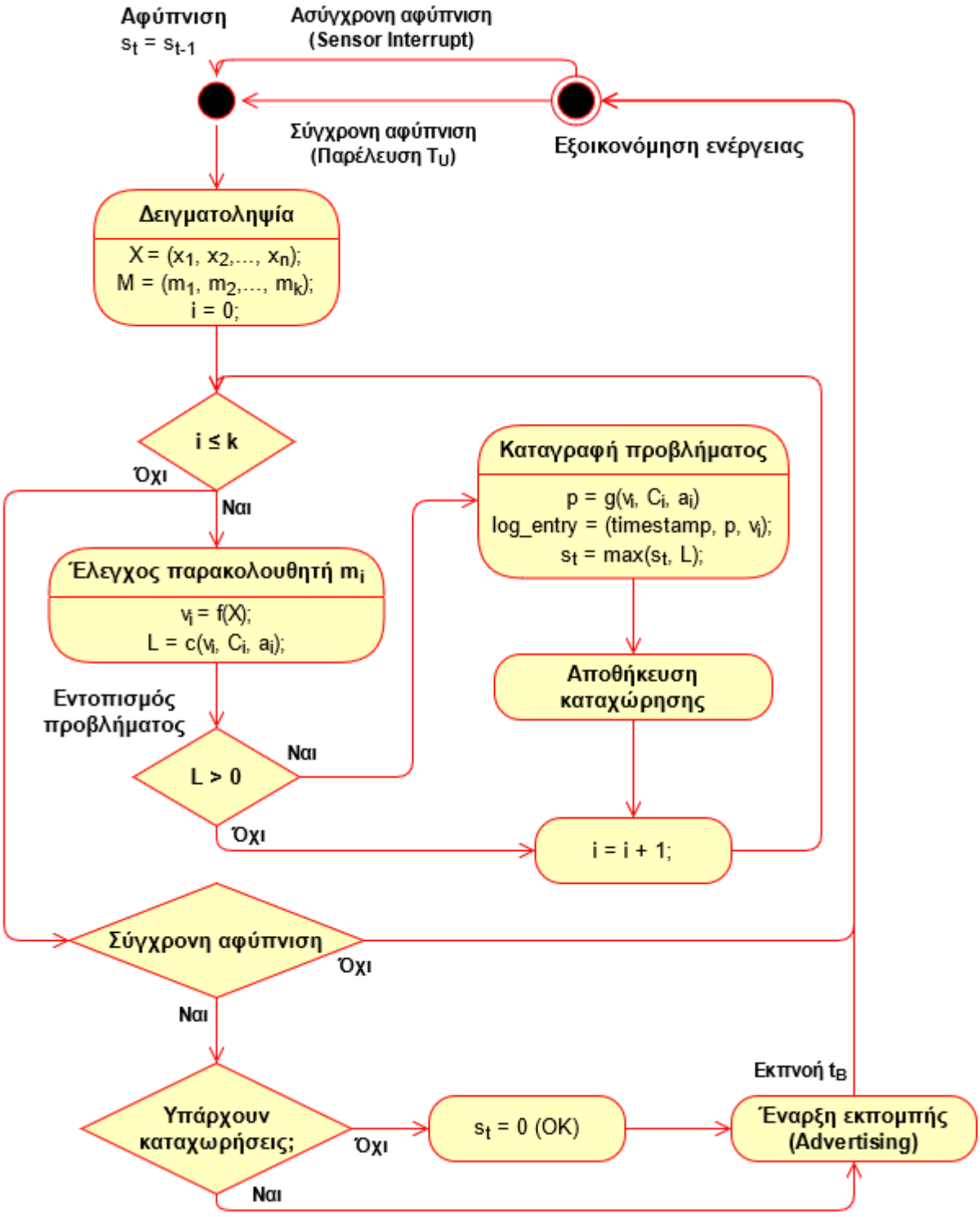
$$c(v, C, l) = \begin{cases} 0\ (OK), & \text{για ορθή λειτουργία} \\ l, & \text{διαφορετικά} \end{cases}$$

- Η κατάσταση $s_{i|t}$ του παρακολουθητή m_i κατά τη χρονική στιγμή ελέγχου t προκύπτει ως εξής: $s_{i|t} = max(s_{i|t-1}, c(v_i, C_i, a_i))$. Η αρχική κατάσταση για μία νέα σύνοδο εποπτείας είναι πάντα OK: $s_{i|0} = 0$.

- Η κατάσταση *s* συσκευής της συσκευής κατά τη χρονική στιγμή ελέγχου t προκύπτει ως εξής: $s_t = max_{i=1\ldots n}\{s_{i|t}\}$, λαμβάνοντας δηλαδή από τις τρέχουσες ενδείξεις όλων των παρακολουθητών τη μέγιστη.

4.3.2.2 Ενημέρωση και κοινοποίηση χωρίς σύνδεση

Η λειτουργία του peripheral κατά το advertising ακολουθεί την εξής FSM:



Σχήμα 4.1: FSM του peripheral εκτός σύνδεσης

Εκτός σύνδεσης, η ένδειξη κατάστασης της συσκευής αποτελεί μέρος του πακέτου διαφήμισης του peripheral, που σημαίνει ότι είναι διαθέσιμη σε κάθε

ενεργό scanner στο δίκτυο κατά τη διάρκεια του advertising. Με ορισμένες τις συνθήκες καλής λειτουργίας, ο χρήστης μπορεί συνεπώς να παρακολουθεί τις καταστάσεις πολλών συσκευών ταυτόχρονα, δίχως να απαιτείται σύνδεση σε κάποια από αυτές.

Περιοδικά, η συσκευή αφυπνίζεται και ενημερώνει την κατάστασή της, ελέγχοντας τους ενεργούς παρακολουθητές με δειγματοληψία των απαιτούμενων μεγεθών από τους αντίστοιχους αισθητήρες, εφόσον οι τελευταίοι βρίσκονται σε λειτουργία. Η *περίοδος ενημέρωσης (update interval)* T_U ορίζεται από το χρήστη μέσω του client. Σε περίπτωση εντοπισμού κάποιου προβλήματος, η συσκευή αποθηκεύει στη μνήμη εργασίας της μια καταχώρηση ιστορικού (log entry) με κατάλληλο κωδικό συμβάντος *p*. Οι καταχωρήσεις αυτές διατηρούνται στη μνήμη της συσκευής μέχρις ότου μεταφορτωθούν σε κάποιον συνδεδεμένο client. Εάν το ιστορικό στη μνήμη της συσκευής είναι άδειο, η συσκευή δηλώνει στο δίκτυο πως βρίσκεται σε κατάσταση κανονικής λειτουργίας (*OK*). Διαφορετικά, δηλώνει ως ένδειξη κατάστασης (status) το μέγιστο επίπεδο ειδοποίησης (*Warning* ή *Critical*) από τις καταχωρήσεις που βρίσκονται στη μνήμη.

Στη συνέχεια, η συσκευή ενσωματώνει την τρέχουσα ένδειξη κατάστασης της συσκευής στο πακέτο advertising και ξεκινά να διαφημίζεται στο δίκτυο για χρονικό διάστημα οριζόμενο από το χρήστη. Καλούμε το διάστημα αυτό *διάρκεια εκπομπής (broadcast timeout)*, t_B. Είναι προφανές από τα παραπάνω πως $t_B \leq T_U$. Για τον υπόλοιπο χρόνο $T_U - t_B$, η συσκευή τοποθετείται σε λειτουργία εξοικονόμησης ενέργειας (ή ύπνου, sleep mode). Τα t_B και T_U μπορούν να λαμβάνουν τιμές από 1 s έως δεκάδες λεπτά, ανάλογα με το χρονιστή (timer) της συσκευής.

Ενδέχεται βέβαια η συσκευή να αφυπνιστεί και ασύγχρονα, εξαιτίας αιτήματος διακοπής (interrupt request, IRQ) από αισθητήρα μετά από κάποιο συμβάν (απότομη μεταβολή μεγέθους κλπ). Κάτι τέτοιο έχει νόημα σε περιπτώσεις αισθητήρων με πολύ υψηλό ρυθμό δειγματοληψίας (π.χ. επιταχυνσιομέτρων), καθώς τότε η περιοδική δειγματοληψία (polling) δεν είναι αρκετά αποδοτική για να εντοπίσει ορισμένους τύπους συμβάντων.

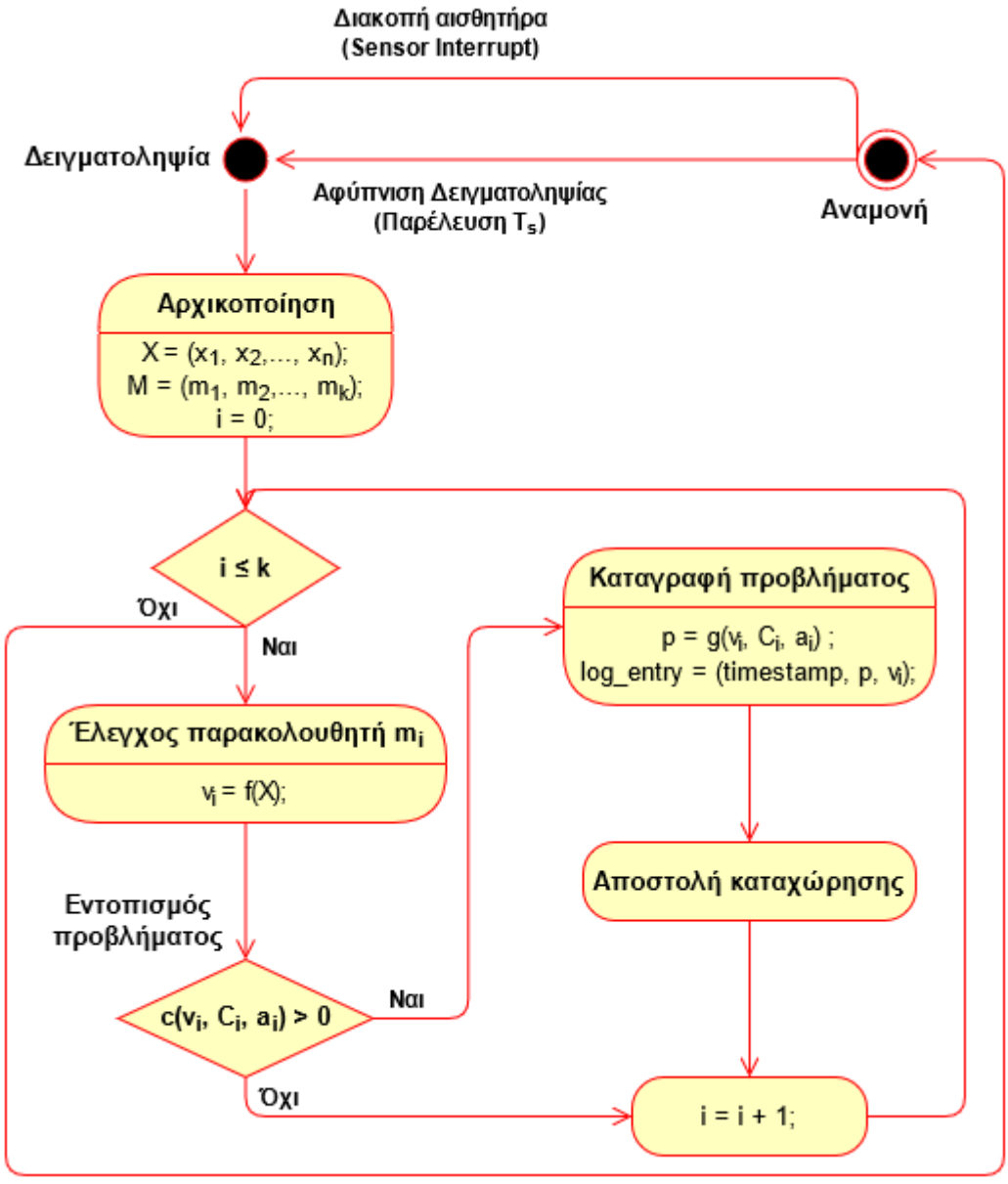
Μακροπρόθεσμα, η σύνδεση σε μία συσκευή εξυπηρετεί μόνο την αλλαγή των ρυθμίσεών της, καθώς και την real-time συλλογή μετρήσεων σε τακτά χρονικά διαστήματα. Οι συνδέσεις αναμένεται λοιπόν να είναι λιγότερο συχνές, και ταυτόχρονα πιο σύντομες. Καθώς μειώνεται τόσο η συχνότητα όσο και η διάρκεια των συνδέσεων με το peripheral, αυξάνεται θεαματικά η διάρκεια ζωής της μπαταρίας του.

4.3.2.3 Ενημέρωση σε σύνδεση

Ακολουθείται η ίδια διαδικασία που περιγράφηκε παραπάνω, με τη διαφορά ότι η δειγματοληψία γίνεται περιοδικά με περίοδο (sampling interval) T_S, κοινή για όλα τα μεγέθη των αισθητήρων, η οποία ορίζεται από το χρήστη για την τρέχουσα σύνοδο εποπτείας. Επιπλέον, κατά τη διάρκεια της συνόδου η πληροφορία για την κατάσταση της συσκευής δεν διαφημίζεται, αλλά διατηρείται στον client και ενημερώνεται από αυτόν βάσει των λαμβανόμενων εγγραφών, συνεπώς κάθε

εγγραφή ιστορικού που παράγεται αποστέλλεται ασύγχρονα στον client και δεν αποθηκεύεται στον server.

Καταλήγουμε έτσι σε μία αρκετά απλούστερη FSM για το peripheral:



Σχήμα 4.2: FSM του peripheral σε σύνδεση

4.4 Ορισμός Condition Monitoring Profile

Το BLE Profile του συστήματός μας, το οποίο ονοματίζουμε Condition Monitoring Profile (ή CM Profile), επιδιώκουμε να είναι επεκτάσιμο σε περισσότερα είδη αισθητήρων. Κάθε είδος αισθητήρα αντιστοιχεί σε ένα GATT Service, που περιλαμβάνει συγκεκριμένα GATT characteristics για τα μεγέθη που απαιτείται να μετρά. Πέραν τούτου, τα Services που ορίζονται με αυτόν τον τρόπο

επιτελούν ένα κοινό σύνολο λειτουργιών, ανεξάρτητο από τα είδη αισθητήρων που αντιπροσωπεύουν.

Μια συσκευή συμβατή με το CM Profile μπορεί να υλοποιεί το πολύ ένα από αυτά τα Services, το οποίο καθορίζει και τον τύπο της συσκευής που αναγνωρίζεται από τον client. Μαζί με το κύριο αυτό CM Service, η συσκευή οφείλει να περιλαμβάνει μια υλοποίηση του Battery Service. Το Battery Service είναι ένα από τα θεμελιώδη Services που έχουν ορισθεί από το Bluetooth SIG, και επιτρέπει την παρακολούθηση της κατάστασης της μπαταρίας μίας συσκευής.

4.4.1 Γενική δομή του κύριου Condition Monitoring Service

Ένα CM Service γενικά αποτελείται από τα παρακάτω characteristics:

UUID (hex)	Περιγραφή	Format	Μήκος (bytes)	Ιδιότητες
2D86-686A-53DC-25B3-0C4A-F0E1-0C8D-EE20	Data In	Byte[]	1…112	Write (Request)
…	Sensor Value 1	Float[L_1]	$4L_1$	Read, Notify
…	Sensor Value 2	Float[L_2]	$4L_2$	Read, Notify
…	…	…	…	…
…	Sensor Value N	Float[L_N]	$4L_N$	Read, Notify
85DA-2210-FBE4-1BA6-9775-0AEE-01BF-1164	Data Out	Byte[]	1…112	Notify

Πίνακας 4.3: Γενική δομή CM Service

Κάθε CM service περιλαμβάνει υποχρεωτικά τα χαρακτηριστικά *Data In* και *Data Out*, για ανταλλαγή μηνυμάτων σε επίπεδο εφαρμογής και γενικού σκοπού είσοδο και έξοδο (I/O) δεδομένων προς και από τον server αντίστοιχα. Αυτά περιγράφονται διεξοδικά στις υποενότητες 4.5.1, 4.5.2.

Η λειτουργικότητα του CM Service ολοκληρώνεται από ένα ή περισσότερα χαρακτηριστικά *Sensor Value*, καθένα από τα οποία αντιπροσωπεύει την τιμή ενός μεγέθους, όπως αυτό μετράται από τον ενσωματωμένο στη συσκευή αισθητήρα. Αν έχουμε N τέτοια μεγέθη, τότε κάθε τιμή V_i , i = 1…N, διάστασης L_i, αποθηκεύεται στη βάση ως ένα διάνυσμα από L_i δεκαδικούς αριθμούς. Οι αριθμοί αυτοί είναι κινητής υποδιαστολής μήκους 32 bit (FLOAT) σύμφωνα με το πρότυπο IEEE-11073, συνεπώς το μήκος της δυαδικής αναπαράστασης της τιμής V_i είναι $4L_i$ bytes.

4.4.2 Environmental Sensor Service

Ως *περιβαλλοντικός αισθητήρας (environmental sensor)* αναφέρεται συνήθως μία διάταξη αποτελούμενη από έναν αισθητήρα ή περισσότερους αισθητήρες με

δυνατότητα μέτρησης παραμέτρων του περιβάλλοντος όπως είναι η θερμοκρασία, η σχετική υγρασία, η ατμοσφαιρική πίεση, η συγκέντρωση αερίων (CO_2, CO, CH_4 κ.ά.) και αιωρούμενων σωματιδίων (particulate matter, PM10 και PM2.5) στην ατμόσφαιρα κλπ.

Στην υλοποίησή μας θα περιοριστούμε στη μέτρηση δύο μόνο βασικών μεγεθών, της θερμοκρασίας και της σχετικής υγρασίας.

4.4.2.1 Ορισμός βάσης δεδομένων GATT

Οι τιμές της θερμοκρασίας και της σχετικής υγρασίας είναι βαθμωτές. Ο πίνακας της παραγράφου 4.4.1 για το ES Service συμπληρώνεται λοιπόν ως εξής:

UUID (hex)	Περιγραφή	Format	Μήκος (bytes)	Ιδιότητες
2D86-686A-53DC-25B3-0C4A-F0E1-0C8D-EE20	Data In	Byte[]	1…112	Write (Request)
1500-5991-B131-3396-014C-664C-9867-B917	Temperature	Float	4	Read, Notify
6EB6-75AB-8BD1-1B9A-7444-621E-52EC-6823	Relative Humidity	Float	4	Read, Notify
85DA-2210-FBE4-1BA6-9775-0AEE-01BF-1164	Data Out	Byte[]	1…112	Notify

Πίνακας 4.4: Δομή ES Service

Σημειώνουμε πως η τιμή της θερμοκρασίας δίνεται πάντα σε βαθμούς Κελσίου (°C), ενώ αυτή της σχετικής υγρασίας είναι καθαρός αριθμός που δίνεται ως ποσοστό επί τοις εκατό (%).

4.4.2.2 Εποπτεία με χρήση του ES Service

Το ES Service παρέχει τους παρακάτω παρακολουθητές για τον εντοπισμό ενδεχόμενης δυσλειτουργίας της συσκευής:

- ***Θερμοκρασία (Temperature)***

Περιγραφή: Κάθε μέτρηση θερμοκρασίας που λαμβάνεται κατά την περιοδική δειγματοληψία συγκρίνεται με τα όρια καλής λειτουργίας.

Μεγέθη: Θερμοκρασία

Τιμές ελέγχου: Άνω και κάτω όρια, v_H και v_L, οριζόμενα από το χρήστη.

Υπολογισμός τιμής: $v = T$.

Συνθήκη ορθής λειτουργίας: $v_L \leq v \leq v_H$.

- ***Μεταβολή θερμοκρασίας (Temperature Change)***

Περιγραφή: Υπολογίζεται ο μέσος ρυθμός μεταβολής της θερμοκρασίας, και στη συνέχεια συγκρίνεται με τα όρια καλής λειτουργίας.

Μεγέθη: Θερμοκρασία

Τιμές ελέγχου: Άνω και κάτω όρια, v_H και v_L, οριζόμενα από το χρήστη.

Υπολογισμός τιμής: $v = f_s(T_k - T_{k-1})$, όπου f_s η συχνότητα δειγματοληψίας σε Hz.

Συνθήκη ορθής λειτουργίας: $v_L \leq v \leq v_H$.

- ***Σχετική υγρασία (Humidity)***

Περιγραφή: Κάθε μέτρηση σχετικής υγρασίας που λαμβάνεται κατά την περιοδική δειγματοληψία συγκρίνεται με τα όρια καλής λειτουργίας.

Μεγέθη: Σχετική υγρασία H, ως ποσοστό επί τοις εκατό (% RH).

Τιμές ελέγχου: Άνω και κάτω όρια, v_H και v_L, οριζόμενα από το χρήστη.

Υπολογισμός τιμής: $v = H$.

Συνθήκη ορθής λειτουργίας: $v_L \leq v \leq v_H$.

- ***Μεταβολή υγρασίας***

Περιγραφή: Υπολογίζεται ο μέσος ρυθμός μεταβολής της σχετικής υγρασίας, και στη συνέχεια συγκρίνεται με τα όρια καλής λειτουργίας.

Μεγέθη: Σχετική υγρασία H, ως ποσοστό επί τοις εκατό (% RH)

Τιμές ελέγχου: Άνω και κάτω όρια, v_H και v_L, οριζόμενα από το χρήστη.

Υπολογισμός τιμής: $v = f_s(H_k - H_{k-1})$, όπου f_s η συχνότητα δειγματοληψίας σε Hz.

Συνθήκη ορθής λειτουργίας: $v_L \leq v \leq v_H$.

Και στις δύο περιπτώσεις, οι υπολογιστικές ενέργειες που απαιτούνται είναι απλές, συνεπώς εκτελούνται απευθείας στον application processor του coin sensor.

4.4.3 Inertial Measurement Service

Μια μονάδα αδρανειακών μετρήσεων (inertial measurement unit, IMU) αποτελείται από ένα σύνολο αισθητήρων με τη βοήθεια των οποίων μπορεί να προσδιορίζεται η μετατόπιση στο χώρο και η περιστροφή ενός σώματος. Γι' αυτό απαιτείται η παρουσία τουλάχιστον ενός επιταχυνσιομέτρου κι ενός γυροσκοπίου. Επιπλέον, αυτά μπορούν να συνοδεύονται από μαγνητόμετρο για τη μέτρηση του μαγνητικού πεδίου.

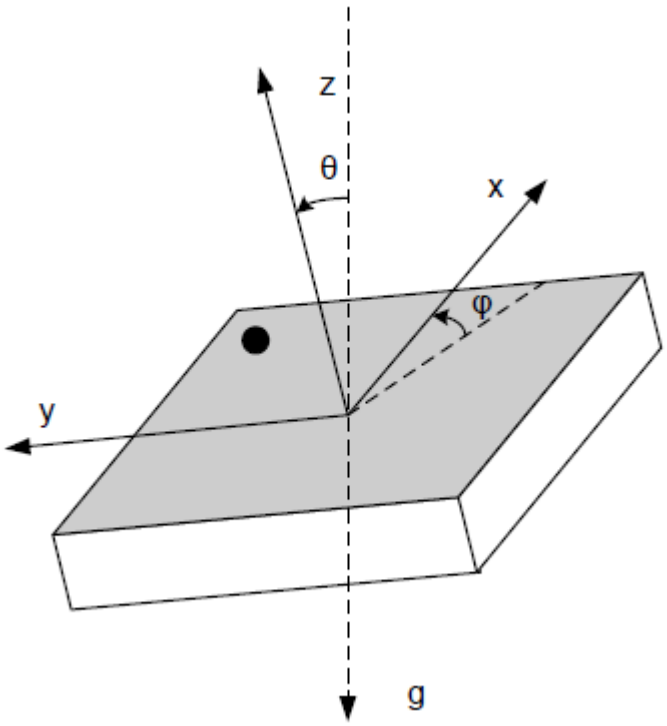
4.4.3.1 Μέτρηση επιτάχυνσης

Το μέγεθος που μετράται από ένα επιταχυνσιόμετρο δεν ταυτίζεται με τη γραμμική επιτάχυνση του σώματος. Πρόκειται για τη *δύναμη επιτάχυνσης* (*gravitational force equivalent, g-force*), δηλαδή τη συνολική δύναμη ανά μονάδα μάζας που εφαρμόζεται σε ένα σώμα, εξαιρουμένων των βαρυτικών και ηλεκτρομαγνητικών δυνάμεων. Ισοδύναμα, θεωρώντας το σύστημα αναφοράς του σώματος ως μη αδρανειακό και επιταχυνόμενο με την επιτάχυνση της βαρύτητας $\vec{g}$, μπορούμε να ορίσουμε τη δύναμη επιτάχυνσης ως την επιτάχυνση του σώματος ως προς το σύστημα αναφοράς του. Προφανώς το μέγεθος αυτό έχει διαστάσεις επιτάχυνσης και σχετίζεται άμεσα με την πραγματική τιμή της επιτάχυνσης. Αγνοώντας τις ηλεκτρομαγνητικές δυνάμεις, και δεδομένου ότι εξαιτίας της επίδρασης της βαρύτητας το σώμα δέχεται ανά πάσα στιγμή δύναμη ανά μονάδα μάζας ίση με την επιτάχυνση της βαρύτητας $\vec{g}$, η μετρώμενη g-force $\overrightarrow{\alpha_p}$ προκύπτει με αφαίρεση της επιτάχυνσης της βαρύτητας από την επιτάχυνση $\vec{\alpha}$ του σώματος:

$$\overrightarrow{F_p} = \Sigma\vec{F} - m\vec{g} \Rightarrow m\overrightarrow{\alpha_p} = m\vec{a} - m\vec{g} \Rightarrow \overrightarrow{\alpha_p} = \vec{a} - \vec{g}$$

Έτσι, μια συσκευή σε ελεύθερη πτώση θα παρουσιάζει $\overrightarrow{\alpha_p} = \vec{0}$. Εάν η συσκευή ισορροπεί σε κάποια θέση, θα παρουσιάζει $\overrightarrow{\alpha_p} = -\vec{g}$.

Όμως τα παραπάνω μεγέθη μετρώνται στους άξονες του συστήματος αναφοράς του σώματος. Έτσι, ενώ η επιτάχυνση της βαρύτητας παραμένει σταθερή, οι συνιστώσες της στους άξονες μεταβάλλονται σε συνάρτηση με τον προσανατολισμό του σώματος. Εδώ λοιπόν η μέτρηση της $\overrightarrow{\alpha_p}$ παρέχει πολύτιμη πληροφορία, καθώς κρατώντας το σώμα σε ηρεμία σε συγκεκριμένη θέση είναι δυνατός ο προσδιορισμός του προσανατολισμού του, όπως φαίνεται στο σχήμα που ακολουθεί:



Σχήμα 4.5: Το σύστημα αναφοράς του σώματος

Εφόσον το σώμα ισορροπεί, είναι $\overrightarrow{\alpha_p} = -\vec{g}$. Για παράδειγμα, αν η συσκευή ηρεμεί τοποθετημένη παράλληλα στη γη με την εμπρόσθια όψη της προς τα πάνω, τότε ο άξονας z είναι κάθετος στη γη και $\theta = 0$, άρα $\overrightarrow{\alpha_p} = (0, 0, g)$. Ο προσανατολισμός του σώματος προσδιορίζεται από τις γωνίες θ, φ που σχηματίζει η $\overrightarrow{\alpha_p}$ με τους άξονες του συστήματος αναφοράς, οι οποίες υπολογίζονται από τις συνιστώσες της:

$$\left.\begin{aligned} a_{px} &= g \sin\theta \cos\varphi \\ a_{py} &= -g \sin\theta \sin\varphi \end{aligned}\right\} \Rightarrow \varphi = -\tan^{-1}\frac{\alpha_{py}}{\alpha_{px}}$$

$$a_{pz} = g\cos\theta \Rightarrow \theta = \cos^{-1}\frac{a_{pz}}{g}$$

Αν το σώμα εκτελεί οποιαδήποτε κίνηση, για την εξαγωγή της πραγματικής επιτάχυνσης $\vec{\alpha}$ από την δύναμη επιτάχυνσης $\overrightarrow{\alpha_p}$ απαιτείται η γνώση ανά πάσα χρονική στιγμή των συνιστωσών της επιτάχυνσης της βαρύτητας στους τρεις άξονες, ισοδύναμα των γωνιών θ, φ. Τότε όμως η $\overrightarrow{\alpha_p}$ γενικά θα περιλαμβάνει μία συνιστώσα κάθετη στην $\vec{g}$, και θα σχηματίζει με τους άξονες γωνίες $\theta' \neq \theta$, $\varphi' \neq \varphi$. Η εύρεση των συνιστωσών της $\vec{g}$ θα μπορούσε θεωρητικά να επιτευχθεί με ολοκλήρωση της γωνιακής ταχύτητας που μετράται από το γυροσκόπιο, ξεκινώντας από κατάσταση ηρεμίας με γνωστές αρχικές συνθήκες για τις θ και φ, στην πράξη

όμως με την εξέλιξη της κίνησης ο θόρυβος πολλαπλασιάζεται με υπερβολικά ταχείς ρυθμούς και τελικά η μέθοδος αποκλίνει.

4.4.3.2 Ορισμός βάσης δεδομένων GATT

Στην υλοποίησή μας θα περιοριστούμε στη μέτρηση της δύναμης επιτάχυνσης και της γωνιακής ταχύτητας στον τρισδιάστατο χώρο. Οι τιμές των δύο αυτών μεγεθών είναι διανύσματα τριών στοιχείων, ενός για καθέναν από τους άξονες x, y, z. Ο πίνακας της παραγράφου 4.4.1 για το IM Service συμπληρώνεται λοιπόν ως εξής:

UUID (hex)	Περιγραφή	Format	Μήκος (bytes)	Ιδιότητες
2D86-686A-53DC-25B3-0C4A-F0E1-0C8D-EE20	Data In	Byte[]	1…112	Write (Request)
AC49-F474-8607-10C2-3BA1-0311-5E45-4C22	G-Force	Float[3]	12	Read, Notify
7081-5332-4691-A218-5534-CCFA-00B1-03FD	Angular Rate	Float[3]	12	Read, Notify
85DA-2210-FBE4-1BA6-9775-0AEE-01BF-1164	Data Out	Byte[]	1…112	Notify

Πίνακας 4.6: Δομή IM Service

Σημειώνουμε πως η δύναμη επιτάχυνσης δίνεται πάντα σε μονάδες επιτάχυνσης της βαρύτητας (g), ενώ η γωνιακή ταχύτητα μετράται σε μοίρες ανά δευτερόλεπτο (°/s ή degrees per second, dps).

4.4.3.3 Εποπτεία με χρήση του IM Service

Η περιοδική δειγματοληψία απαιτεί υψηλούς ρυθμούς, μειώνοντας δραματικά το χρόνο που η συσκευή (τόσο οι αισθητήρες, όσο και ο application processor) μπορεί να παραμείνει σε κατάσταση εξοικονόμησης ενέργειας. Καθίσταται ασύμφορη για ένα ενσωματωμένο σύστημα τροφοδοτούμενο από μπαταρία. Οι αλγόριθμοι εντοπισμού κίνησης και δόνησης επιβάλλεται λοιπόν να υλοποιούνται πάνω στο ίδιο το σύστημα αισθητήρων. Με αυτόν τον τρόπο τόσο ο application processor όσο και οι αισθητήρες μπορούν να παραμένουν σε κατάσταση εξοικονόμησης ενέργειας, και μέσω IRQ να αφυπνίζουν ασύγχρονα τον application processor για την πραγματοποίηση των απαραίτητων ενεργειών (αποθήκευση). Ο αισθητήρας του IM Service παράγει λοιπόν δύο ειδών IRQ, τα οποία τροφοδοτούν τους εξής παρακολουθητές:

- ***Εντοπισμός κίνησης (Motion detection)***

Περιγραφή: Ενεργοποιείται κατά τον εντοπισμό μετατόπισης σε οποιονδήποτε άξονα. Ο εντοπισμός γίνεται με σύγκριση του μέτρου κάθε συνιστώσας της δύναμης επιτάχυνσης με ένα κατώφλι.

Μεγέθη: Δύναμη επιτάχυνσης (α_{px}, a_{py}, α_{pz}), σε μονάδες επιτάχυνσης της βαρύτητας (g).

Τιμές ελέγχου: Κατώφλι v_T, οριζόμενο από το χρήστη.

Υπολογισμός τιμής: $(v_x, v_y, v_z) = (|\alpha_{px}|, |\alpha_{py}|, |\alpha_{pz}|)$.

Συνθήκη ορθής λειτουργίας: $v_x \leq v_T$, $v_y \leq v_T$ και $v_z \leq v_T$.

- ***Εντοπισμός δόνησης (Shock detection detection)***

Περιγραφή: Ενεργοποιείται κατά τον εντοπισμό απότομης αλλαγής της g-force, που υποδηλώνει την ύπαρξη κάποιας δόνησης. Ο εντοπισμός γίνεται με σύγκριση του μέσου ρυθμού μεταβολής του μέτρου κάθε συνιστώσας της δύναμης επιτάχυνσης με ένα κατώφλι.

Μεγέθη: Δύναμη επιτάχυνσης (α_{px}, a_{py}, α_{pz}), σε μονάδες επιτάχυνσης της βαρύτητας (g).

Τιμές ελέγχου: Κατώφλι v_T, οριζόμενο από το χρήστη.

Υπολογισμός τιμής: $(v_x, v_y, v_z) = f_s[(|\alpha_{px}|, |\alpha_{py}|, |\alpha_{pz}|)_k - (|\alpha_{px}|, |\alpha_{py}|, |\alpha_{pz}|)_{k-1}]$, όπου f_s η συχνότητα δειγματοληψίας σε Hz.

Συνθήκη ορθής λειτουργίας: $v_x \leq v_T$, $v_y \leq v_T$ και $v_z \leq v_T$.

4.5 Λειτουργία του CM Service

Αναλύοντας τη λειτουργία του CM Service θα καταδείξουμε τον τρόπο με τον οποίο η εφαρμογή πελάτη μπορεί, αξιοποιώντας τα χαρακτηριστικά του, να αλληλεπιδράσει με μία συσκευή, καθώς και να αντλήσει την επιθυμητή πληροφορία για την κατάσταση και τις ρυθμίσεις της.

4.5.1 Είσοδος δεδομένων - Control Point

Όπως συνηθίζεται σε συστήματα BLE, ορίζουμε ένα characteristic (*Data In*) μέσω του οποίου ο GATT client μπορεί να στέλνει στον GATT server εντολές συνοδευόμενες από δεδομένα. Η δομή ενός πακέτου εισόδου είναι η εξής:

Σχήμα 4.7: Δομή IM Service

Ο κωδικός εντολής (opcode) καταλαμβάνει ένα byte, και τα υπόλοιπα 111 bytes είναι διαθέσιμα για μετάδοση δεδομένων. Ο opcode καθορίζει την εντολή που θα εκτελεστεί από τον server. Παρακάτω παρουσιάζονται συνοπτικά οι εντολές, ενώ στη συνέχεια αναλύεται η καθεμιά ξεχωριστά:

Opcode	**Όνομα**	**Περιγραφή**
0x00	*CP_SAMPLING_START*	Έναρξη δειγματοληψίας
0x01	*CP_SAMPLING_STOP*	Διακοπή δειγματοληψίας
0x02	*CP_SAMPLING_SET_INTERVAL*	Αλλαγή περιόδου δειγματοληψίας
0x03	*CP_RETRIEVE_LOG*	Λήψη των εγγραφών ιστορικού
0x04	*CP_UPDATE_RTC*	Ενημέρωση ημερομηνίας και ώρας
0x05	*CP_GET_DEVICE_INFO*	Λήψη πληροφοριών υλικού
0x06	*CP_GET_DEVICE_CONFIG*	Λήψη τρεχουσών ρυθμίσεων
0x07	*CP_SET_DEVICE_CONFIG*	Ενημέρωση ρυθμίσεων

Πίνακας 4.8: Εντολές προς τον εξυπηρετητή

- ***CP_SAMPLING_START***

Λειτουργία: Ο server ξεκινά τη διαδικασία περιοδικής δειγματοληψίας. Μετά τη λήψη ενός δείγματος από κάθε ενεργό αισθητήρα, ενημερώνονται οι τιμές των αντίστοιχων χαρακτηριστικών και αποστέλλονται στον client μέσω ειδοποιήσεων GATT.

Δεδομένα εισόδου: Όχι.

Πακέτα απόκρισης: Κανένα.

- ***CP_SAMPLING_STOP***

Λειτουργία: Ο server σταματά την περιοδική δειγματοληψία.

Δεδομένα εισόδου: Όχι.

Πακέτα απόκρισης: Κανένα.

- ***CP_SAMPLING_SET_INTERVAL***

Λειτουργία: Ο server αλλάζει την περίοδο δειγματοληψίας. Η νέα περίοδος ισχύει μετά την ολοκλήρωση της τελευταίας προγραμματισμένης λήψης δείγματος.

Δεδομένα εισόδου: Η τιμή της περιόδου δειγματοληψίας σε μορφή απροσήμαστου ακεραίου. Καθώς για κάθε διαφορετικό τύπο αισθητήρα αναμένεται να ενδείκνυται περίοδος δειγματοληψίας συγκεκριμένης τάξης μεγέθους (ms ή s), η τιμή δίνεται σε μονάδες χρονοκαθυστέρησης του χρονιστή της συσκευής.

Πακέτα απόκρισης: Κανένα.

- ***CP_RETRIEVE_LOG***

Λειτουργία: Ο server αποστέλλει στον client όλες τις εγγραφές ιστορικού που είναι αποθηκευμένες στην τοπική του μνήμη, μέσω διαδοχικών GATT notifications.

Δεδομένα εισόδου: Όχι.

Πακέτο απόκρισης: Ένα πακέτο ανά εγγραφή. Η εγγραφή σειριοποιείται και η προκύπτουσα ροή από bytes (bytestream) ενσωματώνεται στο πακέτο.

- ***CP_UPDATE_RTC***

Λειτουργία: Ο server συγχρονίζει την τοπική του ημερομηνία και ώρα με αυτήν του client.

Δεδομένα εισόδου: Ημερομηνία και ώρα από τον client, σε κατάλληλα σειριοποιημένη μορφή.

Πακέτα απόκρισης: Κανένα.

- ***CP_GET_DEVICE_INFO***

Λειτουργία: Ο server αποστέλλει στον client σειριοποιημένη μία σύνθετη εγγραφή (DIS) με πληροφορίες για το υλικό της συσκευής (προδιαγραφές αισθητήρων, χρονιστής κλπ).

Δεδομένα εισόδου: Όχι.

Πακέτα απόκρισης: Ένα πακέτο με ενσωματωμένη την εγγραφή στο payload.

- ***CP_GET_DEVICE_CONFIG***

Λειτουργία: Ο server αποστέλλει στον client σειριοποιημένη μία σύνθετη εγγραφή (DCS) που περιέχει το τρέχον σύνολο ρυθμίσεων της συσκευής (ρυθμίσεις αισθητήρων και εποπτείας, συνθήκες κανονικής λειτουργίας κλπ).

Δεδομένα εισόδου: Όχι.

Πακέτα απόκρισης: Ένα πακέτο με ενσωματωμένη την εγγραφή στο payload.

- ***CP_SET_DEVICE_CONFIG***

Λειτουργία: Ο server εφαρμόζει τις ρυθμίσεις που του δίνονται από τον client.

Δεδομένα εισόδου: Μια σύνθετη εγγραφή (DCS) που περιέχει το νέο σύνολο ρυθμίσεων της συσκευής.

Πακέτα απόκρισης: Κανένα.

4.5.2 Έξοδος δεδομένων

Μέσω του characteristic *Data Out* ο GATT server είναι σε θέση να στέλνει δεδομένα στον GATT client, είτε κατόπιν αιτήματος του τελευταίου, είτε ασύγχρονα με δική του πρωτοβουλία (server-initiated). Η δομή του πακέτου εξόδου είναι η ίδια με αυτήν των πακέτων εισόδου που παρουσιάστηκε προηγουμένως.

Ένα πακέτο απόκρισης χρησιμοποιεί το ίδιο ακριβώς opcode με αυτό του αιτήματος που το προκάλεσε (ειδικότερα, ένα πακέτο με opcode *CP_RETRIEVE_LOG* μπορεί να αποσταλεί όχι μόνο κατόπιν αιτήματος, αλλά και ως επακόλουθο της δημιουργίας μιας νέας εγγραφής ιστορικού). Έτσι, τα πακέτα απόκρισης χρησιμοποιούν ένα υποσύνολο από τα opcodes που ορίσαμε στην προηγούμενη παράγραφο. Αυτά παρουσιάζονται συνοπτικά στον παρακάτω πίνακα:

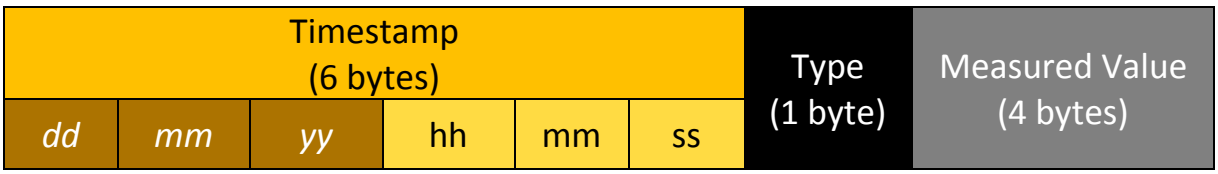
Opcode	Όνομα	Έξοδος δεδομένων
0x03	*CP_RETRIEVE_LOG*	Με πρωτοβουλία του server/ Κατόπιν αιτήματος
0x05	*CP_GET_DEVICE_INFO*	Κατόπιν αιτήματος
0x06	*CP_GET_DEVICE_CONFIG*	Κατόπιν αιτήματος

Πίνακας 4.9: Τύποι πακέτων απόκρισης

Σημειώνουμε πως για την επιτυχή λήψη των δεδομένων από τον client απαιτείται να έχουν ενεργοποιηθεί οι ειδοποίησεις GATT στο χαρακτηριστικό *Data Out* με κατάλληλη τροποποίηση του περιγραφέα *Client Configuration Descriptor*.

4.5.3 Καταγραφή ιστορικού

Το ιστορικό συμβάντων (event log) που διατηρείται στη μνήμη της συσκευής αποτελείται από εγγραφές της παρακάτω μορφής:



Σχήμα 4.10: Δομή εγγραφής ιστορικού

Κάθε εγγραφή συντίθεται από τα εξής πεδία:

- ***Timestamp:*** Η ημερομηνία και ώρα του συμβάντος, μορφοποιημένη ως "*dd*/*mm*/20*yy* *hh*:*mm*:*ss*". Τα bytes *dd* (ημέρα μήνα, 1-31), *mm* (μήνας, 1-12), *yy* (έτος, 0-99), *hh* (ώρες, 0-23), *mm* (λεπτά, 0-59) και *ss* (δευτερόλεπτα, 0-59) είναι κωδικοποιημένα σε BCD. Για παράδειγμα, η Δευτέρα 6 Μαΐου 2019 04:15:33 μορφοποιείται ως "06/05/2019 04:15:33", άρα σειριοποιείται στον πίνακα [0x06, 0x05, 0x19, 0x04, 0x15, 0x33].

- ***Type:*** Ο τύπος του συμβάντος. Για τα ES και IM Services ορίζουμε τα εξής συμβάντα:

Type	**Συμβάν**	**Περιγραφή**
0x00	*TEMP_HIGH*	Τιμή θερμοκρασίας υψηλότερη από το άνω όριο καλής λειτουργίας
0x01	*TEMP_LOW*	Τιμή θερμοκρασίας χαμηλότερη από το κάτω όριο καλής λειτουργίας
0x02	*HUMID_HIGH*	Τιμή υγρασίας υψηλότερη από το άνω όριο καλής λειτουργίας
0x03	*HUMID_LOW*	Τιμή υγρασίας χαμηλότερη από το κάτω όριο καλής λειτουργίας
0x04	*TEMP_CHANGE_HIGH*	Μεταβολή θερμοκρασίας με ρυθμό μεγαλύτερο από το άνω όριο καλής λειτουργίας
0x05	*TEMP_CHANGE_LOW*	Μεταβολή θερμοκρασίας με ρυθμό μικρότερο από το κάτω όριο καλής λειτουργίας
0x06	*HUMID_CHANGE_HIGH*	Μεταβολή υγρασίας με ρυθμό μεγαλύτερο από το άνω όριο καλής λειτουργίας
0x07	*HUMID_CHANGE_LOW*	Μεταβολή υγρασίας με ρυθμό μικρότερο από το κάτω όριο καλής λειτουργίας
0x08	*MOTION_DETECT*	Εντοπισμός κίνησης
0x09	*SHOCK_DETECT*	Εντοπισμός δόνησης

Πίνακας 4.11: Κωδικοί συμβάντων

- ***Measured Value:*** Για το ES Service, είναι η τιμή της μέτρησης που παρέβη τα όρια καλής λειτουργίας με ακρίβεια ενός δεκαδικού ψηφίου. Για το IM Service, πρόκειται για την τιμή του αντίστοιχου ορίου με ακρίβεια δύο δεκαδικών ψηφίων.

4.5.4 Πληροφορίες συσκευής – DIS

Το *Device Information Structure* (DIS) είναι μια δομή δεδομένων που παρέχει στον client την απαραίτητη πληροφορία για τις δυνατότητες του υλικού του server. Περιλαμβάνει απαραίτητα τα εξής πεδία, εκ των οποίων τα δύο πρώτα είναι κοινά για όλα τα CM Services και αφορούν μόνο στον application processor της συσκευής, ενώ το τελευταίο εξαρτάται από το εκάστοτε υλοποιούμενο Service:

- ***Μονάδα χρονιστή (Timer Unit):*** Ο χρόνος σε ms που αντιστοιχεί σε μία περίοδο του χρονιστή της συσκευής, δηλαδή ο χρόνος που απαιτείται για να αυξηθεί η τιμή του χρονιστή κατά μία μονάδα.

- ***Μέγιστη τιμή χρονιστή (Timer Max):*** Δίνεται σε περιόδους χρονιστή, και εκφράζει τη μέγιστη υποστηριζόμενη χρονοκαθυστέρηση.

- ***Τιμές ρυθμίσεων (Setting Values):*** Συγκεντρώνει σε μία συμβολοσειρά τις λίστες των υποστηριζόμενων τιμών για τις διαθέσιμες προς ρύθμιση παραμέτρους των αισθητήρων, ώστε αυτές να μπορούν να εμφανίζονται στα μενού της εφαρμογής πελάτη σε μορφή φιλική προς το χρήστη. Εντός της λίστας για μία συγκεκριμένη παράμετρο, οι δυνατές τιμές διαχωρίζονται με τον χαρακτήρα "," (κόμμα), και οι λίστες με τη σειρά τους οριοθετούνται από τον χαρακτήρα "_" (κάτω παύλα). Για παράδειγμα, αν τα σύνολα τιμών για δύο παραμέτρους είναι αντίστοιχα {1, 4, 11} και {0, 50, 100, 200}, η προκύπτουσα συμβολοσειρά είναι "*1,4,11_0,50,100,200*".

4.5.4.1 Ορισμός για το ES Service

Για το Environmental Sensor Service, το DIS λαμβάνει την εξής μορφή:

Πεδίο	**Τύπος**	**Μήκος (bytes)**	**Περιγραφή**
Timer Unit	Integer	2	Μονάδα χρονοκαθυστέρησης σε ms
Timer Max	Integer	2	Μέγιστη χρονοκαθυστέρηση
Sensor Information: Thermometer			
Setting Values: Bit resolution	String[]	Μεταβλητό	Λίστα όλων των υποστηριζόμενων αναλύσεων δειγματοληψίας για τη θερμοκρασία, σε bits/δείγμα

Sensor Information: Hygrometer			
Setting Values: Bit resolution	String[]	Μεταβλητό	Λίστα όλων των υποστηριζόμενων αναλύσεων δειγματοληψίας για τη σχετική υγρασία, σε bits/δείγμα

Πίνακας 4.12: Ορισμός DIS για το ES Service

4.5.4.2 Ορισμός για το IM Service

Για το Inertial Measurement Service, το DIS λαμβάνει την εξής δομή:

Πεδίο	**Τύπος**	**Μήκος (bytes)**	**Περιγραφή**
Timer Unit	Integer	2	Μονάδα χρονοκαθυστέρησης σε ms
Timer Max	Integer	2	Μέγιστη χρονοκαθυστέρηση
Sensor Information: Accelerometer			
Setting Value Labels: Range	String[]	Μεταβλητό	Λίστα όλων των υποστηριζόμενων τιμών εύρους μέτρησης για τη δύναμη επιτάχυνσης
Sensor Information: Gyroscope			
Setting Value Labels: Range	String[]	Μεταβλητό	Λίστα όλων των υποστηριζόμενων τιμών εύρους μέτρησης για τη γωνιακή ταχύτητα

Πίνακας 4.13: Ορισμός DIS για το IM Service

4.5.5 Ρυθμίσεις συσκευής – DCS

Το *Device Configuration Structure* (DCS) είναι μια δομή δεδομένων που αντιπροσωπεύει το τρέχον σύνολο ρυθμίσεων του server, συμπεριλαμβανομένων των ρυθμίσεων των αισθητήρων, του BLE Host, καθώς και των υπόλοιπων περιφερειακών. Περιλαμβάνει απαραίτητα τα εξής πεδία, τα οποία είναι κοινά για όλα τα CM Services και αφορούν μόνο στον application processor της συσκευής:

- ***Update Interval:*** Η περίοδος ενημέρωσης T_U, όπως ορίζεται στην υποενότητα 4.3.2, σε ms.
- ***Broadcast Timeout:*** Η διάρκεια εκπομπής t_B, όπως ορίζεται στην υποενότητα 4.3.2, σε ms.

Το υπόλοιπο περιεχόμενο του DCS εξαρτάται από την εκάστοτε υλοποιούμενη υπηρεσία και συμπληρώνεται ως εξής:

- Για κάθε *παρακολουθητή* κατασκευάζεται ένα *Monitor Configuration Structure* (MCS), μία δομή που εκφράζει τις ρυθμίσεις του ως ζεύγη της μορφής (*Alert Level*, *Control Values*), όπου τα *Alert Level* και *Control Values* αναπαριστούν αντίστοιχα το επιθυμητό επίπεδο ειδοποίησης και το σύνολο των τιμών ελέγχου.

- Για κάθε *αισθητήρα* κατασκευάζεται ένα *Sensor Configuration Structure* (SCS), μία δομή που εκφράζει τις ρυθμίσεις λειτουργίας του ως ζεύγη της μορφής (*Sensor Mode*, *Setting Values*), όπου τα *Sensor Mode* και *Setting Values* αναπαριστούν αντίστοιχα τον τρόπο λειτουργίας (ON, OFF, εξοικονόμηση ενέργειας ή άλλο) και τις τρέχουσες τιμές των λοιπών ρυθμίσεων προς τροποποίηση, ακολουθώντας την αρίθμηση με την οποία αυτές ορίζονται από την υπηρεσία.

4.5.5.1 Ορισμός για το ES Service

Για το Environmental Sensor Service, το DCS ενσωματώνει τις ρυθμίσεις των παρακολουθητών *Temperature*, *Temperature Change*, *Humidity*, *Humidity Change*. Τόσο για το θερμόμετρο όσο και για το υγρόμετρο είναι επιπλέον δυνατή η επιλογή μίας από τις διαθέσιμες τιμές για την ανάλυση μέτρησης (σε bits/δείγμα), οι οποίες ορίζονται από την υλοποίηση και αντιστοιχούν στο υλικό που χρησιμοποιείται, όπως φαίνεται στον παρακάτω πίνακα:

Πεδίο	**Τύπος**	**Μήκος (bytes)**	**Περιγραφή**
Update Interval	Integer	2	Περίοδος ενημέρωσης (ms)
Broadcast Timeout	Integer	2	Διάρκεια εκπομπής (ms)
Monitor Configuration: Temperature			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Values	Float[2]	8	Άνω και κάτω όρια θερμοκρασίας για κανονική λειτουργία
Monitor Configuration: Temperature Change			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Values	Float[2]	8	Άνω και κάτω όρια ρυθμού μεταβολής θερμοκρασίας για κανονική λειτουργία
Monitor Configuration: Humidity			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Values	Float[2]	8	Άνω και κάτω όρια υγρασίας για κανονική λειτουργία

Monitor Configuration: Humidity Change			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Values	Float[2]	8	Άνω και κάτω όρια ρυθμού μεταβολής υγρασίας για κανονική λειτουργία
Sensor Configuration: Thermometer			
Sensor Mode	Integer	1	Τρόπος λειτουργίας του θερμομέτρου: ON ή OFF
Setting Value: Bit Resolution	Integer	1	Επιλογή μίας από τις αριθμημένες διαθέσιμες τιμές για την ανάλυση μέτρησης
Sensor Configuration: Hygrometer			
Sensor Mode	Integer	1	Τρόπος λειτουργίας του υγρομέτρου: ON ή OFF
Setting Value: Bit Resolution	Integer	1	Επιλογή μίας από τις αριθμημένες διαθέσιμες τιμές για την ανάλυση μέτρησης

Πίνακας 4.14: Ορισμός DCS για το ES Service

4.5.5.2 Ορισμός για το IM Service

Για το Inertial Measurement Service, το DCS περιλαμβάνει τις ρυθμίσεις των παρακολουθητών *Shock Detection* και *Motion Detection*. Τόσο για το επιταχυνσιόμετρο όσο και για το γυροσκόπιο είναι επιπλέον δυνατή η επιλογή μίας από τις διαθέσιμες τιμές για το εύρος μέτρησης (σε g και dps αντίστοιχα), οι οποίες ορίζονται από την υλοποίηση και εξαρτώνται από το υλικό, όπως φαίνεται παρακάτω:

Πεδίο	**Τύπος**	**Μήκος (bytes)**	**Περιγραφή**
Update Interval	Integer	2	Περίοδος ενημέρωσης (ms)
Broadcast Timeout	Integer	2	Διάρκεια εκπομπής (ms)
Monitor Configuration Structure: Motion Detection			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Value	Float	4	Κατώφλι εντοπισμού κίνησης

Monitor Configuration Structure: Shock Detection			
Alert Level	Integer	1	Επίπεδο ειδοποίησης
Control Value	Float	4	Κατώφλι εντοπισμού δόνησης
Sensor Configuration Structure: Accelerometer			
Sensor Mode	Integer	1	Τρόπος λειτουργίας του επιταχυνσιομέτρου: ON ή OFF
Setting Value: Range	Integer	1	Επιλογή μίας από τις αριθμημένες διαθέσιμες τιμές για το εύρος μέτρησης
Sensor Configuration Structure: Gyroscope			
Sensor Mode	Integer	1	Τρόπος λειτουργίας του γυροσκοπίου: ON ή OFF
Setting Values: Range	Integer	1	Επιλογή μίας από τις αριθμημένες διαθέσιμες τιμές για το εύρος μέτρησης

Πίνακας 4.15: Ορισμός DCS για το IM Service

4.5.6 Ειδικά ζητήματα

Στο σημείο αυτό θα αναφερθούμε σύντομα σε κάποια εξειδικευμένα ζητήματα που σχετίζονται άμεσα με τον ορισμό των CM Services που δόθηκε παραπάνω.

4.5.6.1 Δεδομένα εκπομπής

Κατά τη φάση εκπομπής (advertising), το peripheral ενσωματώνει την πληροφορία για την κατάσταση της συσκευής στη δομή *Manufacturer-specific data* του πακέτου εκπομπής. Καθώς δεν είμαστε σε θέση να κατοχυρώσουμε ένα Company ID από το Bluetooth SIG, στο πεδίο Company ID της δομής αυτής αναθέτουμε την τιμή 0xFFFF (*Testing*), ώστε τα δεδομένα να είναι προσβάσιμα από κάθε scanner.

4.5.6.2 Σύνδεση σε συσκευή

Ο ορισμός της MTU του ATT στα 115 bytes είναι αυτός που επιβάλλει μέγιστο μήκος 112 bytes για τις τιμές των characteristics *Data In*, *Data Out*, ταυτόχρονα όμως απαιτείται να υποστηρίζεται και από τις δύο συσκευές που μετέχουν στην επικοινωνία. Η επιλογή αυτής της τιμής για την ATT MTU προκύπτει από την ανάλυση της υποενότητας 5.6.2.

5 Σύστημα αισθητήρων

5.1 Εισαγωγή

Η «έξυπνη συσκευή» που ενσωματώνουμε στο σύστημα προς εποπτεία φιλοξενείται σε μία πλακέτα τυπωμένου κυκλώματος (printed circuit board, PCB) κυκλικού σχήματος και διαμέτρου περίπου 25mm, όπως φαίνεται στην παρακάτω φωτογραφία:

Εικόνα 5.1: Σύστημα αισθητήρων τροφοδοτούμενο από coin cell (coin sensor)

Στο πίσω μέρος της πλακέτας έχει αναρτηθεί η θήκη για την μπαταρία τροφοδοσίας (battery holder). Η τροφοδοσία με τάση 3.0 V γίνεται από μία μπαταρία ιόντων λιθίου της ίδιας περίπου διαμέτρου με αυτήν την πλακέτας. Οι μπαταρίες αυτού του τύπου λόγω του μεγέθους και του σχήματός τους, το οποίο προσομοιάζει με κέρμα, είναι γνωστές ως *coin cells*. Για τον ίδιο λόγο, συστήματα σαν αυτό που κατασκευάστηκε στο πλαίσιο της παρούσας εργασίας καλούνται συχνά coin-sized sensors, ή απλούστερα *coin sensors*. Κατασκευάσαμε δύο ειδών coin sensors:

- **Temperature - Humidity Sensor:** Υλοποιεί το ES Service ενσωματώνοντας στην πλακέτα το IC HDC1008, το οποίο διαθέτει θερμόμετρο και υγρόμετρο.
- **Accelerometer - Gyroscope Sensor:** Υλοποιεί το IM Service ενσωματώνοντας στην πλακέτα το IC BMI160, το οποίο διαθέτει επιταχυνσιόμετρο και γυροσκόπιο τριών αξόνων.

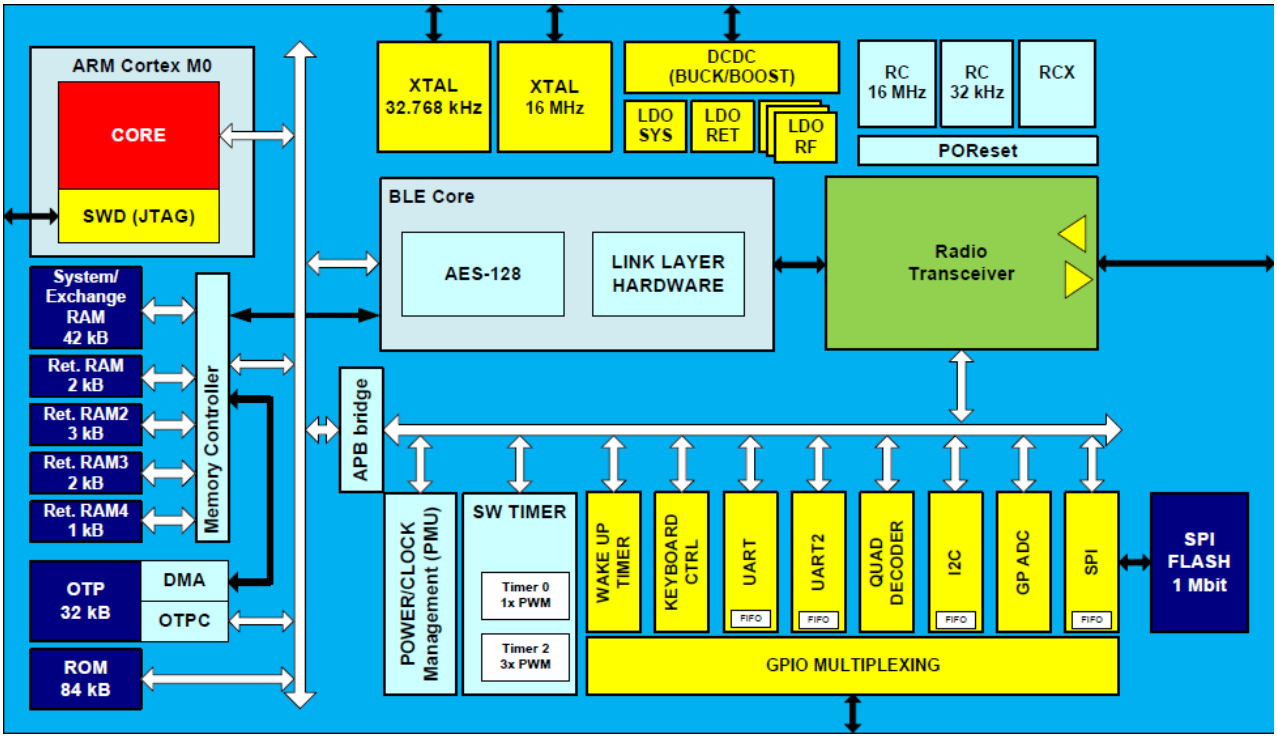
5.2 DA14583 Bluetooth Low Energy SoC

Το σύνολο των coin sensors που κατασκευάσαμε χρησιμοποιούν το IC DA14583 για την υποστήριξη του BLE. Το DA14583 της Dialog Semiconductor είναι ένα πλήρες Bluetooth Low Energy System on Chip (SoC). Περιλαμβάνει δηλαδή σε ένα ολοκληρωμένο κύκλωμα ένα σύστημα με όλα τα απαραίτητα στοιχεία για την ανάπτυξη και την εκτέλεση εφαρμογών BLE:

- Πομποδέκτη ραδιοσυχνοτήτων (RF transceiver) για αμφίδρομη ασύρματη επικοινωνία στη ζώνη συχνοτήτων του BLE.
- Αυτόνομο μικροεπεξεργαστή για την υλοποίηση των λειτουργιών του BLE Host και την εκτέλεση των τελικών εφαρμογών.
- Βοηθητικό κύκλωμα (BLE core) για την υλοποίηση των λειτουργιών του BLE Controller.
- Ενσωματωμένη μνήμη Flash για εύκολο προγραμματισμό και αποσφαλμάτωση.
- Pre-compiled υλοποίηση της πλήρους στοίβας πρωτοκόλλων του BLE, αποθηκευμένη σε ειδική μνήμη ROM.

5.2.1 Αρχιτεκτονική

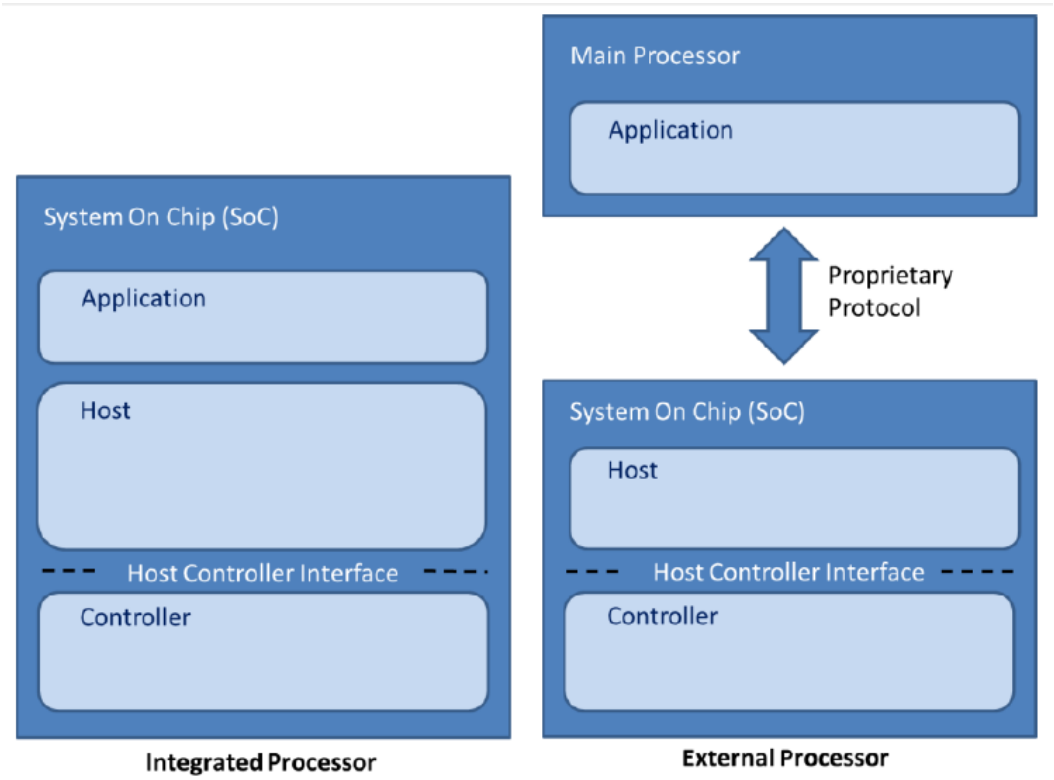
Στο παρακάτω λειτουργικό διάγραμμα (block diagram) παρουσιάζεται η αρχιτεκτονική του DA14583, με τη διασύνδεση των κύριων δομικών μονάδων του. Στη συνέχεια παρουσιάζουμε τα κυριότερα τεχνικά χαρακτηριστικά του:



Σχήμα 5.2: Μπλοκ διάγραμμα του DA14583

Κύρια CPU	32-bit ARM Cortex M0 στα 16 MHz
Μνήμη RAM	42 kB SRAM μνήμη συστήματος 8 kB Retention SRAM για διατήρηση δεδομένων κατά την εξοικονόμηση ενέργειας
Μνήμη ROM	84 kB για τη φιλοξενία του bootloader και της στοίβας του BLE
Εξωτερική μνήμη	125 kB SPI Flash για προγραμματισμό, αποσφαλμάτωση και αποθήκευση κώδικα εφαρμογής και profiles 32 kB OTP (One-Time Programmable Memory) για μόνιμη αποθήκευση της τελικής εφαρμογής
Πομποδέκτης	0 dBm ισχύς εκπομπής εξόδου (TX power) Ευαισθησία -93 dBm
I/O	24 θύρες γενικού σκοπού (GPIO) Διεπαφές UART, SPI, I^2C
Περιφερειακά	3-axis Quadrature Decoder 10-bit A/Ψ Μετατροπέας
Τροφοδοσία	Υποστήριξη τροφοδοσίας 3.0 V από coin cell Υποστήριξη εξωτερικής τροφοδοσίας 3.3 V Ενσωματωμένος μετατροπέας DC-DC
Ρολόγια	HF: 16 MHz με XTAL/RC Oscillator LF: 32 kHz με XTAL/RC Oscillator

Πίνακας 5.3: Τεχνικά χαρακτηριστικά του DA14583



Σχήμα 5.4: Τρόποι λειτουργίας του DA14583

Το DA14583 είναι σε θέση να λειτουργεί τόσο αυτόνομα (*integrated processor configuration*), όσο και επικουρικά προς εξωτερικά συνδεμένο επεξεργαστή (*external processor configuration*). Κατά την αυτόνομη λειτουργία, το DA14583 επιφορτίζεται με τις λειτουργίες BLE Host και Controller, και επιπρόσθετα με την εκτέλεση της εφαρμογής χρήστη στην κύρια CPU (*application processor*). Κατά την επικουρική λειτουργία, το DA14583 αναλαμβάνει τις λειτουργίες BLE Host και Controller, ενώ ο εξωτερικός επεξεργαστής την εκτέλεση της εφαρμογής χρήστη. Ο εξωτερικός επεξεργαστής μπορεί να επικοινωνεί με το DA14583 χρησιμοποιώντας ένα ιδιοταγές πρωτόκολλο πάνω από το δίαυλο SPI ή τη θύρα UART.

Για τη βέλτιστη αξιοποίηση του χώρου πάνω στην PCB, καθώς και για λόγους εξοικονόμησης ενέργειας, στην υλοποίησή μας επιλέξαμε την αυτόνομη λειτουργία, δηλαδή η ανάπτυξη της εφαρμογής μας έγινε πάνω στο DA14583.

5.2.1.1 Εκκίνηση

Κατά την εκκίνηση, ο bootloader του DA14583 εξετάζει την ύπαρξη κώδικα εκκίνησης στη μνήμη OTP. Σε περίπτωση αποτυχίας, η διεπαφή SPI του DA14583 αρχικοποιείται ως SPI Slave, και ο bootloader αναζητά τον κώδικα στη μνήμη SPI Flash, η οποία ενεργεί ως SPI Master. Από τη στιγμή που έχει βρεθεί κώδικας εκκίνησης, αυτός αντιγράφεται αμέσως στη System RAM, και η εκτέλεση ξεκινά από αυτό ακριβώς το σημείο.

Αν δεν υπάρχει τέτοιος κώδικας, δηλαδή καμία από τις δύο μνήμες δεν είναι προγραμματισμένη, απαιτείται η φόρτωσή του (στην SPI Flash, την OTP, ή απευθείας στη System RAM) να γίνει εξωτερικά από άλλη συσκευή συνδεδεμένη στο DA14583 μέσω σειριακής διεπαφής (UART ή SPI).

5.2.1.2 Ροή εκτέλεσης

Η πλατφόρμα λογισμικού του DA14583 βασίζεται σε ένα Real-Time OS (*πυρήνα, kernel*) για την εκτέλεση βασικών λειτουργιών συστήματος (όπως χρονοδρομολόγηση εργασιών και εξυπηρέτηση διακοπών).

Συγκεκριμένα, η κύρια ροή εκτέλεσης συμβαίνει μέσα σε ένα βρόχο (*main loop*), ο οποίος αναλαμβάνει την εξυπηρέτηση των εκκρεμών συμβάντων BLE (advertising ή connection), καθώς και τον προγραμματισμό μελλοντικών συμβάντων με χρήση του χρονοπρογραμματιστή πυρήνα (kernel scheduler). Στα χρονικά διαστήματα που μεσολαβούν, η συσκευή τίθεται σε κατάσταση εξοικονόμησης ενέργειας, και στη συνέχεια αφυπνίζεται σύγχρονα από τον χρονιστή του BLE core για το επόμενο προγραμματισμένο συμβάν, ή ασύγχρονα λόγω άλλου συμβάντος (πχ εξωτερική διακοπή από περιφερειακό).

Οι ανεξάρτητες μονάδες λογισμικού (επίπεδα της στοίβας του BLE, GATT, GAP, εφαρμογή και προφίλ χρήστη) οργανώνονται σε δομές που ονομάζονται *tasks*. Τα tasks επικοινωνούν μεταξύ τους μέσω μηνυμάτων (*messages*), χρησιμοποιώντας τις υπηρεσίες του kernel.

5.2.2 Προγραμματισμός και αποσφαλμάτωση

Για τον coin sensor, χωρίσαμε τη διαδικασία ανάπτυξης λογισμικού σε δύο φάσεις. Στις δύο παραγράφους που ακολουθούν αναλύουμε ξεχωριστά καθεμία από αυτές. Στην παράγραφο 5.2.2.3 παρουσιάζουμε τα βασικά χαρακτηριστικά του παρεχόμενου περιβάλλοντος ανάπτυξης λογισμικού.

5.2.2.1 Πρώτη φάση ανάπτυξης

Η πρώτη φάση ανάπτυξης προηγήθηκε της κατασκευής του coin sensor, ο στόχος της ήταν συνεπώς ο έλεγχος και η αποσφαλμάτωση της εφαρμογής χρήστη και του CM Profile, χωρίς πραγματικά δεδομένα από αισθητήρες. Στη θέση τους χρησιμοποιήθηκαν ενδεικτικές ροές δεδομένων που παράχθηκαν με προγραμματιστικά μέσα στον κώδικα της εφαρμογής, όπως πριονωτές κυματομορφές κλπ.

Για εύκολη αποσφαλμάτωση του DA14583 κατά την πρώτη φάση ανάπτυξης, χρησιμοποιήσαμε μια ειδική αναπτυξιακή πλακέτα (development kit, DK) για το DA14583, με ενσωματωμένη θύρα JTAG για σύνδεση εξωτερικού debugger. Το DK αυτό συνοδεύεται από έναν J-Link debugger με δυνατότητα σύνδεσης μέσω USB στον υπολογιστή όπου γίνεται η ανάπτυξη (development PC).

5.2.2.2 Δεύτερη φάση ανάπτυξης

Κατά τη δεύτερη και τελευταία φάση ανάπτυξης είχαμε πλέον στη διάθεσή μας τους coin sensors. Για τον προγραμματισμό τους κατασκευάσαμε ένα κύκλωμα προγραμματιστή αποτελούμενο από το ολοκληρωμένο FT230X, μια θύρα USB για έξοδο προς το development PC και μια σειρά από ακροδέκτες εξόδου για σύνδεση της διεπαφής UART του FT230X με αυτήν του DA14583: 3V3 για έξοδο τροφοδοσίας, GND για αναφορά σήματος, και τους ακροδέκτες UART RX, TX. Αυτοί συνδέονται αντίστοιχα στους ακροδέκτες 3V3, GND, P0_0 και P0_1 του DA14583.

Ο προγραμματιστής επικοινωνεί με την UART του DA14583 στα 57600 baud/s, δίχως έλεγχο ροής. Το development PC «βλέπει» τον προγραμματιστή ως μία εικονική σειριακή θύρα. Με τη βοήθεια της εφαρμογής *SmartSnippets* της Dialog, το DA14583 επανεκκινείται και στην κύρια μνήμη μεταφορτώνεται μέσω της σειριακής θύρας ένα ειδικό firmware που επιτρέπει τον επαναπρογραμματισμό της SPI Flash, ή τον τελικό προγραμματισμό της OTP, με το τελευταίο εκτελέσιμο (binary) της εφαρμογής.

5.2.2.3 Εργαλεία ανάπτυξης λογισμικού

Η Dialog Semiconductor προσφέρει για το DA14583 ένα πακέτο ανάπτυξης λογισμικού (SDK) που περιλαμβάνει τον πηγαίο κώδικα των tasks για τα θεμελιώδη προφίλ, ένα σύνολο προγραμμάτων οδήγησης (drivers) για τις περιφερειακές συσκευές (ADC, quadrature decoder, διεπαφές I^2C, SPI, UART), την υλοποίηση

κάποιων βασικών υπηρεσιών (*Device Information, Battery* κλπ), καθώς και υποδείγματα πηγαίου κώδικα για μία νέα υπηρεσία GATT και μια απλή εφαρμογή χρήστη.

Τα υποδείγματα (σε γλώσσα C) συγκεντρώνονται σε projects κατάλληλα οργανωμένα για το αναπτυξιακό περιβάλλον (IDE) Keil MDK της ARM, το οποίο αναλαμβάνει και την παραγωγή του τελικού εκτελέσιμου κώδικα.

5.3 Αισθητήρες-περιφερειακά

Παρακάτω θα παρουσιάσουμε τα κυριότερα τεχνικά χαρακτηριστικά των ICs που επιλέχθηκαν για την κατασκευή των coin sensors και των βοηθητικών κυκλωμάτων, καθώς και τον τρόπο διασύνδεσής τους.

5.3.1 Διεπαφές σειριακής επικοινωνίας

Θα ξεκινήσουμε με μια σύντομη αναφορά στις διεπαφές σειριακής επικοινωνίας που χρησιμοποιήθηκαν για τη διασύνδεση των ICs.

5.3.1.1 SPI

Ο δίαυλος SPI *(Serial Peripheral Interface)* αποτελεί μία από τις δημοφιλέστερες σειριακές διεπαφές, κυρίως για επικοινωνία μεταξύ ICs, εσωτερικά πάνω στην ίδια πλακέτα συστήματος. Μέχρι σήμερα, οι διεπαφές SPI ακολουθούν το πρότυπο που προτάθηκε από την Motorola το 1979. Σύμφωνα με αυτό, στο δίαυλο συνδέονται μία συσκευή με αρμοδιότητες ελέγχου (master), και μία ή περισσότερες παθητικές συσκευές (slaves).

Στην απλούστερη εκδοχή του, δηλαδή για την επικοινωνία μεταξύ δύο συσκευών, το SPI χρησιμοποιεί 4 γραμμές (4-wire SPI):

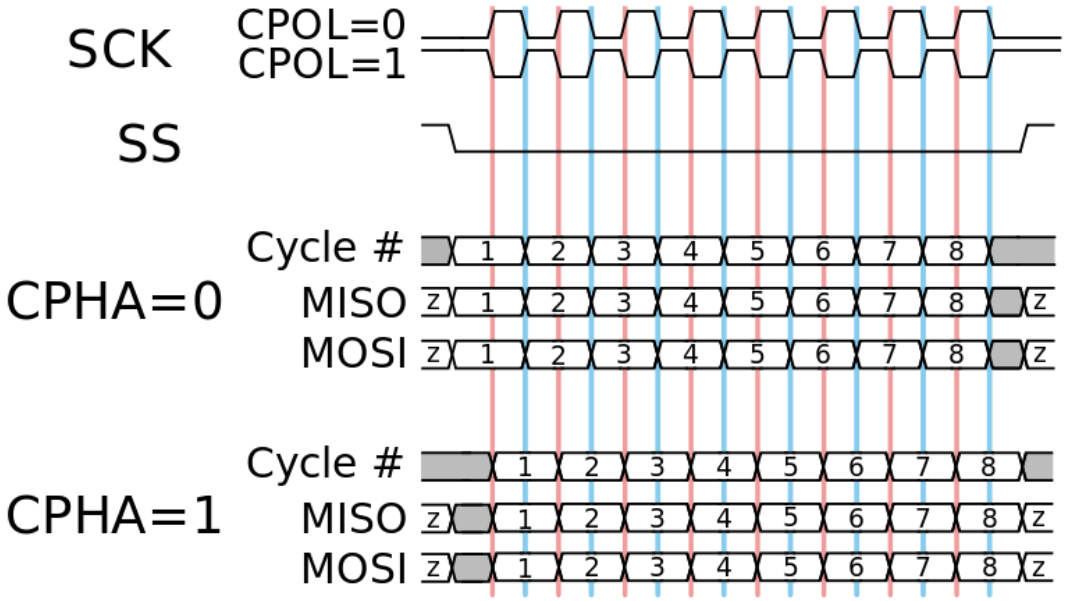
- *SCLK (slave clock):* Το κοινό ρολόι του διαύλου, συνήθως της τάξης του 1-100 MHz, επιβαλλόμενο από τον master.
- *MISO (master input/slave output):* Γραμμή μετάδοσης δεδομένων από τους slaves στον master.
- *MOSI (master output/slave input):* Γραμμή μετάδοσης δεδομένων από τον master στους slaves.
- $\overline{SS}$ *(slave select):* Γραμμή επιλογής του slave από τον master (αρνητική λογική).

Οι γραμμές MISO και MOSI μπορούν να είναι ταυτόχρονα ενεργές, επιτρέποντας πλήρως αμφίδρομη (full-duplex) επικοινωνία. Επειδή όμως ακολουθούν οδήγηση open-drain, η έξοδός τους στην ανενεργή κατάσταση είναι ανοικτοκυκλωμένη (High-Z), άρα η τιμή της είναι απροσδιόριστη. Έτσι, για τον ορισμό του προεπιλεγμένου λογικού επιπέδου των γραμμών απαιτείται η σύνδεση αντιστάσεων πρόσδεσης.

Είναι δυνατό να χρησιμοποιηθεί μόνο η MISO ως γραμμή μετάδοσης. Σε αυτή την εκδοχή, γνωστή ως 3-wire SPI, η επικοινωνία είναι προφανώς μερικώς αμφίδρομη (half-duplex), αφού ο master και ο slave δεν μπορούν να μεταδίδουν ταυτόχρονα.

Η προδιαγραφή του SPI δεν συνοδεύεται από αντίστοιχο πρότυπο για το πρωτόκολλο επικοινωνίας. Εντούτοις, η πλειοψηφία των διαθέσιμων στην αγορά ελεγκτών SPI υλοποιεί την παρακάτω διαδικασία επικοινωνίας:

Βήμα 1: Ο master εκκινεί τη διαδικασία θέτοντας την SS σε λογικό '0'.
Βήμα 2: Η μετάδοση ξεκινά στον αμέσως επόμενο κύκλο ρολογιού.
Βήμα 3: Ο master μεταδίδει λέξεις δεδομένων στη γραμμή MOSI με ρυθμό 1 bit ανά κύκλο ρολογιού.
Βήμα 4: Ο slave μεταδίδει λέξεις δεδομένων στη γραμμή MISO με τον ίδιο ρυθμό.
Βήμα 5: Ο master τερματίζει τη διαδικασία θέτοντας τη γραμμή επιλογής σε λογικό '1'.



Σχήμα 5.5: Δοσοληψία SPI

Για την σειριακή είσοδο/έξοδο των bits δεδομένων και τη μετατροπή τους σε παράλληλη μορφή, και τα δύο άκρα διατηρούν καταχωρητές ολίσθησης, οι οποίοι ορίζουν και το μήκος της λέξης μετάδοσης. Με δεδομένη τη συχνότητα ρολογιού του διαύλου, η διαδικασία ελέγχεται κατά κανόνα από την *πολικότητα* του ρολογιού (CPOL), με τιμή '0' για θετική ή '1' για αρνητική λογική, και τη *φάση* του ρολογιού (CPHA), η οποία ορίζει τη διαφορά φάσης της εναρκτήριας ακμής του bit δεδομένων από την εναρκτήρια ακμή του ρολογιού, με τιμή '0' για 0° ή '1' για 90°.

Για επικοινωνία ενός με πολλούς, περισσότεροι slaves μπορούν να συνδεθούν ως εξής:

- *Σύνδεση αστέρα*: Οι γραμμές MISO και MOSI είναι κοινές σε ολόκληρο το δίαυλο, και ο master χρησιμοποιεί γραμμές επιλογής ισάριθμες του πλήθους των slaves. Η επιλογή των ενεργών slaves γίνεται με ενεργοποίηση των αντίστοιχων γραμμών. Είναι εμφανές πως η έξοδος MOSI του master εκπέμπεται ταυτόχρονα προς όλους τους ενεργούς slaves, ενώ ανά πάσα στιγμή μόνο ένας εξ αυτών είναι δυνατό να μεταδίδει προς τον master.
- *Σύνδεση σε σειρά*: Οι slaves συνδέονται σε κοινή γραμμή επιλογής, και καθένας από αυτούς συνδέει τις εξόδους του MISO/MOSI με τις αντίστοιχες του επόμενου slave, εκτός από τον πρώτο slave που συνδέεται απευθείας με τον master. Η μετάδοση των δεδομένων γίνεται από τον master προς τον τελευταίο slave, και αντίστροφα, με διέλευση από όλους τους ενδιάμεσους slaves.

5.3.1.2 I^2C

Ο δίαυλος I^2C *(Inter-Intergrated Circuit)*, όπως δηλώνει η ονομασία του και όμοια με το SPI, ορίζει μία σειριακή διεπαφή για εσωτερική διασύνδεση ολοκληρωμένων κυκλωμάτων. Σχεδιάστηκε και προτυποποιήθηκε από την Philips το 1982, με την προδιαγραφή να καλύπτει τόσο τη διεπαφή όσο και το πρωτόκολλο επικοινωνίας. Σε έναν δίαυλο I^2C μπορούν να συνδέονται έως 128 συσκευές, διευθυνσιοδοτούμενες μονοσήμαντα με αναγνωριστικά μήκους $\log_2 128 = 7$ bit, η καθεμία είτε με ενεργητικό ρόλο (master), είτε με παθητικό (slave).

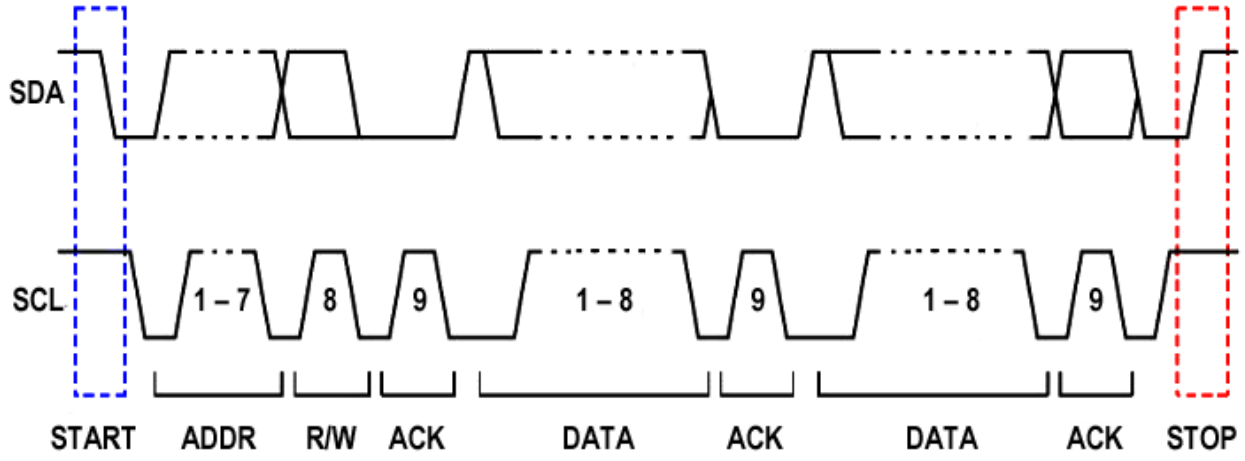
Ανεξάρτητα από το πλήθος των συνδεδεμένων συσκευών, η επικοινωνία γίνεται κάθε φορά μόνο μεταξύ ενός master και ενός slave. Ο δίαυλος προσφέρει τις εξής κοινόχρηστες γραμμές:

- *SCL (serial clock):* Ρολόι διαύλου, της τάξης των 0.1-5 MHz, επιβαλλόμενο από τον εκάστοτε ενεργό master.
- *SDA (serial data):* Γραμμή αμφίδρομης μετάδοσης δεδομένων.

Όμοια με το SPI, η γραμμή SDA οδηγείται ως open-drain, συνεπώς απαιτείται η σύνδεσή της σε αντίσταση πρόσδεσης. Επιπλέον, εφόσον παρέχεται μία μόνο γραμμή δεδομένων, η επικοινωνία μπορεί να είναι μόνο half-duplex. Η διαδικασία επικοινωνίας που ορίζεται από το πρωτόκολλο είναι η εξής:

Βήμα 1: Επιλέγεται ένας master για τον έλεγχο του διαύλου κατά την τρέχουσα δοσοληψία.
Βήμα 2: Ο master δεσμεύει το δίαυλο και ορίζει τη συχνότητα ρολογιού.
Βήμα 3: Ο master εκκινεί τη διαδικασία μεταδίδοντας ένα START bit.
Βήμα 4: Ο master επιλέγει έναν slave για μία δοσοληψία ανάγνωσης ή εγγραφής.
Βήμα 5: Ο master μεταδίδει μία λέξη αποτελούμενη από τη διεύθυνση του slave και το bit ανάγνωσης/εγγραφής $R/\overline{W}$ ('0' για εγγραφή, '1' για ανάγνωση).
Βήμα 6: Σε περίπτωση ανάγνωσης (αντίστοιχα, εγγραφής), ο slave (αντίστοιχα, ο master) μεταδίδει μία λέξη δεδομένων.

Βήμα 7: Σε περίπτωση ανάγνωσης (αντίστοιχα, εγγραφής), ο master (αντίστοιχα, ο slave) αποστέλλει ένδειξη αναγνώρισης της λέξης (ACK).
Βήμα 8: Τα βήματα 6, 7 επαναλαμβάνονται όσο υπάρχουν δεδομένα για ανάγνωση/εγγραφή.
Βήμα 9: Ο master επιστρέφει στο βήμα 4 για πραγματοποίηση νέας δοσοληψίας ανάγνωσης/εγγραφής.
Βήμα 10: Ο master τερματίζει τη διαδικασία μεταδίδοντας ένα STOP bit.
Το πρωτόκολλο ορίζει λέξεις μήκους ενός byte, δεν υπάρχει όμως κανένας περιορισμός στο πλήθος των λέξεων που μπορούν να μεταδοθούν σε μία δοσοληψία.



Σχήμα 5.6: Δοσοληψία I^2C

5.3.1.3 UART

Ως UART *(Universal Asynchronous Receiver/Transmitter)* ήταν παλαιότερα γνωστά τα κυκλώματα αποστολής/λήψης δεδομένων πάνω από τις κλασικές σειριακές θύρες (RS232, RS422, RS485 κλπ) για σύνδεση μόντεμ, εκτυπωτών και άλλων περιφερειακών με προσωπικούς υπολογιστές. Στις μέρες μας, παρά την αντικατάσταση των σειριακών θυρών σε συντριπτικό ποσοστό από το δίαυλο USB, ο όρος παραμένει σε χρήση για κάθε συσκευή συμβατή με αυτές - φυσική ή εικονική - που είναι κατάλληλη για γενικού σκοπού ασύγχρονη σειριακή επικοινωνία. Λόγω τις απλότητάς τους και της υποστήριξης μεγαλύτερων αποστάσεων σε σύγκριση με τις SPI και I^2C, διεπαφές UART ενσωματώνονται στην πλειοψηφία των μικροελεγκτών (μΕ) που κυκλοφορούν στην αγορά, επιτρέποντας τόσο την επικοινωνία αυτών με άλλους μΕ, όσο και με αποσφαλματωτές (debuggers).

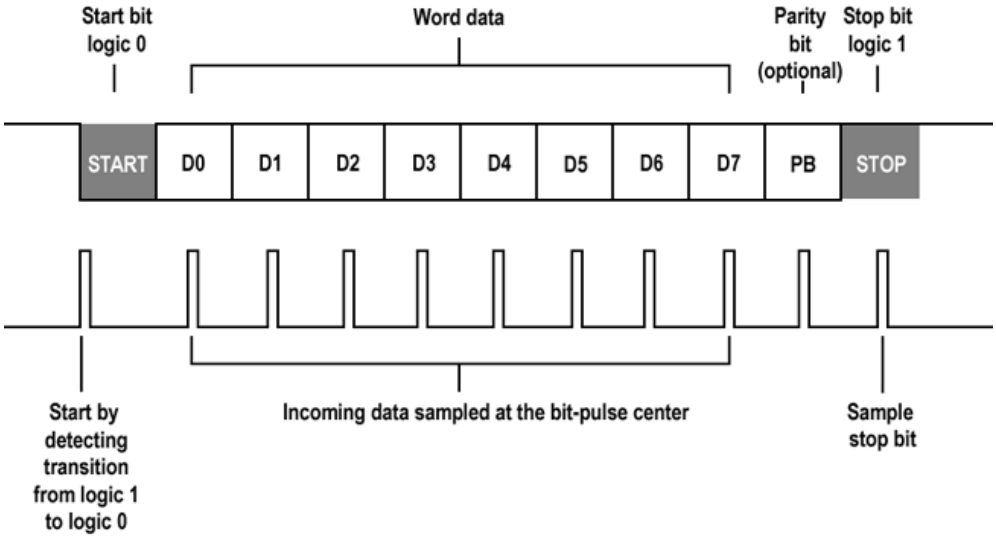
Η σύνδεση μεταξύ δύο UART είναι πάντα σημείου-προς-σημείο. Καθένα από τα δύο άκρα της επικοινωνίας διατηρεί το δικό του εσωτερικό ρολόι, καθώς και έναν καταχωρητή ολίσθησης για την είσοδο/έξοδο των bits δεδομένων και τη μετατροπή τους σε παράλληλη/σειριακή μορφή. Η επικοινωνία είναι full-duplex, και γίνεται μέσω των εξής ακροδεκτών:

- *RX (receive):* Ακροδέκτης λήψης, συνδέεται με τον ακροδέκτη αποστολής (TX) του άλλου άκρου.
- *TX (transmit):* Ακροδέκτης αποστολής, συνδέεται με τον ακροδέκτη λήψης (RX) του άλλου άκρου.

Παρέχονται επιπλέον οι εξής προαιρετικές λειτουργίες:

- *Έλεγχος ροής (flow control):* Υλοποιείται από το υλικό στο επίπεδο συνολικά του καναλιού ή κάθε ξεχωριστής λέξης δεδομένων, διαφορετικά από το λογισμικό με αποστολή των συμβόλων XON/XOFF. Για έλεγχο ροής υλικού, σε επίπεδο καναλιού παρέχονται οι ακροδέκτες RTS/CTS (request/clear to send) για αποστολή/αποδοχή αντίστοιχα του αιτήματος δέσμευσης του καναλιού, ενώ σε επίπεδο λέξης χρησιμοποιούνται οι ακροδέκτες DSR/DTR (data set/data terminal ready) για αποστολή/αποδοχή αντίστοιχα του αιτήματος αποστολής ενός μπλοκ δεδομένων. Αν ο έλεγχος ροής υλικού γίνεται και στα δύο επίπεδα, αρχικά δεσμεύεται ο δίαυλος και στη συνέχεια εφαρμόζεται έλεγχος ροής σε κάθε λέξη ξεχωριστά.
- *Έλεγχος σφαλμάτων (error checking):* Υλοποιείται από το υλικό με την ενσωμάτωση ενός bit ισοτιμίας (άρτιας ή περιττής) στο τέλος της μεταδιδόμενης λέξης. Η χρήση αυτού του bit για σκοπό διαφορετικό από τον έλεγχο ισοτιμίας είναι γνωστή ως *ισοτιμία mark/space* (mark για λογικό '1', space για λογικό '0').

Η επικοινωνία μεταξύ δύο συσκευών, έστω S_1 και S_2, ακολουθεί την παρακάτω διαδικασία:



Σχήμα 5.7: Επικοινωνία μέσω UART

Βήμα 1: Οι S_1, S_2 συμφωνούν σε ένα baud rate.
Βήμα 2: Εάν απαιτείται έλεγχος ροής, η S_1 θέτει την έξοδο RTS/DSR σε λογικό '1' και αναμένει το σήμα CTS/DTR, ή αποστέλλει *XON = 0x11*.

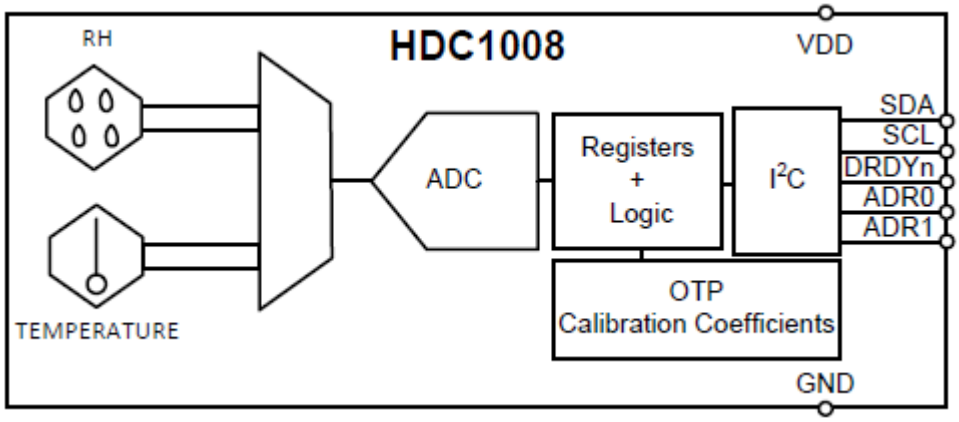
Βήμα 3: Η S_1 αποστέλλει ένα START bit (λογικό '0') για συγχρονισμό των ρολογιών των S_1, S_2.
Βήμα 4: Η S_1 αποστέλλει μία λέξη δεδομένων μήκους 5 έως 9 bit, προαιρετικά συνοδευόμενη από ένα bit ισοτιμίας.
Βήμα 5: Η S_1 ολοκληρώνει τη μετάδοση της λέξης με αποστολή ενός ή περισσότερων STOP bits (λογικό '1').
Βήμα 6: Η S_2 θέτει την έξοδο CTS/DTR σε λογικό '0'.
Βήμα 7: Η S_1 επιστρέφει στο βήμα 2 όσο υπάρχουν επιπλέον δεδομένα προς μετάδοση.
Βήμα 8: Εάν απαιτείται έλεγχος ροής, η S_1 απενεργοποιεί την έξοδο RTS/DSR και CTS/DTR, ή αποστέλλει *XOFF = 0x13*.

5.3.2 Θερμόμετρο/υγρόμετρο HDC1008

Το HDC1008 της Texas Instruments ενσωματώνεται μόνο στον Temperature-Humidity Sensor. Παρακάτω συνοψίζουμε τα τεχνικά χαρακτηριστικά του:

Εύρος μέτρησης	Θερμοκρασία: -40…125 °C Σχετική υγρασία: 0…100%
Δειγματοληψία	Μέγιστος ρυθμός 1 Hz
Ανάλυση	Θερμόμετρο: 11/14-bit Υγρόμετρο: 8/11/14-bit
Πιστότητα	Θερμοκρασία: ± 0.2 °C Σχετική υγρασία: ± 4%
Κατανάλωση ρεύματος	Ανενεργό (sleep mode): 110…220 nA Μέτρηση υγρασίας (11-bit, 1 Hz): 820 nA Μέτρηση θερμοκρασίας και υγρασίας (11-bit, 1 Hz): 1.2 μA
I/O	Διεπαφή I^2C
Τροφοδοσία	3.0…5.0 V

Πίνακας 5.8: Τεχνικά χαρακτηριστικά του HDC1008



Σχήμα 5.9: Μπλοκ διάγραμμα του HDC1008

Πρόκειται για ένα δοκιμασμένο και αξιόπιστο κύκλωμα, αρκετά χαμηλού κόστους, που επιπλέον παρέχει ένα κατάλληλο εύρος μέτρησης και επαρκή ακρίβεια για τις απαιτήσεις της εφαρμογής μας.

Η λειτουργία του HDC1008 είναι απλή, και βασίζεται στο χειρισμό τριών καταχωρητών, δύο για ανάγνωση των τρεχουσών τιμών της θερμοκρασίας και της σχετικής υγρασίας αντίστοιχα, και ενός για ορισμό των επιθυμητών μεγεθών (θερμοκρασία ή/και σχετική υγρασία) και της ανάλυσης της μέτρησης για κάθε μέγεθος.

Παρέχει ως διεπαφή προς το υπόλοιπο σύστημα τους εξής ακροδέκτες, οι συνδέσεις των οποίων παρουσιάζονται στο παράρτημα Α.

- Τις γραμμές *SDA* και *SCL* της διεπαφής I^2C.
- Την τροφοδοσία VCC από την μπαταρία, και την αναφορά σήματος GND.
- Τη γραμμή εξόδου $\overline{DATA\_READY}$, αρνητικής λογικής, η οποία σηματοδοτεί την ολοκλήρωση μίας μέτρησης.
- Τις γραμμές *ADR0* και *ADR1*, καθεμία από τις οποίες προσδένεται στο VCC (λογικό '1') ή στο GND (λογικό '0'). Η δυαδική τιμή (*ADR1|ADR0*) επιλέγει μία από τέσσερις ορισμένες από τον κατασκευαστή διευθύνσεις I^2C, με την οποία η συσκευή θα είναι ορατή στον I^2C master.

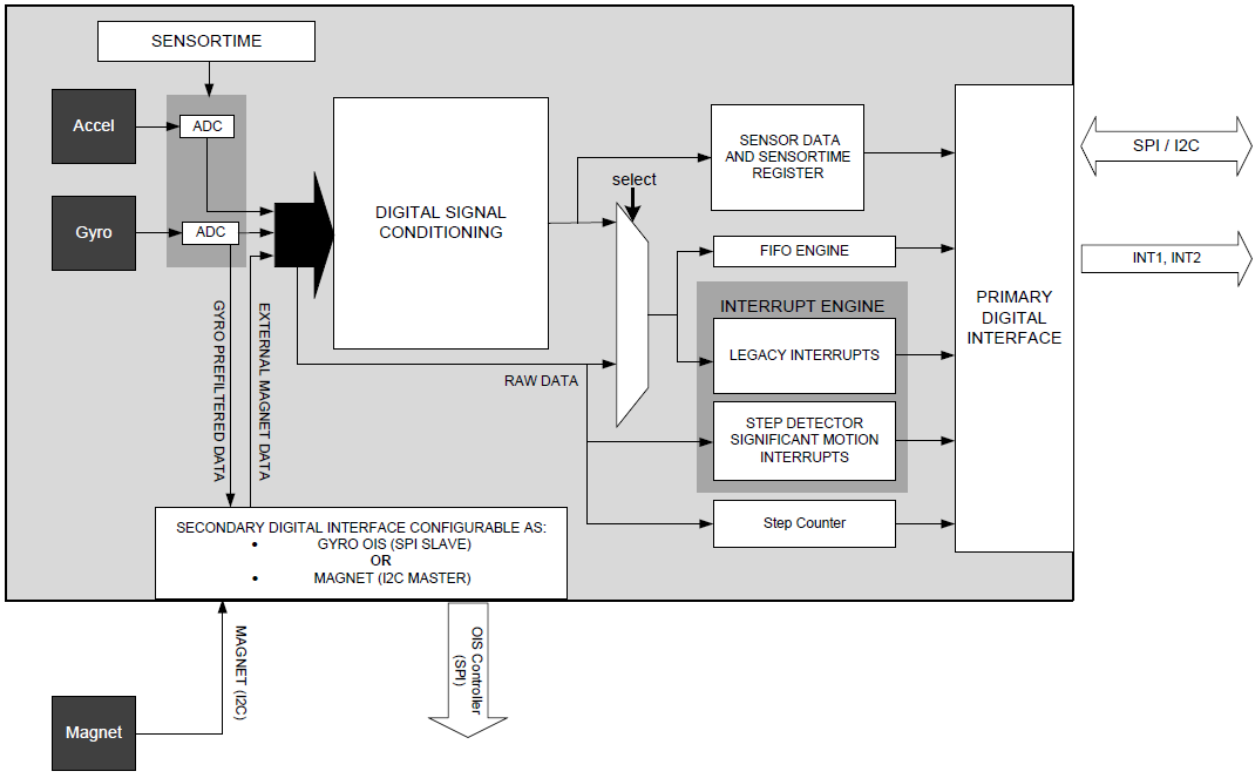
Η μέτρηση γίνεται μόνο κατόπιν αιτήματος δειγματοληψίας, με μέγιστο ρυθμό μία φορά ανά δευτερόλεπτο. Καθώς ο αισθητήρας παραμένει ενεργός μόνο όσο διαρκεί η μέτρηση, η κατανάλωση ισχύος ελαχιστοποιείται. Για την υλοποίηση των παραπάνω κατασκευάσαμε ένα απλό πρόγραμμα οδήγησης, το οποίο παρουσιάζουμε στην υποενότητα 5.5.1.

5.3.3 Επιταχυνσιόμετρο/γυροσκόπιο BMI160

Το BMI160 της Bosch Sensortec ενσωματώνεται μόνο στον Acclerometer-Gyroscope Sensor, και συνιστά μία πλήρους λειτουργικότητας IMU 6 αξόνων (επιταχυνσιόμετρο και γυροσκόπιο 3 αξόνων). Τα κύρια τεχνικά χαρακτηριστικά του είναι τα εξής:

Ανάλυση	Επιταχυνσιόμετρο, γυροσκόπιο: 16-bit
Δειγματοληψία	Μέγιστος ρυθμός 1600 Hz
Ακρίβεια	Επιταχυνσιόμετρο: ± 40 mg Γυροσκόπιο: ± 3 °/s
Κατανάλωση ρεύματος	Ανενεργό (suspend mode): 3 μA Επιταχυνσιόμετρο (1600 Hz): 180 μA Γυροσκόπιο (1600 Hz): 850 μA
I/O	Διεπαφές SPI (3,4-wire), I^2C
Τροφοδοσία	1.7…3.6 V

Πίνακας 5.10: Τεχνικά χαρακτηριστικά του BMI160



Σχήμα 5.11: Μπλοκ διάγραμμα του BMI160

Για την επικοινωνία του BMI160 με το υπόλοιπο σύστημα επιλέγουμε τον δίαυλο I^2C. Οι συνδέσεις του, τις οποίες παρουσιάζουμε στο παράρτημα Β, γίνονται πάνω στους εξής ακροδέκτες:

- Τις γραμμές *SDA* και *SCL* της διεπαφής I^2C.
- Την τροφοδοσία *VDD/VDDIO* και την αναφορά σήματος *GND/GNDIO* για το κύκλωμα και τις θύρες Ε/Ε αντίστοιχα.
- Τις γραμμές εξόδου *INT1* και *INT2* για ασύγχρονη αφύπνιση του application processor με IRQ κατόπιν συγκεκριμένων συμβάντων.

Το επιταχυνσιόμετρο του BMI160 υλοποιεί στο υλικό αλγορίθμους εντοπισμού - μεταξύ άλλων - των παρακάτω συμβάντων, εκ των οποίων χρησιμοποιούμε δύο:

- *Εντοπισμός κίνησης (motion):* Η κλίση της δύναμης επιτάχυνσης σε κάποιον άξονα υπερβαίνει ένα όριο για συγκεκριμένη χρονική διάρκεια. Αυτόν τον αλγόριθμο χρησιμοποιούμε για τη λειτουργία εντοπισμού κίνησης του IM Service, παράγοντας IRQ στην έξοδο *INT1*.
- *Εντοπισμός δόνησης (high-g):* Το μέτρο της δύναμης επιτάχυνσης σε οποιονδήποτε άξονα υπερβαίνει ένα όριο για συγκεκριμένη χρονική διάρκεια. Αυτόν τον αλγόριθμο χρησιμοποιούμε για τη λειτουργία εντοπισμού δόνησης του IM Service, παράγοντας IRQ στην έξοδο *INT2*.

- *Εντοπισμός πατήματος (tap):* Κάθε πάτημα αντιστοιχεί σε γρήγορη εναλλαγή του προσήμου της κλίσης της δύναμης επιτάχυνσης.
- *Μέτρηση βημάτων (step counter):* Κάθε βήμα αντιστοιχεί σε ένα χρονικό διάστημα συγκεκριμένου μήκους κατά το οποίο το μέτρο της δύναμης επιτάχυνσης παρουσιάζει τοπικό μέγιστο.
- *Εντοπισμός ελεύθερης πτώσης (low-g):* Το μέτρο της g-force σε όλους τους άξονες διατηρείται σε τιμές πολύ κοντά στο μηδέν για συγκεκριμένη χρονική διάρκεια.

Η μονάδα διαχείρισης ισχύος (power management unit, PMU) του BMI160 ελέγχει την κατανάλωση ενέργειας ορίζοντας τις παρακάτω καταστάσεις λειτουργίας:

- *Κανονική (normal):* Ο αισθήτηρας πραγματοποιεί συνεχή δειγματοληψία, ανεξάρτητα από το αν υπάρχει αίτημα ανάγνωσης από το χρήστη ή όχι. Η τιμή του εκάστοτε ληφθέντος δείγματος ενημερώνει τον αντίστοιχο καταχωρητή (για την g-force ή τη γωνιακή ταχύτητα), και προαιρετικά εισάγεται σε μία προσωρινή μνήμη ουράς (FIFO buffer) μήκους 1024 bytes. Σε μία δοσοληψία, η έξοδος των δεδομένων μπορεί να γίνει με ανάγνωση ενός δείγματος από τον καταχωρητή, ή πολλών δειγμάτων από την FIFO.
- *Εξοικονόμηση ενέργειας (suspend):* Ο αισθητήρας παραμένει ανενεργός, χωρίς δυνατότητα δειγματοληψίας ή εξόδου δεδομένων.
- *Χαμηλή κατανάλωση (low power):* Προβλέπεται μόνο για το επιταχυνσιόμετρο, και μειώνει την κατανάλωση με εναλλαγή από κανονική κατάσταση σε εξοικονόμηση ενέργειας μεταξύ διαδοχικών μετρήσεων. Σε αυτή την περίπτωση, ο ρυθμός δειγματοληψίας περιορίζεται στα 0.16-400 Hz.
- *Ταχεία εκκίνηση (fast start-up):* Προβλέπεται μόνο για το γυροσκόπιο, και επιταχύνει τη μετάβαση από την εξοικονόμηση ενέργειας στην κανονική λειτουργία.

Σε κάθε περίπτωση, καθώς τα μετρώμενα μεγέθη μεταβάλλονται και δειγματοληπτούνται με εξαιρετικά ταχείς ρυθμούς (σε σύγκριση με ρυθμούς άλλων μεγεθών όπως η θερμοκρασία, η υγρασία κλπ), αναμένεται να προκύπτει ένα σχετικά υψηλό επίπεδο θορύβου. Αυτό μπορεί να περιοριστεί σημαντικά λαμβάνοντας κάθε φορά, αντί για ένα δείγμα, το μέσο όρο των πιο πρόσφατων δειγμάτων εντός ενός παραθύρου σταθερού μήκους. Είναι τότε εμφανές πως ως αντίτιμο, ο ωφέλιμος ρυθμός δειγματοληψίας θα διαιρεθεί με το μήκος του παραθύρου. Στο BMI160 η δυνατότητα αυτή παρέχεται απευθείας στο υλικό, με παράθυρα μήκους 2, 4, 8, 16, 32, 64 ή 128 δειγμάτων.

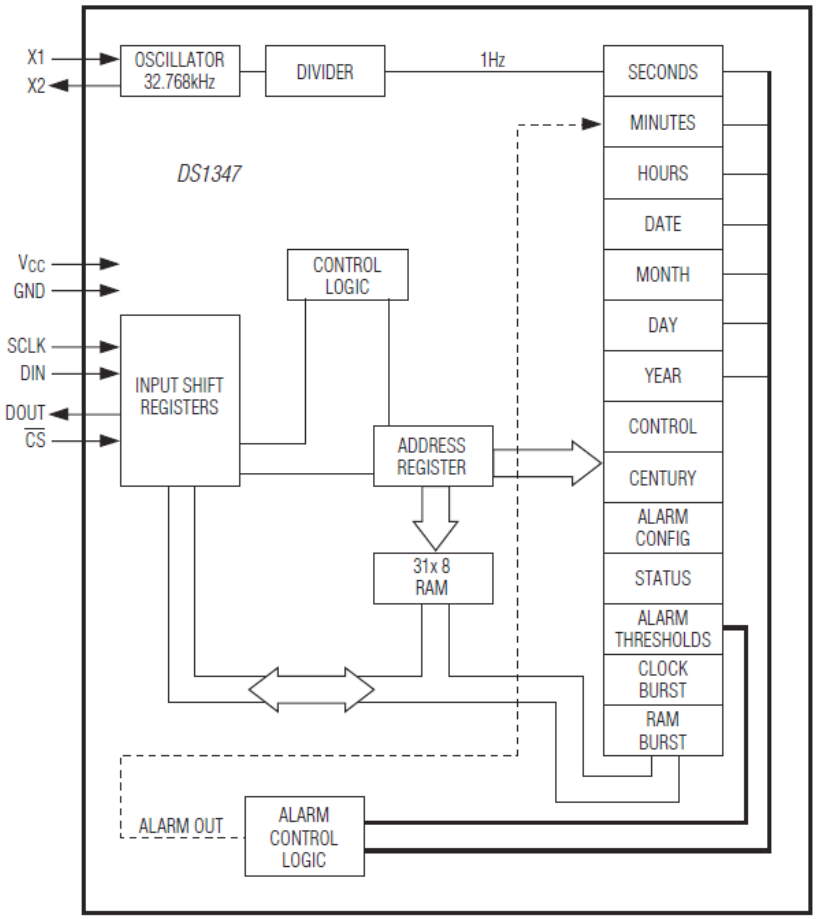
Οι λειτουργίες δειγματοληψίας, ρύθμισης παραμέτρων και ασύγχρονης αφύπνισης επιτελούνται από το πρόγραμμα οδήγησης που παρουσιάζουμε στην υποενότητα 5.5.2.

5.3.4 DS1347 RTC

Το RTC DS1347 της Maxim Integrated σενσωματώνεται και στα δύο είδη coin sensors, παρέχοντας την απαιτούμενη ακρίβεια μέτρησης του χρόνου για τη διατήρηση στη μνήμη και ενημέρωση της τρέχουσας ημερομηνίας και ώρας. Στην εφαρμογή μας χρησιμοποιείται για την παραγωγή των χρονοσφραγίδων των εγγραφών ιστορικού. Παρακάτω παρουσιάζουμε τα τεχνικά χαρακτηριστικά του:

Κατανάλωση ρεύματος	350…800 nA
Λειτουργίες	Λήψη/Ορισμός τρέχουσας ημερομηνίας και ώρας Αφύπνιση
I/O	Διεπαφή SPI (4-wire) με συχνότητα 1 ή 4 MHz
Τροφοδοσία	2…5 V
Ρολόι	32.768 kHz με εξωτερικό XTAL

Πίνακας 5.12: Τεχνικά χαρακτηριστικά του DS1347



Σχήμα 5.13: Μπλοκ διάγραμμα του DS1347

Το DS1347 επικοινωνεί με το υπόλοιπο σύστημα μέσω του διαύλου SPI, ενεργώντας ως slave. Οι συνδέσεις του για τον Temperature Sensor και τον

Accelerometer Sensor, τις οποίες παρουσιάζουμε στα παραρτήματα Α και Β αντίστοιχα, γίνονται πάνω στους εξής ακροδέκτες:

- Τις γραμμές *SS*, *SCLK*, *MISO* και *MOSI* της διεπαφής SPI.
- Την τροφοδοσία *VCC* και την αναφορά σήματος *GND*.
- Τους ακροδέκτες *X1* και *X2* για τη σύνδεση του εξωτερικού κρυστάλλου (XTAL).

Η πληροφορία για την τρέχουσα ημερομηνία/ώρα συντίθεται από διψήφια αριθμητικά πεδία που αντιπροσωπεύουν τα δευτερόλεπτα, τα λεπτά, τις ώρες (0-11 πμ/μμ ή 0-23), την ημέρα του τρέχοντος μήνα, τον μήνα, το έτος του τρέχοντος αιώνα, και τον αιώνα. Τα πεδία κωδικοποιούνται σε BCD, και η τιμή καθενός από αυτά αποθηκεύεται σε ξεχωριστό καταχωρητή. Η ανάγνωση και η εγγραφή μπορούν να γίνονται μεμονωμένα για οποιοδήποτε πεδίο, ή συνολικά σε μία δοσοληψία για όλα τα πεδία.

Η λειτουργία αφύπνισης ενεργοποιείται με εγγραφή σε ειδικό καταχωρητή μίας κατάλληλης τιμής, η οποία περιλαμβάνει απαραίτητα την ημερομηνία και ώρα αφύπνισης. Καθώς όμως το DS1347 δε διαθέτει θύρα εξόδου για πρόκληση διακοπής, ο έλεγχος για αφύπνιση απαιτείται να γίνεται χειροκίνητα από τον μΕ με τακτική ανάγνωση (polling) του αντίστοιχου bit κατάστασης, κάτι που αυξάνει σημαντικά την κατανάλωση ενέργειας στη φάση εξοικονόμησης. Για αυτόν το λόγο επιλέγουμε να μη χρησιμοποιήσουμε τη λειτουργία αυτή στην εφαρμογή μας.

5.3.5 Μετατροπέας USB σε σειριακό FT230X

Στον πίνακα που ακολουθεί δίνουμε τα κύρια τεχνικά χαρακτηριστικά του FT230X:

Ρυθμός μετάδοσης	0.183 – 3000 Kbaud/s
Επιπλέον δυνατότητες	Λειτουργία εικονικής θύρας COM Ενδείξεις LED για RX/TX
I/O	1 x USB 2.0 1 x RS232
Τροφοδοσία	5.0 V (από USB)
Τάση εξόδου	5.0/3.3/2.8/1.8 V
Ρολόγια	48/24/12/6 MHz από εσωτερικό ταλαντωτή 12 MHz

Πίνακας 5.14: Τεχνικά χαρακτηριστικά του FT230X

5.4 Υλοποίηση του CM Profile

Το SDK του DA14583 προσφέρει έναν σκελετό κώδικα (*ble_app_peripheral*) για τον ορισμό και την υλοποίηση από τον προγραμματιστή μίας νέας υπηρεσίας

GATT (CUST1). Ο σκελετός προσφέρει επιπλέον μακροεντολές και συναρτήσεις για τον ορισμό των GATT characteristics και των ιδιοτήτων τους, καθώς και τη δυνατότητα ορισμού συναρτήσεων χειρισμού/επανάκλησης (handler functions ή callbacks) για την εξυπηρέτηση αιτημάτων. Επιλέξαμε να χρησιμοποιήσουμε ως βάση τον κώδικα αυτό για την παραγωγή δύο ανεξάρτητων projects (*ble_app_thermo* και *ble_app_accel*), εφόσον τα δύο είδη coin sensor θα υλοποιούν διαφορετικές υπηρεσίες. Και τα δύο projects ακολουθούν την παρακάτω δομή:

```
ble_app_peripheral
     user_app
          user_custs1_impl.c
          user_custs1_impl.h
          user_peripheral.c
          user_peripheral.h
          user_monitor.c
          user_monitor.h
     user_driver
          user_rtc.c
          user_rtc.h
          user_env_sensor.c
          user_env_sensor.h
          user_imu.c
          user_imu.h
     user_utils
          user_circular_buffer.c
          user_circular_buffer.h
          user_streaming_service.c
          user_streaming_service.h
     user_memory
          user_device_common.c
          user_device_common.h
          user_memory.c
          user_memory.h
     user_app_api
          user_service_def.h
          user_log.h
```

Πίνακας 5.15: Δομή πηγαίου κώδικα της εφαρμογής εξυπηρετητή

Σημειώνουμε πως στη συνέχεια της παρόντος κεφαλαίου δεν γίνεται καμία επιπλέον αναφορά σε μονάδες λογισμικού που αποτελούν στοιχεία του SDK. Αυτή η επιλογή γίνεται αφενός για λόγους απλότητας, αφετέρου επειδή το SDK συνιστά ιδιοταγές λογισμικό.

5.4.1 Έξοδος δεδομένων

Η έξοδος των δεδομένων, όπως περιγράψαμε στο προηγούμενο κεφάλαιο, γίνεται με ειδοποιήσεις GATT είτε στο χαρακτηριστικό *Data Out*, είτε στα χαρακτηριστικά που αντιστοιχούν στα εποπτευόμενα μεγέθη του CM Service. Ανεξάρτητα από το εκάστοτε χαρακτηριστικό προορισμού, γνωρίζουμε πως κάθε ενέργεια GATT θα πρέπει να αναμένει την ολοκλήρωση της προηγούμενής της. Συνεπώς εάν είναι επιθυμητό να αποσταλούν πολλές διαδοχικές ειδοποιήσεις, θα πρέπει αυτές να τοποθετηθούν σε μία ουρά. Το ίδιο απαιτείται και στην περίπτωση που οι ειδοποιήσεις μεταδίδονται περιοδικά, αλλά παράγονται με ρυθμό αρκετά ταχύτερο του μέγιστου ρυθμού διέλευσης της σύνδεσης BLE (για παράδειγμα, δειγματοληψία με περίοδο σημαντικά μικρότερη από το connection interval).

Για την εξασφάλιση της βέλτιστης εξυπηρέτησης και τον έλεγχο όλων των αιτημάτων, οι ροές δεδομένων της εφαρμογής ομαδοποιούνται σε πηγές (sources), ανάλογα με την προέλευση της καθεμιάς. Οι ροές μπορούν να προέρχονται από:

- Τις μετρήσεις των αισθητήρων, ορίζοντας μία πηγή για τα δεδομένα καθενός από αυτούς κατά τη διάρκεια μίας συνόδου εποπτείας:
  - Για το ES Service, ορίζουμε τις πηγές *THERMO_SOURCE* και *HYGRO_SOURCE*, οι οποίες ενημερώνουν αντίστοιχα τα χαρακτηριστικά *Temperature* και *Relative Humidity*.
  - Για το IM Service, ορίζουμε τις πηγές *ACCEL_SOURCE* και *GYRO_SOURCE*, οι οποίες ενημερώνουν αντίστοιχα τα χαρακτηριστικά *Acceleration* και *Angular Rate*.
- Το τοπικά αποθηκευμένο ιστορικό συμβάντων, ορίζοντας μία πηγή για την αποστολή στον client όλων των εγγραφών συμβάντων από την τελευταία σύνδεση, ή κάθε νέου συμβάντος που παράγεται κατά τη διάρκεια μίας συνόδου. Για όλα τα CM Services, ορίζουμε την πηγή *LOG_SOURCE*, η οποία ενημερώνει το χαρακτηριστικό *Data Out*.

Σε λογικό επίπεδο, αναπαριστούμε μία πηγή δεδομένων με τον τύπο `out_data_source` (*user_utils*/*user_streaming_service.h*). Κάθε πηγή μπορεί να δέχεται δεδομένα από οποιαδήποτε δομή δεδομένων (πίνακα, buffer, ουρά κλπ), αρκεί για την τελευταία να δίνονται υλοποιήσεις των παρακάτω συναρτήσεων:

`uint8_t *get_data(uint8_t *data, bool reset)`	
Παράμετροι	`uint8_t *data`: Δείκτης προς τη δομή που περιέχει τα δεδομένα της πηγής. `bool reset`: Τιμή αληθείας για ανάγνωση από την αρχή.
Περιγραφή	Διαβάζει το επόμενο στοιχείο, επιστρέφοντας τη θέση του στη μνήμη ή NULL εάν δεν υπάρχει. Αν η παράμετρος `reset` είναι `true`, η ανάγνωση γίνεται από το πρώτο στοιχείο.

`void clear(uint8_t *data)`	
Παράμετροι	`uint8_t *data`: Δείκτης προς τη δομή που περιέχει τα δεδομένα της πηγής.
Περιγραφή	Διαγράφει τα περιεχόμενα της πηγής. Ολόκληρος ο χώρος της είναι πλέον διαθέσιμος για εγγραφή νέων στοιχείων.

Πίνακας 5.16: Διαπροσωπεία του τύπου `out_data_source`

Ο τύπος `out_data_source` αποτελείται από τα παρακάτω πεδία:

Πεδίο	**Περιγραφή**
`bool enabled`	Τιμή αληθείας για την ενεργοποίηση ή όχι της πηγής δεδομένων.
`uint8_t char_handle`	Το handle του χαρακτηριστικού στο οποίο αποστέλλονται οι ενημερώσεις.
`uint8_t char_len`	Το μήκος (σε bytes) της τιμής του χαρακτηριστικού.
`uint8_t type`	Για ενημερώσεις στο χαρακτηριστικό *Data Out*, αντιστοιχεί στο πεδίο *Opcode* του ορισμού της υποενότητας 4.5.1. Διαφορετικά, τίθεται στην προεπιλεγμένη τιμή *CUSTS1_CP_NONE*.
`uint8_t *raw_data`	Δείκτης προς τη δομή που περιέχει τα δεδομένα της πηγής.
`uint8_t (*get_data)(uint8_t *, bool);`	Δείκτης προς την υλοποίηση της συνάρτησης `get_data()`.
`void (*clear)(uint8_t *)`	Δείκτης προς την υλοποίηση της συνάρτησης `clear()`.
`int length`	Το μήκος (σε bytes) ενός στοιχείου της πηγής.

Πίνακας 5.17: Ο τύπος δεδομένων `out_data_source`

Για διαφορετικούς όμως τύπους δεδομένων, ο προγραμματιστής είναι αναγκαίο να ορίσει τις δικές του δομές. Μια απλή δομή ουράς συμβατή με το πρότυπο της `out_data_source` είναι ένας κυκλικός buffer (`circular_buffer`), για τον οποίο ορίζουμε την παρακάτω δομή (*user_utils/user_circular_buffer.h*):

Πεδίο	Περιγραφή
`int write_idx`	Δείκτης εγγραφής στον εσωτερικό πίνακα. Ισοδυναμεί με τη θέση του επόμενου στοιχείου προς εγγραφή.
`int read_idx`	Δείκτης ανάγνωσης στον εσωτερικό πίνακα. Χρησιμοποιείται για δοσοληψίες σειριακής ανάγνωσης. Με την ολοκλήρωση της διαδικασίας (ανάγνωση του τελευταίου στοιχείου), ο δείκτης λαμβάνει την τιμή-φρουρό *IDX_INVALID* (-1).
`int size`	Ο χώρος του buffer (πλήθος στοιχείων) που βρίσκεται σε χρήση.
`uint8_t *data`	Δείκτης προς τον εσωτερικό πίνακα που περιέχει τα δεδομένα του buffer.
`int length`	Το μήκος (σε bytes) ενός στοιχείου του buffer.
`int capacity`	Η χωρητικότητα του buffer (του εσωτερικού πίνακα), δηλαδή το μέγιστο πλήθος στοιχείων που μπορεί να περιέχει.

Πίνακας 5.18: Η δομή `circular_buffer`

Η δομή του buffer εσωκλείει τον πίνακα `data` με στοιχεία μήκους *length* και χωρητικότητα (μέγιστο πλήθος στοιχείων) *capacity*. Ο δείκτης *read_idx* λειτουργεί ως δείκτης ανάγνωσης του πίνακα, ενώ ο *write_idx* ως δείκτης εγγραφής. Όταν ο buffer είναι πλήρης, ο *write_idx* διαχωρίζει το πρώτο από το τελευταίο στοιχείο της ουράς, το οποίο βρίσκεται αμέσως στα αριστερά του. Έτσι, αν οι δύο δείκτες συμπέσουν, σημαίνει πως το πέρασμα του buffer έχει ολοκληρωθεί. Η εκκαθάριση του buffer (`circ_buf_clear()`) γίνεται με απλή επαναφορά των δεικτών στις αρχικές τους θέσεις. Ακολουθεί η παρουσίαση των βοηθητικών συναρτήσεων του `circular_buffer`:

`uint8_t *circ_buf_get_next(circular_buffer *c, bool reset)`	
Παράμετροι	`uint8_t condition`: Η νέα κατάσταση λειτουργίας.
Περιγραφή	Υλοποιεί την συνάρτηση `get_data()` του προτύπου της `out_data_source`. Εάν η παράμετρος `reset` είναι `true`, με την κλήση `circ_buf_reset()` ο δείκτης *read_idx* επιστρέφει στο πρώτο στοιχείο και η ανάγνωση ξεκινά από την αρχή.
`void circ_buf_clear(circular_buffer *c)`	
Παράμετροι	`circular_buffer *c`: Δείκτης προς τον buffer.
Περιγραφή	Υλοποιεί την συνάρτηση `clear()` του προτύπου της `out_data_source`. Εκκαθαρίζει τον buffer επαναφέροντας τους δείκτες εγγραφής και ανάγνωσης στη θέση 0.

`void circ_buf_add(circular_buffer *c, uint8_t *e)`	
Παράμετροι	`circular_buffer *c`: Δείκτης προς τον buffer. `uint8_t *e`: Δείκτης προς τα δεδομένα που θα εγγραφούν.
Περιγραφή	Προσθέτει ένα στοιχείο στον buffer, αυξάνοντας τον δείκτη εγγραφής *write_idx* (modulo *capacity*). Εάν ο δείκτης ανάγνωσης *read_idx* έχει μη έγκυρη τιμή (*IDX_INVALID*), μετακινείται στη θέση του νέου στοιχείου, ώστε αυτό να μπορεί να διαβαστεί.
`void circ_buf_reset(circular_buffer *c)`	
Παράμετροι	`circular_buffer *c`: Δείκτης προς τον buffer.
Περιγραφή	Επαναφέρει τον δείκτη *read_idx* στο πρώτο στοιχείο.

Πίνακας 5.19: Βοηθητικό API για τη δομή `circular_buffer`

Για τη διαχείριση και την πολυπλεξία ταυτόχρονων ροών από πολλές πηγές ορίζουμε μία βοηθητική *υπηρεσία ροής (streaming service)*. Αυτή διατάσσει τις υφιστάμενες πηγές σε μία *κυκλική* δομή ουράς, και επιτρέπει την αποστολή νέου πακέτου από οποιαδήποτε εξ αυτών μόνο κατόπιν αιτήματος. Παρέχει το εξής API:

`static bool out_data_source_send(out_data_source *source, bool reset)`	
Παράμετροι	`out_data_source *source`: Δείκτης προς την πηγή. `bool reset`: Τιμή αληθείας για ανάγνωση από την αρχή.
Περιγραφή	Έξοδος δεδομένων από την πηγή `source`: μέσω της κλήσης `source->get_data(source->raw_data, reset)` λαμβάνεται το επόμενο στοιχείο της πηγής προς μετάδοση. Αν αυτό υπάρχει, σχηματίζεται ένα πακέτο εξόδου σύμφωνα με τις προδιαγραφές της υποενότητας 4.5.2, και αποστέλλεται με notification στο χαρακτηριστικό του πεδίου `source->char_handle` (επιστρέφεται `true`). Εάν η ουρά είναι άδεια, η `get_data()` επιστρέφει *NULL* και η διαδικασία ολοκληρώνεται (επιστρέφεται `false`). Η συνάρτηση αυτή μπορεί να κληθεί μόνο από την `streaming_request_handler()`.
`void streaming_request_handler()`	
Περιγραφή	Καλείται για την έναρξη νέας μετάδοσης, ή μετά την έξοδο ενός πακέτου από μία πηγή. Εντοπίζει την επόμενη ενεργή πηγή της ουράς η οποία περιέχει δεδομένα προς αποστολή (με κυκλική εναλλαγή), και ξεκινά μετάδοση από αυτήν καλώντας `out_data_source_send()`.

`void streaming_source_enable(int id, bool enabled)`	
Παράμετροι	`int id`: Το αναγνωριστικό της πηγής. `bool reset`: Τιμή αληθείας για ενεργοποίηση της πηγής.
Περιγραφή	Ενεργοποιεί ή απενεργοποιεί την πηγή δεδομένων με αναγνωριστικό `id`.
`void streaming_service_reset()`	
Περιγραφή	Απενεργοποιεί όλες τις πηγές δεδομένων και επαναφέρει την υπηρεσία στην αρχική της κατάσταση, αγνοώντας οποιαδήποτε μετάδοση βρίσκεται σε εξέλιξη.
`bool streaming_service_clear_to_send()`	
Περιγραφή	Πραγματοποιεί έλεγχο για τυχόν μετάδοση που βρίσκεται σε εξέλιξη. Εάν δεν υπάρχει τέτοια επιστρέφεται `true`, οπότε και επιτρέπεται νέα μετάδοση, διαφορετικά `false`. Ο έλεγχος αυτός πρέπει να προηγείται κάθε νέου αιτήματος προς την υπηρεσία.

Πίνακας 5.20: API της υπηρεσίας ροής (user_utils/user_streaming_service.h)

5.4.2 Είσοδος δεδομένων

Ο χειρισμός των δεδομένων εισόδου (εντολών) γίνεται από έναν ειδικά ορισμένο handler (`data_wr_ind_handler()`, *user_app/user_custs1_impl.c*), ο οποίος ενεργοποιείται με κάθε αίτημα εγγραφής (*Write Request*) προς το χαρακτηριστικό *Data In* (*Control Point*), επιστρέφει στον client την απόκριση *Write Response*, και στη συνέχεια εκτελεί την ενέργεια που αντιστοιχεί στο *opcode* του πακέτου εισόδου, όπως ορίζεται στην υποενότητα 4.5.1. Επιπλέον ενέργειες προς πραγματοποίηση κατά τη φάση αυτή περιλαμβάνουν:

➔ Για πακέτο *CP_SAMPLING_START*, ενεργοποίηση όλων των πηγών δεδομένων που αντιστοιχούν στους εκάστοτε ενεργούς αισθητήρες, και απενεργοποίηση των υπολοίπων.

➔ Για πακέτο *CP_SAMPLING_STOP*, εκκαθάριση του περιεχομένου όλων των πηγών δεδομένων.

➔ Για πακέτο *CP_RETRIEVE_LOG*, ενεργοποίηση της πηγής δεδομένων για εγγραφές ιστορικού (*LOG_SOURCE*), και στη συνέχεια απόπειρα αποστολής των εγγραφών με αίτημα προς την υπηρεσία ροής δεδομένων (streaming service).

5.4.3 Καταγραφή ιστορικού

Υλοποιούμε τον τύπο δεδομένων της εγγραφής ιστορικού (`log_entry`, *user_app_api/user_log.h*), σύμφωνα με τον ορισμό της 4.5.3, ως δομή με τα εξής πεδία:

Πεδίο	**Περιγραφή**
`uint8_t timestamp[]`	Αντιστοιχεί στο πεδίο *Timestamp*.
`uint8_t type`	Αντιστοιχεί στο πεδίο *Type*.
`float measured_value`	Αντιστοιχεί στο πεδίο *Measured Value*.

Πίνακας 5.21: Η δομή `log_entry`

Οι εγγραφές ιστορικού παράγονται από το API εποπτείας που παρουσιάζουμε στην υποενότητα 5.4.6.

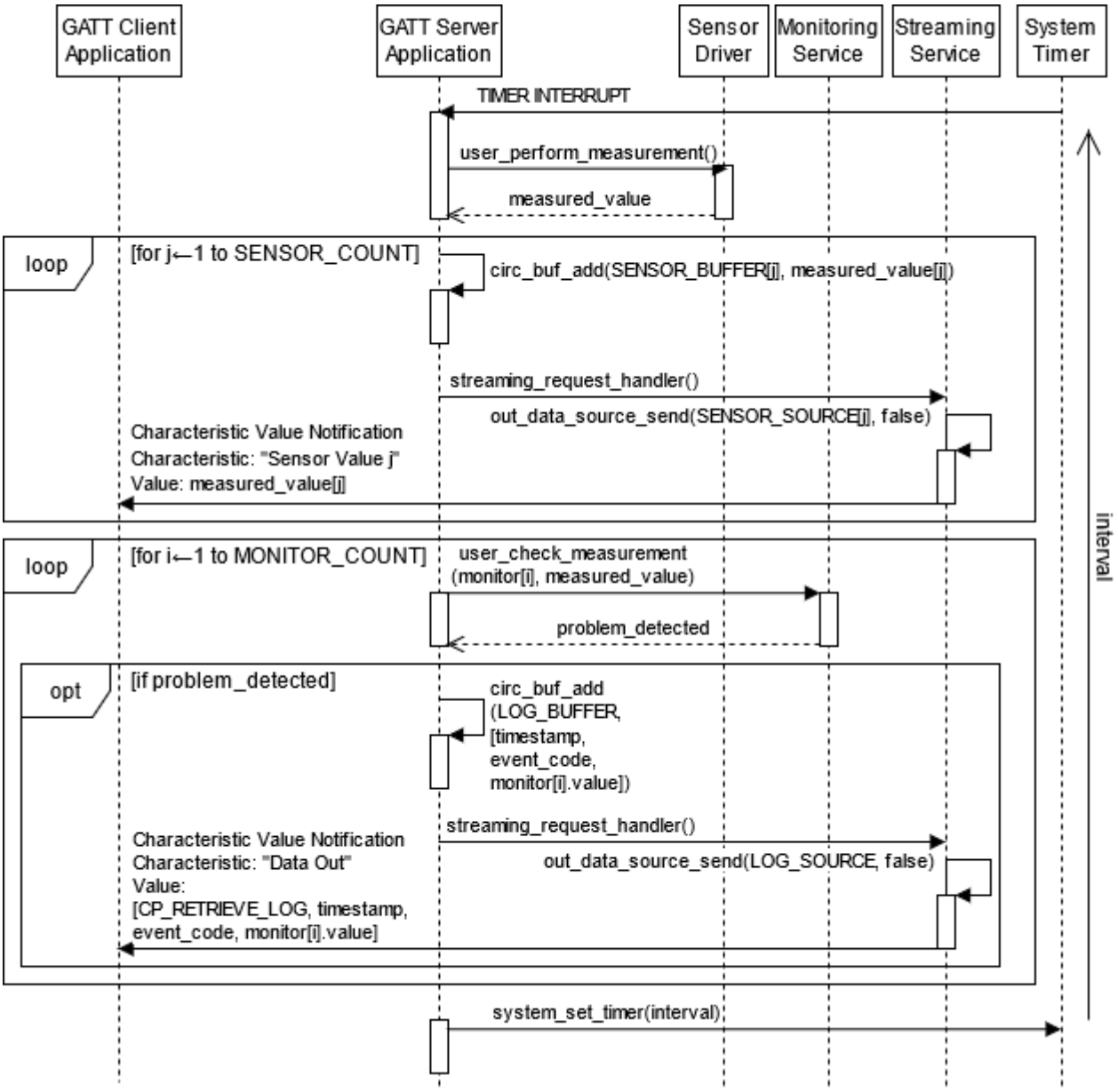
5.4.4 Περιοδική δειγματοληψία

Με τη λήψη του μηνύματος *CP_SAMPLING_START*, το peripheral εκκινεί την περιοδική δειγματοληψία, ενεργοποιώντας τον χρονιστή συστήματος με την συνάρτηση επανάκλησης `user_adc_callback()` (*user_app/user_custs1_impl.c*) και την περίοδο που έχει ορισθεί στο μήνυμα *CP_SAMPLING_SET_INTERVAL*. Αν το τελευταίο δεν έχει ληφθεί, για την περίοδο δειγματοληψίας χρησιμοποιείται από τη συσκευή μία προεπιλεγμένη τιμή αντίστοιχη των μετρώμενων μεγεθών (50 ms για το IM Service, 1 s για το ES Service). Η `user_adc_callback()` πραγματοποιεί τις παρακάτω ενέργειες:

- Κλήση της συνάρτησης `user_perform_measurement()` του προγράμματος οδήγησης του αισθητήρα. Στην προδιαγραφή και την υλοποίηση των προγραμμάτων οδήγησης αναφερόμαστε στις υποενότητες 5.5.1-5.5.2.
- Προσθήκη του τρέχοντος δείγματος για κάθε εποπτευόμενο μέγεθος στην αντίστοιχη πηγή δεδομένων εξόδου.
- Αίτημα προς την υπηρεσία ροής (streaming service) για έναρξη αποστολής των νέων δεδομένων του αισθητήρα στον client, με χρήση της συνάρτησης `streaming_request_handler()`.
- Ενημέρωση των τιμών όλων των ενεργών παρακολουθητών με τις τελευταίες μετρήσεις, εάν αυτό είναι δυνατό (οι παρακολουθητές του IM Service ενημερώνονται μόνο ασύγχρονα), με κλήση της βοηθητικής συνάρτησης `user_check_measurement()`. Αναλυτικά στη διαδικασία ενημέρωσης αναφερόμαστε στην υποενότητα 5.4.6.

- Για κάθε παρακολουθητή που παραβιάζει τα όρια ορθής λειτουργίας, κατασκευή μίας εγγραφής ιστορικού, προσθήκη της στην πηγή *LOG_SOURCE* και αποστολή της στον client με κλήση της `streaming_request_handler()`.
- Επανενεργοποίηση του χρονιστή συστήματος για τη λήψη του επόμενου δείγματος.

Η ροή εκτέλεσης – με ορισμένες απλοποιήσεις – απεικονίζεται στο παρακάτω διάγραμμα:



Σχήμα 5.22: Περιοδική δειγματοληψία

5.4.5 Εκπομπή

Η FSM που παρουσιάσαμε στην παράγραφο 4.3.2.2 υλοποιείται με χρήση ενός callback και δύο βοηθητικών συναρτήσεων στον κύριο κώδικα της εφαρμογής εξυπηρετητή (*user_app*/*user_peripheral.c*). Η συνάρτηση επανάκλησης `adv_data_update_callback()` καλείται με κάθε συμπλήρωση της περιόδου ενημέρωσης T_U για την ενημέρωση της κατάστασης της συσκευής (`adv_data_update()`), και τελικά προγραμματίζει την επόμενη ενημέρωση (`adv_start()`) επανενεργοποιώντας τον kernel timer. Στη συνέχεια, γίνεται μετάδοση των advertising data για διάρκεια ίση με τη διάρκεια εκπομπής t_B.

5.4.6 Ενημέρωση κατάστασης

Η εφαρμογή εξυπηρετητή ενσωματώνει απαραίτητα τις παρακάτω συναρτήσεις, οι οποίες χρησιμοποιούν τους παρακολουθητές της συσκευής για να ενημερώνουν την τοπική ένδειξη της κατάστασης λειτουργίας της. Οι συναρτήσεις αυτές συνθέτουν το τμήμα του *API εποπτείας* που είναι κοινό σε όλα τα CM Services (*user_app*/*user_monitor.h*):

`void user_update_state(uint8_t *data)`	
Παράμετροι	`uint8_t *data`: Δείκτης προς τα δεδομένα εκπομπής.
Περιγραφή	Για εποπτεία χωρίς σύνδεση, ενημερώνει τα δεδομένα εκπομπής της συσκευής με την τρέχουσα κατάσταση λειτουργίας και το επίπεδο φόρτισης της μπαταρίας.
`uint8_t user_get_condition()`	
Περιγραφή	Επιστρέφει την τρέχουσα κατάσταση λειτουργίας.
`void user_set_condition(uint8_t condition)`	
Παράμετροι	`uint8_t condition`: Η νέα κατάσταση λειτουργίας.
Περιγραφή	Ενημερώνει την κατάσταση λειτουργίας της συσκευής.
`void user_get_timestamp(uint8_t *out_data)`	
Παράμετροι	`uint8_t *out_data`: Πίνακας για τα δεδομένα εξόδου.
Περιγραφή	Διαβάζει την τρέχουσα ημερομηνία και ώρα από το RTC, και την αποθηκεύει στον πίνακα `out_data` στη μορφή που ορίζεται στην παράγραφο 4.5.3.

Πίνακας 5.23: Γενικό API εποπτείας

5.4.6.1 Ενημέρωση κατάστασης στο ES Service

Για τους παρακολουθητές του ES Service προστίθενται στο γενικού σκοπού API του προηγούμενου πίνακα οι παρακάτω συναρτήσεις:

`void user_check_measurement(float val, measurement_monitor *monitor)`	
Παράμετροι	`float val`: Νέα μέτρηση μεγέθους από αισθητήρα. `measurement_monitor *monitor`: Δείκτης προς τον παρακολουθητή που θα ενημερωθεί.
Περιγραφή	Ελέγει κατά πόσο η μετρηθείσα τιμή `val` πληροί τις συνθήκες καλής λειτουργίας, και αντίστοιχα ενημερώνει την τιμή ενός παρακολουθητή.
`void user_update_monitor(env_sensor_output *out_data)`	
Παράμετροι	`env_sensor_output *out_data`: Δείκτης προς το διάνυσμα μετρήσεων των αισθητήρων.
Περιγραφή	Ενημερώνει όλους τους παρακολουθητές βάσει των ληφθεισών μετρήσεων, καλώντας κάθε φορά την `user_check_measurement()`.
`void user_reset_monitor(uint16_t interval)`	
Παράμετροι	`uint16_t interval`: Περίοδος δειγματοληψίας για όλα τα μεγέθη του παρακολουθητή.
Περιγραφή	Επαναφέρει κάθε παρακολουθητή στην αρχική του κατάσταση, θέτοντας επιπλέον την τρέχουσα τιμή του που διατηρείται στην προσωρινή μνήμη στην τιμή-φρουρό *MONITOR_SENTINEL = FLT_MAX* (μέγιστη δυνατή τιμή για τον τύπο `float`).

Πίνακας 5.24: Προσθήκες στο API εποπτείας για το ES Service

5.4.6.2 Ενημέρωση κατάστασης στο IM Service

Επιπλέον των βασικών συναρτήσεων, το API εποπτείας για το IM Service περιλαμβάνει την ρουτίνα εξυπηρέτησης διακοπών `user_event_callback()`:

`void user_event_callback()`	
Περιγραφή	Καλείται κατόπιν αιτήματος διακοπής από τον αισθητήρα. Εξετάζει την κατάσταση των εισόδων διακοπής, και προσδιορίζοντας τα αντίστοιχα συμβάντα παράγει κατάλληλες εγγραφές ιστορικού.

Πίνακας 5.25: Προσθήκες στο API εποπτείας για το IM Service

5.5 Επίπεδο αφαίρεσης υλικού

Στο σημείο αυτό θα εξετάσουμε τις απαραίτητες ενέργειες σε επίπεδο λογισμικού για την επικοινωνία της εφαρμογής εξυπηρετητή με τις περιφερειακές συσκευές, καθώς και τον τρόπο με τον οποίο εφαρμόζονται στις τελευταίες οι ρυθμίσεις χρήστη με κάθε τροποποίηση του DCS. Η υλοποίηση των ενεργειών αυτών συνιστά για μία συσκευή ένα *πρόγραμμα οδήγησης* (driver), και εξαρτάται από την υπηρεσία CM και το υλικό που χρησιμοποιείται κάθε φορά. Προκειμένου να διαχωρισθεί ο ορισμός των drivers σε λογικό επίπεδο από τον υπόλοιπο κώδικα της εφαρμογής και τον πραγματικό κώδικα επικοινωνίας με το υλικό, παρεμβάλλουμε μεταξύ τους ένα API, μέσω του οποίου οι drivers θα είναι ορατοί στην εφαρμογή. Η υλοποίηση των συναρτήσεων του API μπορεί να προσαρμόζεται κάθε φορά που αλλάζει το χρησιμοποιούμενο υλικό. Το driver API, το DIS και το DCS συνθέτουν με αυτό τον τρόπο ένα *επίπεδο αφαίρεσης* (abstraction layer) μεταξύ της εφαρμογής και του υλικού. Στο εξής, αν δεν υπάρχει άλλη διευκρίνιση, όποτε αναφερόμαστε σε προγράμματα οδήγησης θα εννοούμε υλοποιήσεις συμβατές με το driver API που ορίζεται εδώ.

5.5.1 Πρόγραμμα οδήγησης Environmental Sensor

Η εφαρμογή επικοινωνεί με το HDC1008 μέσω του διαύλου I^2C. Η δειγματοληψία γίνεται με αίτημα ανάγνωσης του ζεύγους καταχωρητών 0x00-0x01 (ή μόνο του καταχωρητή 0x01, αν απαιτείται μόνον μέτρηση υγρασίας) και αναμονή μέχρι την ολοκλήρωση της μέτρησης, η οποία σηματοδοτείται από την κάθοδο του σήματος $\overline{DATA\_READY}$. Η διάρκεια αναμονής είναι πολύ μικρή σε σύγκριση με την περίοδο δειγματοληψίας. Στη συνέχεια, από το δίαυλο διαβάζονται 4 ή 2 bytes, ανάλογα με τα μεγέθη που ζητήθηκαν.

Οι τιμές της θερμοκρασίας *T* και της σχετικής υγρασίας *RH* υπολογίζονται αντίστοιχα από τα 16-bit απροσήμαστα ψηφιακά δείγματα *TD, HD* βάσει των παρακάτω σχέσεων ανακατασκευής:

Θερμοκρασία (T):
$$T = T_L + (T_H - T_L)\frac{TD[0:15]}{2^{16}}$$

Σχετική υγρασία (RH):
$$RH = 100.0\frac{HD[0:15]}{2^{16}}$$

Σημειώνουμε πως στην πρώτη σχέση, $[T_L, T_H]$ είναι το εύρος τιμών της θερμοκρασίας. Ο οδηγός του HDC1008 εκθέτει στην εφαρμογή το παρακάτω API (*user_driver*/*user_env_sensor.h*):

`void user_perform_measurement(env_sensor_output *output)`	
Παράμετροι	`env_sensor_output *output`: Δείκτης προς τα δεδομένα εξόδου.
Περιγραφή	Λαμβάνει ένα δείγμα για κάθε μέγεθος που ζητήθηκε, με τον τρόπο που περιγράφηκε παραπάνω.
`void user_sensor_update_config()`	
Περιγραφή	Εφαρμόζει τις νέες ρυθμίσεις του DCS στον αισθητήρα. Αυτό ισοδυναμεί με εγγραφή μίας από τις παρακάτω τιμές στον καταχωρητή *Configuration* (0x00) του HDC1008: → 0x00 για μέτρηση μόνο θερμοκρασίας → 0x01 για μέτρηση μόνο υγρασίας → 0x10 για μέτρηση θερμοκρασίας και υγρασίας

Πίνακας 5.26: API του οδηγού του Environmental Sensor

5.5.2 Πρόγραμμα οδήγησης Inertial Measurement Unit

Για το BMI160 έχει ήδη αναπτυχθεί driver ανοικτού κώδικα (https://github.com/BoschSensortec/BMI160_driver) από τον κατασκευαστή. Οι απαραίτητες ενέργειες E/E για την δειγματοληψία και τη μεταβολή των ρυθμίσεων του αισθητήρα (αναγνώσεις/εγγραφές καταχωρητών) ορίζονται στον driver, συνεπώς οι συναρτήσεις που αναπτύσσουμε εδώ λειτουργούν απλώς ως διεπαφή του προς την εφαρμογή.

Εφόσον η καταγραφή και η ενημέρωση για συμβάντα τόσο εντός όσο εκτός σύνδεσης γίνεται ασύγχρονα (μέσω IRQ) και όχι με περιοδική δειγματοληψία, η λειτουργία της `user_perform_measurement()` περιορίζεται στον υπολογισμό των συνιστωσών της δύναμης επιτάχυνσης α_p και της γωνιακής ταχύτητας ω από τα ληφθέντα ψηφιακά δείγματα *AD* και *ΩD* αντίστοιχα. Αυτός γίνεται βάσει των παρακάτω σχέσεων ανακατασκευής:

$$\textit{Δύναμη επιτάχυνσης } (\alpha_p): \qquad (a_{px}, a_{py}, a_{pz}) = R_a \frac{(AD_X, AD_Y, AD_Z)[0:15]}{2^{15}}$$

$$\textit{Γωνιακή ταχύτητα } (\omega): \qquad (\omega_x, \omega_y, \omega_z) = R_\omega \frac{(\Omega D_X, \Omega D_Y, \Omega D_Z)[0:15]}{2^{15}}$$

Σημειώνουμε πως στις παραπάνω σχέσεις, η διαίρεση γίνεται με το 2^{15} διότι τα *AD*, HD είναι 16-bit απροσήμαστοι αριθμοί, συνεπώς το MSB λειτουργεί ως bit προσήμου. Επιπλέον, με R_α και R_ω συμβολίζουμε αντίστοιχα το εύρος μέτρησης του

επιταχυνσιομέτρου και του γυροσκοπίου. Παρακάτω δίνουμε το API του οδηγού της IMU (*user_driver*/*user_imu.h*):

`void user_imu_init()`	
Περιγραφή	Καλείται μόνο κατά την εκκίνηση του συστήματος, και εφαρμόζει τις προεπιλεγμένες ρυθμίσεις στον αισθητήρα.
`void user_sensor_update_config()`	
Περιγραφή	Εφαρμόζει στον αισθητήρα τις ρυθμίσεις που έχουν ορισθεί στο DCS.
`void user_perform_measurement(imu_output *output)`	
Παράμετροι	`imu_output *output`: Δείκτης προς τα δεδομένα εξόδου.
Περιγραφή	Χρησιμοποιείται μόνο κατά τη διάρκεια μίας συνόδου εποπτείας. Λαμβάνει ένα δείγμα από κάθε ενεργό αισθητήρα.
`void user_sensor_power_on()`	
Περιγραφή	Εκκινεί όλους τους αισθητήρες που έχουν ενεργοποιηθεί από το χρήστη. Το επιταχυνσιόμετρο και το γυροσκόπιο του BMI160 εκκινούνται επιλέγοντας τις λειτουργίες κατανάλωσης *Low Power* και *Normal* αντίστοιχα.
`void user_sensor_power_off()`	
Περιγραφή	Σταματά τη λειτουργία όλων των αισθητήρων. Για το BMI160, επιλέγεται η λειτουργία κατανάλωσης *Suspend*.

Πίνακας 5.27: API του οδηγού της IMU

5.5.3 Πρόγραμμα οδήγησης RTC

Η επικοινωνία με το RTC DS1347 γίνεται μέσω του διαύλου SPI. Ορίζουμε τις συναρτήσεις `user_get_date_time()` και `user_set_date_time()` αντίστοιχα για τη λήψη και τη ρύθμιση της τρέχουσας ημερομηνίας και ώρας (*user_driver*/*user_rtc.h*):

`void user_get_date_time(uint8_t *data)`	
Παράμετροι	`uint8_t *data`: Δείκτης προς τα δεδομένα εξόδου, στη μορφή που αυτά γίνονται αντιληπτά από το RTC.
Περιγραφή	Εξάγει την τρέχουσα ημερομηνία και ώρα του RTC.

`void user_set_date_time(uint8_t *data)`	
Παράμετροι	`uint8_t *data`: Δείκτης προς τα δεδομένα εισόδου, στη μορφή που αυτά γίνονται αντιληπτά από το RTC (στην περίπτωσή μας, το DS1347).
Περιγραφή	Επαναρρυθμίζει την τρέχουσα ημερομηνία και ώρα του RTC.

Πίνακας 5.28: API του οδηγού του RTC

Το bytestream εξόδου του RTC μπορεί να μετατραπεί σε timestamp με τη βοήθεια της συνάρτησης `user_get_timestamp()` του API εποπτείας.

5.5.4 Υλοποίηση DIS

Παρακάτω παρουσιάζουμε τη δομή του DIS που χρησιμοποιείται από την εφαρμογή εξυπηρετητή, σύμφωνα με τον ορισμό της υποενότητας 4.5.4 (*user_app_api*/*user_service_def.h*):

Πεδίο	**Περιγραφή**
`uint8_t timer_unit`	Αντιστοιχεί στο πεδίο *Timer Unit*. Για τον kernel timer του DA14583, η τιμή του ισούται με 10.
`uint8_t timer_max`	Αντιστοιχεί στο πεδίο *Timer Max*. Για τον kernel timer του DA14583, η τιμή του ισούται με 30000 (300 s).
`char config_labels[]`	Αντιστοιχεί στο πεδίο *Setting Values*. Για το ES Service με χρήση του HDC1008, η τιμή του ισούται με `"8,11,14_11,14"`. Για το IM Service με χρήση του BMI160, η τιμή του ισούται με `"2,4,8,16_125,250,500,1000,2000"`.

Πίνακας 5.29: Υλοποίηση της δομής του DIS

5.5.5 Υλοποίηση DCS

Παρακάτω παρουσιάζουμε τη δομή του DCS που χρησιμοποιείται από την εφαρμογή εξυπηρετητή, σύμφωνα με τον ορισμό της υποενότητας 4.5.5 (*user_app_api*/*user_service_def.h*):

Πεδίο	**Περιγραφή**
`monitor_configuration_struct monitor_config[]`	Συλλογή των MCS για το σύνολο των παρακολουθητών.
`sensor_configuration_struct sensor_config[]`	Συλλογή των SCS για το σύνολο των αισθητήρων.

`uint16_t update_interval`	Αντιστοιχεί στο πεδίο *Update Interval*. Κατά την εκκίνηση της συσκευής, λαμβάνει την προεπιλεγμένη τιμή 30000 (30 s).
`uint16_t broadcast_timeout`	Αντιστοιχεί στο πεδίο *Broadcast Timeout*. Κατά την εκκίνηση της συσκευής, λαμβάνει την προεπιλεγμένη τιμή 10000 (10 s).

Πίνακας 5.30: Υλοποίηση της δομής του DCS

- Κάθε MCS υλοποιείται ως δομή του τύπου `monitor_configuration_struct`, αποτελούμενη από τα εξής πεδία (*user_app*/*user_monitor.h*):

Πεδίο	**Περιγραφή**	**Προεπιλεγμένη τιμή**
`float values[]`	Αντιστοιχεί στο πεδίο *Control Values*.	Για τον παρακολουθητή *Temperature*, `{10.0f, 30.0f}`. Για τον παρακολουθητή *Temperature Change*, `{-0.1f, 3.0f}`. Για τον παρακολουθητή *Humidity*, `{43.0, 85.0}`. Για τον παρακολουθητή *Humidity Change*, `{-10.0, 10.0f}`. Για τον παρακολουθητή *Motion Detection*, `{1.70f}`. Για τον παρακολουθητή *Shock Detection*, `{4.00f}`.
`uint8_t alert_level`	Αντιστοιχεί στο πεδίο *Alert Level*.	*ALERT_LEVEL_NONE* (0x00).

Πίνακας 5.31: Υλοποίηση MCS

- Κάθε SCS ενσωματώνεται στον driver του αντίστοιχου αισθητήρα, και υλοποιείται ως δομή του τύπου `sensor_configuration_struct`:

Πεδίο	**Περιγραφή**	**Προεπιλεγμένη τιμή**
`uint8_t values[]`	Αντιστοιχεί στο πεδίο *Setting Values*.	Για το θερμόμετρο, `{0}`. Για το υγρόμετρο, `{0}`. Για το επιταχυνσιόμετρο, `{0}`. Για το γυροσκόπιο, `{0}`.
`uint8_t mode`	Αντιστοιχεί στο πεδίο *Sensor Mode*.	Για το γυροσκόπιο, *SENSOR_OFF* (0x01). Για τους υπόλοιπους αισθητήρες, *SENSOR_ON* (0x00).

Πίνακας 5.32: Υλοποίηση SCS

Σημειώνουμε πως οι ρυθμίσεις που ορίζονται σε κάθε SCS εφαρμόζονται με κλήση της συνάρτησης `user_sensor_update_config()` του οδηγού του αντίστοιχου αισθητήρα.

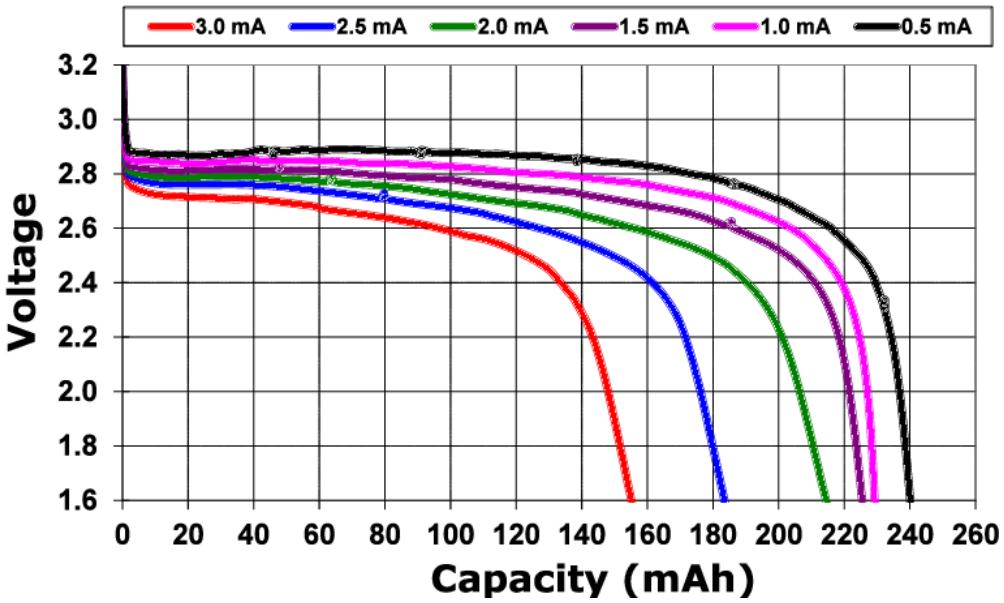
5.6 Ειδικά ζητήματα

Στο σημείο αυτό θα εξετάσουμε κάποια ζητήματα υλοποίησης που επηρεάζουν καθοριστικά την απόδοση των coin sensors.

5.6.1 Διάρκεια ζωής

Όπως έγινε φανερό από την ανάλυση του κεφαλαίου 1, η αυτονομία της πηγής ενέργειας των κόμβων-αισθητήρων αποτελεί βασικό δείκτη απόδοσης για κάθε WSN. Για να εκτιμήσουμε τη διάρκεια ζωής της μπαταρίας της συσκευής που υλοποιούμε στην παρούσα εργασία, θα προσπαθήσουμε να προσεγγίσουμε την κατανάλωση ενέργειας για κάποια εύλογα σενάρια χρήσης.

Η ονομαστική τιμή της *χωρητικότητας* μίας μπαταρίας που δίνεται από τον κατασκευαστή εκφράζει την αποθηκευμένη χημική ενέργεια που μπορεί να μετατραπεί σε ηλεκτρική, αναφέρεται πάντα στην *ονομαστική* τάση λειτουργίας, και δίνεται σε μονάδες ηλεκτρικού φορτίου (συνήθως Ah). Μία μπαταρία CR2032 λειτουργεί συνήθως σε τάσεις 2-3 V, παρέχοντας χωρητικότητα 200-250 mAh, και προορίζεται για καταναλώσεις της τάξης δεκάδων μA έως μερικών mA.



Εικόνα 5.33: Παράδειγμα χαρακτηριστικής εκφόρτισης μπαταρίας CR2032

Είναι εμφανές πως ο ρυθμός με τον οποίο απορροφάται η αποθηκευμένη ενέργεια εξαρτάται τόσο από την στιγμιαία τάση λειτουργίας του τροφοδοτούμενου κυκλώματος, όσο και από το καταναλισκόμενο ρεύμα. Κατά συνέπεια, υπό ταχέως μεταβαλλόμενες συνθήκες λειτουργίας η πραγματική χωρητικότητα της μπαταρίας μπορεί να αποκλίνει σε μεγάλο βαθμό από την ονομαστική τιμή του κατασκευαστή.

Η σχέση μεταξύ της τάσης λειτουργίας, του ρεύματος και της χωρητικότητας μίας μπαταρίας αναπαρίσταται συχνά με τη βοήθεια μίας χαρακτηριστικής *καμπύλης εκφόρτισης*, όπως είναι η παρακάνω.

Διατηρώντας σταθερή την τάση λειτουργίας του συστήματος, το συνολικό φορτίο που έχει προσφέρει η μπαταρία στο κύκλωμα μέχρι τη χρονική στιγμή t εκφράζεται ως εξής:

$$Q(t) = \int_0^t I(\tau)d\tau = \bar{I}t$$

όπου $\bar{I}$ η μέση τιμή του ρεύματος I στο διάστημα $[0, t]$.

Στην περίπτωσή μας, η τάση λειτουργίας παραμένει πράγματι σταθερή στα 3.3 V. Έτσι, αν Q_T είναι η χωρητικότητα που αντιστοιχεί σε αυτή την τιμή της τάσης, η εκφόρτιση ολοκληρώνεται με την κυκλοφορία συνολικού φορτίου ίσου με Q_T, συνεπώς ο χρόνος ζωής της μπαταρίας t_L ισούται με το χρόνο που απαιτείται για αυτό:

$$t_L = \frac{Q_T}{\bar{I}}$$

Αρκεί λοιπόν η μέτρηση του ρεύματος λειτουργίας κατά την εκτέλεση καθενός από τα σενάρια, τα οποία διαφοροποιούνται ως προς τους αισθητήρες που ενεργοποιούνται και τις ρυθμίσεις αυτών (πχ εύρος, ακρίβεια, συχνότητα δειγματοληψίας). Κατά την εκτέλεση ενός σεναρίου μεταβάλλονται τα εξής:

- Το ποσοστό του χρόνου παραμονής σε σύνδεση. Υποθέτουμε πως όσο παραμένουμε σε σύνδεση, δειγματοληπτούμε τα μεγέθη των αισθητήρων.
- Εκτός σύνδεσης, το ποσοστό του χρόνου εκπομπής στο δίκτυο, δηλαδή το λόγο της διάρκειας εκπομπής t_B προς την περίοδο ενημέρωσης T_U.

Πριν παρουσιάσουμε τα σενάρια χρήσης που χρησιμοποιούμε, θα δείξουμε πώς εξαρτάται η κατανάλωση ενέργειας από τις παραπάνω παραμέτρους. Το μέσο ρεύμα λειτουργίας μπορεί να υπολογισθεί θεωρητικά ως εξής:

$$\bar{I} = \sum_{i=1}^{n} d_i I_i + \left(1 - \sum_{i=1}^{n} d_i\right) I_{off} \quad (5.34)$$

όπου

- I_{off} το ρεύμα στην ανενεργό κατάσταση (εξοικονόμηση ενέργειας, sleep mode)
- $\{P_1, ..., P_n\}$ το σύνολο των ενεργειών της συσκευής σε ενεργή κατάσταση
- d_i το ποσοστό του χρόνου ζωής κατά το οποίο η συσκευή πραγματοποιεί την ενέργεια P_i

- I_i το μέσο ρεύμα στο διάστημα αυτό

Στην περίπτωση του coin sensor που κατασκευάσαμε, οι ενέργειες αυτές είναι:

P_1: Ανάγνωση/εγγραφή τιμής από/προς αισθητήρα ή περιφερειακό μέσω των διαύλων SPI/I^2C
P_2: Μετάδοση πακέτων διαφήμισης (advertising event)
P_3: Μετάδοση πακέτων σύνδεσης (connection event)
P_4: Αφύπνιση της συσκευής, είτε σύγχρονα είτε λόγω εξωτερικής διακοπής υλικού

Η σχέση (5.34) μπορεί να χρησιμοποιηθεί λαμβάνοντας για τις αναμενόμενες τιμές του ρεύματος κατά την εκτέλεση των παραπάνω ενεργειών τις αντίστοιχες ονομαστικές τιμές που δίνονται από τους κατασκευαστές, είτε μετρώντας το ρεύμα λειτουργίας σε πραγματικό χρόνο και απομονώνοντας τα αντίστοιχα χρονικά διαστήματα. Μετρώντας όμως σε πιο εκτεταμένα χρονικά διαστήματα, συγκεκριμένα το μέσο ρεύμα I_c κατά την παραμονή σε σύνδεση και το μέσο ρεύμα εκτός σύνδεσης I_a , από τις παραδοχές που κάναμε για τα σενάρια χρήσης είναι εμφανές πως οι ενέργειες $\{P_1, P_3, P_4\}$ ενσωματώνονται στο I_c, ενώ οι ενέργειες $\{P_1, P_2, P_4\}$ στο I_a. Η (5.34) επαναδιατυπώνεται συνεπώς ως εξής:

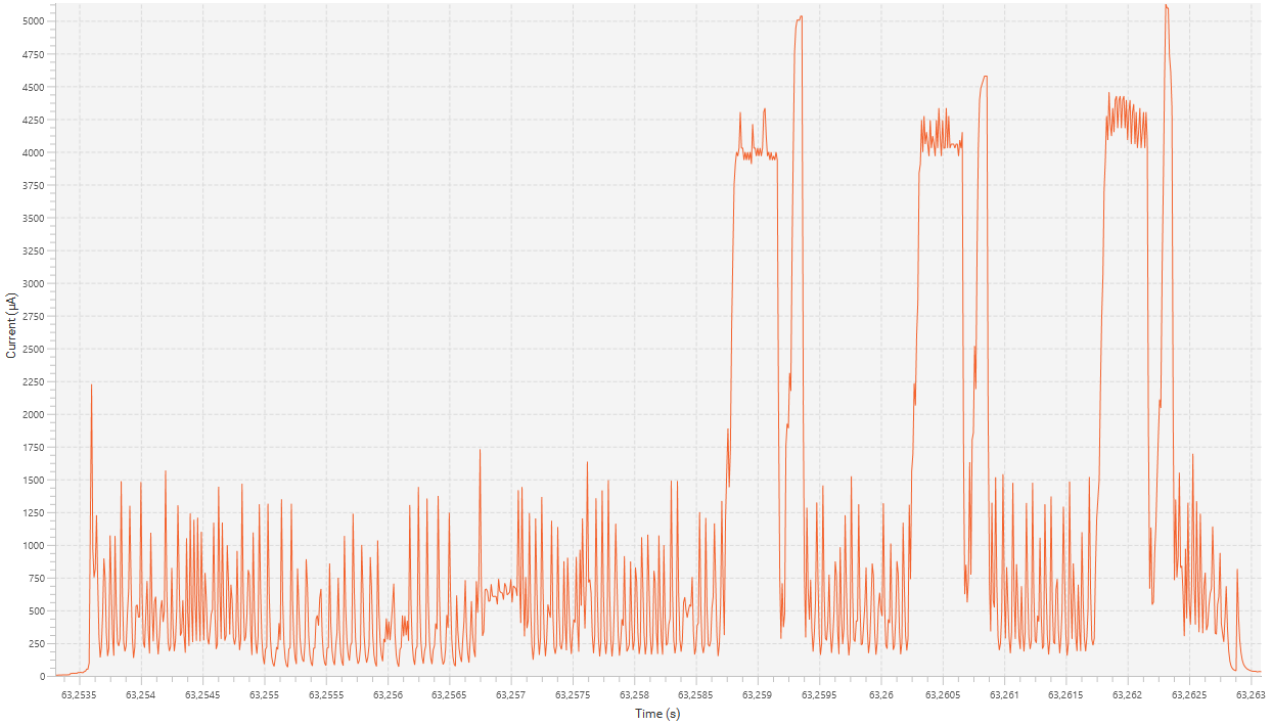
$$\bar{I} = d_c I_c + (1 - d_c)\left[d_a I_a + (1 - d_a) I_{off}\right] \quad (5.35)$$

Διατηρώντας τώρα σταθερά τα BLE connection και advertising intervals στα 50 και 100 ms αντίστοιχα, διαμορφώνουμε τα παρακάτω σενάρια:

- *Σενάριο 1*: Θερμόμετρο και υγρόμετρο με δειγματοληψία στο 1 Hz
- *Σενάριο 2*: Επιταχυνσιόμετρο με δειγματοληψία στα 100 Hz
- *Σενάριο 3*: Επιταχυνσιόμετρο και γυροσκόπιο με δειγματοληψία στα 50 Hz

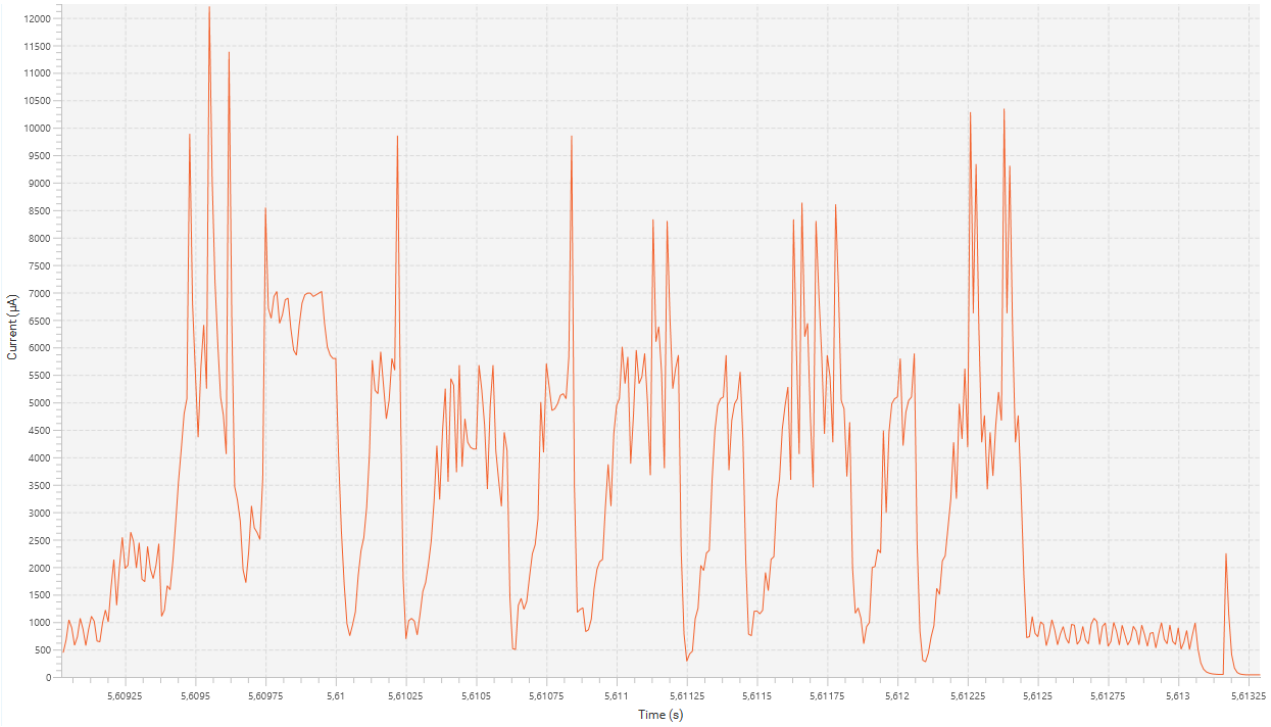
Εκτελούμε τα σενάρια σε συνεργασία με την κινητή συσκευή Samsung J330F ως πελάτη. Για τη μέτρηση του ρεύματος χρησιμοποιούμε την πλακέτα X-NUCLEO-LPM01A της STMicroelectronics.

Παρακάτω φαίνεται το αποτέλεσμα της μέτρησης του καταναλισκόμενου ρεύματος του Accelerometer Sensor κατά τη διάρκεια ενός advertising event. Με την αφύπνιση της συσκευής μεταδίδονται διαδοχικά τρία πακέτα εκπομπής, ένα για καθένα από τα κανάλια εκπομπής 37, 38 και 39:



Σχήμα 5.36: Κατανάλωση ρεύματος κατά την εκπομπή

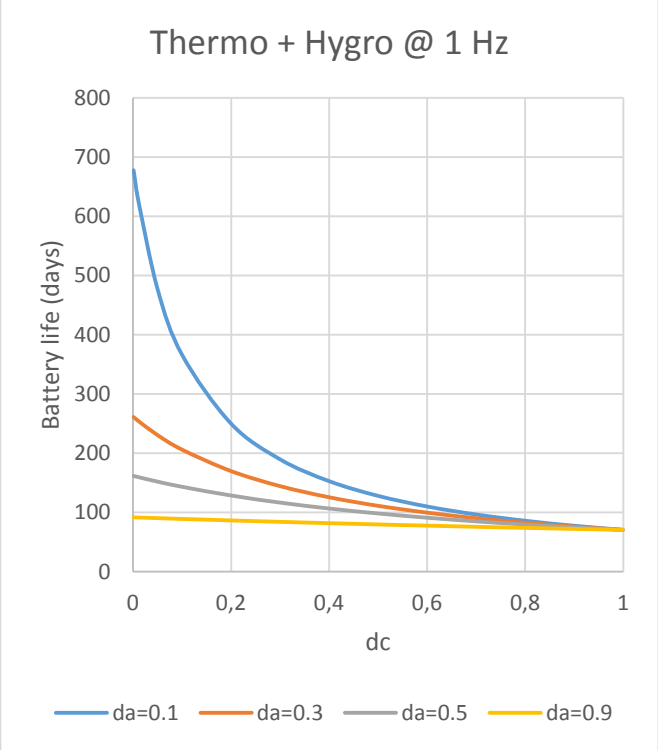
Ακολουθεί το καταναλισκόμενο ρεύμα του Accelerometer Sensor κατά τη συνεχή δειγματοληψία στα 100 Hz, για ένα connection event. Εντός του connection interval (50 ms) μεταδίδονται 5 πακέτα δεδομένων:



Σχήμα 5.37: Κατανάλωση ρεύματος σε σύνδεση

Τέλος, στα παρακάτω σενάρια το μέσο ρεύμα $\overline{I}$ υπολογίζεται από την (5.35):

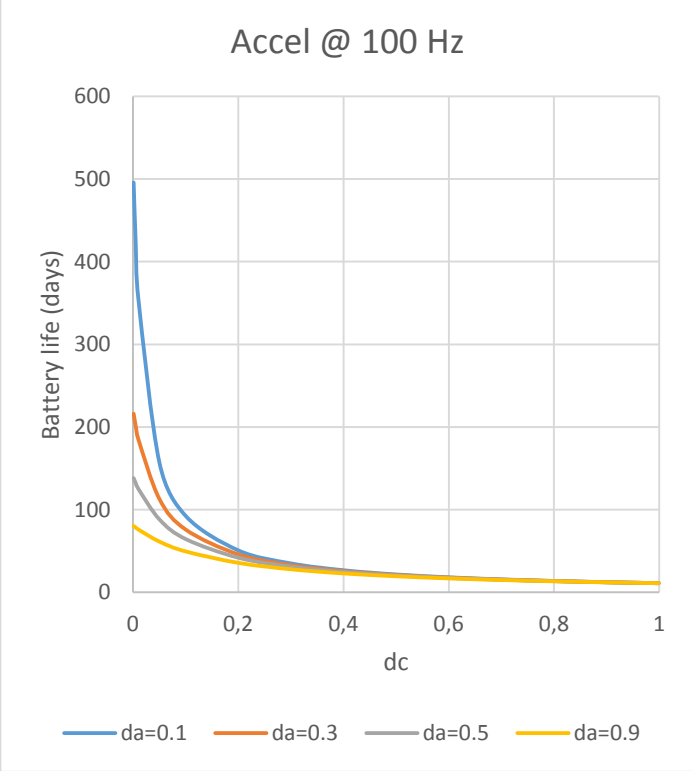
i. Θερμόμετρο/Υγρόμετρο, 1 Hz: I_c = 123,87 μA, I_a = 105,597 μA, I_{off} = 2,487 μA



da	dc	Iavg (μA)	t_L (ημέρες)
0,1	0,1	23,905	366
0,1	0,3	46,120	190
0,1	0,7	90,548	97
0,1	0,9	112,763	78
0,3	0,1	42,465	206
0,3	0,3	60,555	144
0,3	0,7	96,735	90
0,3	0,9	114,825	76
0,5	0,1	61,025	143
0,5	0,3	74,990	117
0,5	0,7	102,922	85
0,5	0,9	116,887	75
0,9	0,1	98,144	89
0,9	0,3	103,861	84
0,9	0,7	115,295	76
0,9	0,9	121,012	72

Σχήμα 5.38, Πίνακας 5.39: Κατανάλωση σεναρίου 1

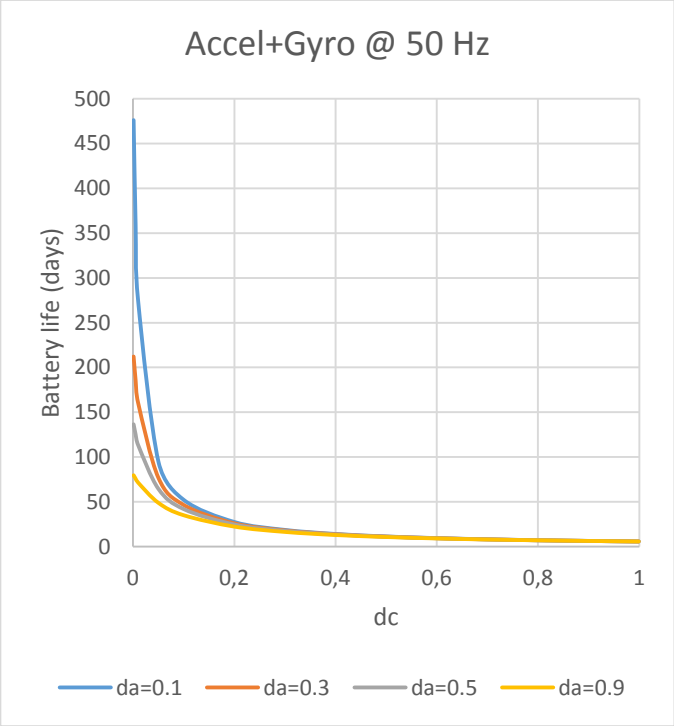
ii. Επιταχυνσιόμετρο, 100 Hz: I_c = 795,53 μA, I_a = 119,804 μA, I_{off} = 5,433 μA



da	dc	Iavg (μA)	t_L (ημέρες)
0,1	0,1	94,736	92
0,1	0,3	250,468	35
0,1	0,7	561,932	16
0,1	0,9	717,664	12
0,3	0,1	115,323	76
0,3	0,3	266,480	33
0,3	0,7	568,794	15
0,3	0,9	719,951	12
0,5	0,1	135,910	64
0,5	0,3	282,492	31
0,5	0,7	575,657	15
0,5	0,9	722,239	12
0,9	0,1	177,083	49
0,9	0,3	314,516	28
0,9	0,7	589,381	15
0,9	0,9	726,814	12

Σχήμα 5.40, Πίνακας 5.41: Κατανάλωση σεναρίου 2

iii. Επιταχυνσιόμετρο/γυροσκόπιο, 50 Hz: I_c = 1523,626 μA, I_a = 119,804 μA, I_{off} = 5,433 μA



da	dc	Iavg (μA)	t_L (ημέρες)
0,1	0,1	167,546	52
0,1	0,3	468,897	19
0,1	0,7	1071,599	8
0,1	0,9	1372,950	6
0,3	0,1	188,132	47
0,3	0,3	484,909	18
0,3	0,7	1078,461	8
0,3	0,9	1375,238	6
0,5	0,1	208,719	42
0,5	0,3	500,921	17
0,5	0,7	1085,324	8
0,5	0,9	1377,525	6
0,9	0,1	249,893	35
0,9	0,3	532,945	16
0,9	0,7	1099,048	8
0,9	0,9	1382,100	6

Σχήμα 5.42, Πίνακας 5.43: Κατανάλωση σεναρίου 3

5.6.2 Διέλευση δεδομένων

Για εφαρμογές με απαιτήσεις ταχείας απόκρισης σε πραγματικό χρόνο, ο ρυθμός δειγματοληψίας που μπορεί να επιτευχθεί πρακτικά περιορίζεται από το μέγιστο ρυθμό διέλευσης δεδομένων πάνω από τη ζεύξη BLE. Ακόμη και αν τα δεδομένα εξόδου διατηρούνται σε ουρές ή buffers μεγάλης χωρητικότητας, ένας ρυθμός παραγωγής δεδομένων σημαντικά μεγαλύτερος από το ρυθμό διέλευσης εξόδου αναπόφευκτα θα οδηγήσει σε εξάντληση του διαθέσιμου χώρου και απώλεια πακέτων.

Αν θεωρήσουμε πως τα δεδομένα εφαρμογής ενσωματώνονται στην ATT PDU ενός πακέτου BLE, ο μέγιστος ρυθμός διέλευσής τους *R* πάνω από μία ζεύξη είναι:

$$R = \frac{min(N_s, N_c)}{t_c} (M - h)$$

όπου

- N_s και N_c είναι το μέγιστο υποστηριζόμενο πλήθος πακέτων ανά connection event για τον server και τον client αντίστοιχα,
- t_c είναι το connection interval της σύνδεσης,
- *M* είναι το μέγιστο μήκος (MTU) της ATT PDU,

- h = 3 bytes είναι το μήκος της επικεφαλίδας ATT.

Τα N_s και N_c γενικά δε μεταβάλλονται για ένα συγκεκριμένο ζεύγος συσκευών, καθώς είναι χαρακτηριστικά της υλοποιούμενης στοίβας BLE σε κάθε πλευρά. Με σταθερό λοιπόν και το t_c, η μεγιστοποίηση του R απαιτεί τη βέλτιστη επιλογή της ATT MTU M.

Η αύξηση της M παύει να επιδρά θετικά από το σημείο στο οποίο η ATT PDU θρυμματίζεται σε πολλά πακέτα του link layer. Εάν μάλιστα το link layer δεν κάνει χρήση της επέκτασης μήκους πακέτου (Bluetooth 4.0/4.1), αυτό συμβαίνει για $M > M_{min}$ = 23 bytes. Έτσι, για λόγους συμβατότητας με το μεγαλύτερο δυνατό πλήθος συσκευών, απαιτούμε επιπλέον $M = min(N_s, N_c)M_{min}$, ώστε κάθε PDU να καταλαμβάνει εγγυημένα το πολύ ένα connection event.

Για την πλειοψηφία των κινητών συσκευών που χρησιμοποιήσαμε ως clients, είναι $min(N_c, N_s) \leq 5$ και $t_c \geq 50$ ms. Με βάση τα παραπάνω, προκύπτουν τελικά τα εξής:

$$M = 115\ B,$$

$$R = \frac{5}{50\ ms}\ 112\ B = 11.2\ kB/s = 89.6\ kbps$$

6 Το λειτουργικό σύστημα Android

6.1 Εισαγωγή

Το Android είναι ένα πλήρες λειτουργικό σύστημα με γραφικό περιβάλλον για φορητές ηλεκτρονικές συσκευές (mobile devices) όπως smartphones, tablets, wearables, smartwatches κλπ.

Αποτελεί στις μέρες μας με διαφορά το πιο δημοφιλές λειτουργικό σύστημα σε ολόκληρο το φάσμα των συσκευών αυτών. Ενδεικτικά, τη στιγμή συγγραφής αυτής της εργασίας, με βάση την παγκόσμια δικτυακή κίνηση, το μερίδιο αγοράς του Android στα κινητά τηλέφωνα υπερβαίνει το 70%, ενώ στα tablets πλησιάζει το 40%, με κύριο ανταγωνιστή και στις δύο περιπτώσεις το iOS. Στο σύνολο της δικτυακής κίνησης, συμπεριλαμβάνοντας δηλαδή τους οικιακούς υπολογιστές και όλες τις υπόλοιπες πλατφόρμες, το μερίδιο του Android πάλι προσεγγίζει το 40%, έχοντας ξεπεράσει αυτό των Windows από το 2019.

Το τελευταίο αυτό στατιστικό στοιχείο αναδεικνύει πόσο δύσκολο είναι πλέον να φανταστεί κανείς το σύγχρονο τρόπο ζωής χωρίς την ευρεία χρήση των mobile devices και τη διείσδυσή τους στην καθημερινότητα. Το Android έχει αποκομίσει τεράστιο όφελος από την εξέλιξη αυτή, στην οποία όμως ταυτόχρονα έχει και καθοριστική συμβολή. Έχει κατορθώσει να ενοποιήσει την πλειοψηφία των φορητών συσκευών, εξασφαλίζοντας τη διαλειτουργικότητα και τη συμβατότητα των εφαρμογών μεταξύ διαφορετικών κατασκευαστών και τύπων συσκευών.

Για smart TVs, media players και άλλες οικιακές συσκευές έχει αναπτυχθεί η ειδική έκδοση Android TV. Παράλληλα, για desktop συστήματα η Google έχει προκρίνει το Chrome OS, το οποίο στις πιο πρόσφατες εκδόσεις παρέχει τη δυνατότητα απόκτησης και εκτέλεσης εφαρμογών Android.

6.1.1 Βασικά χαρακτηριστικά

Το Android αποτελεί ένα λειτουργικό σύστημα βασισμένο σε ελεύθερο και ανοικτού κώδικα λογισμικό (free and open-source software, FOSS), με την έννοια πως συγκεντρώνει τα ακόλουθα χαρακτηριστικά:

- Χρησιμοποιεί τον πυρήνα του Linux, ο οποίος είναι FOSS.
- Οι βασικές συνοδευτικές βοηθητικές εφαρμογές (utilities) είναι FOSS. Η πλειοψηφία των εφαρμογών αυτών προσφέρουν εύκολη και αυτόνομη πρόσβαση στις web/cloud υπηρεσίες της Google.
- Ο πηγαίος κώδικας του Android, αποτελούμενος από τμήματα κώδικα σε Java, C/C++ και άλλες γλώσσες, παρέχεται ως FOSS μέσω της άδειας Apache 2.0.

Παράλληλα όμως, οι βασικές συνοδευτικές εφαρμογές δεν καλύπτουν το σύνολο της λειτουργικότητας μιας κινητής συσκευής. Έτσι, στις συσκευές που κυκλοφορούν στην αγορά ενσωματώνονται από τους κατασκευαστές ιδιοταγείς, κλειστού κώδικα βοηθητικές εφαρμογές με λειτουργίες τηλεφωνίας, μηνυμάτων, αναπαραγωγής πολυμέσων κλπ, σχεδιασμένες για τις συγκεκριμένες συσκευές. Έτσι, ένα προεγκατεστημένο σύστημα Android σε μία κινητή συσκευή δεν απαρτίζεται στο σύνολό του από FOSS.

Επιπρόσθετα στα παραπάνω, ο χρήστης του συστήματος έχει τη δυνατότητα να εγκαθιστά νέες εφαρμογές της επιλογής του, οι οποίες μπορεί να είναι ανοιχτού ή κλειστού κώδικα, να παρέχονται με ή χωρίς κόστος. Η Google ενσωματώνει σε όλες τις εγκαταστάσεις Android το Google Play Store, το οποίο αποτελεί και την κεντρική πλατφόρμα διάθεσης πιστοποιημένων εφαρμογών. Εδώ πρέπει να σημειωθεί πως επίσημες (και όχι παράγωγα) θεωρούνται μόνον οι διανομές του Android που έχουν ελεγχθεί από την Google και περιλαμβάνουν τις βασικές συνοδευτικές εφαρμογές της.

Έχοντας σχεδιαστεί αρχικά για εκτέλεση σε smartphones και tablets με επεξεργαστές ARM, το Android έχει επιδείξει αξιοσημείωτη ευελιξία, επεκτεινόμενο τόσο σε άλλους τύπους συσκευών αυτής της αρχιτεκτονικής (smart TVs), όσο και στις αρχιτεκτονικές x86/64. Το τελευταίο σημαίνει πως είναι δυνατή η εκτέλεση του Android και σε IBM-compatible οικιακούς υπολογιστές, δίχως όμως επίσημη υποστήριξη.

6.1.2 Ιστορικά στοιχεία

Από το 2007, το Android αναπτύσσεται από την *Open Handset Alliance*, μία κοινοπραξία που ιδρύθηκε από την Google και μια σειρά τηλεπικοινωνιακών παρόχων, εταιριών πληροφορικής και κατασκευαστών φορητών συσκευών με σκοπό την ανάπτυξη ενός κοινού, ανεξάρτητου από τον κατασκευαστή λειτουργικού συστήματος ανοιχτού κώδικα για κινητά τηλέφωνα.

Σχεδόν ταυτόχρονα με το Android εμφανίστηκε το iOS, που αποτελεί την πλατφόρμα της Apple για τη δική της σειρά κινητών τηλεφώνων (iPhone), εμπνευσμένο από το MacOS/OS X. Και τα δύο εκτόπισαν από την αγορά ένα σύνολο από λειτουργικά συστήματα που εκτελούνταν τόσο σε smartphones, όσο και στα τελευταία κινητά δεύτερης γενιάς. Από αυτά τα σπουδαιότερα υπήρξαν τα Symbian της Nokia, Windows Mobile/Windows Phone της Microsoft και Blackberry OS, τα οποία στερούνταν υποστήριξης από ικανό αριθμό κατασκευαστών και τελικά έπαψαν να αναπτύσσονται.

Οι πρώτες εκδόσεις του Android (μέχρι και την 9.0-"*Pie*") ήταν γνωστές με κωδικές ονομασίες επιπλέον των αριθμών έκδοσης. Τη στιγμή που γράφονται αυτές οι γραμμές, η πιο πρόσφατη έκδοση είναι η 11.0, και συνολικά η πλατφόρμα του Android είναι εγκατεστημένη σε περισσότερες από 2.5 δισεκατομμύρια συσκευές, με το σύνολο σχεδόν από αυτές (99.8%) να χρησιμοποιεί εκδόσεις νεότερες της 4.0

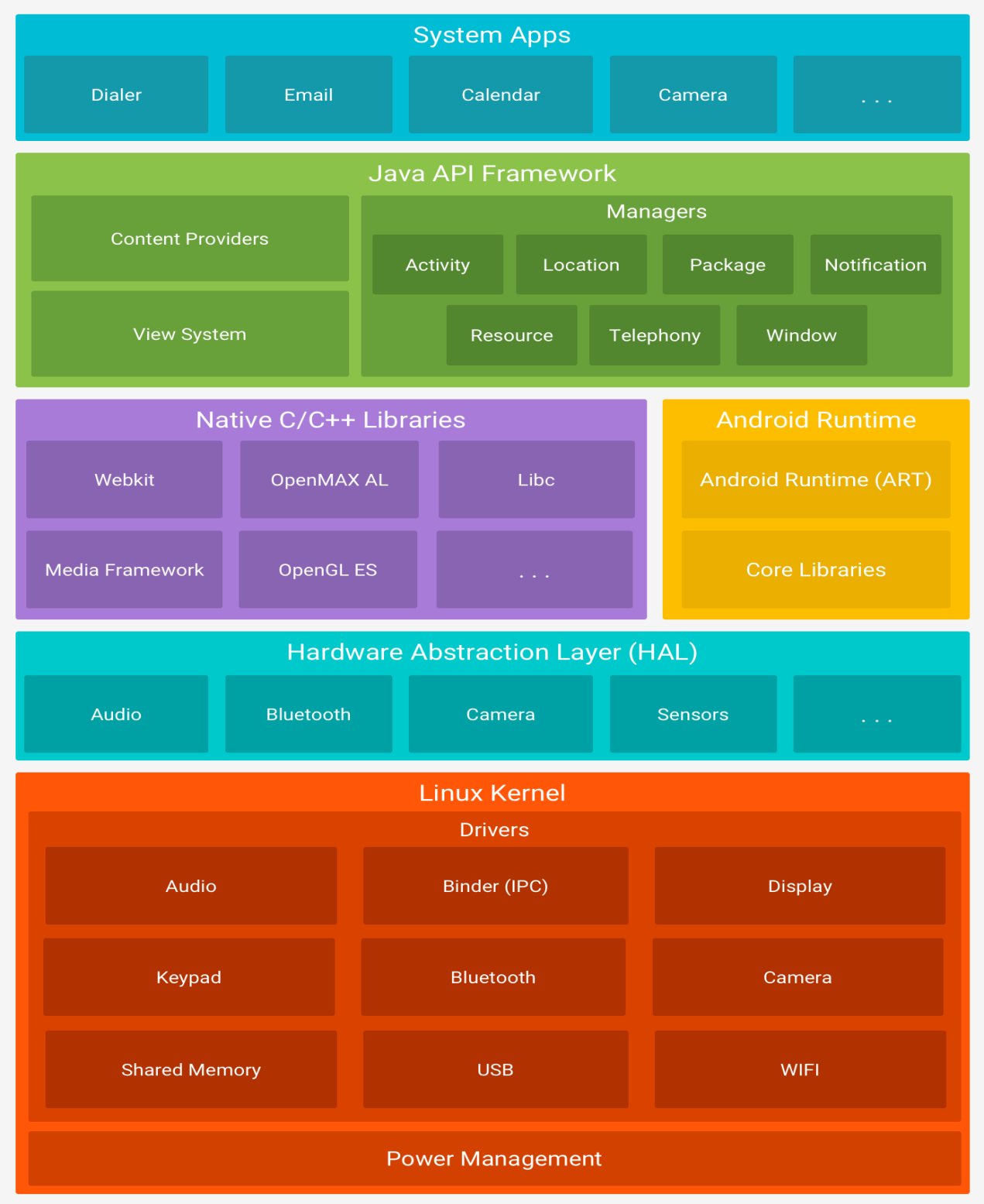
("*Ice Cream Sandwich*"). Παρακάτω σημειώνουμε κάποια ορόσημα στην πορεία του Android και των ανταγωνιστών του:

2003 - Οι Andy Rubin, Rich Miner, Nick Sears και Chris White ιδρύουν την startup Android Inc. για την ανάπτυξη του ομώνυμου project, το οποίο αρχικά σχεδιάστηκε ως ένα έξυπνο λειτουργικό σύστημα για ψηφιακές φωτογραφικές μηχανές. Σύντομα το project προσανατολίζεται στα κινητά τηλέφωνα, αποσκοπώντας στην ανάπτυξη ενός λειτουργικού συστήματος ανοιχτού κώδικα, βασισμένου στον πυρήνα του Linux και ικανού να ανταγωνιστεί τα κυρίαρχα τότε Symbian και Windows Mobile.
2005 - Τα πρώτα απτά αποτελέσματα ελκύουν την προσοχή της Google, η οποία εξαγοράζει την Android Inc. και εντάσσει στο δυναμικό της τους ιδρυτές και την προγραμματιστική ομάδα.
2006 - Σε δικαστική διαμάχη μεταξύ της Google και της Oracle έρχεται στο φως η πρώτη δοκιμαστική συσκευή Android, η οποία είναι όμοια στην εμφάνιση με ένα μοντέλο BlackBerry.
2007 - Η Apple κυκλοφορεί το πρώτο iPhone, με λειτουργικό σύστημα το iOS. Σε αντίθεση με τα μέχρι τότε δημοφιλή μοντέλα smartphones, το iPhone δε διαθέτει πληκτρολόγιο, αλλά παραχωρεί το σύνολο σχεδόν της επιφάνειας του κινητού στην οθόνη αφής. Οι υπόλοιποι κατασκευαστές σύντομα ακολουθούν αυτή τη σχεδιαστική φιλοσοφία.

Ταυτόχρονα, με πρωτοβουλία της Google ιδρύεται η Open Handset Alliance. Η υποστήριξη από πληθώρα κατασκευαστών δίνει σημαντική ώθηση στην ανάπτυξη του Android.
2008 - Κυκλοφορεί η πρώτη επίσημα υποστηριζόμενη (stable) έκδοση του Android (1.0), ταυτόχρονα με την πρώτη συσκευή Android, το κινητό HTC Dream.
2010 - Η έκδοση 2.3 του Android ("Gingerbread") υποστηρίζει προσαρμογείς NFC. Ενώ το iOS ανταγωνίζεται το Symbian για την πρωτοκαθεδρία στην αγορά των smartphones, το Android για πρώτη φορά επιτυγχάνει διψήφιο μερίδιο αγοράς και ξεπερνάει σε δημοτικότητα το mobile λειτουργικό σύστημα της Sony Ericsson.
2011 - Το Android εκτοπίζει το Blackberry OS από την τρίτη θέση στην αγορά.
2012 - Κυκλοφορεί η τελευταία έκδοση του Symbian. Με την πάροδο λίγων μηνών το Android ξεπερνά σε δημοτικότητα τόσο το Symbian όσο και το iOS, κατακτώντας την πρώτη θέση στην αγορά.
2013 - Η έκδοση 4.3 του Android ("*Jelly Bean*") υποστηρίζει το Bluetooth Low Energy.
2014 - Η έκδοση 5.0 ("*Lollipop*") του Android εισάγει ένα βελτιωμένο περιβάλλον εκτέλεσης (ART), καθώς και σημαντικές αλλαγές στη σχεδιαστική φιλοσοφία του γραφικού περιβάλλοντος (Material Design).
2015 - Κυκλοφορεί η τελευταία έκδοση των Windows Phone.
2019 - Κυκλοφορεί η τελευταία έκδοση του Blackberry OS. Μοναδικός ουσιαστικά ανταγωνιστής του Android στα κινητά τηλέφωνα είναι πλέον το iOS.

6.1.3 Αρχιτεκτονική

Η αρχιτεκτονική της στοίβας λογισμικού του Android φαίνεται στο παρακάτω σχήμα:



Σχήμα 6.1: Η στοίβα λογισμικού του Android

Παρουσιάζουμε κάθε επίπεδο της στοίβας ξεχωριστά:

- **Πυρήνας (kernel):** Θεμέλιο του Android αποτελεί ο πυρήνας του Linux, με κάποιες όμως τροποποιήσεις. Ο Linux kernel είναι *μονολιθικός*, που σημαίνει ότι σύνολο των λειτουργιών του συστήματος, συμπεριλαμβανομένων των drivers, εκτελείται στο χώρο πυρήνα (kernel space), όπως φαίνεται στο σχήμα. Οι drivers με την εγκατάσταση ή την απεγκατάστασή τους, αντίστοιχα προστίθενται απευθείας ως δομικές μονάδες (modules) στον πυρήνα ή αφαιρούνται από αυτόν.
- **Επίπεδο αφαίρεσης υλικού (Hardware Abstraction Layer, HAL):** Αποτελεί τη διεπαφή μεταξύ του περιβάλλοντος εκτέλεσης και των κλήσεων συστήματος/προγραμμάτων οδήγησης. Αποτελείται από κατάλληλες διαπροσωπείες (interfaces) Java που επιτρέπουν τη χρήση περιφερειακών συσκευών από τον πηγαίο κώδικα χρήστη.
- **Περιβάλλον εκτέλεσης (Runtime Environment):** Στο Android, περιβάλλον εκτέλεσης δε σημαίνει μόνο τις runtime libraries, την εικονική μνήμη και τους πόρους που παρέχονται στην εφαρμογή/διεργασία από το λειτουργικό σύστημα, ακριβώς γιατί η εφαρμογή δε βρίσκεται άμεσα διαθέσιμη σε κώδικα μηχανής, αλλά σε bytecode. Οι εφαρμογές του Android εκτελούνται λοιπόν από εικονική μηχανή, καθεμία στο δικό της αντίγραφο. Από το Android 5.0 και μετά, το Android Runtime (ART) αποτελεί το περιβάλλον εκτέλεσης, διαδεχόμενο το Dalvik. Στο περιβάλλον εκτέλεσης του Android αναλυτικά θα αναφερθούμε στην επόμενη παράγραφο.
- **Χαμηλού επιπέδου βιβλιοθήκες C/C++:** Τμήματα πηγαίου κώδικα που απαιτούν εξειδικευμένες ως προς τη συσκευή βελτιστοποιήσεις ή χαμηλού επιπέδου πρόσβαση στο υλικό, μπορούν να γραφούν σε C/C++. Για το σκοπό παρέχεται μια ειδική προγραμματιστική πλατφόρμα αντίστοιχη του Android SDK για Java, το Android NDK (Native Development Kit).
- **Προγραμματιστική πλατφόρμα (framework):** Όλες οι λειτουργίες συστήματος του Android είναι διαθέσιμες στον προγραμματιστή εφαρμογών μέσω ενός συνόλου από Java APIs (Java API Framework), το οποίο περιέχεται στο Android SDK (Software Development Kit). Αυτό καθιστά την ανάπτυξη εφαρμογών για Android εξαιρετικά απλή, καθώς για την παραγωγή κώδικα αρκεί το Java Development Kit (JDK) και το Android SDK.
- **Εφαρμογές συστήματος/προεγκατεστημένες εφαρμογές:** Κάθε εγκατάσταση Android ενσωματώνει βασικές βοηθητικές εφαρμογές (utilities), όπως email client, φυλλομετρητή (browser), διαχείριση επαφών κλπ.

6.1.4 Περιβάλλον εκτέλεσης

Το περιβάλλον εκτέλεσης του Android ισοδυναμεί ουσιαστικά με μια εικονική μηχανή, αποστολή της οποίας είναι η εξομοίωση ενός συστήματος με κώδικα μηχανής τον bytecode της εφαρμογής, μαζί με τις απαραίτητες binary βιβλιοθήκες. Στις βιβλιοθήκες αυτές συγκαταλέγεται φυσικά η υλοποίηση του API της γλώσσας προγραμματισμού Java, αφού η εφαρμογή δεν εκτελείται σε JVM.

Παρακάτω αναφερόμαστε σύντομα στις βασικές λειτουργίες του περιβάλλοντος εκτέλεσης και τις βελτιώσεις που παρέχει το ART για καθεμιά από αυτές, σε σύγκριση πάντα με το παλαιότερο Dalvik:

- ***Παραγωγή κώδικα μηχανής***

Το ART δέχεται ως είσοδο την ίδια μορφή bytecode (DEX - Dalvik Executable Format) με το Dalvik, γενικά όμως δεν υπάρχει προς τα πίσω συμβατότητα μεταξύ τους.

Ενώ το Dalvik ακολουθεί αποκλειστικά τη λογική ενός διερμηνέα, δηλαδή μεταγλωττίζει κάθε τμήμα bytecode σε κώδικα μηχανής τη στιγμή που το συγκεκριμένο τμήμα εκτελείται και μόνο (just-in-time compilation, JIT), το ART παρέχει τη δυνατότητα εκ των προτέρων μεταγλώττισης ολόκληρου του bytecode κατά την εκκίνηση της εφαρμογής (ahead-of-time compliation, AOT), ακολουθώντας τη λογική ενός μεταγλωττιστή. Συνήθως εκτιμάται η πιθανή επιτάχυνση ή επιβάρυνση του συστήματος, και ακολουθείται μία από τις δύο προσεγγίσεις, ή συνδυασμός τους, όπως φαίνεται στο σχήμα 6.2.

Εξάλλου από την έκδοση 9.0 του Android και μετά, κατά την εγκατατάσταση μιας εφαρμογής δίνεται η επιλογή απευθείας μετατροπής μέρους των DEX αρχείων σε συμπαγέστερα αρχεία κώδικα μηχανής (ELF), μειώνοντας τις απαιτήσεις τόσο σε αποθηκευτικό χώρο όσο και σε μνήμη RAM. Η μετατροπή γίνεται με χρήση του εργαλείου *dex2oat*.

- ***Διαχείριση μνήμης***

Κατά το χρόνο εκτέλεσης, ο μηχανισμός συλλογής απορριμμάτων (garbage collection, GC) ελέγχει περιοδικά για αχρησιμοποίητα ή περιττά αντικείμενα και τα καταστρέφει, αποδεσμεύοντας αντίστοιχα τη μνήμη που καταλαμβάνουν.

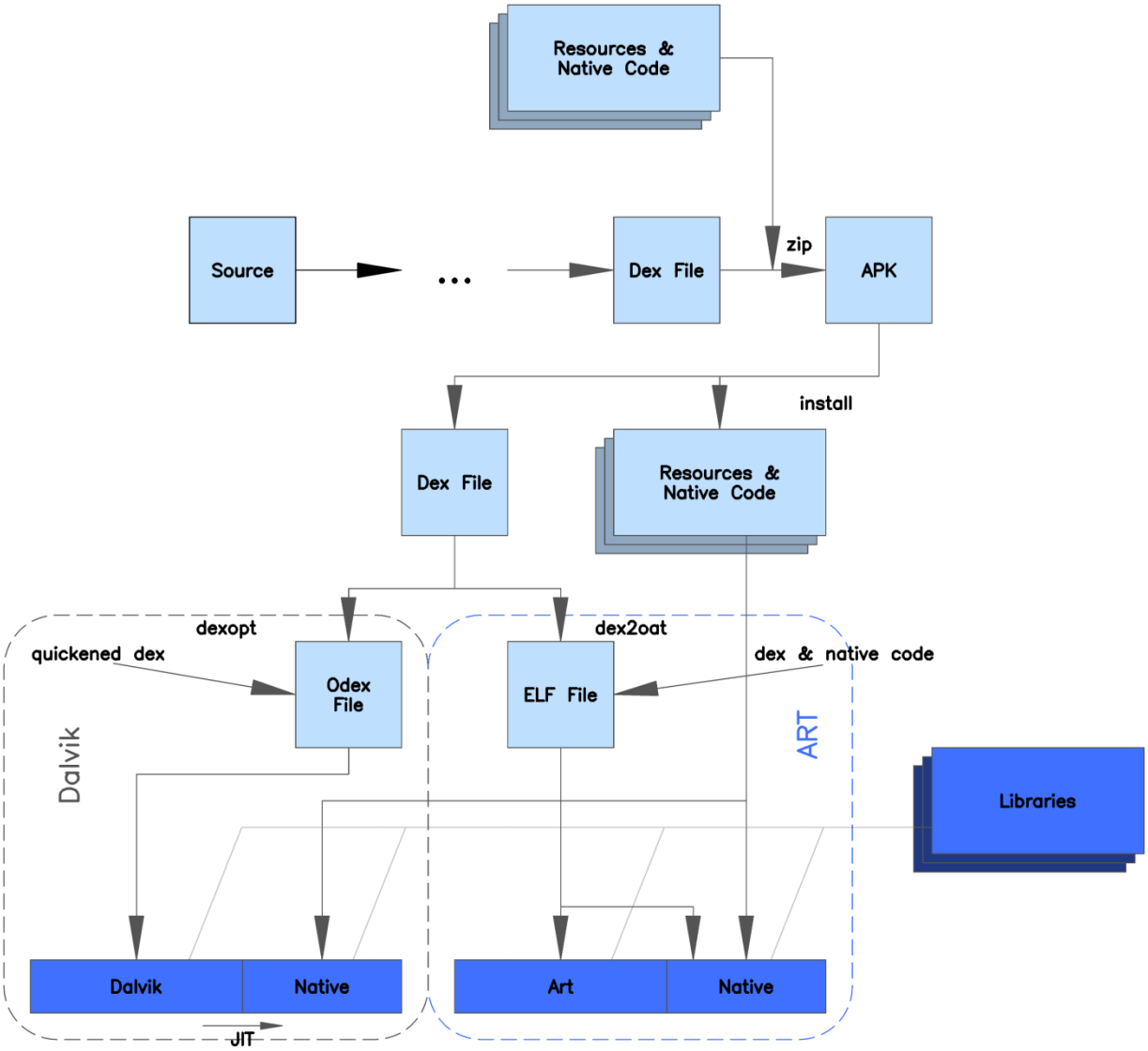
Η διαδικασία αυτή οφείλει να είναι όσο το δυνατόν πιο ακριβής, ώστε να ελαχιστοποιείται το αποτύπωμα μνήμης της εφαρμογής και να εξοικονομούνται πόροι, αλλά και σύντομη, ώστε η εφαρμογή και το γραφικό περιβάλλον να επιτυγχάνουν καλούς χρόνους απόκρισης.

Το ART υπερβαίνει αυτή τη σχέση ανταλλαγής πραγματοποιώντας κάθε φορά μία παύση GC αντί για δύο, και παραλληλοποιώντας τη διαδικασία. Έτσι, το κύριο νήμα απαλλάσσεται από κάθε αναμονή που οφείλεται σε GC, συνεχίζοντας κανονικά την εκτέλεσή του. Ταυτόχρονα η διαδικασία GC επιταχύνεται θεαματικά, διατηρώντας την απαραίτητη ακρίβεια.

- ***Αποσφαλμάτωση***

Το ART ενισχύει τις δυνατότητες του debugger για εκτύπωση λεπτομερών διαγνωστικών μηνυμάτων (logging), αποδοτική εποπτεία της εκτέλεσης (monitoring), και εκτίμηση κατανάλωσης πόρων (profiling). Εκτός από τα ορισμένα από τον προγραμματιστή σημεία παύσης εκτέλεσης (breakpoints), είναι δυνατή η εισαγωγή περισσότερων ειδικών σημείων ελέγχου, όπως για την καταγραφή της

τιμής επιστροφής μιας μεθόδου (method-exit) ή την παρακολούθηση της μεταβολής ενός πεδίου κλάσης.



Σχήμα 6.2: Παραγωγή εκτελέσιμου κώδικα στο ART και στο Dalvik

6.1.5 Περιβάλλον προγραμματισμού

Η ανάπτυξη εφαρμογών Android μπορεί να γίνει στις γλώσσες Java ή Kotlin, αρκεί ο προγραμματιστής να έχει στη διάθεσή του το παρεχόμενο από την Google Android SDK, το JDK, ένα συμβατό περιβάλλον ανάπτυξης (πχ Eclipse) και ένα σύστημα αυτοματοποίησης (build automation system) για την παραγωγή του εκτελέσιμου κώδικα (πχ Maven, Gradle). Το Android Studio IDE, βασισμένο στο IntelliJ IDEA, υποστηρίζεται επίσημα από την Google και ενσωματώνει το Android SDK και το Gradle σε μία ολοκληρωμένη και αρκετά εύχρηστη λύση.

Το Android SDK περιλαμβάνει την υλοποίηση των απαραίτητων *βιβλιοθηκών συστήματος (framework libraries)* για μία ή περισσότερες εκδόσεις της πλατφόρμας του Android, καθώς και τα APIs των βιβλιοθηκών αυτών για Java ή

Kotlin. Ενσωματώνει επιπλέον βοηθητικά binaries (build tools, platform tools) για την παραγωγή του τελικού κώδικα και την αποσφαλμάτωσή του (με εγκατάσταση και εκτέλεση σε εξομοιωτή ή πραγματική συσκευή), με σπουδαιότερο αυτών το εργαλείο *adb*. Σε πρώτη φάση παράγεται Java bytecode από τον πηγαίο κώδικα, και από την μεταγλώττιση του bytecode σε δεύτερη φάση προκύπτει ο τελικός εκτελέσιμος κώδικας για ART/Dalvik.

Κάθε στοιχείο λογισμικού (βιβλιοθήκη, κλάση, πεδίο ή μέθοδος κλάσης) του SDK προσδιορίζει στη δήλωσή του το δικό του ελάχιστο συμβατό *επίπεδο API (API level)*, το οποίο αντιστοιχεί μονοσήμαντα στην ελάχιστη έκδοση της πλατφόρμας του Android που απαιτείται για τη λειτουργία του. Για παράδειγμα, το API level 30 είναι διαθέσιμο στο Android 11. Το ελάχιστο απαιτούμενο API level για την ανάπτυξη και την εκτέλεση μίας εφαρμογής προκύπτει από τις απαιτήσεις των στοιχείων που ενσωματώνει. Ορισμένες βελτιώσεις και στοιχεία που έχουν προστεθεί σε νεότερες εκδόσεις των framework libraries παρέχονται επιπλέον σε εκδόσεις συμβατές με παλαιότερα API levels (backporting) μέσω των *βιβλιοθηκών υποστήριξης (support libraries)* και της βιβλιοθήκης *AndroidX*.

6.2 Ανάπτυξη εφαρμογών για Android

Στο σημείο αυτό είναι χρήσιμο να απαριθμήσουμε τα σημαντικότερα APIs που παρέχονται από το Android framework στις εφαρμογές για την πραγματοποίηση εργασιών με χρήση των βασικών μηχανισμών του λειτουργικού συστήματος.

6.2.1 Θεμελιώδη δομικά στοιχεία

Η επικοινωνία του περιβάλλοντος εκτέλεσης του Android με τον κώδικα της εφαρμογής γίνεται μέσα από κάποιες ειδικά ορισμένες κλάσεις για το σκοπό αυτό, οι οποίες είναι ορατές στον προγραμματιστή μέσω του Android SDK. Τα στιγμιότυπά τους αποτελούν τα θεμελιώδη δομικά *στοιχεία* της εφαρμογής *(app components)*.

Καθεμία από αυτές τις κλάσεις, τις οποίες παρουσιάζουμε παρακάτω, λειτουργεί ως *σημείο εισόδου (entry point)* στην εφαρμογή κατόπιν μιας συγκεκριμένης ενέργειας. Μία τέτοια ενέργεια μπορεί να έχει αναληφθεί από το σύστημα κατόπιν ενός συμβάντος (αλλαγή σύνδεσης δικτύου ή πηγής τροφοδοσίας), ή να οφείλεται σε αλληλεπίδραση του χρήστη με το σύστημα (πχ επιλογή του εικονιδίου της εφαρμογής, άνοιγμα αρχείου ή συνδέσμου συσχετισμένου με την εφαρμογή).

- ***Δραστηριότητα γραφικού περιβάλλοντος (Activity)***

Το γραφικό περιβάλλον μίας εφαρμογής Android γενικά αποτελείται από μία ή περισσότερες οθόνες. Μια δραστηριότητα (activity) αντιπροσωπεύει τον κύκλο ζωής (lifecycle) μίας οθόνης της εφαρμογής. Την λειτουργία των activities και γενικότερα το χειρισμό του γραφικού περιβάλλοντος πραγματευόμαστε στην υποενότητα 6.2.2.

- ***Υπηρεσία (Service)***

Αντιπροσωπεύει μία λειτουργία που εκτελείται στο παρασκήνιο, ανεξάρτητα από το αν το γραφικό περιβάλλον της εφαρμογής είναι ενεργό ή όχι. Τη λειτουργία των services πραγματευόμαστε στην υποενότητα 6.2.3.

- ***Αποδέκτης μηνυμάτων εκπομπής (Broadcast Receiver)***

Παραλαμβάνει για λογαριασμό της εφαρμογής μηνύματα εκπομπής (broadcasts) που αναφέρονται σε συγκεκριμένα συμβάντα και προέρχονται από το λειτουργικό σύστημα, από άλλες ενεργές διεργασίες, ή και από άλλα app compontents της ίδιας εφαρμογής. Τη λειτουργία των broadcast receivers και γενικά τη διαδιεργασιακή επικοινωνία πραγματευόμαστε στην υποενότητα 6.2.5.

- ***Πάροχος περιεχομένου (content provider)***

Καθιστά τα μόνιμα αποθηκευμένα δεδομένα της εφαρμογής (αρχεία, βάσεις δεδομένων κλπ) ορατά στο υπόλοιπο σύστημα. Την αποθήκευση δεδομένων πραγματευόμαστε στην υποενότητα 6.2.6.

Για την ενσωμάτωση ενός app component στην τελική εφαρμογή, απαιτείται κατά τη διαδικασία ανάπτυξης η κλάση υλοποίησής του να καταχωρείται σε ένα ειδικό αρχείο XML, το *αρχείο δηλώσεων* (*manifest file*). Κάθε καταχώρηση στο αρχείο αυτό ισοδυναμεί με ένα στοιχείο (element) XML, με κατάλληλες ιδιότητες (attributes) οριζόμενες από το Android SDK.

Κατά το χρόνο εκτέλεσης, μία εφαρμογή Android ενδέχεται να είναι πιο αποδοτικό να φορτώνει δυναμικά κάποια σταθερά αντικείμενα που είναι αναγκαία για την πραγματοποίηση συγκεκριμένων λειτουργιών (συμβολοσειρές, σχήματα, γραφικά στοιχεία, εικόνες), αντί να τα κατασκευάζει προγραμματιστικά ή να χρησιμοποιεί απευθείας το API για Ε/Ε από αρχεία. Αυτό διευκολύνεται σημαντικά ορίζοντας - πριν τη μεταγλώττιση - τα αντικείμενα αυτά με τη μορφή *πόρων* (*resources*) σε κατάλληλους υποκαταλόγους του καταλόγου `source_path/res`, όπου `source_path` είναι η διαδρομή προς τον πηγαίο κώδικα της εφαρμογής. Οι πιο σημαντικοί τύποι πόρων περιλαμβάνουν:

- *Λίστες τιμών σε XML*: αποθηκεύονται στον κατάλογο *res*/*values* (για παράδειγμα, οι συμβολοσειρές βρίσκονται στο αρχείο *strings.xml*).
- *Ορισμοί γραφικών αντικειμένων σε XML*: αποθηκεύονται στον κατάλογο *res*/*layout*.
- *Εικόνες (bitmap ή vector)*: αποθηκεύονται στον κατάλογο *res*/*drawable*.

Ο κώδικας της εφαρμογής μπορεί να αναφέρεται σε τέτοια αρχεία δημιουργώντας και χρησιμοποιώντας *αναγνωριστικά πόρων* (*Resource IDs*) της μορφής `R.<res_type>.<file>`, όπου `res_type` είναι ο τύπος του πόρου, `file` το όνομα του αρχείου, και `R` είναι κλάση Java που παράγεται από τη μεταγλώττιση της εφαρμογής και περιέχει όλα τα Resource IDs. Για παράδειγμα, αν στους πόρους της εφαρμογής περιλαμβάνεται η εικόνα *myimg.png*, κατά τη μεταγλώττιση η εφαρμογή θα παραγάγει για αυτήν το αναγνωριστικό `R.drawable.myimg`.

Για πόρους που βρίσκονται εντός του καταλόγου `res/values`, η αναφορά γίνεται απευθείας στο αρχείο, και στη συνέχεια στο στοιχείο XML που περιέχεται σε αυτό: `R.<file>.<element>`. Για παράδειγμα, για τη φόρτωση της συμβολοσειράς με κλειδί "*mystr*" από το *strings.xml* δίνεται το Resource ID `R.string.mystr`.

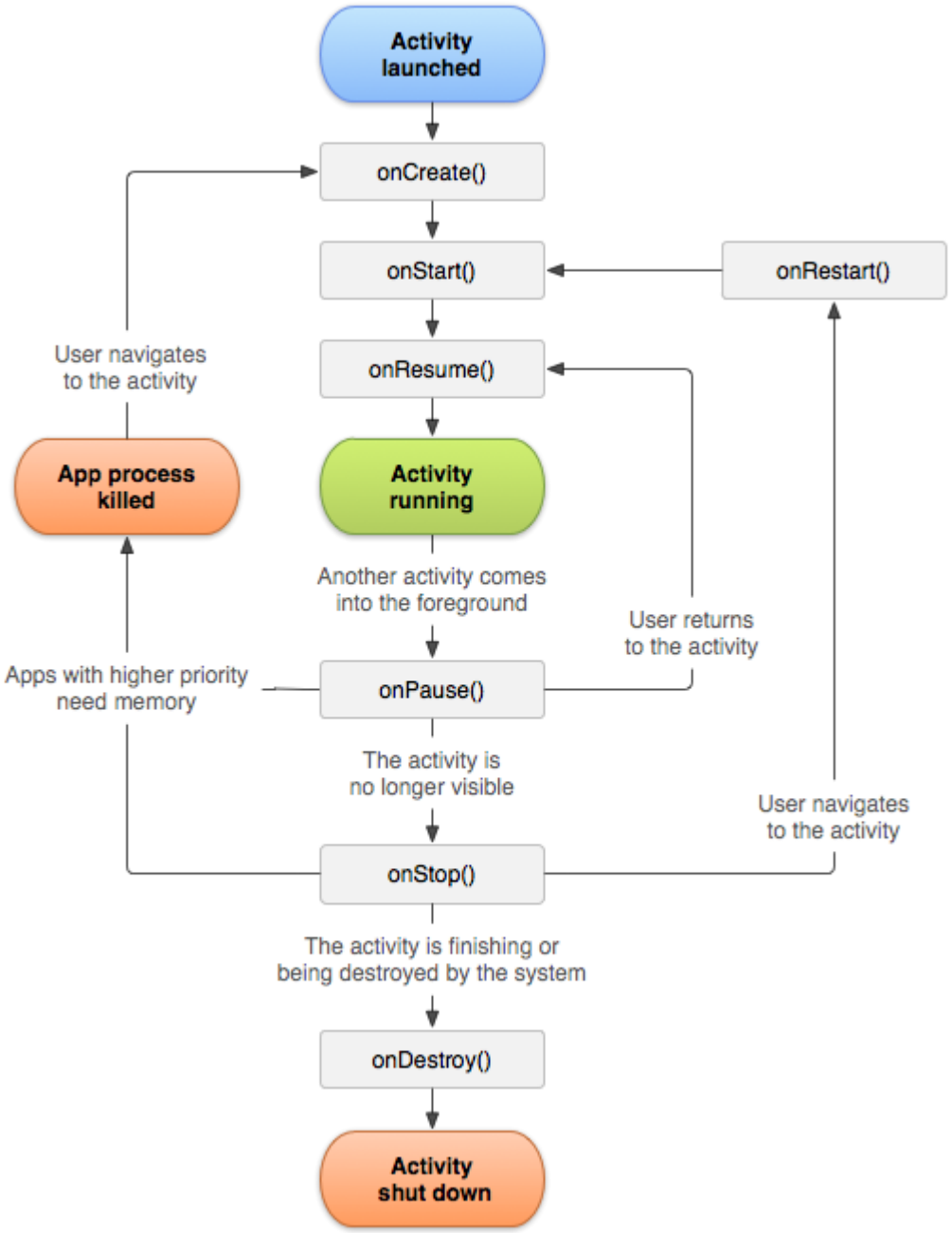
Σε κάθε περίπτωση, για αντικείμενα που ορίζονται σε αρχεία XML είναι δυνατή η αναφορά στα περιεχόμενα στοιχεία τους (*elements*), εφόσον τα τελευταία περιλαμβάνουν κάποια ιδιότητα (*attribute*) με ισχύ αναγνωριστικού (συνήθως `android:id`). Το αναγνωριστικό κάθε τέτοιου στοιχείου έχει τοπική εμβέλεια εντός του αντικειμένου στο οποίο ανήκει.

6.2.2 Διαχείριση γραφικού περιβάλλοντος

Κατά τη χρήση μίας εφαρμογής Android, ο χρήστης περιηγείται μέσα σε μία ακολουθία από «παράθυρα» που καλύπτουν συνήθως το σύνολο του χώρου της οθόνης της συσκευής. Στο εξής θα αναφερόμαστε στα παράθυρα αυτά ως τις *οθόνες (screens)* της εφαρμογής.

6.2.2.1 Activities

Κάθε οθόνη της εφαρμογής αντιστοιχεί σε ένα στιγμιότυπο της κλάσης *Activity*. Η κλάση *Activity* παρέχει μία σειρά από callbacks που αντιστοιχούν σε μεταβάσεις σε συγκεκριμένες καταστάσεις του *κύκλου ζωής* μίας οθόνης, επιτρέποντας στον προγραμματιστή τον χειρισμό των αντίστοιχων συμβάντων και τη σύνδεση της UI engine με τον κώδικα χρήστη. Παρακάτω φαίνεται ο κύκλος ζωής (lifecycle) μίας οθόνης:



Σχήμα 6.3: Κύκλος ζωής οθόνης

Μία activity μπορεί να βρίσκεται ανά πάσα χρονική στιγμή σε μία από τις ακόλουθες καταστάσεις:

- **Created:** Πρόκειται για την αρχή του κύκλου ζωής της οθόνης. Το σύστημα έχει δεσμεύσει τους απαραίτητους πόρους για τη δημιουργία της οθόνης. Εδώ είναι συνήθως (ολοκλήρωση του `onCreate()` callback) το σημείο όπου ορίζεται το layout και αρχικοποιούνται αναφορές σε αντικείμενα γραφικού περιβάλλοντος (views), καθώς και σε άλλα app components που το activity ενδεχομένως χρειάζεται.

- **Started:** Η οθόνη έχει εκκινηθεί και είναι ορατή στο χρήστη, δε βρίσκεται όμως απαραίτητα στο προσκήνιο.

- **Resumed:** Με την ολοκλήρωση του `onResume()` callback, η οθόνη έρχεται στο προσκήνιο.

- **Paused:** Με την ολοκλήρωση του `onPause()` callback, η οθόνη περνά στο παρασκήνιο, παραμένει όμως ορατή.

- **Stopped:** Με την ολοκλήρωση του `onStop()` callback, η οθόνη χάνεται από το οπτικό πεδίο του χρήστη και αναστέλλει τη λειτουργία της. Αν υπάρχει επείγουσα ανάγκη εξοικονόμησης μνήμης από το σύστημα, η διεργασία της εφαρμογής μπορεί να τερματιστεί εκείνη τη στιγμή. Σε αυτή την περίπτωση, η κατάσταση της οθόνης αποθηκεύεται προσωρινά, ώστε να είναι δυνατή η ανακατασκευή της μόλις ο χρήστης επανέλθει στην εφαρμογή.

- **Destroyed:** Με την ολοκλήρωση του `onDestroy()` callback, η οθόνη φτάνει στο τέλος του κύκλου ζωής της και αποδεσμεύονται όλοι οι πόροι που αυτή κατέχει. Αυτό συμβαίνει όταν ο χρήστης με επιλογή του κλείνει τη συγκεκριμένη οθόνη, ή συνολικά την εφαρμογή.

Η εξειδίκευση της συμπεριφοράς κάθε ξεχωριστής οθόνης που σχεδιάζουμε επιτυγχάνεται με την επέκταση της κλάσης `android.app.Activity`.

6.2.2.2 Στοιχεία γραφικού περιβάλλοντος

Τα αντικείμενα που εμφανίζονται στο γραφικό περιβάλλον και με τα οποία αλληλεπιδρά ο χρήστης της εφαρμογής αναφέρονται ως *όψεις (views)*.

Τα views μπορούν να είναι *απλά αντικείμενα (widgets)*, όπως κουμπιά, λίστες, ενδείκτες, πεδία πολλαπλών επιλογών, εμφάνισης/εισαγωγής κειμένου, εικόνες, σχήματα κλπ, ή *περιέκτες αντικειμένων (containers)*, οι οποίοι τοποθετούν τα θυγατρικά τους αντικείμενα στο χώρο που τους έχει διατεθεί σύμφωνα με μία ορισμένη από τον σχεδιαστή *διάταξη (layout)*.

Η τοποθέτηση των views στο χώρο γίνεται ιεραρχικά, συνεπώς κάθε διάταξη ισοδυναμεί με μία δενδρική δομή. Έτσι είναι δυνατός - και συνήθως πιο συμφέρων - ο ορισμός του layout για κάθε view σε αντίστοιχο αρχείο XML, με elements τα views και attributes ιδιότητες των views όπως διαστάσεις, χρώμα ταπετσαρίας, μέγεθος κειμένου, περιθώρια κλπ.

Βέβαια, είναι δυνατός ο ορισμός όλων των παραπάνω προγραμματιστικά, κάτι είναι που είναι χρήσιμο κυρίως σε περιπτώσεις όπου η διατάξη των αντικειμένων ή οι ιδιότητές τους αλλάζουν δυναμικά κατά το χρόνο εκτέλεσης.

6.2.2.3 Fragments

Στο εσωτερικό μίας activity, είναι δυνατό εκτός από κοινά views να ενσωματώσουμε αυτόνομες υπομονάδες με δικό τους κύκλο ζωής και δυνατότητα αλληλεπίδρασης με το χρήστη, δηλαδή με λειτουργία παρόμοια με αυτήν μίας activity. Αυτές οι υπομονάδες συνιστούν τα *τμήματα γραφικού περιβάλλοντος (fragments).* Η χρήση fragments ευνοεί την επαναχρησιμοποίηση στοιχείων γραφικού περιβάλλοντος που απαιτείται να έχουν την ίδια εμφάνιση και συμπεριφορά σε περισσότερα από ένα activities.

Ο κύκλος ζωής ενός fragment - τον οποίο θα αναλύσουμε αμέσως παρακάτω - ξεκινά από την επισύναψη του fragment και τη δέσμευση του περιεχομένου του σε κάποιον container (πχ ένα *FrameLayout*), μέχρι την απόσπασή του από το activity, είτε λόγω ενέργειας του χρήστη ή κάποιου συμβάντος, είτε λόγω καταστροφής της activity που το φιλοξενεί. Από την έκδοση 4.2 του Android (επίπεδο API 17) και μετά, είναι μάλιστα δυνατό σε ένα fragment να ενσωματώνονται άλλα θυγατρικά fragments, ορίζοντας βαθύτερες και πιο περίπλοκες ιεραρχίες οθονών-υποοθονών.

Οι δυνατές καταστάσεις ενός fragment κατά τον κύκλο ζωής του είναι οι εξής:

- **Attached:** Κατασκευάζεται το Java αντικείμενο που αντιστοιχεί στο νέο fragment, και το fragment προστίθεται στην τρέχουσα activity.

- **Created:** Δεσμεύονται οι απαραίτητοι πόροι για την ενσωμάτωση του fragment στην ιεραρχία του γραφικού περιβάλλοντος της activity.

- **View created:** Κατασκευάζεται το γραφικό περιεχόμενο του fragment και είναι έτοιμο προς εμφάνιση.

- **Started:** Το fragment παραμένει σε αυτή την κατάσταση ενόσω η activity που το φιλοξενεί βρίσκεται επίσης στην κατάσταση *Started*, δηλαδή είναι ορατή αλλά όχι στο προσκήνιο.

- **Resumed:** Η activity που φιλοξενεί το fragment βρίσκεται στο προσκήνιο.

- **Paused:** Η activity που φιλοξενεί το fragment περνά στο παρασκήνιο.

- **Stopped:** Η activity που φιλοξενεί το fragment βρίσκεται στην κατάσταση Stopped.

- **View destroyed:** Το γραφικό περιεχόμενο του fragment καταστρέφεται.

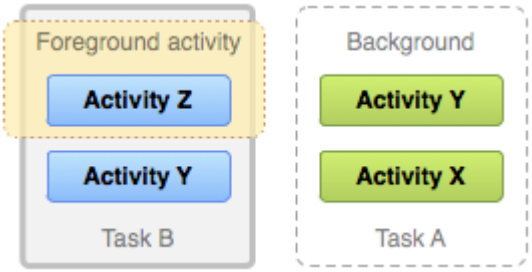
- **Destroyed:** Αποδεσμεύονται οι πόροι που ανήκουν στο fragment.

- **Detached:** Το fragment αφαιρείται από την activity, και το αντίστοιχο αντικείμενο στη μνήμη καταστρέφεται.

6.2.2.4 Πλοήγηση

Η ακολουθία των *ενεργών* οθονών μίας εφαρμογής κατά τη διάρκεια μίας συνόδου περιήγησης περικλείεται σε μία δομή που ονομάζεται *εργασία (task)*, η οποία ανήκει στη διεργασία της εφαρμογής. Έτσι, ο χρήστης είναι δυνατό να περιηγείται σε πρώτο επίπεδο μεταξύ πολλών ενεργών εφαρμογών, δηλαδή πολλών tasks, και σε δεύτερο επιπεδο μεταξύ πολλών οθονών εντός του task, δηλαδή εντός κάθε εφαρμογής.

Στο παρακάτω παράδειγμα φαίνονται δύο ενεργές εφαρμογές A και B, με τα αντίστοιχα tasks και τα περιεχόμενα activities. Στο προσκήνιο βρίσκεται η εφαρμογή B, με την activity Z ορατή στο χρήστη:



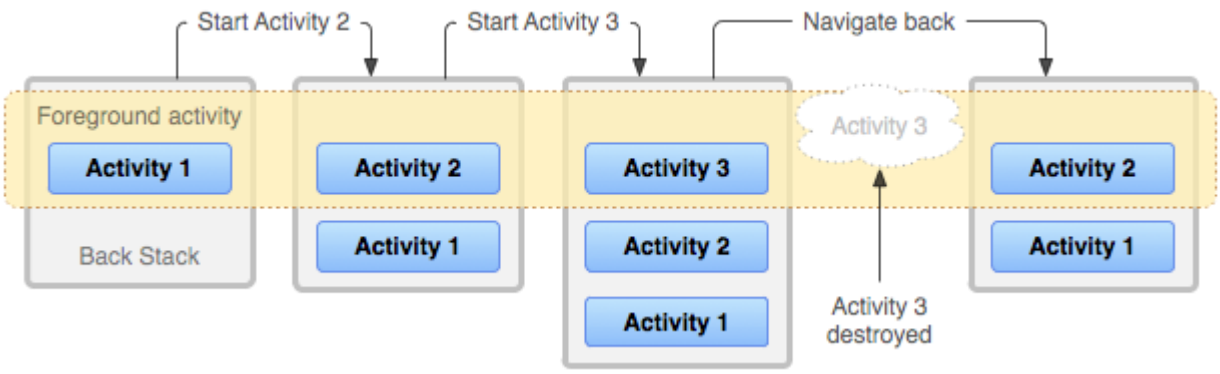
Σχήμα 6.4: Tasks και activities

Η πλοήγηση μεταξύ activities γίνεται σειριακά, είτε προς τα εμπρός είτε προς τα πίσω:

- ***Πλοήγηση εμπρός (Forward/Down-navigation):*** Ο χρήστης ανοίγει μία καινούρια οθόνη, η οποία και έρχεται στο προσκήνιο.

- ***Πλοήγηση πίσω (Back/Up-navigation):*** Με πάτημα του πλήκτρου *Back*, ο χρήστης επανέρχεται στην αμέσως προηγούμενη οθόνη στο ιστορικό.

Το ιστορικό των ενεργών activities και fragments διατηρείται σε μία δομή στοίβας, την *πίσω-στοίβα* (*back-stack*). Με κάθε ενέργεια πλοήγησης προς τα πίσω, το στοιχείο στην κορυφή της στοίβας εξάγεται και καταστρέφεται. Έτσι, αν στην τρέχουσα activity έχουν προστεθεί ένα ή περισσότερα fragments, απαιτείται ο χρήστης ισάριθμες φορές να πατήσει *Back* προτού βρεθεί σε θέση να επανέλθει στην προηγούμενη activity.



Σχήμα 6.5: Πίσω-στοίβα (back stack)

6.2.3 Υπηρεσίες

Όπως και σε άλλα λειτουργικά συστήματα, έτσι και στο Android οι υπηρεσίες (services) αποτελούν τον πρωταρχικό μηχανισμό εκτέλεσης εργασιών χωρίς γραφικό περιβάλλον.

Μία υπηρεσία Android εκπροσωπείται από ένα αντικείμενο της κλάσης *Service* (`android.app.Service`). Για την εξειδίκευση της συμπεριφοράς μίας υπηρεσίας, αρκεί να οριστεί μία κατάλληλη υποκλάση της κλάσης αυτής. Σε αντίθεση όμως με τις activities, ανά πάσα χρονική στιγμή μόνο ένα αντίγραφο της υπηρεσίας μπορεί να βρίσκεται ενεργό στη μνήμη.

Υπάρχουν δύο τρόποι με τους οποίους μπορεί μία υπηρεσία να ενεργοποιηθεί και να χρησιμοποιηθεί από κάποιο άλλο app component:

- ***Εκκίνηση (start)***

Η υπηρεσία εκκινείται για συνεχή εκτέλεση από κάποιο component μέσω της κλήσης `Context.startService()`, και σταματά να λειτουργεί μόνο με ρητή εντολή τερματισμού μέσω της κλήσης `Service.stopSelf()` από τον κώδικα της ίδιας της υπηρεσίας, ή της κλήσης `Context.stopService()` από τον κώδικα οποιουδήποτε άλλου component.

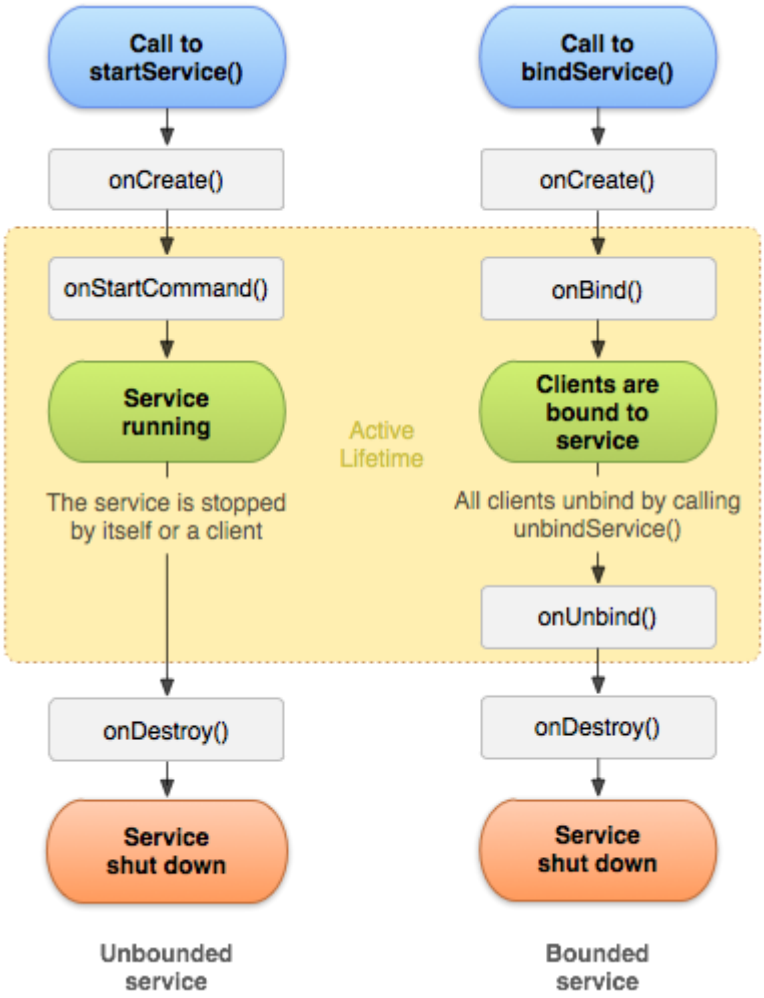
- ***Δέσμευση (binding)***

Ο μηχανισμός αυτός βασίζεται στην αρχιτεκτονική εξυπηρετητή-πελάτη. Η υπηρεσία, ενεργώντας ως εξυπηρετητής, δεσμεύεται από ένα ή περισσότερα component-πελάτες, για περιορισμένη χρονικά εκτέλεση μέσω της κλήσης `Context.bindService()`, και παραμένει ενεργή μέχρι να αποδεσμευτεί από όλους τους πελάτες μέσω της κλήσης `Context.unbindService()`, διαφορετικά μέχρι τον τερματισμό τους.

Το σπουδαιότερο προτέρημα της μεθόδου αυτής δεν βρίσκεται όμως στον έλεγχο της διάρκειας ζωής της υπηρεσίας από τον πελάτη, αλλά στη διεπαφή εξυπηρετητή-πελάτη που προσφέρει: η κλήση `Context.bindService()` επιστρέφει ένα αντικείμενο τύπου `android.os.IBinder`, πάνω στο οποίο μπορεί

να εκτελεί μεθόδους της υπηρεσίας όχι μόνο ο πελάτης, αλλά και διεργασίες άλλων εφαρμογών.

Με βάση τα παραπάνω, ο κύκλος ζωής μίας υπηρεσίας εξελίσσεται ως εξής:



Σχήμα 6.6: Κύκλος ζωής υπηρεσίας

Μία υπηρεσία μπορεί να παραμένει ενεργή ταυτόχρονα και με τους δύο τρόπους που περιγράφηκαν, να είναι δηλαδή και εκκινημένη και δεσμευμένη. Τότε η συνθήκη τερματισμού είναι η υπηρεσία να έχει λάβει ρητή εντολή τερματισμού (`Service.stopSelf()` ή `Context.stopService()`) *και* να έχει αποδεσμευθεί από όλους τους πελάτες της (δηλαδή όλοι οι πελάτες να έχουν καλέσει `Context.unbindService()`). Οι παραπάνω ενέργειες μπορούν να πραγματοποιηθούν με οποιαδήποτε επιθυμητή σειρά.

Αξίζει να σημειωθεί πως οι υπηρεσίες παραμένουν ενεργές στο παρασκήνιο, ο κώδικάς τους όμως δεν εκτελείται ούτε σε ξεχωριστή διεργασία, ούτε σε ξεχωριστό νήμα εντός της διεργασίας, αλλά στο κύριο νήμα. Κατά συνέπεια, για την αποσυμφόρηση του κύριου νήματος από χρονοβόρες ενέργειες απαιτείται ο προγραμματιστής να ορίσει νέα νήματα εντός του κώδικα της υπηρεσίας, ή να

χρησιμοποιήσει κάποιον από τους μηχανισμούς ασύγχρονης εκτέλεσης που περιγράφονται στην επόμενη παράγραφο.

Για σκοπούς εξοικονόμησης μνήμης και ενέργειας, η έκδοση 8.0 του Android (API level 26) επιβάλλει νέους περιορισμούς στην εκτέλεση υπηρεσιών στο παρασκήνιο. Κάθε εκκινούμενη υπηρεσία οφείλει πλέον να εμφανίζει μία κατάλληλη ειδοποίηση στο χρήστη, η οποία παραμένει ορατή καθ΄όλη τη διάρκεια ζωής της υπηρεσίας, διαφορετικά τερματίζεται από το σύστημα. Με αυτό τον τρόπο ο χρήστης έχει επίγνωση της εκτέλεσης της υπηρεσίας στο παρασκήνιο, και μπορεί ανά πάσα στιγμή να την τερματίσει.

6.2.4 Ασύγχρονη εκτέλεση εργασιών

Από προεπιλογή, ο κώδικας στο εσωτερικό ενός app component εκτελείται στο κύριο νήμα της διεργασίας της εφαρμογής. Το κύριο νήμα (main thread ή UI thread) είναι πρωταρχικά υπεύθυνο για τη διαχείριση του γραφικού περιβάλλοντος, όπως το χειρισμό συμβάντων αλληλεπίδρασης με το χρήστη (είσοδος κειμένου, εντοπισμός κίνησης χεριού κλπ) και την ανανέωση του περιεχομένου του. Αντίστροφα, κάθε ενέργεια τροποποίησης του γραφικού περιβάλλοντος πρέπει να εκτελείται από το κύριο νήμα, διαφορετικά θα αποτύχει. Εύκολα λοιπόν μπορεί να αντιληφθεί κανείς πως αν το κύριο νήμα αφεθεί να αναλώνεται σε εκτέλεση χρονοβόρων εργασιών (όπως είναι πολύπλοκοι μαθηματικοί υπολογισμοί ή είσοδος/έξοδος δεδομένων από/προς το σύστημα αρχείων, άλλες περιφερειακές συσκευές, το δίκτυο κλπ), το γραφικό περιβάλλον της εφαρμογής θα καταστεί αργό και μη αποκρίσιμο.

Προκύπτει συνεπώς η ανάγκη τέτοιες χρονοβόρες εργασίες να ανατίθενται σε ξεχωριστά νήματα, και μετά την ολοκλήρωσή τους να είναι σε θέση ασύγχρονα να ενημερώνουν το κύριο νήμα, αν αυτό είναι απαραίτητο. Για το σκοπό αυτό διατίθενται οι εξής μηχανισμοί:

- ***Απευθείας χειρισμός νημάτων***

Ο προγραμματιστής κατασκευάζει ένα ή περισσότερα νήματα χρησιμοποιώντας την παρεχόμενη από το Java Framework κλάση `java.lang.Thread`. Αν τα νήματα είναι αναγκαίο να επικοινωνούν ή να συγχρονίζονται μεταξύ τους - λόγου χάρη αναλαμβάνοντας ρόλους παραγωγών και καταναλωτών - τότε, πέραν των πρωτόγονων μηχανισμών κλειδωμάτων και αναμονής της Java (`Object.wait()`, `Object.notify()`), μπορεί να χρησιμοποιηθεί ο μηχανισμός ανταλλαγής μηνυμάτων του Android framework. Ο μηχανισμός αυτός συντίθεται από τις κλάσεις `Looper`, `Handler`, `Message` και `MessageQueue` του πακέτου `android.os`, και ακολουθεί το παρακάτω μοντέλο εκτέλεσης:

  - Σε κάθε νήμα αντιστοιχίζεται ένα αντικείμενο τύπου `Looper`. Με την κλήση `Looper.loop()` το συγκεκριμένο αντικείμενο επιτρέπει στο νήμα

να εκτελείται στο διηνεκές και να λαμβάνει μηνύματα ως αντικείμενα τύπου `Message`.
- Για τον χειρισμό των εισερχόμενων μηνυμάτων ορίζεται για το νήμα ένα αντικείμενο `Handler`. Ο χειρισμός γίνεται με τη βοήθεια της μεθόδου `Handler.handleMessage()`.
- Τα εισερχόμενα μηνύματα που αναμένουν χειρισμό διατηρούνται σε μία δομή ουράς τύπου `MessageQueue`.
- Με χρήση του αντικειμένου τύπου `Handler` του παραλήπτη είναι δυνατή η αποστολή μηνυμάτων καλώντας τη μέθοδο `Handler.sendMessage()`.

Η διαχείριση νημάτων από τον προγραμματιστή σε τόσο χαμηλό επίπεδο μπορεί να αυξάνει το μέγεθος και την πολυπλοκότητα του κώδικα για απλές εργασίες, ταυτόχρονα όμως δίνει πλήρη έλεγχο πάνω στη διαδικασία και τους πόρους που αυτή καταναλώνει, επιτρέποντας τη βελτιστοποίηση και την εξειδίκευσή της για τις ανάγκες μίας δοσμένης εργασίας. Εξάλλου, ο βασικός μηχανισμός που μόλις παρουσιάσαμε αποτελεί τη βάση για κάθε απλούστερη υλοποίηση υψηλού επιπέδου.

Μια απλοποίηση σε πρώτο επίπεδο γίνεται με την βοήθεια της κλάσης `android.os.HandlerThread`, η οποία ενσωματώνει σε ένα αντικείμενο ένα Java `Thread`, έναν `Looper` και έναν `Handler`, επιφορτίζοντας τον προγραμματιστή μόνο με τον ορισμό και το χειρισμό των απαραίτητων μηνυμάτων.

- ***Μηχανισμοί ασύχρονης εκτέλεσης της Java***

Η βιβλιοθήκη `java.util.concurrent` παρέχει μία σειρά από μηχανισμούς παράλληλης εκτέλεσης, βασισμένους στην κλάση `Executor`. Η εκτέλεση μίας εργασίας σε έναν Executor γίνεται μέσω της κλήσης `Executor.execute()`. Ορίζοντας κατάλληλα τις παραμέτρους κατασκευής του αντικειμένου `Executor`, δίνεται η επιλογή η εκτέλεση να ανατίθεται τόσο στο νήμα του καλούντος component, όσο και σε μία ομάδα από νήματα (thread pool) σταθερού ή δυναμικά μεταβαλλόμενου μεγέθους. Οι υπόλοιπες κλάσεις της βιβλιοθήκης υλοποιούν προηγμένους μηχανισμούς συγχρονισμού (όπως κλειδώματα, ουρές αναμονής και σημαφόρους).

- ***Μηχανισμοί ασύγχρονης εκτέλεσης του Android Framework***

Στην πλειοψηφία των περιπτώσεων, οι εργασίες προς εκτέλεση απαιτείται απλώς να φορτώνουν δεδομένα στο παρασκήνιο και με αυτά να ενημερώνουν κάποιο στοιχείο του UI, και αναμένεται να διαρκούν λίγο (συνήθως μέχρι μερικά δευτερόλεπτα) σε σύγκριση με το μέσο χρόνο χρήσης της εφαρμογής. Για τέτοιες εργασίες, το Android framework προσφέρει έναν ευέλικτο και εξαιρετικά εύκολο στη χρήση μηχανισμό εκτέλεσης υψηλού επιπέδου μέσω της κλάσης `android.os.AsyncTask`. Μία εργασία μπορεί να εκτελεστεί ορίζοντας ένα στιγμιότυπο της κλάσης αυτής, και καλώντας πάνω σε αυτό τη μέθοδο `execute()`.

Τότε αυτόματα δημιουργείται ένα πρόσθετο νήμα με την κατάλληλη διάρκεια ζωής, και ο χειρισμός των απαραίτητων μηνυμάτων για την επικοινωνία με το κύριο νήμα αφήνεται στο framework. Η ενημέρωση του κύριου νήματος για την πρόοδο εκτέλεσης της εργασίας γίνεται από τα παρακάτω callbacks:

- `onPreExecute()`: Καλείται από το UI thread πριν την έναρξη της εκτέλεσης, και χρησιμοποιείται προαιρετικά για την αρχικοποίηση της ένδειξης της πρόοδου εκτέλεσης.
- `doInBackground()`: Καλείται από το νήμα παρασκηνίου, και περιέχει τον κώδικα της εργασίας που επιθυμούμε να εκτελεστεί. Συνιστά τη μοναδική μέθοδο που απαιτείται να υλοποιεί ένα στιγμιότυπο της `AsyncTask`.
- `onProgressUpdate()`: Καλείται από το UI thread, και ενημερώνει την ένδειξη της προόδου εκτέλεσης κατόπιν κλήσης της `AsyncTask.publishProgress()`.
- `onPostExecute()`: Καλείται από το UI thread, και παρέχει στην εφαρμογή το συνολικό αποτέλεσμα της εργασίας. Υλοποιείται στις περιπτώσεις όπου η εργασία επιστρέφει δεδομένα, ή υπάρχουν ενέργειες που οφείλουν να πραγματοποιηθούν μετά την ολοκλήρωσή της.

Για εκτέλεση και χρονοπρογραμματισμό εργασιών οι οποίες δεν απαιτείται να ενημερώνουν το UI, το Android framework παρέχει επιπλέον τις κλάσεις `IntentService`, `WorkManager` και `JobScheduler`.

6.2.5 Διαδιεργασιακή επικοινωνία

Για την επικοινωνία μεταξύ των app components - και σε δεύτερο επίπεδο μεταξύ διεργασιών - το Android framework ορίζει έναν στοιχειώδη τύπο μηνύματος μέσω της κλάσης `android.content.Intent`.

Ένα αντικείμενο τύπου `Intent` περιγράφει μία ενέργεια την πραγματοποίηση της οποίας ένα component αναθέτει σε κάποιο άλλο, συνοδευόμενη από τα απαραίτητα δεδομένα για το σκοπό αυτό. Για παράδειγμα, μία activity μπορεί να εκκινήσει μία άλλη activity για την εμφάνιση μίας οθόνης που έχει επιλέξει ο χρήστης, ή μία υπηρεσία για την αναπαραγωγή πολυμέσων, τη μεταφόρτωση ενός αρχείου από το δίκτυο κλπ.

Έτσι, κάθε τέτοιο αντικείμενο συντίθεται από ένα κλειδί σε μορφή συμβολοσειράς που περιγράφει την ενέργεια (*action*) προς εκτέλεση, και από ένα σύνολο ζευγών κλειδιών-τιμών που αναπαριστούν τα συνοδευτικά δεδομένα. Ο τύπος των συνοδευτικών δεδομένων μπορεί να είναι οποιοσδήποτε από τους βασικούς τύπους της Java (`Integer`, `Float`, `String`, `Boolean` κλπ), ή κάποια κλάση που υλοποιεί τη διαπροσωπεία `android.os.Parcelable`.

Εάν κατά την κατασκευή ενός αντικειμένου *Intent* προσδιορίζεται ο παραλήπτης, λέμε ότι πρόκειται για ένα *ρητό* (*explicit*) Intent, διαφορετικά για ένα *υποδηλούμενο* (*implicit*) Intent. Στη δεύτερη περίπτωση, το σύστημα αναζητά ένα component το οποίο μπορεί να εκτελέσει την περιγραφόμενη ενέργεια.

Μία ειδική κατηγορία μηνυμάτων στα οποία ενσωματώνονται αντικείμενα implicit intent είναι τα μηνύματα εκπομπής (broadcasts), τα οποία απευθύνονται προς το σύνολο του συστήματος. Για την παραλαβή τέτοιων μηνυμάτων το Android framework προβλέπει εξειδικευμένα components, τους broadcast receivers (αντικείμενα της κλάσης `android.content.BroadcastReceiver`). Οι broadcast receivers λειτουργούν σύμφωνα με το πρότυπο publish-subscribe: κάθε receiver δηλώνει μία λίστα (*intent filter*) από τύπους μηνυμάτων (intents) τα οποία ενδιαφέρεται να χειρίζεται. Με την αποστολή ενός broadcast, το μήνυμα οφείλει να παραληφθεί από όλους τους ενδιαφερόμενους ενεργούς broadcast receivers. Ένας receiver μπορεί να ενεργοποιείται στατικά με δήλωσή του στο αρχείο manifest της εφαρμογής, ή να χρησιμοποιείται δυναμικά από τον κώδικα της εφαρμογής με χρήση των κλήσεων `Context.registerReceiver()` και `Context.unregisterReceiver()`.

Ο βασικός μηχανισμός διαδιεργασιακής επικοινωνίας μέσω intents παρέχεται μέσω αντικειμένων της κλάσης `android.os.Binder`, η οποία υλοποιεί την διαπροσωπεία `IBinder`. Αποτελεί επέκταση του μηχανισμού που παρέχεται για δέσμευση (binding) υπηρεσιών εντός της ίδιας εφαρμογής, και επιτρέπει την εκτέλεση μεθόδων μίας υπηρεσίας που ανήκει σε διαφορετική διεργασία μέσω κλήσεων RPC (*remote procedure call*). Η διεπαφή της υπηρεσίας για κλήσεις RPC συνήθως ορίζεται μέσω της γλώσσας AIDL (*Android Interface Definition Language*).

6.2.6 Μόνιμη αποθήκευση δεδομένων

Κάθε εγκατάσταση Android ενσωματώνει στα βασικά εργαλεία συστήματος μία διανομή του SQLite RDBMS, η οποία εκτίθεται και στο API του Android SDK. Αποτελεί λοιπόν για τους προγραμματιστές τη βασική επιλογή για τη διαχείριση και αποθήκευση των δεδομένων της εφαρμογής τοπικά στη συσκευή, όταν οι λειτουργίες αυτές είναι επιθυμητό να γίνονται με τη βοήθεια βάσεων δεδομένων.

Βέβαια, μία εφαρμογή Android μπορεί πάντα να διαβάζει και να αποθηκεύει δεδομένα από και προς το σύστημα αρχείων, χρησιμοποιώντας τις βιβλιοθήκες της Java για File I/O. Καθώς το Android βασίζεται στον πυρήνα του Linux, είναι συμβατό με όλα τα συστήματα αρχείων που υποστηρίζονται από αυτόν (πχ ext2/ext3/ext4, FAT, yaffs), μπορεί όμως να χρησιμοποιεί και ιδιοταγή συστήματα ενσωματωμένα στις συσκευές από τους κατασκευαστές τους (πχ το RFS για συσκευές Samsung). Σε νεότερα συστήματα επικρατεί η τάση να χρησιμοποιείται το ext4 για τα σταθερά αποθηκευτικά μέσα (εσωτερική μνήμη συσκευής) λόγω της αξιοπιστίας και της ασφάλειας που παρέχει. Σε λογικό επίπεδο, ο αποθηκευτικός χώρος μίας συσκευής είναι ορατός στις εφαρμογές με δύο μορφές:

- *Εσωτερικός χώρος (internal storage):* Περιλαμβάνει μόνο τον κατάλογο εγκατάστασης της εφαρμογής που περιέχεται στον κατάλογο συστήματος */Android/data/<package_name>*, όπου *<package_name>* είναι η ονομασία του ριζικού πακέτου (package). Η εφαρμογή μπορεί στο χώρο αυτό ελεύθερα να

δημιουργεί, να διαβάζει και να γράφει βάσεις δεδομένων ή αρχεία για προσωρινή ή μόνιμη χρήση, και καμία άλλη εφαρμογή δεν είναι δυνατό να αποκτήσει πρόσβαση στα δεδομένα αυτά, πρόκειται δηλαδή για ιδιωτικά δεδομένα.

- *Εξωτερικός χώρος (external storage):* Περιλαμβάνει τα αφαιρούμενα αποθηκευτικά μέσα (κάρτες SD, μνήμες flash κλπ), καθώς και τους δημόσιους καταλόγους πολυμέσων, εγγράφων και λήψεων (*DCIM/, Download/, Documents/, Pictures/, Videos/* κλπ). Η εφαρμογή είναι ελεύθερη να αποθηκεύει δημόσια ή ιδιωτικά δεδομένα στο χώρο αυτό, και να έχει πρόσβαση στα δεδομένα που έχει παραγάγει. Για την πρόσβαση σε δεδομένα που έχουν δημιουργηθεί από άλλες εφαρμογές, αποκτείται η παροχή αντίστοιχης εξουσιοδότησης από το χρήση κατά το χρόνο εκτέλεσης.

Συχνά είναι επιθυμητό τα τοπικά αποθηκευμένα δεδομένα μιας εφαρμογής να είναι διαθέσιμα - με ελεγχόμενη πρόσβαση - σε components άλλων εφαρμογών, ανεξάρτητα από τη φυσική μορφή αποθήκευσής τους στη συσκευή. Τέτοια δεδομένα μπορούν να είναι αρχεία, απλές εγγραφές βάσεων δεδομένων, καρτέλες επαφών (contacts) κλπ. Ένα τέτοιο επίπεδο αφαίρεσης εισάγουν οι *πάροχοι δεδομένων (content providers)*.

Ένας content provider μπορεί να ανήκει στο σύστημα ή να αποτελεί app component μίας εφαρμογής, και εκθέτει τα δεδομένα στα οποία έχει δικαιοδοσία μέσω μίας διεπαφής με τη μορφή βάσης δεδομένων SQLite, επιτρέποντας σε app components εφαρμογών να αποκτούν πρόσβαση σε αυτά πραγματοποιώντας ερωτήματα SQL. Ο προγραμματιστής έχει τη δυνατότητα να κατασκευάζει δικούς του content providers για τις εφαρμογές του επεκτείνοντας την κλάση `android.content.ContentProvider` και ορίζοντας την ερμηνεία των ερωτημάτων SQL, τη σύνδεση με την πραγματική πηγή των δεδομένων και την πραγματοποίηση των απαραίτητων ενεργειών για την άντληση και τη μετατροπή τους σε μορφή εγγραφών SQLite. Το Android framework παρέχει από προεπιλογή δύο τύπους content provider για πρόσβαση σε αρχεία πολυμέσων και έγγραφα, μέσω των APIs *MediaStore* και *DocumentsProvider* αντίστοιχα.

Τα ερωτήματα των components-πελατών γίνονται πάνω σε ένα καθολικά ορατό URI *(content URI)*. Κάθε content provider ορίζει ένα ριζικό (root) URI για τον έλεγχο της πρόσβασης, και θυγατρικά URIs για τον προσδιορισμό των ζητούμενων δεδομένων. Ένα content URI έχει την εξής γενική μορφή:

`content://<authority>/<path>`

Το ριζικό URI (κόκκινο) σχηματίζεται από το πρόθεμα `content://` και την δικαιοδοσία `<authority>` του παρόχου. Αυτή λαμβάνει μία προκαθορισμένη τιμή εάν ο πάροχος ανήκει στο σύστημα, διαφορετικά - στην περίπτωση δηλαδή που ο πάροχος ανήκει σε εφαρμογή - ταυτίζεται με την ονομασία του πακέτου της εφαρμογής. Τα θυγατρικά URIs σχηματίζονται προσθέτοντας κάθε φορά στο ριζικό

URI τη διαδρομή `<path>` (μπλε) που έχει ορισθεί για το επιθυμητό σύνολο δεδομένων.

Για τη βελτιστοποίηση της συνεργασίας μεταξύ ενός content provider και των υπόλοιπων components της εφαρμογής, και ιδιαίτερα του UI, παρέχεται ένα σύνολο από βοηθητικούς μηχανισμούς, με σπουδαιότερο αυτόν των *φορτωτών* (*loaders*). Ένας φορτωτής υλοποιείται ως υποκλάση της `android.content.Loader<D>`, όπου *D* είναι ο τύπος των δεδομένων που επιστρέφεται από τον provider, και αναλαμβάνει να παρακολουθεί συγκεκριμένα content URIs. Σε περίπτωση μεταβολής των δεδομένων που αντιπροσωπεύει κάποιο URI, ο φορτωτής ανανεώνει αυτόματα το τοπικό αντίγραφο που διατηρεί από αυτά, εκτελώντας το αντίστοιχο ερώτημα σε ένα προσωρινό νήμα παρασκηνίου. Για εκτέλεση μεμονωμένων ερωτημάτων στο παρασκήνιο δίνεται επιπλέον η κλάση `android.content.AsyncQueryHandler`.

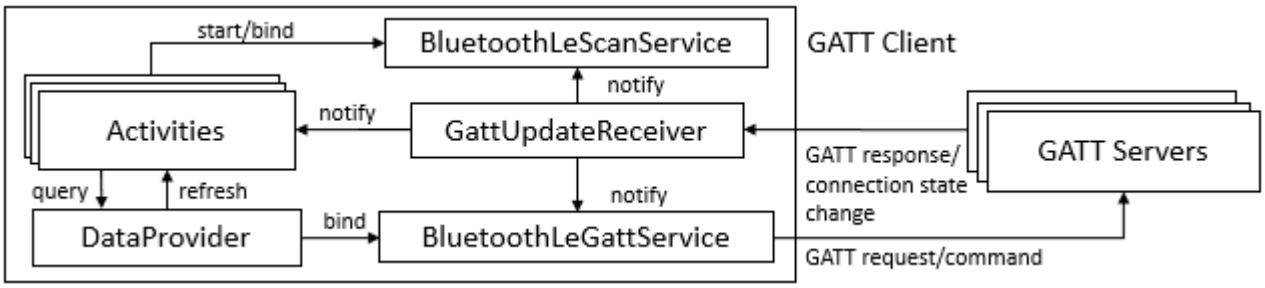
7 Εφαρμογή πελάτη

7.1 Δομή εφαρμογής

Η εφαρμογή πελάτη έχει αναπτυχθεί για το λειτουργικό σύστημα Android, απαιτώντας για την εκτέλεσή της την έκδοση 5.0 αυτού ή οποιαδήποτε νεότερη, καθώς και υποστήριξη του Bluetooth Low Energy. Εισαγωγικά παρουσιάζονται οι μονάδες λογισμικού που την αποτελούν.

7.1.1 Συστατικά μέρη

Η εφαρμογή μας βασίζεται στη συνεργασία κάποιων θεμελιωδών συστατικών μερών (app components) του περιβάλλοντος Android, με τον τρόπο που παρουσιάζεται στο διάγραμμα που ακολουθεί:



Σχήμα 7.1: Συστατικά μέρη εφαρμογής πελάτη

Τα συστατικά αυτά μέρη περιλαμβάνουν:

- Τις απαραίτητες *δραστηριότητες γραφικού περιβάλλοντος (activities)* για την οπτικοποίηση της πληροφορίας και την αλληλεπίδραση με το χρήστη. Αυτές παρουσιάζονται αναλυτικά στην υποενότητα 7.2.1.

- Έναν *πάροχο περιεχομένου (content provider)* με την ονομασία *DataProvider*, τον οποίο παρουσιάζουμε στην υποενότητα 7.3.7, για εύκολη και αποδοτική πρόσβαση στα μόνιμα αποθηκευμένα δεδομένα της εφαρμογής.

- Έναν *broadcast receiver* με την ονομασία *GattUpdateReceiver*, τον οποίο παρουσιάζουμε στην υποενότητα 7.3.2, για την κοινοποίηση συμβάντων σχετικών με τις ενεργές συνδέσεις (ολοκλήρωση/αποτυχία σύνδεσης, αποσύνδεση, είσοδος δεδομένων από κάποιο χαρακτηριστικό κλπ) στα υπόλοιπα components της εφαρμογής.

- Δύο *υπηρεσίες (services)* για την υλοποίηση - ανεξάρτητα από το γραφικό περιβάλλον - λειτουργιών για την επικοινωνία με τις απομακρυσμένες συσκευές. Οι υπηρεσίες αυτές είναι:

(i) Η υπηρεσία *BluetoothLeGattService*, την οποία παρουσιάζουμε στην υποενότητα 7.3.3, για τη διαχείριση των ενεργών συνδέσεων και την εκτέλεση ενεργειών GATT.
(ii) Η υπηρεσία *BluetoothLeScanService*, την οποία παρουσιάζουμε στην υποενότητα 7.3.1, για τη σάρωση του δικτύου και τον εντοπισμό των συσκευών που διαφημίζονται σε αυτό.

7.1.2 Πακέτα-κλάσεις

Παρακάτω δίνουμε την ιεραρχία των πακέτων και των κλάσεων Java που απαρτίζουν την εφαρμογή μας. Με τον συμβολισμό `Class2:Class1` δηλώνουμε πως η κλάση `Class2` αποτελεί υποκλάση της κλάσης `Class1` (αντίστοιχα, υλοποιεί μία διαπροσωπεία):

```
com.example.android.blesense
     connectivity
          BackgroundScanCallback:ScanCallback
          ForegroundScanCallback:ScanCallback
          BluetoothLeGattService:Service
          BluetoothLeScanService:Service
          ConnectionManager
          ScanConfiguration
     device
          DeviceConfig
          DeviceEntry
          DeviceListAdapter:RecyclerView.Adapter,
                            storage.ListRefreshable
          DeviceHistoryAdapter:RecyclerView.Adapter,
                            storage.ListRefreshable
          DeviceInfo
          DeviceWrapper
          SensorConfig
     dialog
          AlertDialogFragment:DialogFragment
          InputDialogFragment:DialogFragment
          SimpleInputDialogFragment:DialogFragment
     log
          LogAdapter:RecyclerView.Adapter,
                     storage.ListRefreshable
          LogEntry
          LogItemLookup:ItemDetailsLookup
          LogSelectionObserver:SelectionTracker.SelectionObserver
          LogMenuController:ActionMode.Callback
          RecordingEntry
```

```
monitor
        LiveDataReceiver
        LiveRecorder
        Monitor
        MonitorConfig
        MonitoringManager
        MonitoringSession
        Recording
plot
        MultiValuePlot:Plot
        Plot
        interface PlotAdapter
        PlotConfig
        PlotData
        PlotFactory
        PlotFragment:Fragment
        PlotStats
        PlotToGraphViewAdapter:PlotAdapter
        StatsPlot:Plot
storage
        BackgroundLoadCallbacks:LoaderManager.LoaderCallbacks
        BackgroundQueryHandler:AsyncQueryHandler
        interface ListRefreshable
        DataConnector:SQLiteOpenHelper
        DataProvider:ContentProvider
        DataSchema:BaseColumns
        DataService
ui
        AppInfoActivity:AppCompatActivity
        AppSettingsActivity:AppCompatActivity
        AppSettingsFragment:PreferenceFragmentCompat
        DeviceControlActivity:AppCompatActivity
        DeviceDashboardActivity:AppCompatActivity
        DeviceDashboardFragment:Fragment
        DeviceHistoryActivity:AppCompatActivity
        LogViewerActivity:AppCompatActivity
        DeviceScanActivity:AppCompatActivity
        RecordingViewerActivity:AppCompatActivity
        SelectableList
        SettingsListView:RelativeLayout
        SplashActivity:AppCompatActivity
utils
        CalendarUtils
        GattAttributes
        GlobalDefs
        ImageMappingUtils
```

Πίνακας 7.2: Πακέτα και κλάσεις Java της εφαρμογής πελάτη

Την υλοποίηση των σπουδαιότερων κλάσεων από τις παραπάνω πραγματευόμαστε στις ενότητες που ακολουθούν.

7.2 Γραφικό περιβάλλον

Θα ξεκινήσουμε την παρουσίαση του γραφικού περιβάλλοντος από την οργάνωση των οθονών. Αυτή γίνεται σε τρία επίπεδα:

(i) Top-level activities

Είναι τρεις οθόνες με τις οποίες αλληλεπιδρά ο χρήστης κατά την εκκίνηση της εφαρμογής. Η εναλλαγή μεταξύ των τριών οθονών γίνεται επιλέγοντας το κατάλληλο στοιχείο της μπάρας πλοήγησης (navigation bar) που βρίσκεται στο κάτω μέρος τους. Οι οθόνες αυτές είναι οι εξής:

- *DeviceScanActivity*, για αναζήτηση συσκευών και σύνδεση σε αυτές.
- *DeviceHistoryActivity*, για προβολή του ιστορικού των συσκευών.
- *SettingsAppActivity*, για αλλαγή των προτιμήσεων της εφαρμογής.

Εικόνα 7.3: Μπάρα πλοήγησης

(ii) Δεύτερο επίπεδο

Περιλαμβάνει τις οθόνες που είναι προσβάσιμες σε ένα βήμα από μία ή περισσότερες οθόνες του πρώτου επίπεδου:

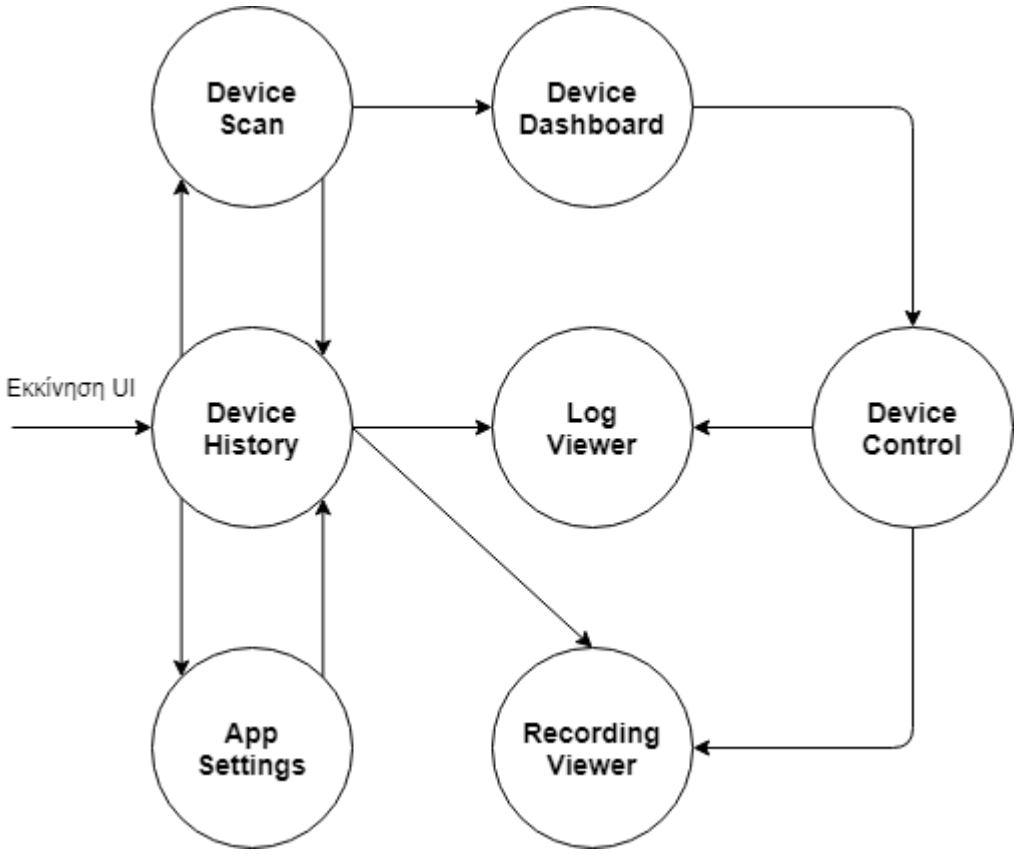
- *DeviceDashboardActivity*, για προβολή και διαχείριση συνόδων εποπτείας.
- *LogViewerActivity*, για προβολή του ιστορικού συμβάντων μίας συσκευής.
- *RecordingViewerActivity*, για προβολή των αποθηκευμένων συνόδων εποπτείας μίας συσκευής.

(iii) Τρίτο επίπεδο

Περιλαμβάνει τις οθόνες που είναι προσβάσιμες σε ένα βήμα από μία ή περισσότερες οθόνες του δεύτερου επίπεδου:

- *DeviceControlActivity*, για προβολή και τροποποίηση των ρυθμίσεων μίας συσκευής.

Οι δυνατές μεταβάσεις μεταξύ των οθονών της εφαρμογής φαίνονται στον παρακάτω *γράφο πλοήγησης*:



Σχήμα 7.4: Γράφος πλοήγησης

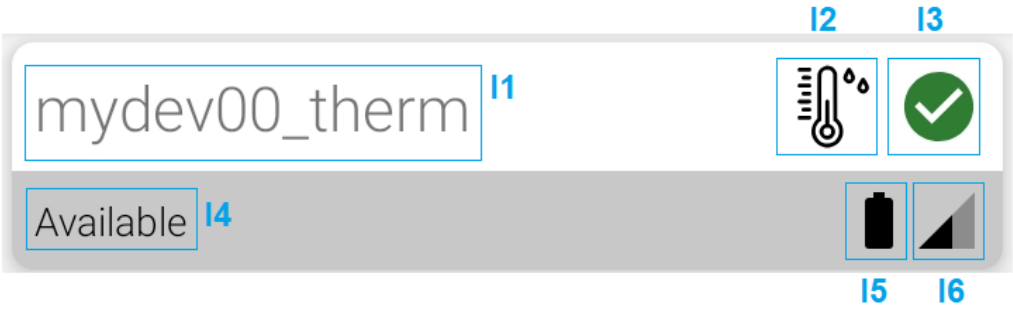
Σημειώνουμε πως στον παραπάνω γράφο η ύπαρξη κατευθυνόμενης ακμής *A* → *B* σημαίνει πως η οθόνη *A* μπορεί να *εκκινήσει* την οθόνη *B*. Σε κάθε περίπτωση, από την τρέχουσα οθόνη μπορούμε να επιστρέψουμε στη γονική της πατώντας το πλήκτρο "*Home*".

7.2.1 Οθόνες και πλοήγηση

Παρουσιάζουμε τώρα κάθε οθόνη αναλυτικά. Όλες οι κλάσεις που περιγράφονται παρακάτω αποτελούν μέλη του πακέτου `com.example.android.blesense.ui`.

7.2.1.1 DeviceScanActivity

Ο χρήστης κατευθύνεται σε αυτή την οθόνη κατά την εκκίνηση της εφαρμογής, ή με επιλογή του στοιχείου *"Network" (Δίκτυο)* της αρχικής μπάρας πλοήγησης. Η οθόνη εμφανίζει μια – ανανεούμενη σε πραγματικό χρόνο – λίστα όλων των διαθέσιμων συσκευών στο δίκτυο, τόσο των συνδεδεμένων με την εφαρμογή, όσο και των advertisers. Ο εντοπισμός των advertisers γίνεται με BLE σάρωση (scanning). Αναλυτικά παρουσιάζουμε τη διαδικασία εντοπισμού των συσκευών και ανανέωσης της λίστας στην υποενότητα 7.3.1.



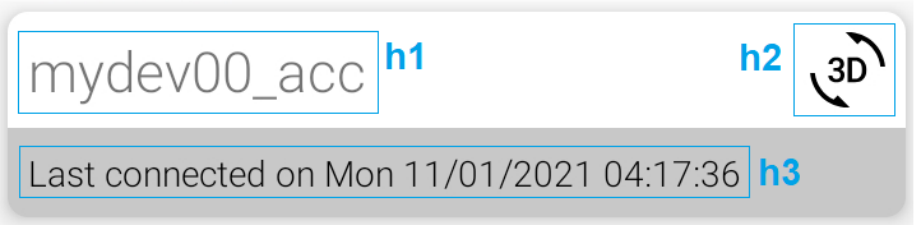
Εικόνα 7.5: Δείγμα στοιχείου λίστας συσκευών

Η αλληλεπίδραση του χρήστη με τη λίστα είναι εξαιρετικά απλή: πατώντας στιγμιαία πάνω στο πλακίδιο μίας συσκευής που εκπέμπει στο δίκτυο, η εφαρμογή επιχειρεί να συνδεθεί με τη συσκευή. Με την ολοκλήρωση της σύνδεσης, μεταβάλλεται η αντίστοιχη ένδειξη κατάστασης σύνδεσης. Πατώντας παρατεταμένα πάνω στο πλακίδιο μίας συνδεδεμένης συσκευής, η εφαρμογή αποσυνδέεται από τη συσκευή και αποδεσμεύει όλους τους σχετιζόμενους με αυτήν πόρους. Κάθε στοιχείο της λίστας υλοποιείται ως στιγμιότυπο της κλάσης `CardView`, και ενσωματώνει τα εξής γραφικά αντικείμενα στο εσωτερικό του, όπως σημειώνονται και αριθμούνται στην παραπάνω εικόνα:

- [*I1*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο το όνομα της συσκευής, όπως αυτό εμφανίζεται κατά το advertising.
- [*I2*] Ένα εικονίδιο αντιπροσωπευτικό του τύπου της συσκευής, τύπου `ImageView`, αντίστοιχο του CM Service το οποίο αυτή υλοποιεί.
- [*I3*] Ένα εικονίδιο, τύπου `ImageView`, με περιεχόμενο την ένδειξη κατάστασης καλής λειτουργίας της συσκευής (*OK*/*Warning*/*Critical*).
- [*I4*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο την ένδειξη της κατάστασης σύνδεσης της συσκευής (*διαθέσιμη* – "*Available*", *σύνδεση σε εξέλιξη* – "*Connecting*", *συνδεδεμένη* – "*Connected*", *αποσυνδεδεμένη* – "*Disconnected*").
- [*I5*] Ένα εικονίδιο, τύπου `ImageView`, με περιεχόμενο την ένδειξη του επιπέδου φόρτισης της μπαταρίας της συσκευής.
- [*I6*] Ένα εικονίδιο, τύπου `ImageView`, με περιεχόμενο την ένδειξη της ισχύος λήψης σήματος.

7.2.1.2 DeviceHistoryActivity

Ο χρήστης κατευθύνεται σε αυτή την οθόνη με επιλογή του στοιχείου "*History*" (*Ιστορικό*) της αρχικής μπάρας πλοήγησης. Από αυτή την οθόνη είναι δυνατή η προβολή του συνόλου του αποθηκευμένου ιστορικού – συμβάντων και καταγεγραμμένων χρονοσειρών – για όλες τις συσκευές με τις οποίες η εφαρμογή έχει συνδεθεί από τη στιγμή της τελευταίας της εγκατάστασης. Η λίστα των συσκευών αποτελείται από στοιχεία της εξής μορφής:



Εικόνα 7.6: Δείγμα στοιχείου λίστας ιστορικού συσκευών

Κάθε στοιχείο της λίστας υλοποιείται ως στιγμιότυπο της κλάσης `CardView`, και ενσωματώνει τα εξής γραφικά αντικείμενα στο εσωτερικό του, όπως σημειώνονται και αριθμούνται στην παραπάνω εικόνα:

- [*h1*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο το όνομα της συσκευής, όπως αυτό έχει αποθηκευθεί από την πρώτη σύνδεση με αυτήν.
- [*h2*] Ένα εικονίδιο αντιπροσωπευτικό του τύπου της συσκευής, τύπου `ImageView`, αντίστοιχο του CM Service το οποίο υλοποιεί.
- [*h3*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο την ημερομηνία και ώρα της τελευταίας σύνδεσης της εφαρμογής με τη συσκευή.

Για το φιλτράρισμα των αποτελεσμάτων, ο χρήστης έχει τη δυνατότητα, μέσω του επιλογέα που βρίσκεται στο πάνω μέρος της οθόνης, να προσδιορίσει το χρονικό εύρος της αναζήτησης. Τον τρόπο αποθήκευσης και ανάκλησης του ιστορικού πραγματευόμαστε αναλυτικά στην υποενότητα 7.3.7.

7.2.1.3 AppSettingsActivity

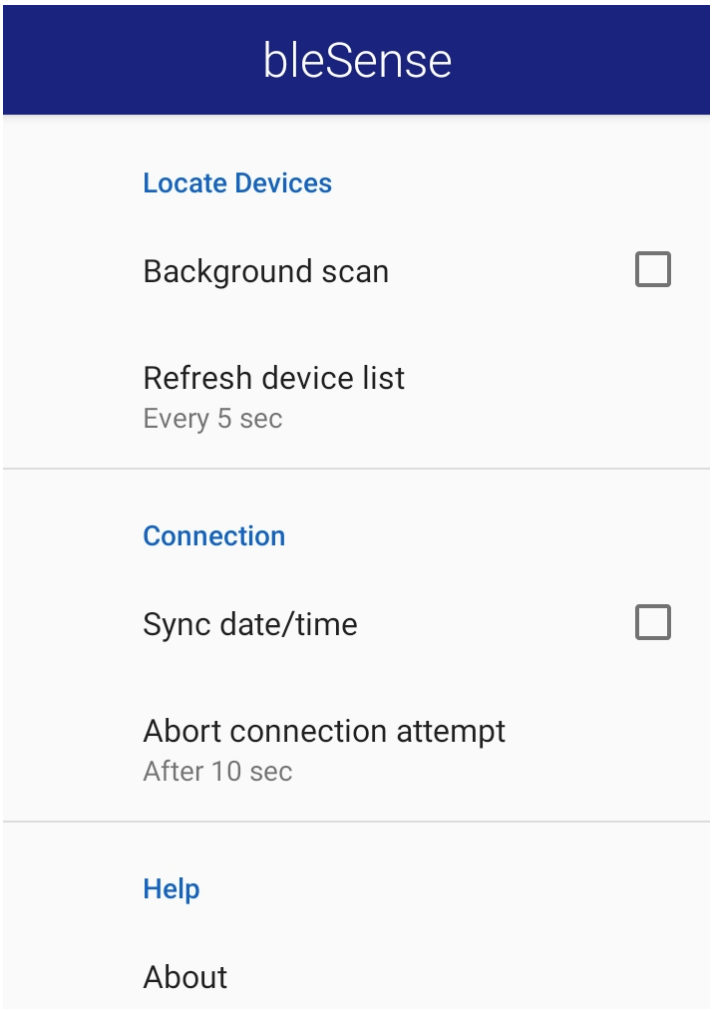
Ο χρήστης κατευθύνεται σε αυτή την οθόνη με επιλογή του στοιχείου "*Settings*" (*Ρυθμίσεις*) της αρχικής μπάρας πλοήγησης. Από αυτή την οθόνη είναι δυνατή η τροποποίηση των εξής ρυθμίσεων, οι οποίες αποθηκεύονται μόνιμα και αφορούν το σύνολο των λειτουργιών της εφαρμογής:

- "*Background scan*": Η επιλογή αυτού του checkbox ενεργοποιεί τη λειτουργία σάρωσης για συσκευές στο παρασκήνιο μετά τον τερματισμό του UI της εφαρμογής, με τον τρόπο που περιγράφεται στην υποενότητα 7.3.1.

- "*Refresh device list*": Με επιλογή αυτού του στοιχείου εμφανίζεται ένα παράθυρο διαλόγου, στο οποίο ο χρήστης εισάγει την επιθυμητή περίοδο ανανέωσης της λίστας των διαθέσιμων συσκευών. Η ανανέωση της λίστας γίνεται με τον τρόπο που περιγράφεται στην υποενότητα 7.3.1.

- "*Sync date/time*": Κατά τη διαδικασία εγκατάστασης της σύνδεσης, επιτρέπει στον απομακρυσμένο server να συγχρονισθεί με την ημερομηνία και ώρα του client, με αποστολή από τον δεύτερο ενός πακέτου *CP_UPDATE_RTC*.

- "*Abort connection attempt*": Με επιλογή αυτού του στοιχείου εμφανίζεται ένα παράθυρο διαλόγου, στο οποίο ο χρήστης εισάγει το επιθυμητό χρονικό όριο για την εγκατάσταση σύνδεσης με τις διαθέσιμες συσκευές. Αναλυτικά στη διαχείριση συνδέσεων αναφερόμαστε στην υποενότητα 7.3.2.



Εικόνα 7.7: Στιγμιότυπο της AppSettingsActivity

Τον τρόπο αποθήκευσης και διαχείρισης των παραπάνω προτιμήσεων, καθώς και τη χρήση τους από τον κώδικα της εφαρμογής πραγματευόμαστε στην υποενότητα 7.3.8.

7.2.1.4 DeviceDashboardActivity

Εικόνα 7.8: Στιγμιότυπο της DeviceDashboardActivity

Ο χρήστης κατευθύνεται σε αυτή την οθόνη με επιλογή μίας συνδεδεμένης συσκευής από τη λίστα της *DeviceScanActivity*. Αποτελεί το κεντρικό σημείο από το οποίο είναι δυνατή η συνεχής εποπτεία όλων των συνδεδεμένων συσκευών σε πραγματικό χρόνο. Κάθε στιγμή στην οθόνη μπορεί να εμφανίζεται η τρέχουσα σύνοδος εποπτείας για μία συσκευή, χωρίς αυτό να σημαίνει ότι οι σύνοδοι εποπτείας των υπόλοιπων σταματούν να εκτελούνται παράλληλα.

Ο χρήστης έχει τη δυνατότητα να μεταβαίνει από συσκευή σε συσκευή με οριζόντια κύλιση της οθόνης. Οι υποοθόνες στις οποίες πλοηγείται ο χρήστης έχουν την ίδια δομή και παρουσιάζουν την ίδια συμπεριφορά, είναι συνεπώς εύκολο να οριστούν ως fragments και να προσαρτηθούν σε ένα αντικείμενο τύπου `ViewPager`,

μίας ειδικής κλάσης περιέκτη που παρέχεται από το Android framework και επιτρέπει την εναλλαγή σελίδων στην οθόνη με κινήσεις κύλισης.

Για τις υποοθόνες ορίζουμε την κλάση `DeviceDashboardFragment` ως υποκλάση της `Fragment`. Το περιεχόμενό τους είναι το εξής:

(i) Στοιχεία ελέγχου στην γραμμή τίτλου

Τα ακόλουθα εικονίδια της γραμμής τίτλου επιτρέπουν τον έλεγχο της τρέχουσας συνόδου εποπτείας:

- ▶ ("*Start/Resume*") για εκκίνηση ή συνέχιση της συνόδου,
- ■ ("*Stop/Pause*") για παύση καταγραφής ή διακοπή της συνόδου,
- ● ("*Record*") για έναρξη καταγραφής της συνόδου,
- 💾 ("*Save*") για αποθήκευση στην ΒΔ όλων των καταγεγραμμένων χρονοσειρών κατά την τρέχουσα σύνοδο.

(ii) Περιοχή απεικόνισης χρονοσειρών

Πρόκειται για έναν απλό περιέκτη τύπου `LinearLayout` με κατακόρυφη διάταξη. Για κάθε ενεργό αισθητήρα, στον περιέκτη προστίθεται ένα διάγραμμα του αντίστοιχου μετρώμενου μεγέθους, που ανανεώνεται σε πραγματικό χρόνο κατά τη διάρκεια της συνόδου εποπτείας.

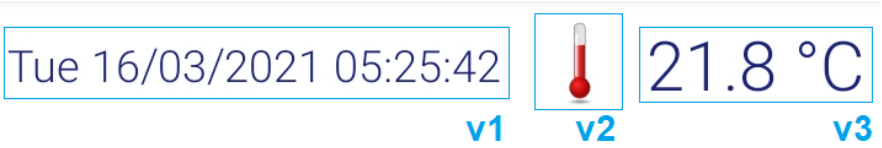
(iii) Μπάρα κατάστασης

Παρουσιάζει συνοπτικά την τρέχουσα κατάσταση ορθής λειτουργίας των εποπτευόμενων παραμέτρων, παρέχοντας αντίστοιχα μία ένδειξη ("*OK*"/"*Warning*"/"*Critical*") για κάθε παρακολουθητή, συνοδευόμενη από ένα αντιπροσωπευτικό εικονίδιο, όπως ορίζεται στην υποενότητα 7.3.7. Περιλαμβάνει επιπλέον το εικονίδιο ⧉ ("*Launch*") για μετάβαση στην οθόνη *DeviceControlActivity*.

7.2.1.5 LogViewerActivity

Ο χρήστης κατευθύνεται σε αυτή την οθόνη επιλέγοντας μία συσκευή από τη λίστα της *DeviceHistoryActivity* και πατώντας "*View events*" στο drop-down menu που εμφανίζεται, ή επιλέγοντας "*Events*"→"*View All*" από το UI της *DeviceControlActivity*.

Η οθόνη εμφανίζει στο χρήστη το καταγεγραμμένο ιστορικό συμβάντων για τη συσκευή σε μορφή λίστας (τύπου `RecyclerView`), πάντα για το χρονικό εύρος που έχει επιλεγεί από το περιβάλλον της προηγούμενης οθόνης. Εάν η συσκευή βρίσκεται σε σύνδεση, η λίστα ανανεώνεται σε πραγματικό χρόνο με την άφιξη νέων εγγραφών συμβάντων. Κάθε στοιχείο της προβάλλεται ως γραφικό στοιχείο της παρακάτω μορφής:



Εικόνα 7.9: Δείγμα στοιχείου λίστας συμβάντων

Κάθε στοιχείο της λίστας υλοποιείται ως στιγμιότυπο της κλάσης περιέκτη `ConstraintLayout`, και ενσωματώνει τα εξής αντικείμενα UI στο εσωτερικό του, όπως σημειώνονται και αριθμούνται στην παραπάνω εικόνα:

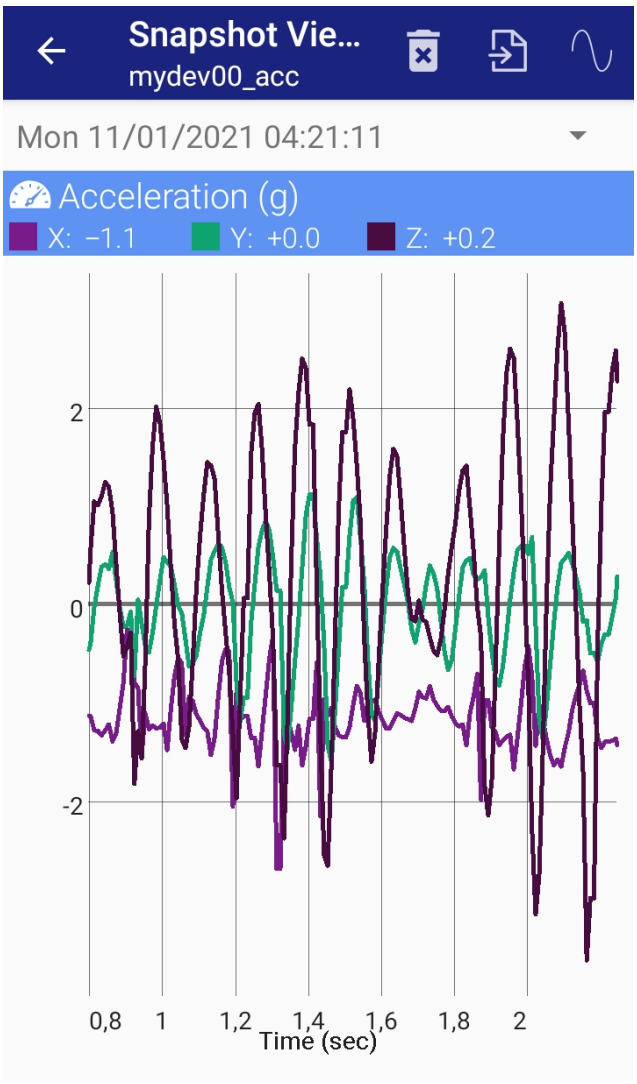
- [*v1*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο τη χρονοσφραγίδα του συμβάντος.
- [*v2*] Ένα εικονίδιο αντιπροσωπευτικό του συμβάντος, τύπου `ImageView`, όπως ορίζεται στην υποενότητα 7.3.6.
- [*v3*] Ένα πεδίο προβολής κειμένου, τύπου `TextView`, με περιεχόμενο την τιμή του παρακολουθητή προέλευσης του συμβάντος.

Η λίστα παρέχει επιπλέον τη δυνατότητα επιλογής ενός ή περισσότερων στοιχείων για την πραγματοποίηση ενεργειών πάνω σε αυτά. Η επιλογή του πρώτου στοιχείου γίνεται με παρατεταμένο πάτημα, και των υπόλοιπων στοιχείων με διαδοχικά απλά πατήματα. Ο χρήστης μπορεί κάθε στιγμή να ακυρώσει την τρέχουσα επιλογή πατώντας "←" στη γραμμή τίτλου, ή το πλήκτρο "*Home*". Όσο υπάρχει ενεργή επιλογή, στη γραμμή τίτλου εμφανίζεται ένα μενού αποτελούμενο από τα εξής πλήκτρα:

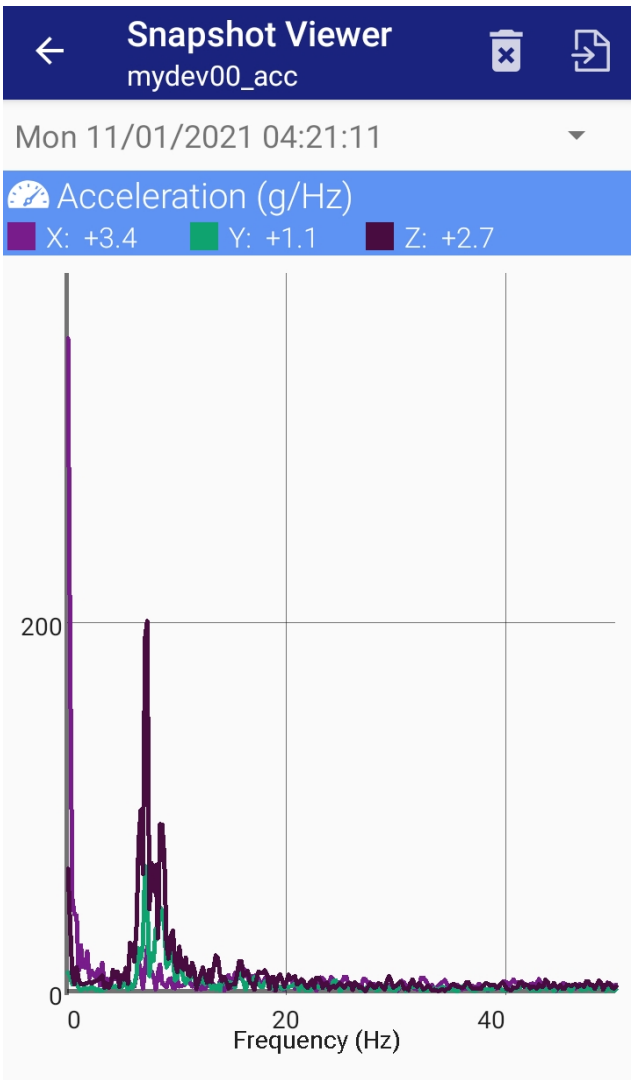
- *(Επιλογή όλων, "Select all")*: Επεκτείνει την τρέχουσα επιλογή στο σύνολο των στοιχείων της λίστας.
- *(Διαγραφή, "Delete")*: Διαγράφει τα επιλεγμένα στοιχεία από την ΒΔ της εφαρμογής.
- *(Επισήμανση ως αναγνωσμένο, "Mark as read")*: Επιτρέπει την επισήμανση των επιλεγμένων εγγραφών ως αναγνωσμένων, δίνοντας τη δυνατότητα στο χρήστη να επικεντρώσει την προσοχή του σε πρόσφατα συμβάντα, ή παλαιότερα συμβάντα για την αντιμετώπιση των οποίων εκκρεμεί η πραγματοποίηση κατάλληλων ενεργειών.

7.2.1.6 RecordingViewerActivity

Ο χρήστης κατευθύνεται σε αυτή την οθόνη επιλέγοντας μία συσκευή από τη λίστα της *DeviceHistoryActivity* και πατώντας "*View recordings*" στο drop-down menu που εμφανίζεται, ή επιλέγοντας "*Recorded Sessions*"→"*View All*" από το UI της *DeviceControlActivity*.



Εικόνα 7.10: Αποθηκευμένη κυματομορφή στο πεδίο του χρόνου



Εικόνα 7.11: Η ίδια κυματομορφή στο πεδίο της συχνότητας

Όπως φαίνεται στις παραπάνω εικόνες, η οθόνη παρέχει έναν επιλογέα σε μορφή αναπτυσσόμενης λίστας (τύπου `Spinner`), με στοιχεία τις χρονοσφραγίδες έναρξης όλων των αποθηκευμένων συνόδων εποπτείας για το χρονικό εύρος που έχει οριστεί από το UI της προηγούμενης οθόνης. Με την επιλογή ενός στοιχείου από τη λίστα, φορτώνονται από την ΒΔ και εμφανίζονται στην περιοχή της οθόνης αμέσως κάτω από τον επιλογέα οι χρονοσειρές που καταγράφηκαν κατά τη διάρκεια της αντίστοιχης συνόδου. Για την προβολή των χρονοσειρών σε διαγράμματα, όπως και στην περίπτωση της εποπτείας σε πραγματικό χρόνο, χρησιμοποιούνται αντικείμενα τύπου `PlotFragment` (βλ. επόμενη υποενότητα).

Για την επιλεγμένη σύνοδο, εμφανίζονται επιπλέον εικονίδια στη γραμμή τίτλου που επιτρέπουν τις εξής ενέργειες:

- *(Διαγραφή, "Delete"):* Διαγράφει τη σύνοδο από το ιστορικό, καθώς και όλες τις περιέχομενες χρονοσειρές.

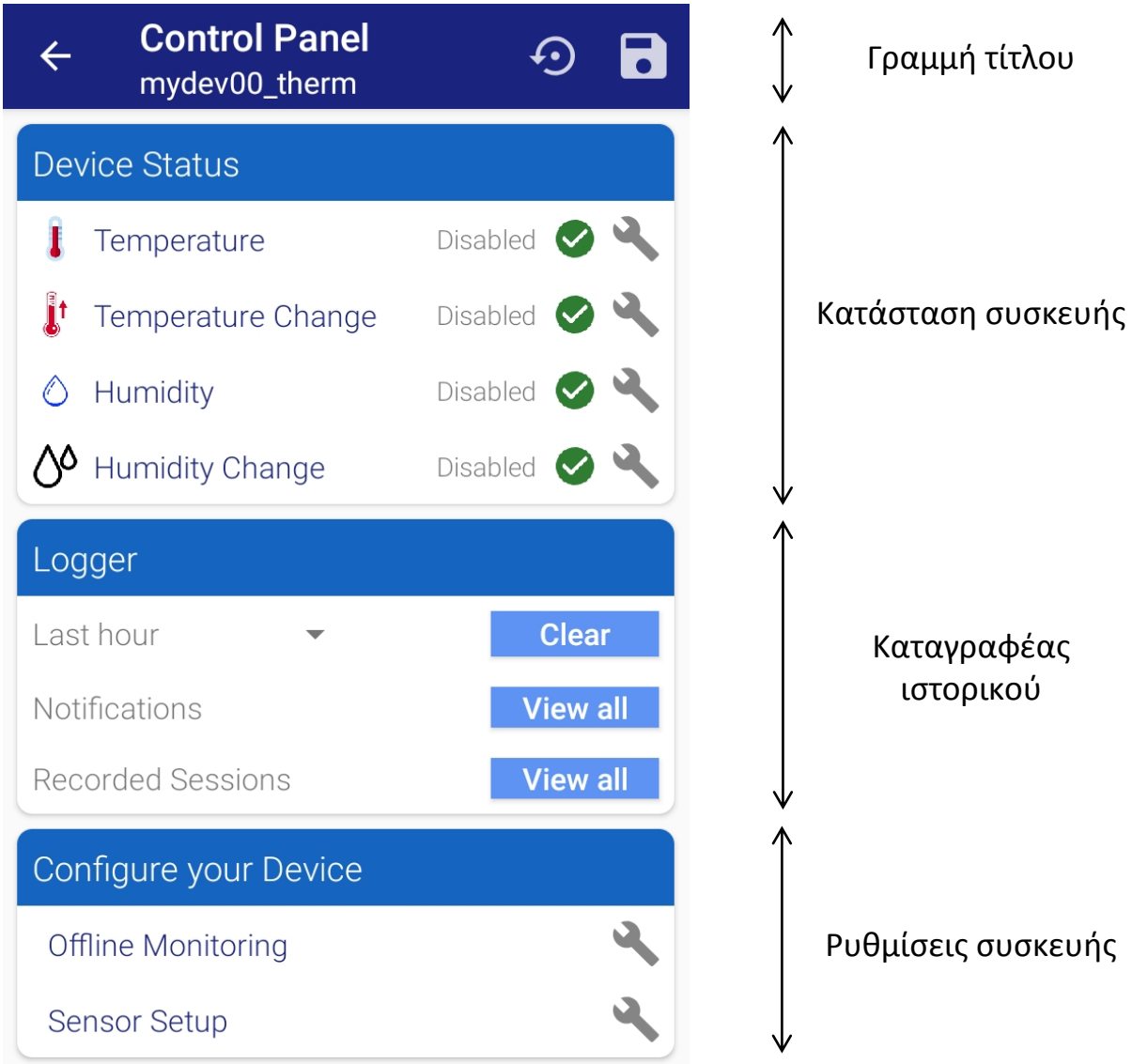
- *(Εξαγωγή σε αρχείο, "Export"):* Εμφανίζει ένα παράθυρο διαλόγου για την εξαγωγή των χρονοσειρών σε αρχεία CSV, ένα για κάθε διάγραμμα. Κάθε γραμμή του προκύπτοντος αρχείου έχει τη μορφή:

  ```
  t, v1, … , vN
  ```

  όπου t ο χρόνος σε δευτερόλεπτα, N η διάσταση του φυσικού μεγέθους που απεικονίζεται στο διάγραμμα, και v_1, …, v_N οι τιμές των χρονοσειρών κατά τη χρονική στιγμή t.

- *(Ανάλυση στο πεδίο της συχνότητας, "Frequency analysis"):* Μετασχηματίζει το ορατό μέρος των χρονοσειρών στο πεδίο της συχνότητας, και αντίστοιχα ανανεώνει τα διαγράμματα.

7.2.1.7 DeviceControlActivity

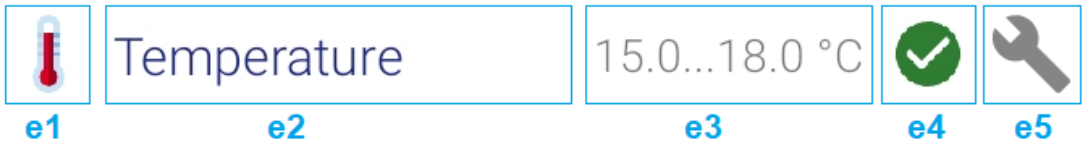


Εικόνα 7.12: Στιγμιότυπο της DeviceControlActivity για το ES Service

Ο χρήστης κατευθύνεται σε αυτή την οθόνη πατώντας το εικονίδιο ("*Launch*") από την μπάρα κατάστασης της *DeviceDashboardActivity*. Αποτελεί το κεντρικό σημείο από το οποίο ο χρήστης μπορεί να έχει πρόσβαση στο ιστορικό και τις ρυθμίσεις μίας συνδεδεμένης συσκευής. Το UI της οθόνης χωρίζεται σε τρεις ενότητες, οι οποίες εμφανίζονται στο χρήστη ως καρτέλες (τύπου `CardView`) η μία κάτω από την άλλη, όπως φαίνεται στην παραπάνω εικόνα:

(i) Κατάσταση συσκευής (Device Status)

Εμφανίζει μία λίστα που περιέχει το σύνολο των παρακολουθητών του CM Service που υλοποιείται από τη συσκευή. Κάθε καταχώρηση της λίστας υλοποιείται ως στιγμιότυπο της κλάσης `SettingsListView`, την οποία ορίζουμε κατάλληλα ως υποκλάση της `View` ακριβώς για το σκοπό αυτό, και περιλαμβάνει τα γραφικά αντικείμενα που σημειώνονται αριθμημένα και κυκλωμένα στην ακόλουθη εικόνα:

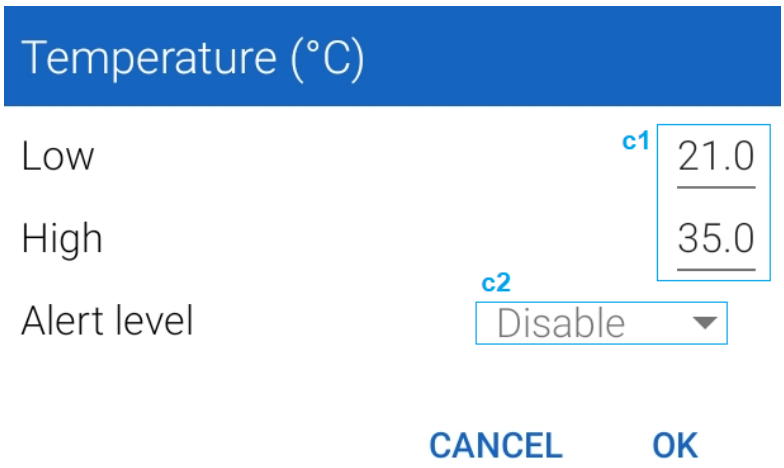


Εικόνα 7.13: Δείγμα στοιχείου λίστας παρακολουθητών

Παρουσιάζουμε καθένα από τα αντικείμενα αυτά ξεχωριστά:

- [*e1*] Ένα αντιπροσωπευτικό εικονίδιο τύπου `ImageView` για τον παρακολουθητή.
- [*e2*] Ένα πεδίο προβολής κειμένου τύπου `TextView` με περιεχόμενο τη φιλική ονομασία του παρακολουθητή.
- [*e3*] Ένα πεδίο προβολής κειμένου τύπου `TextView` με περιεχόμενο μία σύνοψη των ρυθμίσεων εποπτείας για τη συγκεκριμένη παράμετρο. Εάν η εποπτεία είναι ενεργοποιημένη, το στοιχείο αυτό εμφανίζει – με κατάλληλη μορφοποίηση – τα ορισμένα από το χρήστη όρια καλής λειτουργίας για τον παρακολουθητή, διαφορετικά την ένδειξη "*Disabled*".
- [e4] Ένα εικονίδιο τύπου `ImageView`, που αναπαριστά την τρέχουσα κατάσταση του παρακολουθητή (*OK*, *Warning* ή *Critical*).
- [e5] Ένα εικονίδιο τύπου `ImageView`, με το πάτημα του οποίου εμφανίζεται ένα παράθυρο διάλογου για τη ρύθμιση των παραμέτρων εποπτείας της παραμέτρου.

Το παράθυρο διαλόγου για την αλλαγή των ρυθμίσεων εποπτείας ενός παρακολουθητή έχει την ακόλουθη μορφή:



Εικόνα 7.14: Δείγμα διαλόγου ρύθμισης παρακολουθητή

Τα παράθυρα αυτά ενσωματώνονται σε fragments, και ειδικότερα σε αντικείμενα τύπου `android.app.DialogFragment`, τα οποία επεκτείνουμε ορίζοντας την κλάση `dialog.InputDialogFragment`. Η ρύθμιση των παραμέτρων γίνεται από τα παρακάτω UI controls του παραθύρου, τα οποία τοποθετούνται δυναμικά - με κατακόρυφη διάταξη - σε έναν περιέκτη τύπου `LinearLayout`:

- [*c1*] Ένα ή περισσότερα πεδία εισαγωγής κειμένου τύπου `EditText` για τη ρύθμιση των τιμών ελέγχου (ορίων ορθής λειτουργίας).
- [*c2*] Μία αναπτυσσόμενη λίστα τύπου `Spinner` για την επιλογή του επιπέδου ειδοποίησης ("*Disable*" για *ALERT_NONE*, "*Warning*" για *ALERT_WARNING*, ή "*Critical*" για *ALERT_CRITICAL*).

Τα παράθυρα αρχικοποιούνται με τις τρέχουσες τιμές των ρυθμίσεων, και ανταποκρίνονται στην είσοδο του χρήστη διαμέσου ενός αντικειμένου που υλοποιεί διαπροσωπεία `InputDialogFragment.InputListener`. Αυτή παρέχει το εξής API:

`onInitialize(View v)`	
Παράμετροι	`View v`: Το παράθυρο-περιέκτης.
Περιγραφή	Αρχικοποιεί τα πεδία εισαγωγής των ορίων ορθής λειτουργίας με τις τρέχουσες τιμές.
`onInputComplete(View v)`	
Παράμετροι	`View v`: Το παράθυρο-περιέκτης.
Περιγραφή	Ενημερώνει την καταχώρηση της λίστας για τον επιλεγέντα παρακολουθητή με τις τιμές που δόθηκαν από το χρήστη.

Πίνακας 7.15: API της `InputDialogFragment.InputListener`

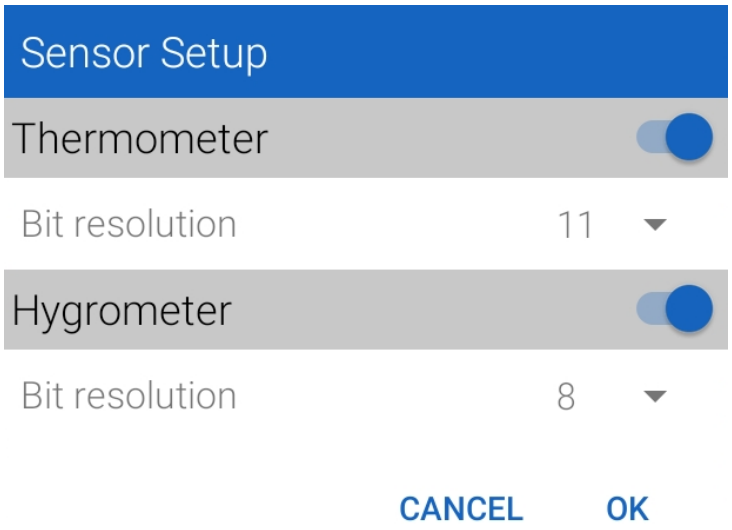
(ii) Καταγραφέας ιστορικού (Logger)

Επιτρέπει στο χρήστη την προβολή του ιστορικού του συνόλου των συμβάντων ("*Notifications*") ή των καταγεγραμμένων χρονοσειρών ("*Recorded Sessions*") με το πάτημα του κουμπιού "*View all*" ακριβώς δίπλα από την αντίστοιχη ένδειξη. Το χρονικό εύρος αναζήτησης (τελευταία ώρα/ημέρα/μήνας/έτος, ή από την αρχή) ορίζεται από την αναπτυσσόμενη λίστα που βρίσκεται στην κορυφή της καρτέλας.

(iii) Ρυθμίσεις συσκευής (Device Configuration)

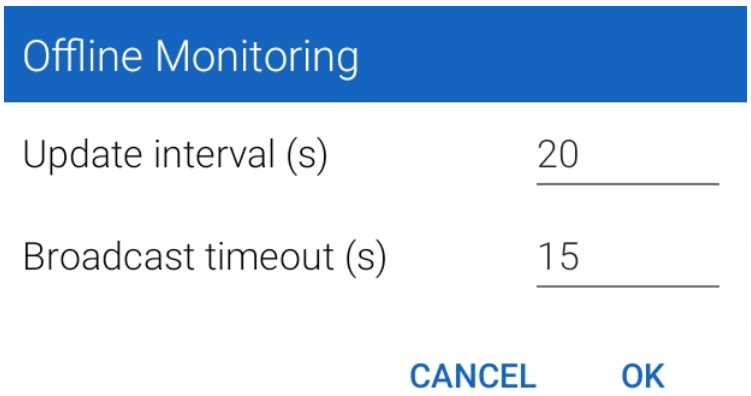
Επιτρέπει στο χρήστη τη ρύθμιση των αισθητήρων και της εκτός σύνδεσης εποπτείας, εμφανίζοντας αντίστοιχα δύο στοιχεία τύπου `SettingsListView` με τις εξής ενδείξεις:

- *Ρύθμιση αισθητήρων* ("*Sensor Setup*"): με πάτημα στο εικονίδιο 🔧, εμφανίζεται ένα παράθυρο διαλόγου που για κάθε αισθητήρα περιλαμβάνει έναν διακόπτη ενεργοποίησης/απενεργοποίησης και μία ή περισσότερες αναπτυσσόμενες λίστες, μία για κάθε στοιχείο του αντίστοιχου SCS:



Εικόνα 7.16: Διάλογος ρύθμισης αισθητήρων για το ES Service

- *Εποπτεία χωρίς σύνδεση* ("*Offline Monitoring*"): με πάτημα στο εικονίδιο 🔧, εμφανίζεται ένα παράθυρο διαλόγου που περιλαμβάνει δύο πεδία εισαγωγής κειμένου τύπου `EditText`, για τον ορισμό αντίστοιχα της περιόδου ενημέρωσης και της διάρκειας εκπομπής.



Εικόνα 7.17: Ρυθμίσεις εποπτείας χωρίς σύνδεση

7.2.2 Παραγωγή και προβολή διαγραμμάτων

Για την παραγωγή στοιχείων UI κατάλληλων για απεικόνιση διαγραμμάτων, απαιτείται είτε να ενσωματώσουμε στο σύστημα μία υπάρχουσα βιβλιοθήκη που ορίζει προγραμματιστικά και σχεδιάζει τέτοια στοιχεία, είτε να κατασκευάσουμε εμείς μία βιβλιοθήκη για αυτό το σκοπό. Για τη σχεδίαση σημείων, γραμμών και στοιχειωδών σχημάτων, κάθε τέτοιο εργαλείο αναμένεται να χρησιμοποιεί χαμηλού επιπέδου βιβλιοθήκες του Android framework ή της OpenGL ES.

Η διαθεσιμότητα ικανοποιητικού πλήθους ανοικτού κώδικα βιβλιοθηκών σχεδίασης διαγραμμάτων, με δυνατότητες επαρκείς για τις απαιτήσεις της εφαρμογής μας, οδηγεί στην πρώτη επιλογή. Στο σύστημά μας ενσωματώνουμε τη βιβλιοθήκη σχεδίασης *GraphView* (github.com/jjoe64/graphview), η οποία αναλαμβάνει τη σχεδίαση διαγραμμάτων στο εσωτερικό στοιχείων UI που αποτελούν μέλη της ομώνυμης κλάσης.

Θα ήταν όμως επιθυμητό - και αναμφίβολα πιο αποδοτικό - η διαδικασία παραγωγής των διαγραμμάτων να είναι ανεξάρτητη από τη βιβλιοθήκη σχεδίασης που χρησιμοποιούμε ως βάση, ώστε η τελευταία να μπορεί εύκολα να αντικατασταθεί από κάποια άλλη που προσφέρει παρόμοια ή βελτιωμένη λειτουργικότητα. Για το λόγο αυτό, όπως και συνολικά για λόγους ευελιξίας και αυστηρότερης δόμησης του κώδικα, παρεμβάλλουμε μία διεπαφή (interface) μεταξύ της βιβλιοθήκης σχεδίασης και των υπόλοιπων components της εφαρμογής. Αυτή αποτελείται από τα εξής:

(i) Για την *κατασκευή* των διαγραμμάτων:

Ένα επίπεδο αφαίρεσης μεταξύ της βιβλιοθήκης σχεδίασης με τα στοιχεία UI που ορίζει, από τη μία πλευρά, και του διαγράμματος – όπως αυτό είναι ορατό ως αντικείμενο στα υπόλοιπα components της εφαρμογής – μαζί με τα δεδομένα (χρονοσειρές) που περιέχει, από την άλλη πλευρά. Προσφέρει περισσότερη ευκολία στην παραγωγή διαγραμμάτων και επιτρέπει την παραμετροποίηση της διαδικασίας, ελαχιστοποιώντας την επαναχρησιμοποίηση κώδικα. Περιλαμβάνει τις παρακάτω κλάσεις, τις οποίες ορίζουμε ως μέλη του πακέτου `com.android.blesense.plot`:

- `Plot`: Αντιπροσωπεύει ένα διάγραμμα σε λογικό επίπεδο, με δυνατότητα ενημέρωσης των περιεχομένων του σε πραγματικό χρόνο, καθώς και προβολής συναρτήσεων στο πεδίο του χρόνου ή της συχνότητας. Η επικοινωνία με τον κώδικα της εφαρμογής γίνεται μέσω του ακόλουθου API:

`Plot(View v, int resId, PlotConfig config)`	
Παράμετροι	`View v`: Το στοιχείο-περιέκτης του διαγράμματος. `int resId`: Resource ID του στοιχείου UI του διαγράμματος. `PlotConfig config`: Προτιμήσεις κατασκευής για το διάγραμμα.
Περιγραφή	Παραγωγή ενός νέου διαγράμματος με τις προτιμήσεις που περιγράφονται στο `config`, και αντιστοίχισή του στο στοιχείο UI εντός του περιέκτη `v` με αναγνωριστικό `resId`.
`void updateData(PlotData[] data)`	
Παράμετροι	`PlotData[] data`: διάνυσμα από ζεύγη $(t, x_i(t))$ για κάθε κυματομορφή x_i κατά τη χρονική στιγμή t.
Περιγραφή	Προσθήκη του διανύσματος `data` στο διάγραμμα και προσαρμογή των ορίων του κατακόρυφου άξονα. Κάθε στοιχείο του διανύσματος αποτελεί και σημείο μίας κυματομορφής.
`List<List<PlotData>> getData()`	
Περιγραφή	Επιστροφή των περιεχόμενων στο διάγραμμα κυματομορφών σε μορφή λίστας από ακολουθίες σημείων τύπου `PlotData`.
`void setOnClickListener(Plot.OnClickListener listener)`	
Παράμετροι	`Plot.OnClickListener listener`: Ακροατής (*listener*) για συμβάντα πατήματος πάνω σε σημεία του διαγράμματος. Περιλαμβάνει τη μέθοδο επανάκλησης `onClick(PlotData[] data)`.
Περιγραφή	Προσθήκη ενός listener για απόκριση σε πατήματα πάνω σε σημεία του διαγράμματος.

Πίνακας 7.18: API της κλάσης `ui.Plot`

Για την εξειδίκευση της συμπεριφοράς ενός διαγράμματος, αρκεί να ορισθεί μία κατάλληλη υποκλάση. Για τις ανάγκες των δύο υπηρεσιών CM που υλοποιήθηκαν στην παρούσα εργασία, κατασκευάζουμε τις γενικού σκοπού υποκλάσεις `MultiValuePlot` και `StatsPlot`. Η πρώτη χρησιμοποιείται για γραφικές παραστάσεις διανυσματικών μεγεθών, και απεικονίζει στο διάγραμμα χρονοσειρές ισάριθμες της διάστασης του μετρώμενου μεγέθους, τυπώνοντας στη γραμμή τίτλου τις τιμές των συνιστωσών του διανύσματος. Η δεύτερη χρησιμοποιείται για γραφικές παραστάσεις βαθμωτών μεγεθών, και απεικονίζει στο

διάγραμμα μία μόνο χρονοσειρά, τυπώνοντας στη γραμμή τίτλου την ελάχιστη, τη μέση και τη μέγιστη τιμή του μεγέθους για την τρέχουσα σύνοδο.

- `PlotConfig`: Αποτελείται από τα παρακάτω ιδιωτικά πεδία:

Πεδίο	**Περιγραφή**
`int icon`	Αναγνωριστικό (Resource ID) του εικονιδίου που εμφανίζεται στη γραμμή τίτλου του διαγράμματος.
`String xAxisLabel`	Τίτλος του οριζόντιου άξονα.
`String yAxisLabel`	Τίτλος του κατακόρυφου άξονα.
`int numSeries`	Πλήθος των ενεργών και έτοιμων προς εμφάνιση χρονοσειρών που περιέχονται στο διάγραμμα.
`int[] seriesColor`	Κωδικοί χρωμάτων για τις χρονοσειρές.
`String[] seriesLabel`	Για ένα διανυσματικό μέγεθος, περιέχει τις περιγραφές (φιλικές ονομασίες) των συνιστωσών.
`int maxDataPoints`	Μέγιστη χωρητικότητα (σε σημεία) για το εκάστοτε ορατό τμήμα του διαγράμματος.
`double minXValue`	Ελάχιστη τιμή του οριζόντιου άξονα.
`double maxXValue`	Μέγιστη τιμή του οριζόντιου άξονα.
`double minYValue`	Ελάχιστη τιμή του κατακόρυφου άξονα.
`double maxYValue`	Μέγιστη τιμή του κατακόρυφου άξονα.
`int mode`	Ορίζει τον τρόπο εισαγωγής και εμφάνισης των δεδομένων, λαμβάνοντας μία από τις παρακάτω τιμές: • *PLOT_MODE_LIVE* (0): Το διάγραμμα ενημερώνεται σε πραγματικό χρόνο. Κατά συνέπεια, μόνο τα δεδομένα του εκάστοτε ορατού του τμήματος διατηρούνται στη μνήμη. • *PLOT_MODE_STATIC* (1): Τα δεδομένα φορτώνονται κατά την κατασκευή του διαγράμματος, και το σύνολό τους διατηρείται στη μνήμη. Δεν είναι δυνατή η προσθήκη νέων δεδομένων στη συνέχεια.
`boolean displayStats`	Επιτρέπει την εμφάνιση βασικών δεικτών για το μέγεθος που απεικονίζεται (μέγιστη, ελάχιστη και μέση τιμή). Τίθεται στην τιμή `false` εάν το μέγεθος είναι διανυσματικό.
`boolean scrollable`	Επιτρέπει την οριζόντια ολίσθηση του ορατού τμήματος του διαγράμματος με την κατάλληλη κίνηση του χεριού.
`boolean zoomable`	Επιτρέπει την οριζόντια μεγέθυνση/σμίκρυνση του διαγράμματος με την κατάλληλη κίνηση του χεριού.

Πίνακας 7.19: Η κλάση `ui.PlotConfig`

Η `PlotConfig` ορίζει το παραπάνω σύνολο προτιμήσεων για την προσαρμογή των παραμέτρων κατασκευής ενός διαγράμματος τύπου `Plot`. Οι τιμές όλων των πεδίων είναι προσβάσιμες μέσω μεθόδων *getter*/*setter* του API.

- `PlotFactory`: Παρέχει την factory method `Plot createPlot(View v, int resId, int plotType, Params params)`, η οποία κατασκευάζει ένα αντικείμενο `config` τύπου `PlotConfig` με τις προεπιλεγμένες προτιμήσεις για τον τύπο `plotType` και το σύνολο παραμέτρων `params`, και στη συνέχεια καλεί τον κατασκευαστή `Plot(v, resId, config)` για την παραγωγή του διαγράμματος.

(ii) Για την *προβολή* των διαγραμμάτων:

Έναν επαναχρησιμοποιήσιμο περιέκτη για τα διαγράμματα τύπου `Plot`, με δυνατότητα δυναμικής προσθαφαίρεσης από το UI. Η απαίτηση αυτή εύκολα ικανοποιείται με τη χρήση fragments, συνεπώς για το σκοπό αυτό ορίζουμε την κλάση `PlotFragment:Fragment`.

Παρακάτω δίνουμε το API της:

`static PlotFragment newInstance(DeviceWrapper device, int layoutResId, int plotResId, int dataType, PlotFactory.Params params)`	
Παράμετροι	`DeviceWrapper device`: Η συσκευή στην οποία αναφέρεται η τρέχουσα σύνοδος εποπτείας (βλ. υποενότητα 7.3.2). `int layoutResId`: Το αναγνωριστικό (Resouce ID) του layout του διαγράμματος. `int plotResId`: Το αναγνωριστικό του περιέκτη. `int dataType`: Ο τύπος των δεδομένων που φιλοξενούνται στο διάγραμμα. Αντιστοιχεί στο UUID του χαρακτηριστικού για το απεικονιζόμενο μέγεθος. `PlotFactory.Params params`: Παράμετροι κατασκευής του διαγράμματος.
Περιγραφή	Factory method που είναι ορατή αντί του κατασκευαστή της κλάσης. Με την προσάρτηση του fragment στο UI, δηλαδή κατά την κλήση της `Fragment.onCreateView()`, ο περιέκτης συμπληρώνεται με το layout που δόθηκε από το χρήστη, και στη συνέχεια συνδέεται με ένα νέο διάγραμμα τύπου `Plot` με χρήση της `PlotFactory.createPlot()`.
`Plot getPlot()`	
Περιγραφή	Επιστρέφει το περιεχόμενο διάγραμμα τύπου `Plot`.

`void updateData(double timestamp, double[] data)`	
Παράμετροι	`double timestamp`: Χρονοσφραγίδα λήψης των δεδομένων. `double[] data`: Οι τιμές των χρονοσειρών.
Περιγραφή	Για κάθε στοιχείο *value* του διανύσματος `data`, κατασκευάζει ζεύγος (*timestamp*, *value*) τύπου `PlotData`. Με αυτά τα ζεύγη τιμών ενημερώνει το περιεχόμενο διάγραμμα καλώντας `Plot.updateData()`.

Πίνακας 7.20: API της κλάσης `PlotFragment`

- Μία διαπροσωπεία `PlotAdapter`, κάθε υλοποίηση της οποίας προσαρμόζει τις κλήσεις πάνω σε αντικείμενα `Plot` σε κατάλληλες κλήσεις του API της βιβλιοθήκης σχεδίασης. Η υλοποίηση της `PlotAdapter` αποτελεί το μοναδικό στοιχείο που εξαρτάται από την βιβλιοθήκη σχεδίασης, συνεπώς είναι και το μοναδικό στοιχείο που απαιτείται για τη χρήση οποιασδήποτε συμβατής βιβλιοθήκης. Εν προκειμένω, για την βιβλιοθήκη *GraphView* ορίζουμε την υλοποίηση `PlotToGraphViewAdapter`.

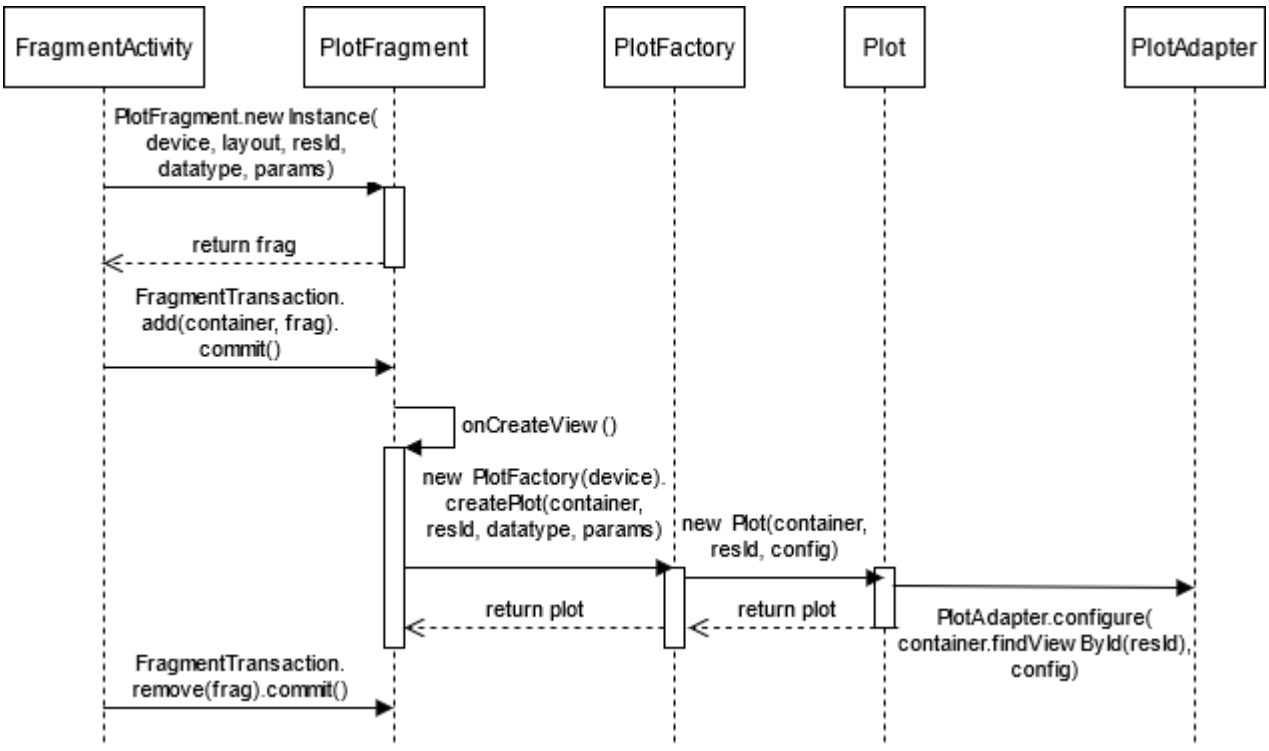
  Ακολουθεί το API της `PlotAdapter`:

`void configure(View v, PlotConfig config)`	
Παράμετροι	`View v`: Το στοιχείο-περιέκτης του διαγράμματος. `PlotConfig config`: Προτιμήσεις κατασκευής.
Περιγραφή	Καλείται από τον κατασκευαστή της `Plot`, πραγματοποιώντας με τη σειρά της όλες τις απαραίτητες για την κατασκευή του διαγράμματος κλήσεις στο API της βιβλιοθήκης σχεδίασης.
`void setVisible(boolean visible)`	
Παράμετροι	`boolean visible`: Τιμή αληθείας για την ορατότητα του διαγράμματος.
Περιγραφή	Εμφανίζει ή αποκρύπτει το γραφικό στοιχείο που αντιστοιχεί στο διάγραμμα.
`void updateData(PlotData[] data)`	
Παράμετροι	`PlotData[] data`: Διάνυσμα δεδομένων εισόδου.
Περιγραφή	Προσαρμογέας της `Plot.updateData()`.
`void clearData()`	
Περιγραφή	Προσαρμογέας της `Plot.clearData()`.

`void rescale()`	
Περιγραφή	Καλείται από την `Plot.updateData()` για την αναπροσαρμογή των ορίων του κατακόρυφου άξονα.
`List<List<PlotData>> getData()`	
Περιγραφή	Προσαρμογέας της `Plot.getData()`.
`void setOnClickListener(Plot.OnClickListener listener)`	
Παράμετροι	`Plot.OnClickListener listener`: Listener για απόκριση σε πατήματα πάνω σε σημεία του διαγράμματος
Περιγραφή	Προσαρμογέας της `Plot.setOnClickListener()`.

Πίνακας 7.21: API της διαπροσωπείας `PlotAdapter`

Οι διαδικασίες παραγωγής και προβολής ενός διαγράμματος σε μία activity, καθώς και αφαίρεσης και καταστροφής του, με τη βοήθεια των παραπάνω κλάσεων παρουσιάζονται συνοπτικά στο διάγραμμα που ακολουθεί:



Σχήμα 7.22: Προσθαφαίρεση και προβολή διαγραμμάτων

7.3 Υλοποίηση λειτουργιών

Στις υποενότητες που ακολουθούν παρουσιάζουμε τον τρόπο υλοποίησης βασικών λειτουργιών της εφαρμογής, καθώς και την τεκμηρίωση του σχετικού κώδικα.

7.3.1 Αναζήτηση για συσκευές

Η λίστα διαθέσιμων συσκευών συντίθεται από τις συνδεδεμένες συσκευές, και αυτές που πραγματοποιούν εκπομπή (advertising). Ο εντοπισμός των τελευταίων γίνεται με σάρωση BLE (scanning). Η σάρωση είναι μία διαδικασία που οφείλει να τρέχει στο παρασκήνιο όσο την χρειαζόμαστε. Επιπλέον επιδιώκουμε να παρέχουμε στο χρήστη την επιλογή η σάρωση να συνεχίζεται και μετά το κλείσιμο του UI, ώστε έγκαιρα να λαμβάνονται κατάλληλες ειδοποιήσεις σε περίπτωση εντοπισμού συσκευής με προβληματική κατάσταση λειτουργίας. Ο ενδεδειγμένος λοιπόν τρόπος υλοποίησης της διαδικασίας είναι με χρήση μίας υπηρεσίας, η οποία θα αξιοποιείται από την οθόνη *DeviceScanActivity* και θα μπορεί να παραμένει ενεργή χωρίς UI.

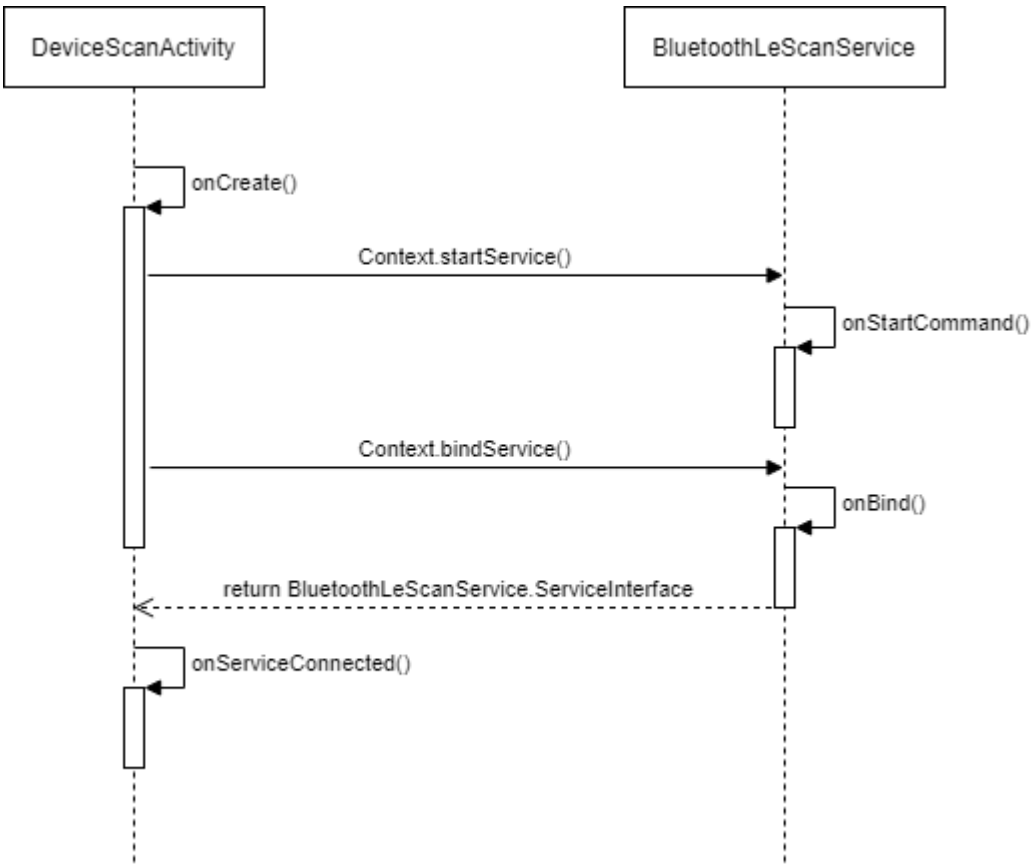
Ορίζουμε για αυτό το σκοπό την υπηρεσία *BluetoothLeScanService*, την οποία χρησιμοποιούμε ως εξής:

1. Κατά την έναρξη της εφαρμογής, συγκεκριμένα με την είσοδο στην *DeviceScanActivity*, η υπηρεσία *εκκινείται* με την κλήση `Context.startService()`. Με αυτό τον τρόπο παραμένει διαθέσιμη για εκτέλεση και μετά το κλείσιμο του UI. Η διαδικασία σάρωσης όμως δεν ξεκινά από αυτό το σημείο, αλλά ελέγχεται από τα components που κάνουν χρήση της υπηρεσίας.

Από την έκδοση 8.0 και μετά, το Android framework επιβάλλει περιορισμούς στην εκτέλεση εκείνων των υπηρεσιών οι οποίες εκκινούνται στο παρασκήνιο χωρίς τη συγκατάθεση του χρήστη. Για συμμόρφωση με τους περιορισμούς των εκδόσεων αυτών, κατά την εκκίνηση της υπηρεσίας μας αναρτάται ειδικό μήνυμα στην περιοχή ειδοποιήσεων του συστήματος, το οποίο διατηρείται ορατό όσο η υπηρεσία παραμένει ενεργή:

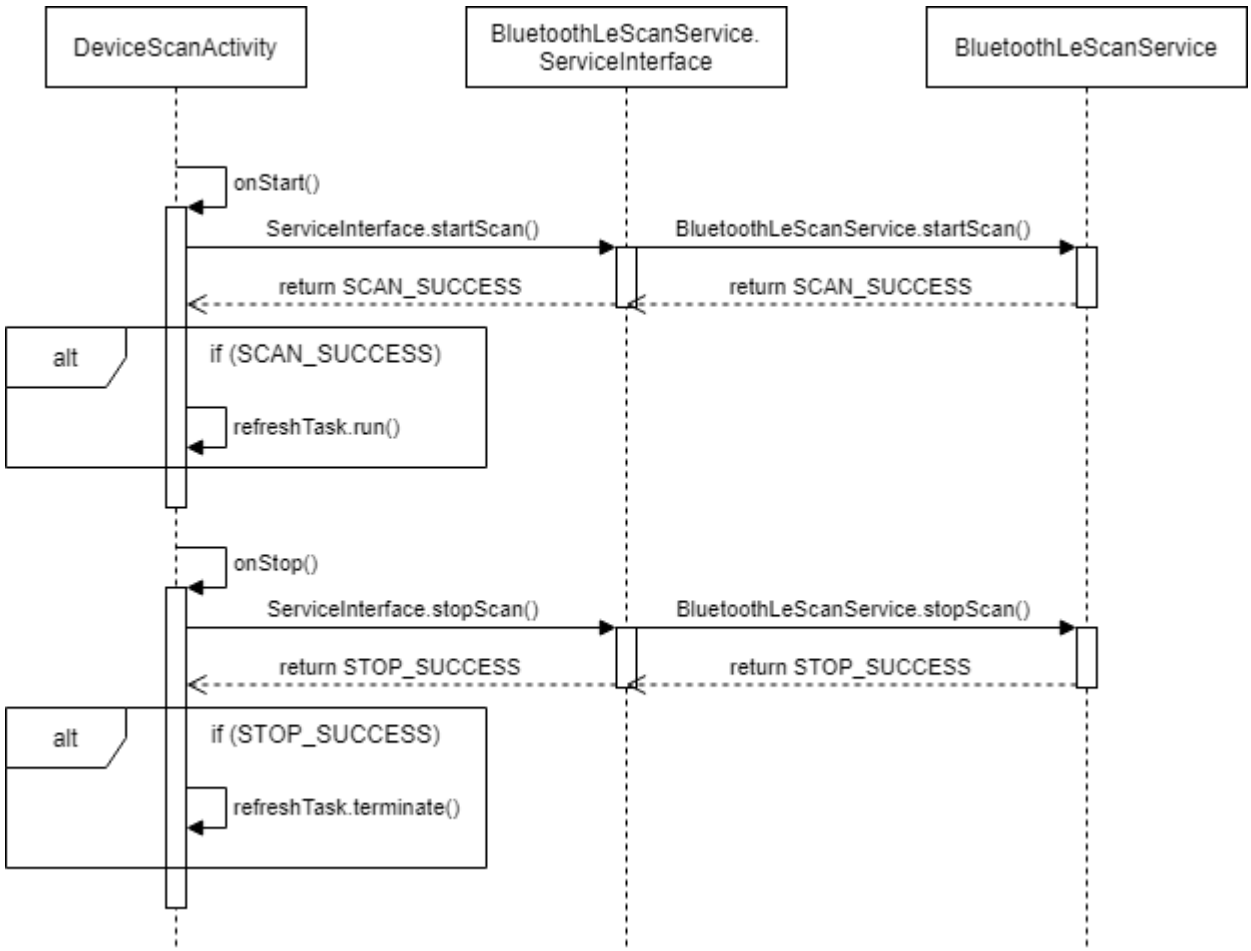
Εικόνα 7.23: Μήνυμα στην περιοχή ειδοποιήσεων

2. Η *DeviceScanActivity* δεσμεύει την υπηρεσία με την κλήση `Context.bindService()`, αποκτώντας τον έλεγχο της διαδικασίας σάρωσης για το σύνολο της διάρκειας ζωής του UI. Όπως φαίνεται στα αποσπάσματα που ακολουθούν, η κλήση δέσμευσης επιστρέφει ένα αντικείμενο τύπου `BluetoothLeScanService.ServiceInterface:IBinder`, το οποίο εκθέτει κατάλληλες δημόσιες (public) μεθόδους προς κλήση. Τα παραπάνω συνοψίζονται στο ακόλουθο διάγραμμα:



Σχήμα 7.24: Χρήση της υπηρεσίας BluetoothLeScanService

Με τη μετάβαση της *DeviceScanActivity* στις καταστάσεις *Started* και *Stopped*, ο κώδικας της εφαρμογής αντίστοιχα ξεκινά ή διακόπτει τη σάρωση με τον τρόπο που φαίνεται παρακάτω:



Σχήμα 7.25: Έναρξη και διακοπή αναζήτησης

Ακολουθεί το API της `ServiceInterface`:

`boolean startScan()`	
Περιγραφή	Εκκινεί σάρωση για συσκευές καλώντας τη μέθοδο `BluetoothLeScanner.startScan()` του BLE API. Επιστρέφει `true` για επιτυχή έναρξη της διαδικασίας, διαφορετικά `false`.
`boolean stopScan()`	
Περιγραφή	Σταματά την τρέχουσα σάρωση καλώντας τη μέθοδο `BluetoothLeScanner.stopScan()` του BLE API. Επιστρέφει `true` σε περίπτωση επιτυχίας, διαφορετικά `false`.
`void configure(ScanCallback callback)`	
Παράμετροι	`ScanCallback callback`: Περιέχει τη μέθοδο επανάκλησης `onScanResult()`, η οποία καλείται με λήψη κάθε νέου πακέτου διαφήμισης.
Περιγραφή	Ορίζει τη μέθοδο επανάκλησης `callback` η οποία θα περαστεί ως παράμετρος στις `BluetoothLeScanner.startScan()`, `BluetoothLeScanner.stopScan()`.

`void start()`	
Περιγραφή	Καλείται μετά τη δέσμευση της υπηρεσίας. Διακόπτει τυχόν σάρωση που εκτελείται στο παρασκήνιο, και ξεκινά εκ νέου σάρωση.
`void stop()`	
Περιγραφή	Καλείται μετά την αποδέσμευση της υπηρεσίας από όλους τους πελάτες της. Εάν είναι επιθυμητή η συνέχιση της σάρωσης στο παρασκήνιο, η υπηρεσία διατηρείται στην κατάσταση *Started* και ξεκινά εκ νέου σάρωση, διαφορετικά τερματίζει τον εαυτό της με την κλήση `Service.stopSelf()`.

Πίνακας 7.26: API της `BluetoothLeScanService.ServiceInterface`

3. Με τον τερματισμό της *DeviceScanActivity* και τη διαγραφή της από τη μνήμη, η υπηρεσία αποδεσμεύεται με την κλήση `Context.unbindService()`. Εδώ πρέπει να σημειώσουμε πως η *DeviceScanActivity* αποτελεί τη ριζική οθόνη της εφαρμογής, κατά συνέπεια ο τερματισμός της σηματοδοτεί το κλείσιμο συνολικά του UI. Επειδή όμως η *DeviceScanActivity* είναι ταυτόχρονα ο μοναδικός πελάτης της υπηρεσίας, εκείνη ακριβώς τη στιγμή καλείται η `BluetoothLeScanService.onUnbind()`. Σε αυτό το σημείο, με την κλήση `ServiceInterface.stop()` γίνεται η επιλογή για την παραμονή της υπηρεσίας σε ενεργή κατάσταση ή όχι, ανάλογα με την τιμή που έχει ορίσει ο χρήστης για την αντίστοιχη ρύθμιση της υποενότητας 7.3.8. Στην πρώτη περίπτωση, η υπηρεσία ξεκινά σάρωση στο παρασκήνιο και διατηρεί ορατό το μήνυμά της στην περιοχή ειδοποίησης του συστήματος. Διαφορετικά, καταστρέφεται πλήρως και αποδεσμεύεται το σύνολο της μνήμης που καταλαμβάνει η εφαρμογή.

Για το μενού της *DeviceScanActivity*, η λίστα των διαθέσιμων συσκευών υλοποιείται ως στιγμιότυπο της κλάσης `RecyclerView`, και συμπληρώνεται με στοιχεία τύπου `DeviceEntry` της παρακάτω μορφής:

Πεδίο	**Περιγραφή**
`String deviceName`	Το όνομα με το οποίο εμφανίζεται η συσκευή στη λίστα. Με το ίδιο όνομα αποθηκεύεται και η καταχώρηση της συσκευής στην ΒΔ.
`String deviceAddress`	Η δημόσια διεύθυνση BLE της συσκευής, η οποία χρησιμοποιείται για την εγκατάσταση σύνδεσης.
`String deviceType`	Ο τύπος της συσκευής: →"*THERMO*" για Temperature/Humidity Sensor →"*ACCEL*" για Accelerometer/Gyroscope Sensor

`String connectionState`	Ένδειξη κατάστασης σύνδεσης της συσκευής: →"*Connected*", αν η σύνδεση έχει ολοκληρωθεί →"*Disconnected*", αμέσως μετά την αποσύνδεση →"*Available*", όσο η συσκευή διαφημίζεται →"*Connecting*", όσο εξελίσσεται η εγκατάσταση της σύνδεσης
`int signalStrength`	Ένδειξη επιπέδου ισχύος λήψης: → : *0* (έως -90 dBm) → : *1* (-90…-70 dBm) → : *2* (-70…-50 dBm) → : *3* (-50…-30 dBm) → : *4* (πάνω από -30 dBm)
`int condition`	Ένδειξη κατάστασης λειτουργίας της συσκευής: → : *0* (*CONDITION_OK*) → : *1* (*CONDITION_WARNING*) → : *2* (*CONDITION_CRITICAL*)
`int batteryLevel`	Ένδειξη επιπέδου φόρτισης της μπαταρίας της συσκευής: → : *0* (0…10%) → : *1* (10…20%) → : *2* (20…30%) → : *3* (30…50%) → : *4* (50…60%) → : *5* (60…80%) → : *6* (80…90%) → : *7* (90…100%)
`int id`	Το αναγνωριστικό της συσκευής, το οποίο αποθηκεύεται στην ΒΔ και χρησιμοποιείται από τα υπόλοιπα στοιχεία της εφαρμογής. Προκύπτει με κλήση της `BluetoothDevice.hashCode()`.

Πίνακας 7.27: Μέλη της κλάσης `DeviceEntry`

Ο κώδικας της *DeviceScanActivity* ανανεώνει περιοδικά τη λίστα των διαθέσιμων συσκευών εκτελώντας με τη βοήθεια ενός `Handler` την εσωτερική διαδικασία *refreshTask* (τύπου `Runnable`), η οποία αποτελείται από την παρακάτω ακολουθία ενεργειών:

1. Λήψη του συνόλου αποτελεσμάτων *R* της σάρωσης από την τελευταία ανανέωση της λίστας.
2. Ανανέωση του σύνολου των διαθέσιμων συσκευών *D* ως εξής:
   $D \leftarrow D - A^c - R^c$, όπου *A* είναι το σύνολο των συσκευών σε κατάσταση σύνδεσης *Connected* ή *Connecting*.
3. Εκκαθάριση του συνόλου αποτελεσμάτων: $R \leftarrow \varnothing$.
4. Ενημέρωση της ένδειξης ισχύος λήψης (RSSI) για όλες τις συσκευές του *D*.
5. Προγραμματισμός της επόμενης ενημέρωσης βάσει της τιμής που έχει ορίσει ο χρήστης για την αντίστοιχη προτίμηση της υποενότητας 7.3.8.

Πάντα διατηρούμε τη λίστα ταξινομημένη σε φθίνουσα σειρά, εισάγοντας τα αντικείμενα `DeviceEntry` σε μία δομή τύπου `RecyclerView.SortedList`. Τα κριτήρια ταξινόμησης ορίζονται στη μέθοδο σύγκρισης `DeviceEntry.compareTo()`, και ακολουθούνται με τη σειρά που δίνεται παρακάτω:

1. Κατάσταση σύνδεσης, με προτεραιότητα
   Connected (3) > *Disconnected* (2) > *Connecting* (1) > *Available* (0).
2. Κατάσταση λειτουργίας, με προτεραιότητα *Critical* (2) > *Warning* (1) > *OK* (0).
3. Όνομα συσκευής, με αλφαβητική σειρά.

7.3.2 Διαχείριση συνδέσεων

Η εγκατάσταση σύνδεσης με μία διαθέσιμη συσκευή του δικτύου ακολουθεί την εξής διαδικασία:

- Σύνδεση με τον εξυπηρετητή GATT της συσκευής. Η συσκευή απαιτείται να διαθέτει δημόσια διεύθυνση για το σκοπό αυτό.
- Έλεγχος ταυτότητας και εξουσιοδότηση με αντιστοίχιση και σύζευξη (bonding), με χρήση της μεθόδου PKE (*Passkey Entry*).
- Ορισμός της μέγιστης ATT MTU στα 115 bytes.
- Εύρεση υπηρεσιών και χαρακτηριστικών (service discovery).
- Προαιρετικά, ενημέρωση της τοπικά διατηρούμενης στη συσκευή. ημερομηνίας/ώρας, με χρήση της εντολής *CP_UPDATE_RTC*.
- Λήψη του DIS με χρήση της εντολής *CP_DEVICE_GET_INFO*.
- Λήψη του DCS με χρήση της εντολής *CP_DEVICE_GET_CONFIG*.

Τα παραπάνω βήματα εκτελούνται σειριακά, συνεπώς η εγκατάσταση της σύνδεσης ολοκληρώνεται με τη λήψη του DCS. Στη συνέχεια, με αποστολή της εντολής *CP_RETRIEVE_LOG* λαμβάνεται το σύνολο των αποθηκευμένων στη συσκευή εγγραφών ιστορικού συμβάντων.

Η εφαρμογή διατηρεί τον κατάλογο των συνδεδεμένων συσκευών με τη μορφή μίας συλλογής αντικειμένων τύπου `DeviceWrapper`. Κάθε τέτοιο

αντικείμενο ενσωματώνει την απαιτούμενη πληροφορία για την αντίστοιχη συσκευή στα παρακάτω πεδία:

Πεδίο	**Περιγραφή**
`String name`	Το όνομα με το οποίο έχει καταχωρηθεί η συσκευή στην ΒΔ, όπως προκύπτει από το αντίστοιχο πεδίο του αντικειμένου `DeviceEntry`.
`String address`	Η διεύθυνση της συσκευής, όπως προκύπτει από το αντίστοιχο πεδίο του αντικειμένου `DeviceEntry`.
`String type`	Ο τύπος της συσκευής, όπως προκύπτει από το αντίστοιχο πεδίο του αντικειμένου `DeviceEntry`.
`UUID serviceUuid`	Το UUID της υπηρεσίας CM που υλοποιείται από τη συσκευή.
`DeviceInfo info`	Το DIS της συσκευής, στη λογική του αναπαράσταση (μετά την αποσειριοποίηση).
`DeviceConfig config`	Το DCS της συσκευής, στη λογική του αναπαράσταση (μετά την αποσειριοποίηση).
`Monitor.Status status`	Η τρέχουσα κατάσταση λειτουργίας της συσκευής.
`List<Monitor> monitors`	Λίστα των ενεργών παρακολουθητών κατά τη διάρκεια μίας συνόδου.
`int id`	Το αναγνωριστικό της συσκευής, όπως προκύπτει από το αντίστοιχο πεδίο του αντικειμένου `DeviceEntry`.

Πίνακας 7.28: Μέλη της κλάσης `DeviceWrapper`

Για τη διαχείριση των συνδέσεων της εφαρμογής με τις συσκευές-εξυπηρετητές και την επικοινωνία του κώδικα των υπόλοιπων components με το BLE API του Android framework, ορίζουμε την κλάση `connectivity.ConnectionManager`. Η κλάση αυτή ακολουθεί το πρότυπο σχεδίασης *singleton*, δηλαδή ανά πάσα στιγμή μπορεί να υπάρχει το πολύ ένα στιγμιότυπό της, το οποίο διατηρεί επιπλέον στο εσωτερικό του τον κατάλογο των συνδεδεμένων συσκευών με τη μορφή ενός πίνακα αντιστοίχισης αναγνωριστικών σε αντικείμενα `DeviceWrapper` (τύπου `Map<Integer, DeviceWrapper>`).

Η κλάση `ConnectionManager` παρέχει ως API τις παρακάτω μεθόδους:

`static ConnectionManager getInstance()`	
Περιγραφή	Επιστρέφει το τρέχον στιγμιότυπο της κλάσης, διαφορετικά ένα νέο αντικείμενο. Στη δεύτερη περίπτωση, αρχικοποιεί τον κατάλογο των συνδεδεμένων συσκευών.

`void open(Context context)`	
Παράμετροι	`Context context`: Το περιβάλλον του καλούντος component.
Περιγραφή	Ενεργοποίηση του διαχειριστή συνδέσεων. Δεσμεύει την υπηρεσία *BluetoothLeGattService*.
`void close(Context context)`	
Παράμετροι	`Context context`: Το περιβάλλον του καλούντος component.
Περιγραφή	Τερματίζει όλες τις ενεργές συνδέσεις, αποδεσμεύει την υπηρεσία *BluetoothLeGattService* και απελευθερώνει όλους τους σχετικούς πόρους (κατάλογος συσκευών κλπ). Έπειτα καταστρέφει το τρέχον στιγμιότυπο της κλάσης.
`void connectDevice(DeviceWrapper device)`	
Παράμετροι	`DeviceWrapper device`: Η συσκευή προς σύνδεση.
Περιγραφή	Επιχειρεί σύνδεση με την συσκευή `device`, και εισάγει την εγγραφή της στον κατάλογο των συσκευών.
`DeviceWrapper getDevice(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Επιστρέφει τη συσκευή που έχει καταχωρηθεί στον κατάλογο με αναγνωριστικό `deviceId`.
`void disconnectDevice(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Απολύει τη σύνδεση με τη συσκευή που είναι καταχωρημένη με αναγνωριστικό `deviceId`. Οι πόροι της σύνδεσης διατηρούνται διαθέσιμοι για χρήση σε ενδεχόμενο επανασύνδεσης.
`void releaseDevice(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Επιτελεί την ίδια λειτουργία με την `disconnectDevice()`, απελευθερώνοντας επιπλέον τους πόρους της σύνδεσης.

`BluetoothLeGattService.ServiceInterface getService()`	
Περιγραφή	Επιστρέφει μία διεπαφή τύπου `IBinder` προς την υπηρεσία *BluetoothLeGattService*, για την εκτέλεση των δημοσίων μεθόδων που αυτή παρέχει.
`Set<Integer> getDevices()`	
Περιγραφή	Επιστρέφει τα αναγνωριστικά όλων των καταχωρημένων συσκευών.
`void registerGuiCallback(GuiCallback cb)`	
Παράμετροι	`GuiCallback cb`: Το νέο GUI callback.
Περιγραφή	Ενεργοποίηση του GUI callback `cb` για την ενημέρωση του γραφικού περιβάλλοντος.
`void unregisterGuiCallback()`	
Περιγραφή	Απενεργοποίηση του τρέχοντος GUI callback.

Πίνακας 7.29: API της κλάσης `ConnectionManager`

Με την ενεργοποίησή του (κλήση της `ConnectionManager.open()`), ο διαχειριστής συνδέσεων εκκινεί τον αποδέκτη μηνυμάτων *GattUpdateReceiver*, ο οποίος με τη σειρά του διατηρείται ενεργός μέχρι το κλείσιμο του πρώτου (κλήση της `ConnectionManager.close()`). Ο *GattUpdateReceiver* λαμβάνει από το σύστημα Android και τα υπόλοιπα components της εφαρμογής τα ακόλουθα μηνύματα:

- ***ACTION_GATT_SERVICES_DISCOVERED***

Περιγραφή: Αποστέλλεται από την υπηρεσία *BluetoothLeGattService* με την ολοκλήρωση της διαδικασίας εύρεσης υπηρεσιών (service discovery).

Δεδομένα εισόδου: Το αναγνωριστικό της συσκευής με την οποία εγκαθίσταται σύνδεση.

Ενέργειες: Ολοκληρώνει τη διαδικασία εγκατάστασης σύνδεσης δίνοντας τις εντολές *CP_UPDATE_RTC* (προαιρετικά), *CP_GET_DEVICE_INFO* και *CP_GET_DEVICE_CONFIG*.

- ***ACTION_GATT_DISCONNECTED***

Περιγραφή: Αποστέλλεται από την υπηρεσία *BluetoothLeGattService* με την αποσύνδεση μίας συσκευής.

Δεδομένα εισόδου: Το αναγνωριστικό της συσκευής.

Ενέργειες: Εάν είναι αναγκαίο, ενημερώνεται το UI.

- ***ACTION_READ_RSSI***

Περιγραφή: Αποστέλλεται από την υπηρεσία *BluetoothLeGattService* με την ανάγνωση της τρέχουσας τιμής της ισχύος λήψης (RSSI) από μία συνδεδεμένη συσκευή.

Δεδομένα εισόδου: Το αναγνωριστικό της συσκευής.

Ενέργειες: Εάν είναι αναγκαίο, ενημερώνεται το UI.

- ***ACTION_DATA_AVAILABLE***

Περιγραφή: Αποστέλλεται από την υπηρεσία *BluetoothLeGattService* με τη λήψη ενός πακέτου απόκρισης σε εντολή, δηλαδή με την έλευση νέας ειδοποίησης τιμής στο χαρακτηριστικό *Data Out*.

Δεδομένα εισόδου: Το αναγνωριστικό της συσκευής, καθώς και το περιεχόμενο πακέτου.

Ενέργειες: Η απόφαση λαμβάνεται βάσει του τύπου του ληφθέντος πακέτου, ως εξής:

➔ Για πακέτο *CP_GET_DEVICE_INFO*, η εφαρμογή ενημερώνεται για το περιεχόμενο του DIS.

➔ Για πακέτο *CP_GET_DEVICE_CONFIG*, η εφαρμογή ενημερώνεται για το περιεχόμενο του DCS και στη συνέχεια ολοκληρώνει τη διαδικασία εγκατάστασης, ενημερώνοντας επιπλέον το UI αν αυτό είναι αναγκαίο.

➔ Για πακέτο *CP_RETRIEVE_LOG*, η εφαρμογή αποθηκεύει τη ληφθείσα εγγραφή ιστορικού στη ΒΔ και στη συνέχεια ενημερώνει τους ενεργούς παρακολουθητές, ενημερώνοντας επιπλέον το UI αν αυτό είναι αναγκαίο.

Στο σημείο αυτό αξίζει να σημειωθεί πως οι ενημερώσεις πάνω στα χαρακτηριστικά που αντιπροσωπεύουν τα μετρώμενα μεγέθη του CM Service διαβιβάζονται από την *BluetoothGattService* απευθείας στα ενδιαφερόμενα αντικείμενα (πχ `MonitoringSession`), δίχως δηλαδή να μεσολαβεί ο *GattUpdateReceiver*.

Σε περίπτωση που είναι αναγκαία η ενημέρωση του UI κατόπιν ενός από τα παραπάνω συμβάντα, αυτή γίνεται μέσω των ακόλουθων μεθόδων επανάκλησης (*callbacks*) της κλάσης `ConnectionManager.GuiCallback`:

`void onConnectionComplete(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Καλείται από την `GattUpdateReceiver.onReceive()` με την ολοκλήρωση της διαδικασίας εγκατάστασης σύνδεσης (λήψη του DCS).
`void onDisconnect(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Καλείται από την `GattUpdateReceiver.onReceive()` κατόπιν αποσύνδεσης της συσκευής με αναγνωριστικό `deviceId`.
`void onStatusChanged(int deviceId, Monitor.Type mon, Monitor.Status status)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `Monitor.Type mon`: Ένας από τους τύπους παρακολουθητών της υποενότητας 7.3.6. `Monitor.Status status`: Η νέα κατάσταση του παρακολουθητή.
Περιγραφή	Καλείται από την `GattUpdateReceiver.onReceive()` κατόπιν μεταβολής της κατάστασης του παρακολουθητή `mon` λόγω άφιξης εγγραφής ιστορικού για σχετικό συμβάν.
`void onRssiUpdate(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Καλείται από την `GattUpdateReceiver.onReceive()` με τη λήψη νέας τιμής για το RSSI της συσκευής με αναγνωριστικό `deviceId`.

Πίνακας 7.30: Η κλάση `ConnectionManager.GuiCallback`

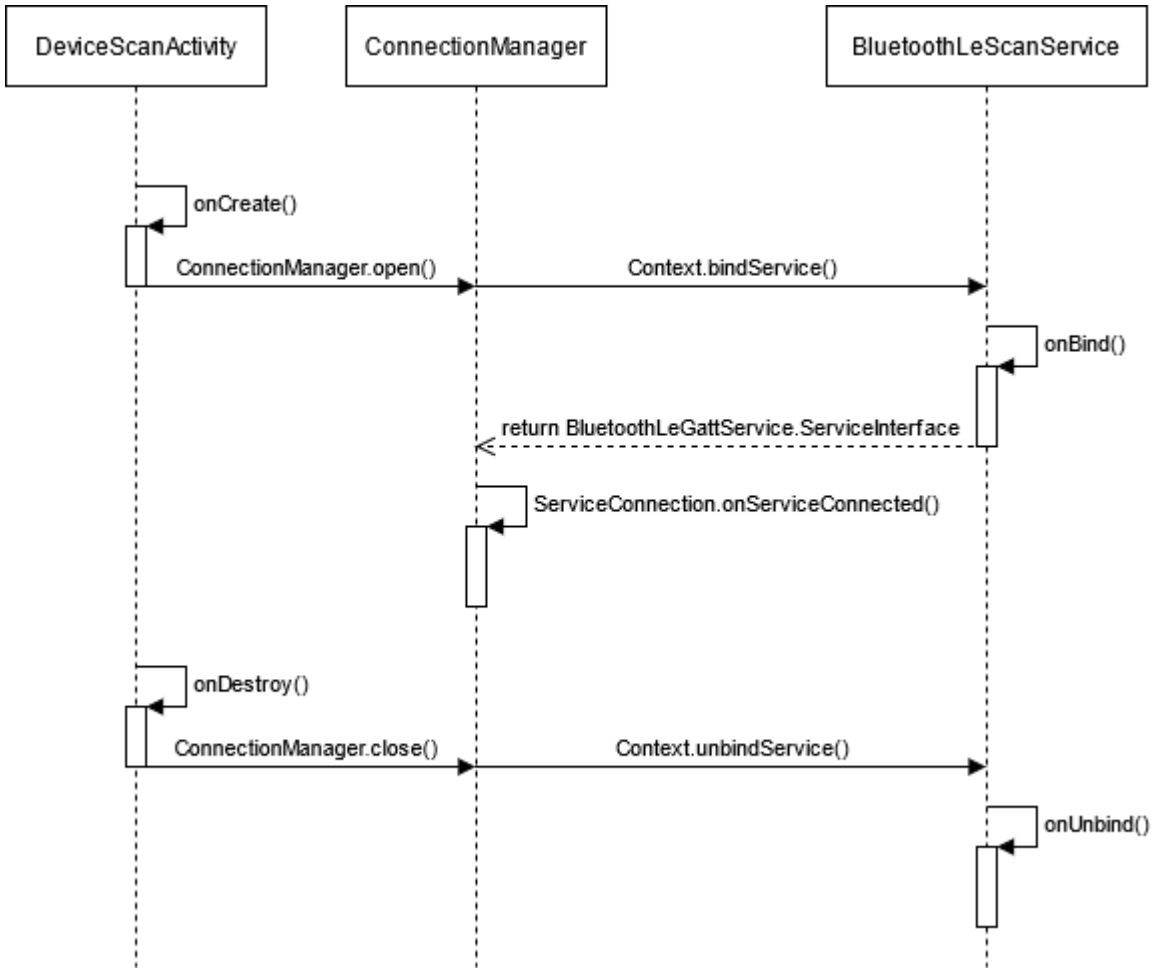
Ανά πάσα στιγμή είναι δυνατό να υπάρχει το πολύ ένα ενεργό στιγμιότυπο της εν λόγω κλάσης, το οποίο αντιστοιχεί στην τρέχουσα οθόνη της εφαρμογής και δηλώνεται καλώντας `ConnectionManager.registerGuiCallback()`. Το σώμα των μεθόδων της κλάσης είναι από προεπιλογή κενό, με τον προγραμματιστή να είναι σε θέση να ορίζει τις δικές του υλοποιήσεις κατά την κατασκευή κάθε νέου στιγμιοτύπου της.

7.3.3 Διαχείριση Services και Characteristics

Η αποστολή εντολών στη συσκευή και η λήψη δεδομένων από αυτή εμπεριέχει την εκτέλεση ενεργειών GATT πάνω σε χαρακτηριστικά του CM

Service. Για το σκοπό αυτό ορίζουμε την υπηρεσία `BluetoothLeGattService`, η οποία δεσμεύεται με την κλήση `ConnectionManager.open()`, διατηρώντας στη μνήμη τους εξυπηρετητές GATT (τύπου `android.bluetooth.BluetoothGatt`) για το σύνολο των ενεργών συνδέσεων, και εκθέτοντας στην εφαρμογή τις μεθόδους του BLE API του Android framework για τη διαχείριση υπηρεσιών και χαρακτηριστικών.

Η εκτέλεση των εντολών γίνεται σειριακά, με αναμονή σε μία εσωτερικά διατηρούμενη ουρά προτεραιότητας (τύπου `SynchronousQueue`). Η `BluetoothLeGattService` δεν εκκινεί κάποιο ξεχωριστό νήμα, συνεπώς κάθε ακολουθία κλήσεων των μεθόδων της πρέπει να εκτελείται με τη βοήθεια μίας εργασίας `AsyncTask`, ή κάποιου άλλου μηχανισμού ασύγχρονης εκτέλεσης της υποενότητας 6.2.4. Με την αποδέσμευση της υπηρεσίας από όλους τους πελάτες της, δηλαδή με τον τερματισμό της εφαρμογής και την κλήση `ConnectionManager.close()`, τερματίζονται όσες συνδέσεις διατηρούνται ενεργές και απελευθερώνονται οι σχετικοί πόροι.



Σχήμα 7.31: Χρήση της υπηρεσίας `BluetoothLeGattService`

Η δέσμευση της `BluetoothLeGattService` επιστρέφει μία διεπαφή τύπου `BluetoothLeGattService.ServiceInterface`:

`boolean initialize()`	
Περιγραφή	Αρχικοποίηση της υπηρεσίας μετά τη δέσμευση, με επιστροφή κατάλληλης τιμής αληθείας για έλεγχο επιτυχίας. Η διαδικασία αποτυγχάνει εάν ο BLE Controller του συστήματος δεν είναι διαθέσιμος ή δυσλειτουργεί.
`void connect(DeviceWrapper device)`	
Παράμετροι	`DeviceWrapper device`: Η συσκευή προς σύνδεση.
Περιγραφή	Επιχειρεί σύνδεση στον GATT server της συσκευής `device` μέσω της κλήσης `BluetoothDevice.connectGatt()` του BLE API. Σε περίπτωση επιτυχούς σύνδεσης, το αντικείμενο που αντιπροσωπεύει τον GATT server καταχωρείται στη μνήμη με αναγνωριστικό `device.id`.
`void disconnect(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Αποσυνδέεται από τον GATT server της συσκευής με αναγνωριστικό `deviceId` καλώντας τη μέθοδο `BluetoothGatt.disconnect()` του BLE API. Ο GATT server διατηρείται στη μνήμη για χρήση σε ενδεχόμενο επανασύνδεσης.
`void close(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Αποσυνδέεται από τον GATT server με αναγνωριστικό `deviceId`, και επιπρόσθετα αποδεσμεύει όλους τους σχετικούς πόρους καλώντας τη μέθοδο `BluetoothGatt.close()` του BLE API.
`void setCharacteristicNotification(int deviceId, UUID serviceUuid, UUID characteristicUuid, boolean enabled)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `UUID serviceUuid`: Το UUID της υπηρεσίας GATT. `UUID characteristicUuid`: Το UUID του χαρακτηριστικού. `boolean enabled`: Τιμή αληθείας για ενεργοποίηση των ειδοποιήσεων.
Περιγραφή	Ο client ενεργοποιεί ή απενεργοποιεί τις ειδοποιήσεις για ένα χαρακτηριστικό, καλώντας τη μέθοδο `BluetoothGatt.setCharacteristicNotification()` του BLE API. Απαιτείται επιπλέον η εγγραφή της κατάλληλης τιμής στον περιγραφέα *Client Characteristic Configuration*.

`void enqueueCharRead(int deviceId, UUID serviceUuid, UUID characteristicUuid)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `UUID serviceUuid`: Το UUID της υπηρεσίας GATT. `UUID characteristicUuid`: Το UUID του χαρακτηριστικού.
Περιγραφή	Προσθέτει ένα αίτημα ανάγνωσης τιμής χαρακτηριστικού στην ουρά των ενεργειών GATT. Η ανάγνωση γίνεται με κλήση της μεθόδου `BluetoothGatt.readCharacteristic()` του BLE API.
`void enqueueCharWrite(int deviceId, UUID serviceUuid, UUID characteristicUuid, byte[] value)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `UUID serviceUuid`: Το UUID της υπηρεσίας GATT. `UUID characteristicUuid`: Το UUID του χαρακτηριστικού. `byte[] value`: Η τιμή προς εγγραφή.
Περιγραφή	Προσθέτει ένα αίτημα εγγραφής τιμής χαρακτηριστικού στην ουρά των ενεργειών GATT. Η εγγραφή γίνεται με κλήση της μεθόδου `BluetoothGatt.writeCharacteristic()` του BLE API.
`void enqueueDescrRead(int deviceId, UUID serviceUUID, UUID characteristicUuid, UUID descriptorUuid)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `UUID serviceUuid`: Το UUID της υπηρεσίας GATT. `UUID characteristicUuid`: Το UUID του χαρακτηριστικού. `UUID descriptorUuid`: Το UUID του περιγραφέα.
Περιγραφή	Προσθέτει ένα αίτημα ανάγνωσης περιγραφέα στην ουρά των ενεργειών GATT. Η ανάγνωση γίνεται με κλήση της μεθόδου `BluetoothGatt.readDescriptor()` του BLE API.
`void enqueueDescrWrite(int deviceId, UUID serviceUUID, UUID characteristicUuid, UUID descriptorUuid, byte[] value)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `UUID serviceUuid`: Το UUID της υπηρεσίας GATT. `UUID characteristicUuid`: Το UUID του χαρακτηριστικού. `UUID descriptorUuid`: Το UUID του περιγραφέα. `byte[] value`: Η τιμή προς εγγραφή.
Περιγραφή	Προσθέτει ένα αίτημα εγγραφής περιγραφέα στην ουρά των ενεργειών GATT. Η εγγραφή γίνεται με κλήση της μεθόδου `BluetoothGatt.writeDescriptor()` του BLE API.
`void getRssi()`	
Περιγραφή	Μέτρηση της ισχύος λήψης σήματος (RSSI) για όλες τις ενεργές συνδέσεις, με χρήση της μεθόδου `BluetoothGatt.readRemoteRssi()` του BLE API.

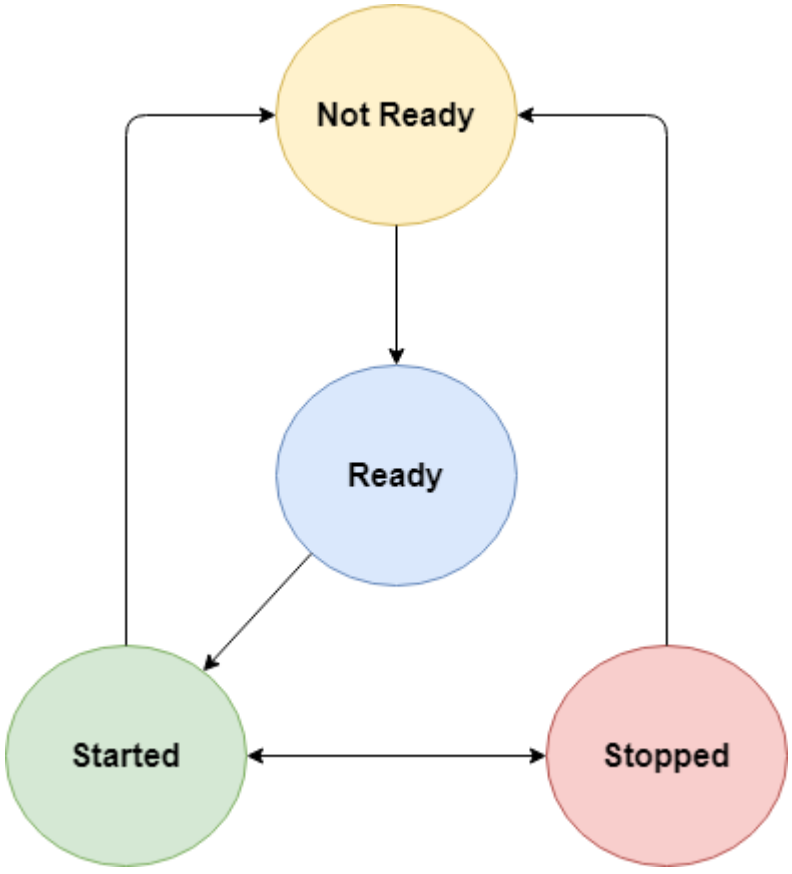
Πίνακας 7.32: API της `BluetoothLeGattService.ServiceInterface`

7.3.4 Ρυθμίσεις συσκευής

Η λογική αναπαράσταση του DCS ενσωματώνεται σε αντικείμενα τύπου `device.DeviceConfig`. Τα στοιχεία UI της οθόνης ρυθμίσεων *DeviceControlActivity* επιτρέπουν την τροποποίηση του DCS με τον τρόπο που περιγράφεται στην παράγραφο 7.2.1.7. Με την έξοδο από την οθόνη αυτή, από τις ρυθμίσεις που ορίστηκαν από το χρήστη κατασκευάζεται ένα καινούριο αντικείμενο `DeviceConfig`, το οποίο στη συνέχεια συγκρίνεται με το DCS των τρεχουσών ρυθμίσεων. Εφόσον τα δύο αντικείμενα διαφέρουν, το νέο DCS σειριοποιείται και αποστέλλεται στη συσκευή μέσω ενός πακέτου που περιέχει την εντολή *CP_SET_DEVICE_CONFIG*.

7.3.5 Διαχείριση συνόδου

Κατά τη διάρκεια μίας συνόδου συνεχούς εποπτείας, η εφαρμογή λαμβάνει σε πραγματικό χρόνο ενημερώσεις για τα μετρώμενα από τους αισθητήρες μεγέθη. Οι ενημερώσεις γίνονται πάνω στα αντίστοιχα characteristics του εκάστοτε CM Service, και στη συνέχεια οι ληφθείσες τιμές μπορούν να οπτικοποιούνται σε διαγράμματα και να αποθηκεύονται σε χρονοσειρές.



Σχήμα 7.33: FSM συνόδου εποπτείας

Για τον έλεγχο και τη ρύθμιση παραμέτρων της διαδικασίας ορίζουμε την κλάση `monitor.MonitoringSession`, η οποία αναπαριστά τον κύκλο ζωής μίας συνόδου, λαμβάνει τις ενημερώσεις των characteristics και τις προωθεί στους επιθυμητούς παραλήπτες. Κάθε στιγμιότυπο της `MonitoringSession` μπορεί να βρίσκεται ανά πάσα στιγμή σε μία από τις κατάστασεις που ορίζει η FSM του παραπάνω σχήματος:

- *Όχι έτοιμη (STATE_NOT_READY)*: Οι παράμετροι της συνόδου δεν έχουν ακόμη ρυθμιστεί, ή η σύνοδος δεν επιθυμούμε να λαμβάνει δεδομένα. Οι εντολές εκκίνησης/διακοπής δεν έχουν αποτέλεσμα.
- *Έτοιμη (STATE_READY)*: Η σύνοδος έχει ρυθμιστεί σωστά και είναι έτοιμη να λάβει εντολή εκκίνησης (*CP_SAMPLING_START*).
- *Ενεργή (STATE_STARTED)*: Η σύνοδος έχει λάβει εντολή εκκίνησης και είναι σε θέση λαμβάνει δεδομένα.
- *Ανενεργή (STATE_STOPPED)*: Η σύνοδος έχει λάβει εντολή διακοπής (*CP_SAMPLING_STOP*) και δεν θα πρέπει να δέχεται δεδομένα. Αν ακόμα λαμβάνονται δεδομένα, αυτά απορρίπτονται.

Παρακάτω παρουσιάζουμε το API της `MonitoringSession`:

`void setDeviceId(int deviceId)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής.
Περιγραφή	Συσχετίζει τη σύνοδο με τη συσκευή με αναγνωριστικό `deviceId`.
`int getDeviceId()`	
Περιγραφή	Επιστρέφει το αναγνωριστικό της συσκευής στην οποία ανήκει αυτή η σύνοδος.
`int getState()`	
Περιγραφή	Επιστρέφει την τρέχουσα κατάσταση της FSM της συνόδου.
`void setReady(boolean ready)`	
Παράμετροι	`boolean ready`: Τιμή αληθείας για την ενεργοποίηση της λήψης δεδομένων από τη σύνοδο, δηλαδή για μετάβαση τουλάχιστον σε κατάσταση *STATE_READY*.
Περιγραφή	Εάν η παράμετρος `ready` έχει την τιμή `false`, η σύνοδος μεταβαίνει στην κατάσταση *STATE_NOT_READY*. Διαφορετικά, αν η κατάσταση της συνόδου είναι *STATE_NOT_READY*, μεταβαίνει στην *STATE_READY*.

`void start(OnCompleteCallback cb)`	
Παράμετροι	`OnCompleteCallback cb`: Μέθοδος επανάκλησης για την ολοκλήρωση της διαδικασίας.
Περιγραφή	Εκκίνηση της συνόδου, δηλαδή μετάβαση στην κατάσταση *STATE_STARTED*, αν αυτό είναι δυνατό.
`void stop(OnCompleteCallback cb)`	
Παράμετροι	`OnCompleteCallback cb`: Μέθοδος επανάκλησης για την ολοκλήρωση της διαδικασίας.
Περιγραφή	Διακοπή της συνόδου, δηλαδή μετάβαση στην κατάσταση *STATE_STOPPED*.
`double getInterval()`	
Περιγραφή	Επιστρέφει την περίοδο δειγματοληψίας σε δευτερόλεπτα.
`void setInterval(double interval)`	
Παράμετροι	`double interval`: Η νέα περίοδος δειγματοληψίας σε δευτερόλεπτα.
Περιγραφή	Αλλαγή της περιόδου δειγματοληψίας στην τιμή `interval`. Στη συσκευή αποστέλλεται η εντολή *CP_SAMPLING_SET_INTERVAL*, αφού η τιμή έχει μετατραπεί σε μονάδες χρονιστή του server (διαίρεση με την τιμή *Timer Unit* του DIS).
`void addDataType(UUID type)`	
Παράμετροι	`UUID type`: Το UUID του χαρακτηριστικού.
Περιγραφή	Η σύνοδος επιτρέπεται να λαμβάνει ενημερώσεις από το χαρακτηριστικό με UUID `type`.
`void addPlot(UUID type, PlotFragment p)`	
Παράμετροι	`UUID type`: Το UUID του χαρακτηριστικού. `PlotFragment p`: Ο περιέκτης του διαγράμματος.
Περιγραφή	Επιτρέπει την προβολή των τιμών από το χαρακτηριστικό με UUID `type` στο διάγραμμα που φιλοξενείται στον περιέκτη `p`. Θα πρέπει να έχει προηγηθεί η κλήση `addDataType(type)`.
`Collection<UUID> getDataTypes()`	
Περιγραφή	Επιστρέφει τα χαρακτηριστικά από τα οποία η σύνοδος δέχεται ενημερώσεις.

`void removeDataTypes()`	
Περιγραφή	Καταργεί τη λήψη ενημερώσεων από όλα τα characteristics με τα οποία η σύνοδος μέχρι τώρα είχε συσχετιστεί.
`void addRecorder(LiveRecorder recorder)`	
Παράμετροι	`LiveRecorder recorder`: Καταγραφέας για την τρέχουσα σύνοδο.
Περιγραφή	Επιτρέπει την καταγραφή της συνόδου με τη βοήθεια ενός αντικειμένου `monitor.LiveRecorder`. Κάθε σύνοδος μπορεί να κατέχει το πολύ έναν `LiveRecorder`.
`void removeRecorder()`	
Περιγραφή	Αφαιρεί και καταστρέφει το αντικείμενο τύπου `Recorder` που ανήκει σε αυτή τη σύνοδο.
`double getTimestamp()`	
Περιγραφή	Επιστρέφει την τρέχουσα χρονοσφραγίδα της συνόδου, δηλαδή τη χρονική στιγμή λήψης της τελευταίας ενημέρωσης, σε σχέση με τη χρονική στιγμή έναρξης της συνόδου.
`void reset()`	
Περιγραφή	Επαναφέρει την τρέχουσα χρονοσφραγίδα της συνόδου στο μηδέν.

Πίνακας 7.34: API της κλάσης `MonitoringSession`

Ορίζουμε επιπλέον την κλάση `monitor.MonitoringManager`, η οποία παρέχει στατικές μεθόδους με αρμοδιότητα την κατασκευή και τη διαχείριση αντικειμένων `MonitoringSession`, καθώς και τη σύνδεση αυτών με εξαρτημένα στοιχεία γραφικού περιβάλλοντος (πχ διάγραμματα):

`MonitoringSession setupSession(DeviceWrapper device, boolean reset)`	
Παράμετροι	`DeviceWrapper device`: Η συσκευή προς εποπτεία. `boolean reset`: Τιμή αληθείας για επαναφορά των προεπιλεγμένων ρυθμίσεων.
Περιγραφή	Επιστρέφει την τρέχουσα σύνοδο εποπτείας της συσκευής `device` εάν αυτή υπάρχει, διαφορετικά επιστρέφει μία νέα συνοδο και την συσχετίζει με τη συσκευή.

`void destroySession(Context context, DeviceWrapper device)`	
Παράμετροι	`Context context`: Το περιβάλλον του καλούντος component. `DeviceWrapper device`: Η εποπτευόμενη συσκευή.
Περιγραφή	Αφαιρεί από τη λίστα τη σύνοδο εποπτείας της συσκευής `device`, και αποδεσμεύει τους σχετικούς με αυτήν πόρους.
`void close(Context context)`	
Παράμετροι	`Context context`: Το περιβάλλον του καλούντος component.
Περιγραφή	Τερματίζει όλες τις ενεργές συνόδους εποπτείας, και εκκαθαρίζει τη λίστα.
`static void Utils.setupPlots(MonitoringSession session, Fragment fragment, int containerResId, int layoutResId)`	
Παράμετροι	`MonitoringSession session`: Η τρέχουσα σύνοδος εποπτείας. `Fragment fragment`: Το fragment στο οποίο ανήκει ο περιέκτης. `int containerResId`: Resource ID του περιέκτη. `int layoutResId`: Resource ID του layout του διαγράμματος.
Περιγραφή	Βάσει των ενεργών μεγεθών της συνόδου `session`, κατασκευάζει διαγράμματα και τα εισάγει στο στοιχείο-περιέκτη της υποοθόνης `fragment` με αναγνωριστικό `containerResId`, χρησιμοποιώντας το API που παρουσιάστηκε στην υποενότητα 7.2.2.
`static void Utils.destroyPlots(MonitoringSession session, Fragment fragment)`	
Παράμετροι	`MonitoringSession session`: Η συσκευή προς εποπτεία. `Fragment fragment`: Η συσκευή προς εποπτεία.
Περιγραφή	Αφαιρεί από την υποοθόνη `fragment` όλα τα διαγράμματα που έχουν παραχθεί για τη σύνοδο `session`, χρησιμοποιώντας το API που παρουσιάστηκε στην υποενότητα 7.2.2.
`static void Utils.resetPlots(MonitoringSession session)`	
Παράμετροι	`MonitoringSession session`: Η συσκευή προς εποπτεία.
Περιγραφή	Εκκαθαρίζει τα περιεχόμενα (χρονοσειρές) όλων των διαγραμμάτων της συνόδου `session`, επαναφέροντάς τα στη χρονική στιγμή μηδέν. Χρησιμοποιείται συνήθως κατά την πίσω-πλοήγηση προς μία οθόνη που περιέχει διαγράμματα.

Πίνακας 7.35: API της κλάσης `MonitoringManager`

7.3.6 Κατάσταση συσκευής

Όπως παρουσιάστηκε στο κεφάλαιο 4, ο έλεγχος καλής λειτουργίας γίνεται όχι από την εφαρμογή, αλλά από τη συσκευή με τη βοήθεια παρακολουθητών. Σε περίπτωση παράβασης κάποιας συνθήκης καλής λειτουργίας, παράγεται και αποστέλλεται στην εφαρμογή κατάλληλη εγγραφή ιστορικού για το συμβάν. Ακολούθως, η κατάσταση της συσκευής ενημερώνεται σύμφωνα με τις σχέσεις της παραγράφου 4.3.2.1. Όσον αφορά την εφαρμογή πελάτη, κάθε παρακολουθητής αναπαρίσταται από αντικείμενα τύπου `monitor.Monitor` της παρακάτω μορφής:

Πεδίο	**Περιγραφή**
`Monitor.Status status`	Αναπαριστά τον τύπο του παρακολουθητή, και λαμβάνει μία από τις προκαθορισμένες τιμές της απαρίθμησης `Monitor.Status`. Μέλη της απαρίθμησης αποτελούν: • Ο κωδικός κατάστασης (*OK*, *Warning* ή *Critical*). • Το αναγνωριστικό του εικονιδίου για την κατάσταση.
`Monitor.Type type`	Αναπαριστά τον τύπο του παρακολουθητή, και λαμβάνει μία από τις προκαθορισμένες τιμές της απαρίθμησης `Monitor.Type`. Μέλη της απαρίθμησης αποτελούν: • Το αναγνωριστικό του παρακολουθητή. • Η φιλική ονομασία του παρακολουθητή. • Οι φιλικές ονομασίες των τιμών ελέγχου του παρακολουθητή. • Μία συμβολοσειρά μορφοποίησης για την εμφάνιση των τρεχουσών τιμών ελέγχου. • Η μονάδα μέτρησης της τιμής του παρακολουθητή. • Το αναγνωριστικό ενός αντιπροσωπευτικού εικονιδίου. Τα εικονίδια για τις υπηρεσίες ES, IM περιλαμβάνουν:  *Θερμοκρασία*  *Μεταβολή υγρασίας*  *Μεταβολή θερμοκρασίας*  *Εντοπισμός κίνησης*  *Υγρασία*  *Εντοπισμός δόνησης*
`float measuredValue`	Αντιστοιχεί στο πεδίο *Measured Value*.

Πίνακας 7.36: Δομή της κλάσης `Monitor`

Ακολουθεί το API της κλάσης `Monitor`:

`Monitor.Type getType()`	
Περιγραφή	Επιστρέφει τον τύπο του παρακολουθητή, ο οποίος μπορεί να ανήκει στις τιμές που δίνονται στον προηγούμενο πίνακα.
`Monitor.Status getStatus()`	
Περιγραφή	Επιστρέφει την κατάσταση της παραμέτρου που επιβλέπει αυτός ο παρακολουθητής.
`void setStatus(Monitor.Status status)`	
Παράμετροι	`Monitor.Status status`: Η νέα κατάσταση του παρακολουθητή.
Περιγραφή	Ενημερώνει την κατάσταση της παραμέτρου που επιβλέπει αυτός ο παρακολουθητής.
`static List<Monitor> values(String deviceType)`	
Παράμετροι	`String deviceType`: Ο τύπος της συσκευής, όπως ορίζεται στον πίνακα 7.27.
Περιγραφή	Επιστρέφει ένα νέο σύνολο παρακολουθητών για μία συσκευή τύπου `deviceType`, αρχικοποιημένων στην κατάσταση *STATUS_OK*.

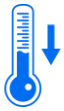
Πίνακας 7.37: API της κλάσης `Monitor`

Η ενεργοποίηση και η απενεργοποίηση των παρακολουθητών γίνεται τροποποιώντας κατάλληλα το DCS, όπως περιγράφεται στην υποπαράγραφο 7.1.2.1.7.

7.3.7 Ιστορικό συσκευής

Κατά το χρόνο εκτέλεσης, η εφαρμογή τροφοδοτεί το UI με την απαραίτητη πληροφορία για το ιστορικό της συσκευής μέσω κατάλληλα κατασκευασμένων αντικειμένων που αντιπροσωπεύουν συμβάντα, καταγεγραμμένες συνόδους κλπ. Τα αντικείμενα αυτά αποτελούν μέλη των παρακάτω κλάσεων του πακέτου `com.android.blesense.log`:

- `log.LogEntry`: Αντιπροσωπεύει καταχωρήσεις συμβάντων στο ιστορικό, σύμφωνα με τον ορισμό της υποενότητας 4.5.3. Αποτελείται από τα εξής πεδία, οι τιμές των οποίων προκύπτουν από κατάλληλη αποσειριοποίηση του ληφθέντος πακέτου *CP_RETRIEVE_LOG*:

Πεδίο	Περιγραφή
`String timestamp`	Αντιστοιχεί στο πεδίο *Timestamp*.
`LogEntry.Type type`	Αναπαριστά τον τύπο του συμβάντος, και λαμβάνει μία από τις προκαθορισμένες τιμές της απαρίθμησης `LogEntry.Type`. Μέλη της απαρίθμησης αποτελούν: • Ο κωδικός του συμβάντος (τύπου `int`), όπως δίνεται στο πεδίο *Type* της καταχώρησης. • Το είδος του παρακολουθητή που παράγει το συμβάν (τύπου `Monitor.Type`). • Η μονάδα μέτρησης της τιμής του πεδίου *Measured Value* της καταχώρησης. • Το αναγνωριστικό ενός αντιπροσωπευτικού εικονιδίου για το συμβάν. Τα εικονίδια για τις υπηρεσίες ES, IM περιλαμβάνουν:  *Temperature High*  *Temperature Change High*  *Temperature Low*  *Temperature Change Low*  *Humidity High*  *Humidity Change High*  *Humidity Low*  *Humidity Change Low*  *Motion Detected*  *Shock Detected*
`float measuredValue`	Αντιστοιχεί στο πεδίο *Measured Value*.

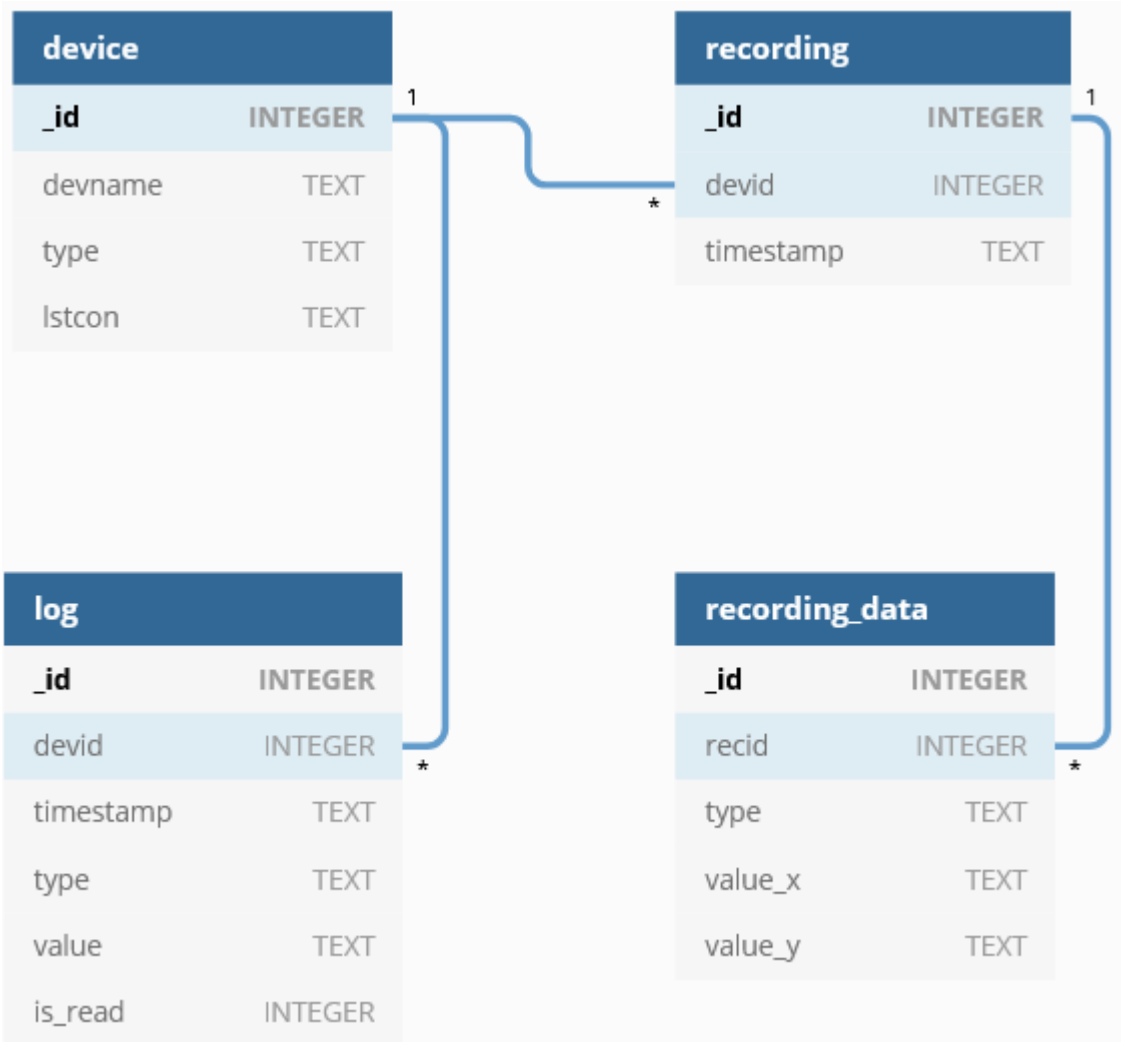
Πίνακας 7.38: Η κλάση `LogEntry`

- `log.RecordingEntry`: Αντιπροσωπεύει στοιχεία αποθηκευμένων κυματομορφών από συνόδους εποπτείας. Κάθε στιγμιότυπο της κλάσης αντιστοιχεί σε συγκεκριμένη χρονική στιγμή και συγκεκριμένο εποπτευόμενο μέγεθος. Για βαθμωτά μεγέθη περιέχει ένα σημείο, διαφορετικά ένα διάνυσμα από σημεία.

Πεδίο	**Περιγραφή**
`String timestamp`	Η κοινή χρονοσφραγίδα (τετμημένη) των σημείων: προκύπτει από τη διαφορά της χρονικής στιγμής λήψης των μετρήσεων από την χρονική στιγμή έναρξης της συνόδου.
`String measuredValue`	Οι τεταγμένες των σημείων (μετρήσεις για τη συγκεκριμένη χρονική στιγμή).
`String type`	Αντιπροσωπεύει το εποπτευόμενο μέγεθος, και παράγεται από το UUID του αντίστοιχου χαρακτηριστικού του CM Service.

Πίνακας 7.39: Η κλάση `RecordingEntry`

Το περιεχόμενο όλων των παραπάνω αντικειμένων αντλείται από τα μόνιμα αποθηκευμένα δεδομένα της εφαρμογής, τα οποία διατηρούνται στη βάση δεδομένων *DataStore*. Παρακάτω δίνουμε το σχεσιακό σχήμα της ΒΔ:



Πίνακας 7.40: Σχεσιακό σχήμα ΒΔ

Παρουσιάζουμε τώρα κάθε πίνακα της ΒΔ ξεχωριστά:

- *device*: Διατηρεί τις καταχωρήσεις των συσκευών με τις οποίες η εφαρμογή έχει συνδεθεί τουλάχιστον μία φορά.

Στήλη	**Τύπος**	**Περιγραφή**
_id	INTEGER	Πρωτεύον κλειδί (αναγνωριστικό συσκευής).
devname	TEXT	Όνομα συσκευής.
lstcon	TEXT	Ημερομηνία και ώρα τελευταίας σύνδεσης, μορφοποιημένη ως "*dd*/*mm*/*yyyy hh*:*mm*:*ss*".
type	TEXT	Τύπος συσκευής, όπως ορίζεται στον πίνακα 7.27.

Πίνακας 7.41: Ο πίνακας "device"

- *log*: Διατηρεί το ιστορικό συμβάντων από όλες τις συσκευές με τις οποίες έχει συνδεθεί η εφαρμογή. Οι εγγραφές ακολουθούν τη δομή που περιγράφεται στην υποενότητα 4.5.3.

Στήλη	**Τύπος**	**Περιγραφή**
_id	INTEGER	Πρωτεύον κλειδί.
devid	INTEGER	Ξένο κλειδί, αναφέρεται στο *device*.*_id*.
timestamp	TEXT	Χρονοσφραγίδα συμβάντος, μορφοποιημένη ως "*dd*/*mm*/*yyyy hh*:*mm*:*ss*".
type	TEXT	Τύπος συμβάντος, όπως ορίζεται στην παράγραφο 4.5.3.
value	TEXT	Η τιμή του μεγέθους που μετρήθηκε, σειριοποιημένη ως "[v1, …,vN]".
is_read	INTEGER	Δυαδική τιμή αληθείας για την επισήμανση μιας εγγραφής ως αναγνωσμένης (1) ή μη (0).

Πίνακας 7.42: Ο πίνακας " log"

- *recording*: Για όλες τις συσκευές, διατηρεί μία λίστα των κυματομορφών που προέρχονται από καταγραφές συνόδων εποπτείας.

Στήλη	**Τύπος**	**Περιγραφή**
_id	INTEGER	Πρωτεύον κλειδί.
devid	INTEGER	Ξένο κλειδί, αναφέρεται στο *device*.*_id*.
timestamp	TEXT	Χρονοσφραγίδα έναρξης της κυματομορφής.

Πίνακας 7.43: Ο πίνακας "recording"

- Πίνακας *recording_data*: Διατηρεί τα περιεχόμενα των κυματομορφών του πίνακα *recording*. Σημειώνουμε πως εδώ οι κυματομορφές αποθηκεύονται μόνο στο πεδίο του χρόνου. Η μετατροπή στο πεδίο της συχνότητας γίνεται προσωρινά, μόνο για προβολή σε διάγραμμα, με τη βοήθεια της βιβλιοθήκης *noise* (github.com/paramsen/noise).

Στήλη	**Τύπος**	**Περιγραφή**
_id	INTEGER	Πρωτεύον κλειδί.
recid	INTEGER	Ξένο κλειδί, αναφέρεται στο *recording_entry._id*.
type	TEXT	Τύπος δεδομένων. Αντιστοιχεί στο UUID του χαρακτηριστικού που αντιπροσωπεύει το μετρηθέν μέγεθος.
value_x	TEXT	Χρονική στιγμή μέτρησης, σε δευτερόλεπτα.
value_y	TEXT	Η τιμή του μεγέθους που μετρήθηκε, σειριοποιημένη ως "[v1, …,vN]".

Πίνακας 7.44: Ο πίνακας "recording_data"

Για τη διευκόλυνση της επικοινωνίας του κώδικα της εφαρμογής με τη ΒΔ και την αξιοποίηση δυνατοτήτων όπως είναι η φόρτωση στο παρασκήνιο και η αυτόματη ανανέωση των δεδομένων, εκθέτουμε το σχήμα της ΒΔ στον πάροχο περιεχομένου `DataProvider:ContentProvider`, για τον οποίο ορίζουμε το ριζικό URI `root` με τιμή `content://com.example.android.blesense/`, καθώς και τα εξής θυγατρικά URI:

- `root/device`: Επιστρέφει τον πίνακα *device*.
- `root/log`: Επιστρέφει τον πίνακα *log*.
- `root/recording`: Επιστρέφει τον πίνακα *recording*.
- `root/recording_data`: Επιστρέφει τον πίνακα *recording_data*.
- `root/history`: Επιστρέφει την αριστερή συνένωση (left join) $device \bowtie_{\_id=devid} log \bowtie_{devid=devid} recording$.

Η επικοινωνία με τη ΒΔ και η διαχείριση του παρόχου *DataProvider* γίνονται με κατάλληλες βοηθητικές μεθόδους, μέλη της κλάσης `DataService`, για την οποία ανά πάσα στιγμή διατηρείται στη μνήμη το πολύ ένα στιγμιότυπο (ακολουθώντας το πρότυπο σχεδίασης *singleton*). Για την αυτόματη επαναφόρτωση των δεδομένων της ΒΔ από τα ενδιαφερόμενα components, η `DataService` χρησιμοποιεί τους παρακάτω φορτωτές (loaders):

- *LOG_LOADER_ID* (0): Παρακολουθεί το URI `root/log`.
- *RECORDING_LOADER_ID* (1): Παρακολουθεί το URI `root/recording`.
- *HISTORY_LOADER_ID* (2): Παρακολουθεί το URI `root/history`.

Εάν οι τελικοί αποδέκτες των δεδομένων είναι στοιχεία γραφικού περιβάλλοντος, αυτά μπορούν να χρησιμοποιούν τις κλάσεις `CursorAdapter`, `RecyclerView.Adapter`, ή οποιαδήποτε άλλη κλάση μπορεί να συνεργαστεί με τους φορτωτές υλοποιώντας μία μέθοδο ανανέωσης `refresh(Cursor cursor)`, την οποία ορίζουμε ως το μοναδικό στοιχείο της διαπροσωπείας `ListRefreshable`.

Ακολουθεί το API της `DataService`:

`static DataService getInstance(Context context)`	
Παράμετροι	`Context context`: Περιβάλλον του καλούντος component.
Περιγραφή	Επιστρέφει το τρέχον στιγμιότυπο της κλάσης εάν αυτό υπάρχει, διαφορετικά ένα νέο.
`void close()`	
Περιγραφή	Καταστροφή του τρέχοντος στιγμιοτύπου.
`void findDevice(int deviceId, DataService.QueryCallback callback)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `DataService.QueryCallback callback`: Μέθοδος επανάκλησης για την ολοκλήρωση της ενέργειας.
Περιγραφή	Αναζήτηση στον πίνακα *device* για τη συσκευή με αναγνωριστικό `deviceId`. Το αποτέλεσμα διαβιβάζεται ασύχρονα από τη μέθοδο επανάκλησης `callback`.
`void addDevice(int deviceId, DeviceEntry deviceData, DataService.QueryCallback callback)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `DeviceEntry deviceData`: Τα δεδομένα της εγγραφής. `DataService.QueryCallback callback`: Μέθοδος επανάκλησης για την ολοκλήρωση της ενέργειας.
Περιγραφή	Προσθήκη μιας εγγραφής για τη συσκευή με αναγνωριστικό `deviceId` στον πίνακα *device*. Το αποτέλεσμα διαβιβάζεται ασύγχρονα από τη μέθοδο επανάκλησης `callback`.
`void updateDevice(int deviceId, long timestamp, DataService.QueryCallback callback)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `long timestamp`: Χρονική στιγμή της τελευταίας επιτυχούς σύνδεσης. `DataService.QueryCallback callback`: Μέθοδος επανάκλησης για την ολοκλήρωση της ενέργειας.
Περιγραφή	Ενημέρωση του πεδίου *lstcon* της εγγραφής του πίνακα *device* για τη συσκευή με αναγνωριστικό `deviceId`.

`void insertLogEntry(int deviceId, LogEntry logEntry)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `LogEntry logEntry`: Τα δεδομένα της εγγραφής ιστορικού.
Περιγραφή	Προσθήκη της νέας εγγραφής ιστορικού `logEntry` για τη συσκευή με αναγνωριστικό `deviceId` στον πίνακα *log*.
`void deleteLogEntry(int entryId)`	
Παράμετροι	`int entryId`: Το κλειδί της εγγραφής.
Περιγραφή	Διαγραφή της εγγραφής ιστορικού με κλειδί `entryId` από τον πίνακα *log*.
`void updateLogEntry(int entryId, int value)`	
Παράμετροι	`int entryId`: Το κλειδί της εγγραφής. `int value`: Επισήμανση της εγγραφής ως αναγνωσμένης (1) ή όχι (0).
Περιγραφή	Ενημέρωση της επισήμανσης ανάγνωσης της εγγραφής ιστορικού με αναγνωριστικό `entryId`.
`void addRecording(int deviceId, int recId, long timestamp)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `int recId`: Το αναγνωριστικό της συνόδου. `long timestamp`: Η χρονοσφραγίδα έναρξης της συνόδου.
Περιγραφή	Δημιουργία καταχώρησης για τη σύνοδο με αναγνωριστικό `recId`, αναγνωριστικό συσκευής `deviceId`, και στιγμή έναρξης `timestamp`.
`void insertRecordingEntry(long recordingId, RecordingEntry entry)`	
Παράμετροι	`long recordingId`: Το αναγνωριστικό της συνόδου. `RecordingEntry entry`: Καταχώρηση για μία μέτρηση της συνόδου.
Περιγραφή	Προσθήκη μίας μέτρησης της συνόδου με αναγνωριστικό `recordingId` στον πίνακα *recording_data*.
`void deleteRecording(long recordingId)`	
Παράμετροι	`long recordingId`: Το αναγνωριστικό της συνόδου.
Περιγραφή	Διαγραφή της καταχώρησης για τη σύνοδο με αναγνωριστικό `recordingId`.

`void getRecordings(LoaderManager loaderManager, int deviceId, long timestamp, BackgroundLoadCallbacks callbacks)`	
Παράμετροι	`LoaderManager loaderManager`: Βοηθητική υπηρεσία του Android framework για την κατασκευή ενός `Loader`. `long timestamp`: Χρονοσφραγίδα αναφοράς. `BackgroundLoadCallbacks callbacks`: Μέθοδοι επανάκλησης για αυτόματη επαναφόρτωση με χρήση ενός `Loader`.
Περιγραφή	Κατασκευάζει και εκκινεί το φορτωτή με αναγνωριστικό *RECORDING_LOADER_ID* για τη φόρτωση και την αυτόματη ανανέωση της λίστας καταγεγραμμένων χρονοσειρών με χρονοσφραγίδα μεταγενέστερη της `timestamp`, για τη συσκευή με αναγνωριστικό `deviceId`.
`void loadHistory(LoaderManager loaderManager, long timestamp, BackgroundLoadCallbacks callbacks)`	
Παράμετροι	`LoaderManager loaderManager`: Βοηθητική υπηρεσία του Android framework για την κατασκευή ενός `Loader`. `long timestamp`: Χρονοσφραγίδα αναφοράς. `BackgroundLoadCallbacks callbacks`: Μέθοδοι επανάκλησης για την αυτόματη φόρτωση με χρήση ενός `Loader`.
Περιγραφή	Κατασκευάζει και εκκινεί το φορτωτή με αναγνωριστικό *HISTORY_LOADER_ID* για τη φόρτωση και την αυτόματη ανανέωση της λίστας συσκευών για τις οποίες υπάρχουν καταγεγραμμένες χρονοσειρές ή εγγραφές ιστορικού με χρονοσφραγίδα μεταγενέστερη της `timestamp`.
`void deleteHistory(int deviceId, long timestamp)`	
Παράμετροι	`int deviceId`: Το αναγνωριστικό της συσκευής. `long timestamp`: Η χρονοσφραγίδα αναφοράς.
Περιγραφή	Διαγράφει όλες τις καταγεγραμμένες χρονοσειρές και όλες τις εγγραφές ιστορικού με χρονοσφραγίδα μεταγενέστερη της `timestamp`.
`void getRecordingData(long recordingId, DataService.QueryCallback callback)`	
Παράμετροι	`long recordingId`: Το αναγνωριστικό της συνόδου. `DataService.QueryCallback callback`: Μέθοδος επανάκλησης για την ολοκλήρωση της διαδικασίας.
Περιγραφή	Επιστρέφει το σύνολο των αποθηκευμένων στον πίνακα *recording_data* μετρήσεων της συνόδου με αναγνωριστικό `recordingId`. Το αποτέλεσμα διαβιβάζεται ασύγχρονα από τη μέθοδο επανάκλησης `callback`.

Πίνακας 7.45: API της κλάσης `DataService`

Οι μέθοδοι που πραγματοποιούν μεμονωμένα ερωτήματα κατασκευάζουν κατάλληλα στιγμιότυπα της κλάσης `android.content.AsyncQueryHandler` για εκτέλεσή τους σε ξεχωριστό νήμα. Η ολοκλήρωση κάθε ερωτήματος πυροδοτεί την κατάλληλη μέθοδο χειρισμού της `AsyncQueryHandler`, η οποία με τη σειρά της ενημερώνει τον καλούντα κώδικα μέσω μίας μεθόδου επανάκλησης της διαπροσωπείας `DataService.QueryCallback`:

`void onQueryComplete(Cursor data)`	
Παράμετροι	`Cursor data`: Δείκτης στο αποτέλεσμα του ερωτήματος.
Περιγραφή	Καλείται με την ολοκλήρωση ενός ερωτήματος που επιστρέφει δεδομένα (για παράδειγμα, από την `AsyncQueryHandler.onQueryComplete()`).
`void onUpdateComplete()`	
Περιγραφή	Καλείται με την ολοκλήρωση ενός ερωτήματος ενημέρωσης (για παράδειγμα, από τις `AsyncQueryHandler.onInsertComplete()`, `AsyncQueryHandler.onDeleteComplete()`, `AsyncQueryHandler.onUpdateComplete()`).

Πίνακας 7.46: API της διαπροσωπείας `DataService.QueryCallback`

7.3.8 Προτιμήσεις εφαρμογής

Οι προτιμήσεις που εκτίθενται στο χρήστη μέσω του UI της *AppSettingsActivity* (παράγραφος 7.2.1.3) διατηρούνται στον ιδιωτικό αποθηκευτικό χώρο της εφαρμογής ως ζεύγη κλειδιών-τιμών, και ανακαλούνται ή μεταβάλλονται ως στοιχεία του καταλόγου *κοινών προτιμήσεων* (*shared preferences*) με χρήση του API των κλάσεων `SharedPreferences` και `PreferenceManager`, όποτε αυτό απαιτείται από την εκτέλεση κάποιας ενέργειας. Αναλυτικά οι προτιμήσεις είναι:

- ***Σάρωση στο παρασκήνιο (Background scan)***

Περιγραφή: Ενεργοποιεί/απενεργοποιεί τη σάρωση στο παρασκήνιο. Η τιμή της προτίμησης διατηρείται ως τιμή αλήθειας (`boolean`), και εκτίθεται στο χρήστη ως στοιχείο UI τύπου `CheckBoxPreference`.

Κλειδί: "*background_scan*"

Ενημέρωση τιμής: Μόνο από το UI της *AppSettingsActivity*.

Ανάγνωση τιμής: Κατά την έναρξη και τον τερματισμό της *DeviceScanActivity*, όπως περιγράφεται στην υποενότητα 7.3.1.

- ***Ανανέωση λίστας συσκευών (Refresh device list)***

Περιγραφή: Ορίζει την περίοδο ανανέωσης της λίστας συσκευών (σε s). Η τιμή της προτίμησης διατηρείται ως συμβολοσειρά, και εκτίθεται στο χρήστη ως στοιχείο UI τύπου `EditTextPreference`.

Κλειδί: "*refresh_list*"

Ενημέρωση τιμής: Μόνο από το UI της *AppSettingsActivity*.

Ανάγνωση τιμής: Κατά την περιοδική ανανέωση της λίστας των διαθέσιμων συσκευών, όπως περιγράφεται στην υποενότητα 7.3.1.

- ***Συγχρονισμός ημερομηνίας και ώρας (Sync date/time)***

Περιγραφή: Ενεργοποιεί/απενεργοποιεί τον αυτόματο συγχρονισμό της ημερομηνίας/ώρας του server με αυτήν του client. Η τιμή της προτίμησης διατηρείται ως τιμή αληθείας (`boolean`), και εκτίθεται στο χρήστη ως στοιχείο UI τύπου `CheckBoxPreference`.

Κλειδί: "*sync_date_time*"

Ενημέρωση τιμής: Μόνο από το UI της *AppSettingsActivity*.

Ανάγνωση τιμής: Κατά την εγκατάσταση σύνδεσης με απομακρυσμένη συσκευή, όπως περιγράφεται στην υποενότητα 7.3.2.

- ***Χρονικό όριο ολοκλήρωσης σύνδεσης (Abort connection attempt)***

Περιγραφή: Ορίζει το χρονικό όριο (σε s) με την παρέλευση του οποίου αποτυγχάνει μία εγκατάσταση σύνδεσης που βρίσκεται σε εξέλιξη. Η τιμή της προτίμησης διατηρείται ως ακέραιος αριθμός (`long`), και εκτίθεται στο χρήστη ως στοιχείο UI τύπου `EditTextPreference`.

Κλειδί: "*connection_timeout*"

Ενημέρωση τιμής: Μόνο από το UI της *AppSettingsActivity*.

Ανάγνωση τιμής: Κατά την εγκατάσταση σύνδεσης με απομακρυσμένη συσκευή, όπως περιγράφεται στην υποενότητα 7.3.2.

8 Συμπεράσματα-προτάσεις

8.1 Επίλογος

Στο πλαίσιο της παρούσας εργασίας υλοποιήθηκε ένα όσο το δυνατόν πιο πλήρες σύστημα CM. Το σύστημα της εργασίας μας δεν είναι καθεαυτό σύστημα PdM, ωστόσο αποτελεί θεμελιώδες μέρος κάθε τέτοιας εφαρμογής. Επιπλέον, δεν στηρίζεται σε ένα αυτορρυθμιζόμενο δίκτυο αισθητήρων, αλλά σε ένα απλό, συγκεντρωτικό δίκτυο αστέρα, καθώς όλοι οι κόμβοι επικοινωνούν απευθείας μόνο με την πύλη (εφαρμογή πελάτη).

Παρ' όλα αυτά οι ρυθμοί μετάδοσης κρίνονται ικανοποιητικοί για τις εφαρμογές που μελετήθηκαν, το ίδιο και η διάρκεια ζωής της πηγής τροφοδοσίας που επιτυγχάνεται. Το σπουδαιότερο πλεονέκτημα άλλωστε του συστήματός μας βρίσκεται στην ίδια του την υποδομή, η οποία επιτρέπει επεκτάσεις τόσο στις υπηρεσίες CM, οι οποίες ορίζονται ανεξάρτητα από το υλικό, όσο και στη λειτουργικότητα και τη δομή του δικτύου.

8.2 Προτάσεις για επέκταση

Για περαιτέρω ανάπτυξη προτείνονται οι παρακάτω επεκτάσεις στο σύστημα, στην κατεύθυνση της σταδιακής μετατροπής του σε ένα πλήρες σύστημα PdM:

- Ενσωμάτωση λογικής για την αλληλεπίδραση και την επικοινωνία μεταξύ των κόμβων. Αυτό μπορεί να γίνει είτε κεντρικά μέσω της πύλης για διαβίβαση εντολών (πχ η εφαρμογή δίνει εντολή σε μια απομακρυσμένη συσκευή μετά τη λήψη μίας αναφοράς συμβάντος από άλλη συνδεδεμένη συσκευή), είτε με απευθείας ανταλλαγή μηνυμάτων. Στην τελευταία περίπτωση, οι συσκευές του δικτύου οφείλουν να υποστηρίζουν την εναλλαγή των ρόλων τους στο επίπεδο του GAP.

- Ανάλυση των δεδομένων (χρονοσειρών, καταγραφών συμβάντων) με μία από τις μεθόδους τις υποενότητας 1.2.3 (πχ εκπαίδευση νευρωνικού δικτύου),

- Αξιοποίηση μεταγενέστερων εκδόσεων της τεχνολογίας BLE (Bluetooth 5.x ή νεότερο) για αυξημένο εύρος ζώνης, σύνδεση σε κατανεμημένη τοπολογία κλπ.

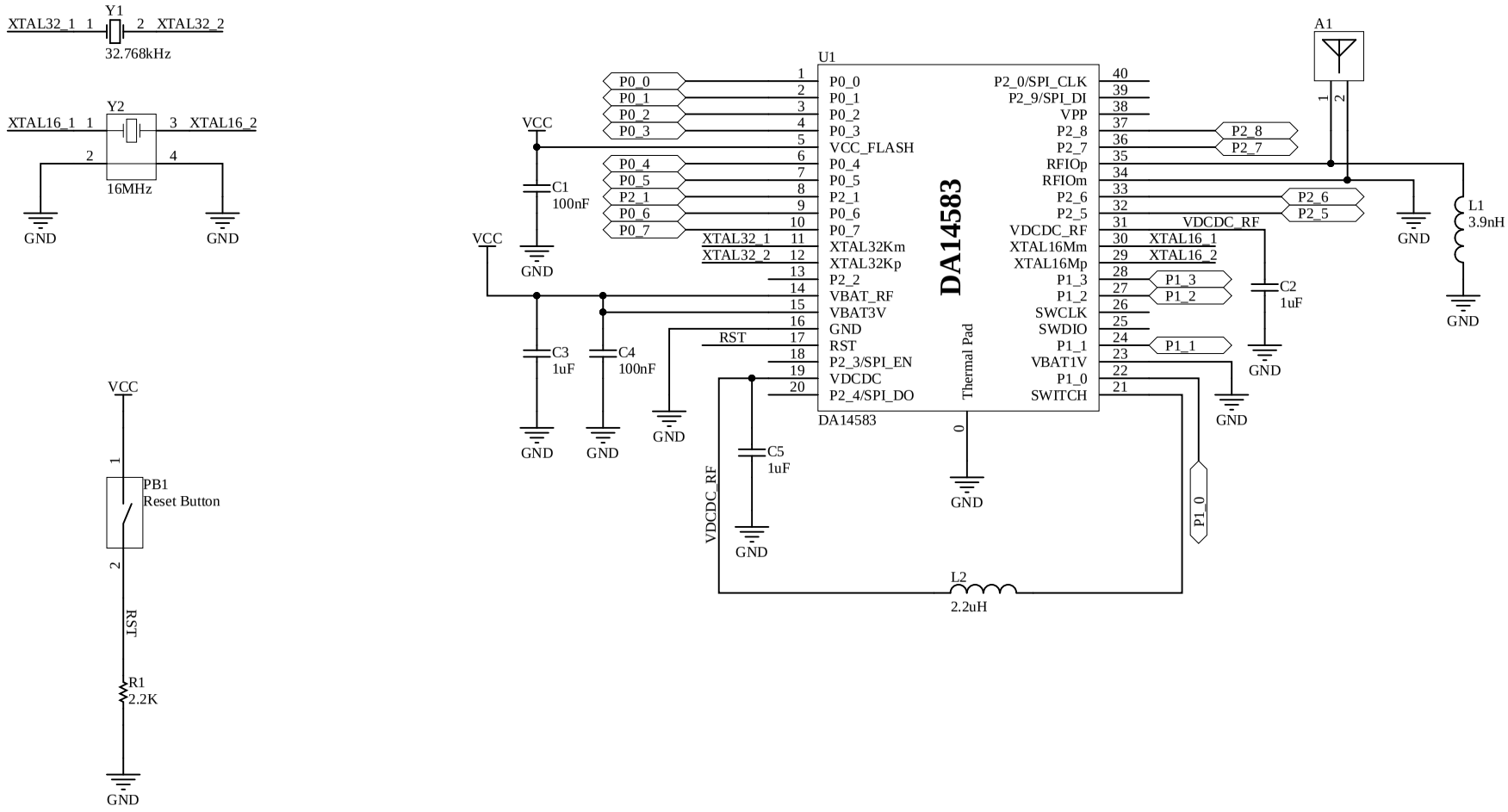
- Υλοποίηση ενός κεντρικού σημείου συλλογής των δεδομένων, πχ με επικοινωνία της εφαρμογής πελάτη με κάποιον απομακρυσμένο διακομιστή ή μία *υπηρεσία νέφους* (*cloud service*).

- Βελτίωση της λειτουργικότητας και της αξιοπιστίας των αισθητήρων με ενσωμάτωση λειτουργιών επαναβαθμονόμησης (*calibration*), απαλοιφή της βαρυτικής συνιστώσας από τις μετρήσεις του επιταχυνσιομέτρου κ.ά.

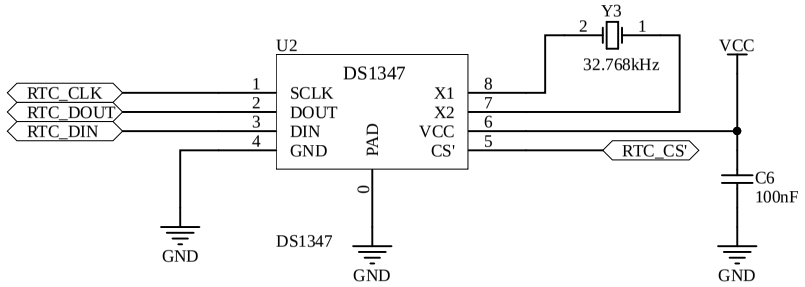
Βιβλιογραφία

1. Robin Heydon, "Bluetooth Low Energy: The Developer's Handbook", Prentice Hall, 2012
2. R. Keith Mobley, "An Introduction to Predictive Maintenance", Butterworth-Heinemann, 2012
3. Tom M. Mitchell, "Machine Learning", McGraw-Hill, 1997
4. Erik Hellman, "Android Programming: Pushing the Limits", Wiley & Sons, 2014
5. Anders Göransson, "Efficient Android Threading", O'Reilly, 2014
6. Phil Dutson, "Android Development Patterns", Addison-Wesley, 2016
7. Luigi Atzori et al., "The Internet of Things: A Survey", 2010
8. C.K.M. Lee, Yi Chao and Kam Hung Ng, "Big Data Analytics for Predictive Maintenance Strategies", 2017
9. Paul Bowman, Jason Ng, Mark Harrison, Tomás Sánchez López and Alexander Ilic, "Sensor-based Condition Monitoring", 2009
10. Mardiana Mohamad Noor and Wan Haslina Hassan, "Current Threats of Wireless Networks", 2008
11. Ruben Sipos et al., "Log-based Predictive Maintenance", 2014
12. Carles Gomez, Joaquim Oller and Josep Paradell, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology", 2012
13. "Internet of Things: Wireless Sensor Networks", International Electrotechnical Commission, 2014
14. "The Intel Mote Platform: A Bluetooth-based Sensor System for Industrial Monitoring", Intel Corporation, 2005
15. Bluetooth Core Specification v5.2, Bluetooth SIG
16. DA14583 Datasheet, Dialog Semiconductor
17. BMI160 Datasheet, Bosch Sensortec GmbH
18. DS1347 Datasheet, Maxim Integrated
19. HDC1008 Datasheet, Texas Instruments
20. FT230X Datasheet, FTDI Ltd.

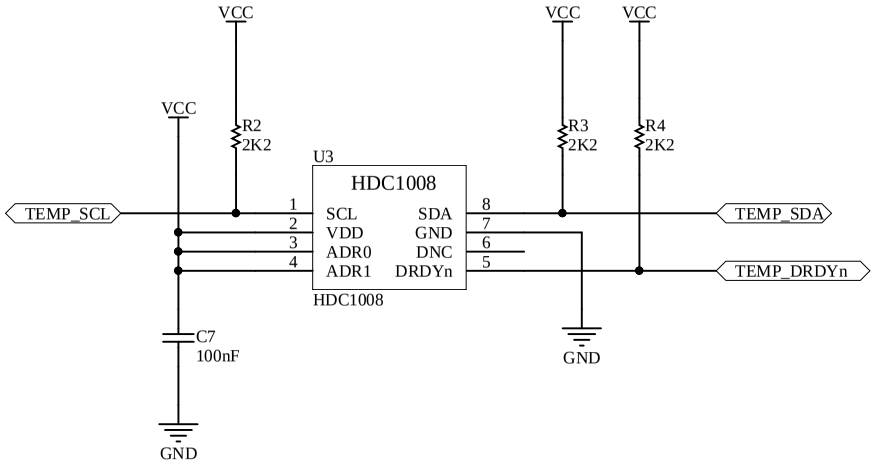
Παράρτημα Α:
Σχηματικό διάγραμμα Temperature Sensor



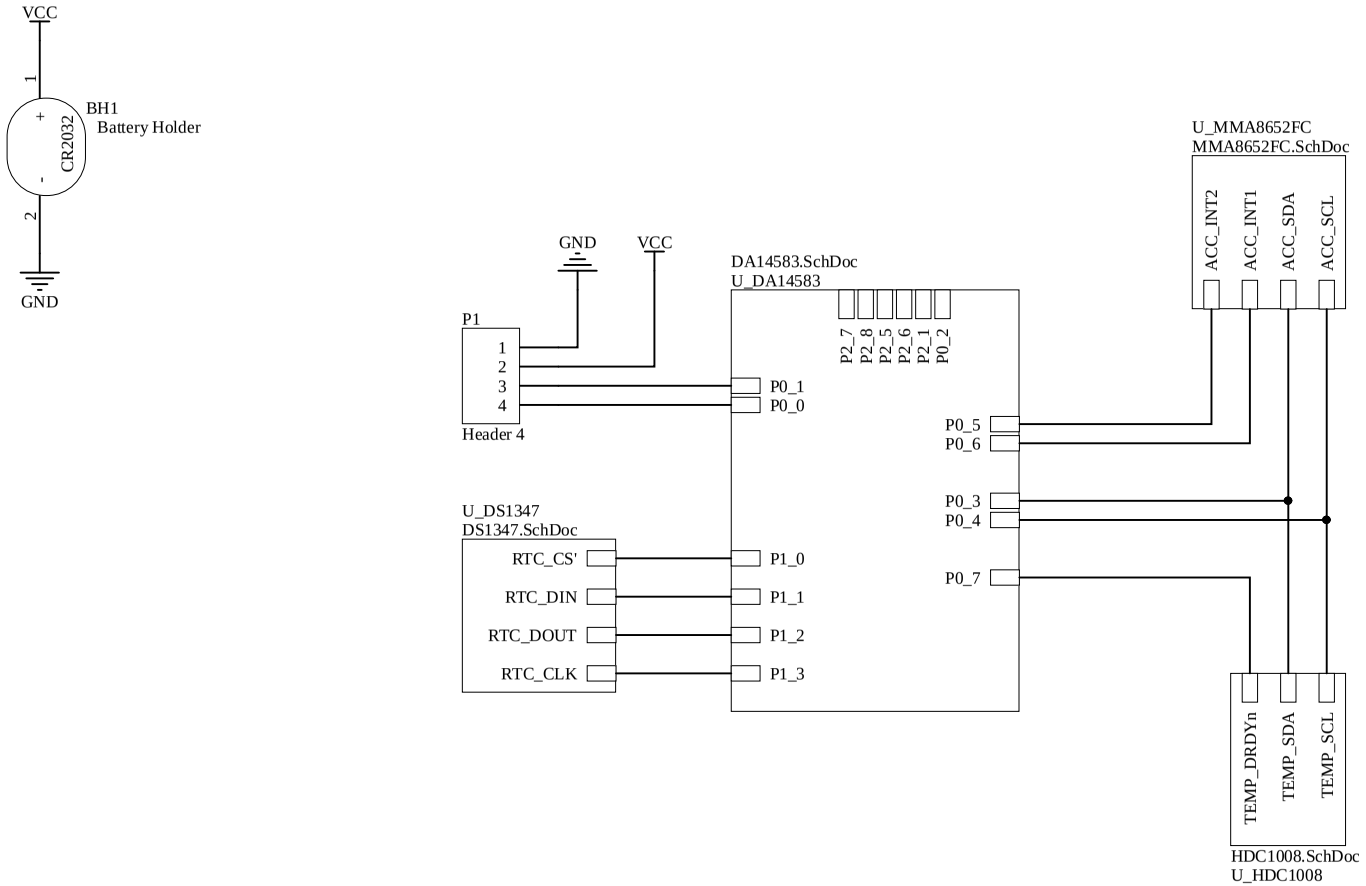
Title	**DA14583 BLE SOC**	
Size: A4	Number	Revision
Date: 10/6/2016	Sheet of	
File: C:\Users\..\DA14583.SchDoc	Drawn By:	



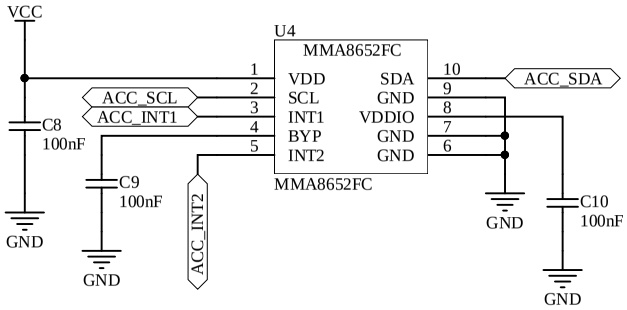
Title	**DS1347 Real-Time Clock**		
Size: A4	Number:		Revision:
Date:	10/6/2016	Sheet of	
File:	C:\Users\..\DS1347.SchDoc	Drawn By:	



Title	**HDC1008 Temperature - Humidity Sensor**		
Size A4	Number		Revision
Date:	10/6/2016	Sheet of	
File:	C:\Users\..\HDC1008.SchDoc	Drawn By:	

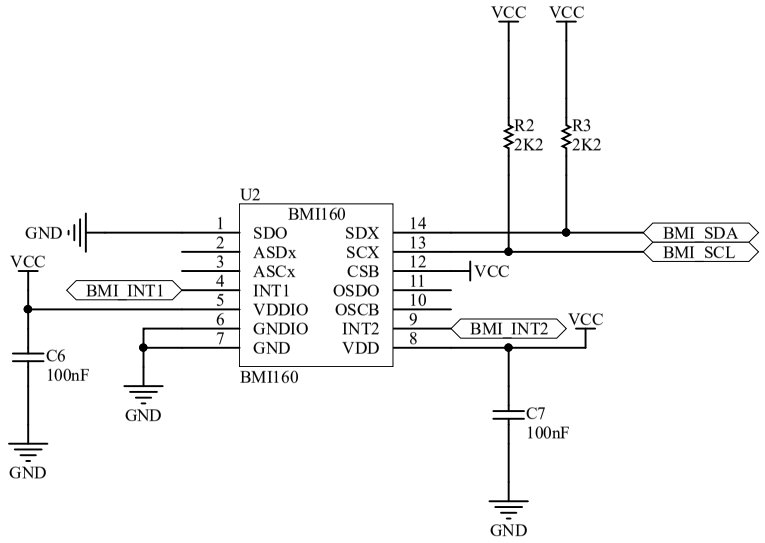


Title		
Size: A4	Number	Revision
Date: 10/6/2016		Sheet of
File: C:\Users\..\Main.SchDoc		Drawn By:

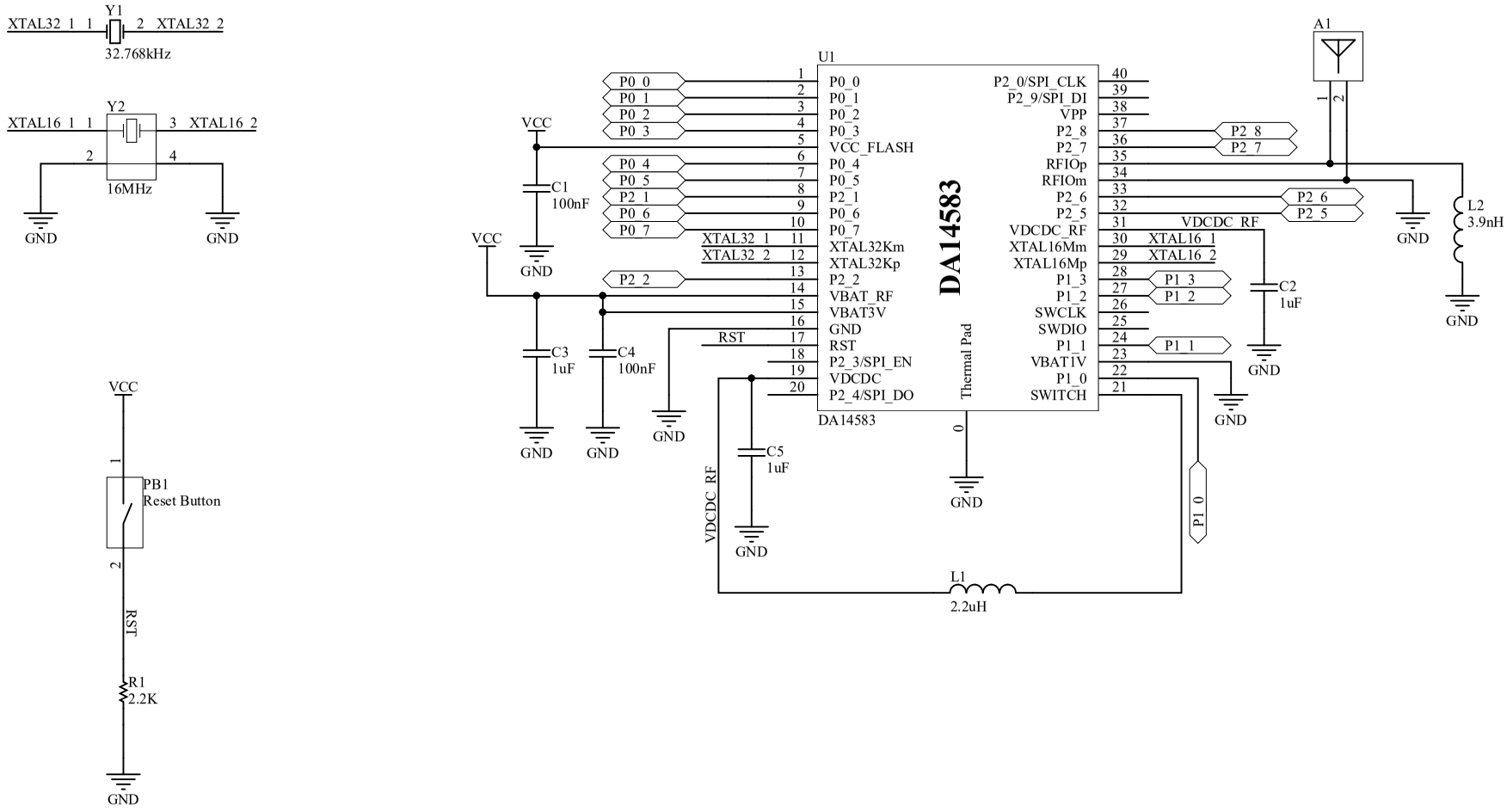


Title		
Size: A4	Number	Revision
Date: 10/6/2016		Sheet of
File: C:\Users\..\MMA8652FC.SchDoc		Drawn By:

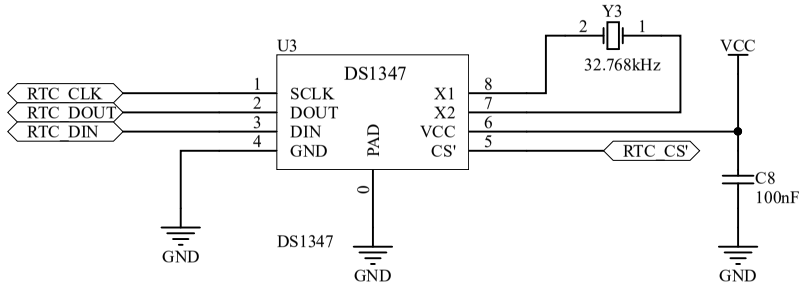
Παράρτημα Β:
Σχηματικό διάγραμμα Accelerometer Sensor



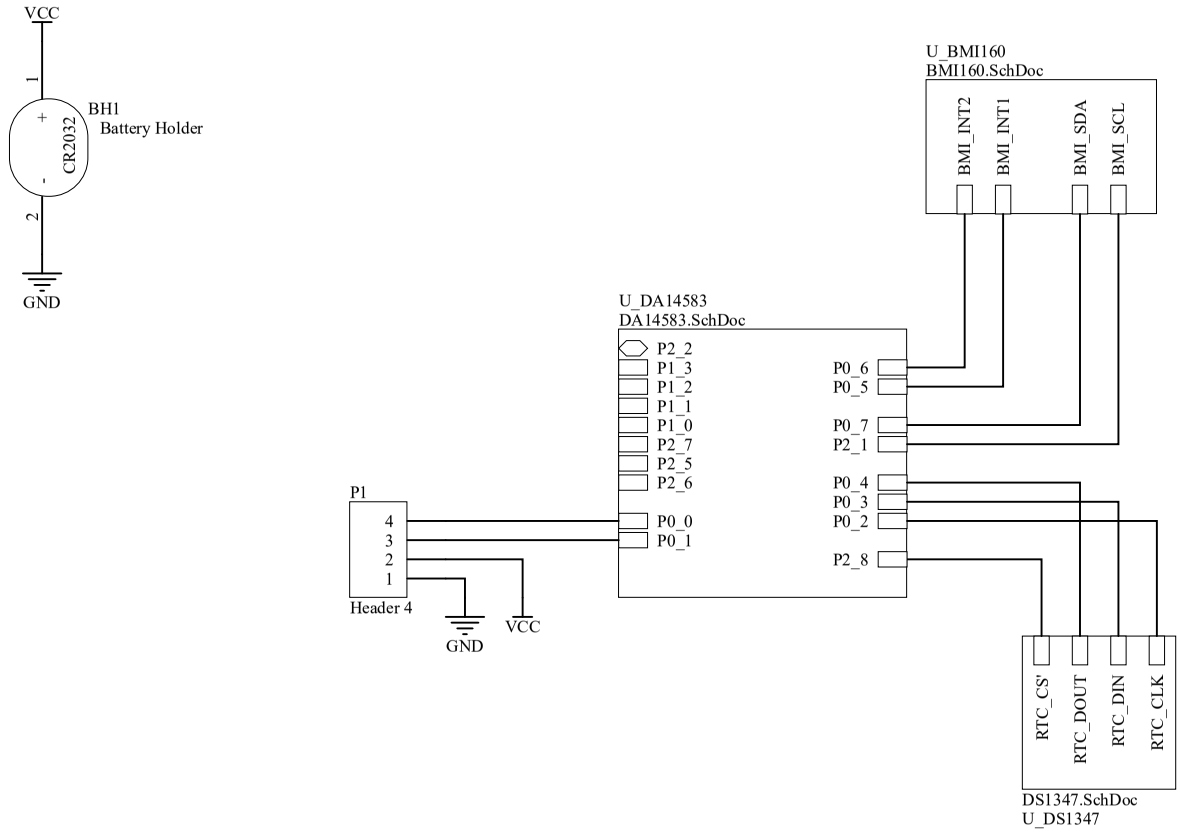
Title		
Size A4	Number	Revision
Date: 27/3/2016		Sheet of
File: C:\Users\..\BMI160.SchDoc		Drawn By:



Title	**DA14583 BLE SOC**	
Size: A4	Number	Revision
Date: 27/3/2016		Sheet of
File: C:\Users\..\DA14583.SchDoc		Drawn By:



Title	**DS1347 Real-Time Clock**		
Size A4	Number		Revision
Date:	27/3/2016	Sheet of	
File:	C:\Users\..\DS1347.SchDoc	Drawn By:	



Title	**COIN SIZE ACCELEROMETER SENSOR**	
Size: A4	Number:	Revision:
Date: 27/3/2016		Sheet of
File: C:\Users\..\Main.SchDoc		Drawn By: