Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Παιχνίδια, ψυχαγωγικές και εκπαιδευτικές διαδραστικές εγκαταστάσεις με τη χρήση 'διάχυτων' υπολογιστικών συστημάτων

Πρωτόπαπα Αναστασία
Α.Μ : 3735

Επιβλέπων: Καθηγητής Σπυράκης Παύλος
Συνεπιβλέπων: Δρ Χατζηγιαννάκης Ιωάννης

Πάτρα, Σεπτέμβριος 2011

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Παύλο Σπυράκη, Καθηγητή του τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και επιβλέποντα καθηγητή μου για την δυνατότητα να εντρυφήσω σε ένα πολύ ενδιαφέρον γνωστικό πεδίο. Θα ήθελα επίσης να ευχαριστήσω το συνεπιβλέποντα Δρ. Χατζηγιαννάκη Ιωάννη για την εμπιστοσύνη του, την βοηθειά του, τις γνώσεις που μου μετέδωσε, όπως και επίσης την συνεχή παρότρυνση του.

Επίσης, θέλω να ευχαριστήσω τον Ακριβόπουλο Ορέστη για την υπομονή του, την μεγάλη υποστήριξη και βοήθεια που μου χάρισε καθ'όλη την διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας. Επίσης και τον Γεωργιτζίκη Βασίλειο για την συναδελφική του συμπαράσταση.

Περίληψη

Η παρούσα διπλωματική εργασία ασχολείται με την δημιουργία και την εγκατάσταση ψυχαγωγικών και εκπαιδευτικών διαδραστικών εγκαταστάσεων. Αρχικά παρουσιάζεται η τεχνολογία και τα χαρακτηριστικά των διάχυτων υπολογιστικών συστημάτων όπως και επίσης το περιβάλλον γραφικών Processing.org. Στη συνέχεια συστήνεται και αναλύεται ο σχεδιασμός και η υλοποιήση μιας βιβλιοθήκης για τον over-the-air προγραμματισμό και έλεγχο των αισθητήρων, με σκοπό την εύκολη υλοποιήση διαδραστικών εφαρμογών. Τέλος εκτίθεται ένα πρότυπο παιχνίδι χρησιμοποιώντας τα παραπάνω εργαλεία.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο και Σημασία

Οι υπολογιστες ξεκίνησαν σαν τεράστια μαύρα κουτιά που επεξεργάζονταν κεντρικά τις πληροφορίες, τις οποίες παρείχε ο χρήστης με τους τυπικούς τρόπους αλληλεπίδρασης - ποντίκι, πλητρολόγιο. Τα τελευταία όμως χρόνια έχει κυριαρχήσει η τάση για την συνένωση του φυσικού κόσμου με το ψηφιακό, όπου η επεξεργασία πληροφορίας έχει ενσωματωθεί σε καθημερινά αντικείμενα τα οποία ο άνθρωπος χρησιμοποιεί και επηρεάζει μέσα από τις δραστηριότητες του. Η νέα αυτή γενική κατεύθυνση έχει επηρέασει πολλούς τομείς, όπως για παράδειγμα την βιομηχανία, τον αθλητισμό και την ιατρική.

Ένας σημαντικός τομέας που έχει επίσης δραματικά επηρεαστεί είναι αυτός των ηλεκτρονικών παιχνιδιών. Τα παιχνίδια αποτελούν ένα μεγάλο και σημαντικό κομμάτι της βιομηχανίας των υπολογιστών και η εξέλιξη τους έχει ακολουθήσει παράλληλο δρόμο με την πρόοδο των ηλεκτρονικών υπολογιστών. Έχουν γίνει αρκετές προσπάθειες εισαγωγής των διάχυτων υπολογιστικών συστημάτων στην ηλεκτρονική διασκέδαση με κύριο σκοπό την αντικατάσταση του πληκτρολογίου και του ποντικιού με συσκευές που αναγνωρίζουν την τοποθεσία και την κίνηση των παιχτών στο φυσικό χώρο. Ιδιαίτερα αντιπροσωπευτικά και γνωστά παραδείγματα είναι το Wii της Nintedo και το Kinect της Microsoft.

Σε έναν ακόμα τομέα που έχει επιρροή η νέα μόδα που περιγράφουμε, είναι αυτός της κινητής τηλεφωνίας. Έχουν αναπτυχθεί πολλές mobile εφαρμογές οι οποίες στηρίζονται στην αντίληψη του περιβάλλοντος και της συμπεριφορά του χρήστη και έχουν μεταφέρει τα ηλεκτρονικά παιχνίδια απο τους τυπικούς υπολογιστές στα κινητά τηλέφωνα.

Όλες αυτές οι εξελίξεις έχουν θέσει τις βάσεις για ένα νέο είδος ψυχαγωγίας που αποτελείται από πανταχού παρών ψηφιακές μονάδες, χαρακτηρίζεται

“mixed-reality” και περιγράφεται από τον όρο Pervasive Games .

1.2 Στόχοι της διπλωματικής εργασίας

Στόχος της παρούσας διπλωματικής εργασίας είναι η αναπτυξη ψυχαγωγικών και διαδραστικών εγκαταστάσεων με την χρήση διάχυτων υπολογιστικών συστημάτων. Ουσιαστικά ο στόχος είναι η αξιοποιήση των υπάρχοντων τεχνολογιών ασύρματων αισθητήρων για την δημιουργία εφαρμογών.

Οι εφαρμογές αλληλεπίδρασης αυτές, θα χρησιμοποιούν συσκευές που υποστηρίζουν ασύρματη επικοινωνία με πομποδέκτες που ακολουθούν το πρότυπο IEEE 802.15.4. Οι αισθητήρες αυτοί χαρακτηρίζονται απο ετερογένεια στον τρόπο με τον οποίο προγραμματίζονται, στις υπηρεσίες που προσφέρουν και στον τρόπο επικοινωνίας με τον κεντρικό υπολογιστή. Ευθύνη λοιπόν αυτής της διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποιήση μιας βιβλιοθήκης που θα προσφέρει ένα σύνολο λειτουργιών για τον ενιαίο προγραμματισμό και έλεγχο των προτεινόμενων πλατφόρμων, με απώτερο στόχο την χρήση της με το περιβάλλον γραφικών Processing.org.

Η Processing.org, είναι μια open-source γλώσσα προγραμματισμού και ένα ψηφιακό εργαλείο που στηρίζεται στην Java και απλοποιεί την δημιουργία διαδραστικών γραφικών.

Τέλος μελετήθηκαν οι συσκευές sunSPOT και Arduino για την ανάπτυξη λογισμικού για αυτά, με σκοπό την διαφανή επικοινωνία τους με τον κετρικό υπολογιστή και το περιβάλλον γραφικών, χρησιμοποιώντας την βιβλιοθήκη, και κυρίως την ενοποιήση των λειτουργιών τους και τον τρόπο χειρισμού τους.

1.3 Δομή της διπλωματικής εργασίας

Σε αυτό τον τομέα παρουσιάζεται η δομή της διπλωματικής εργασίας. Στην πρώτη ενότητα έγινε η εισαγωγική παρουσιάση της εργασίας. Στο δεύτερο κεφάλαιο παρουσιάζονται οι έννοιες Pervasive computing και Pervasive gaming ενώ δίνονται και κάποια παραδείγματα παιχνιδιών.

Στο τρίτο κεφάλαιο αναφέρονται και αναλύονται οι απαραίτητες τεχνολογίες για την εκπλήρωση των διαδραστικών εφαρμογών. Συγκεκριμένα στα υποκεφάλαια 3.2 και 3.3 αναφέρονται οι βασικοί αισθητήρες που χρησιμοποιεί αυτή η εργασία - SunSPOT και Arduino . Στο υποκεφάλαιο 3.4 παρατίθεται μια στοιχειώδης σύγκριση μεταξύ των δύο συσκευών. Τελος στο 3.5 παρουσιάζεται η Processing.org και δίνονται επίσης κάποια sketches .

Στο κεφάλαιο 4 παρουσιάζεται το application programming interface , η αρχιτεκοτινική, η δομή του κώδικα και το πρωτόκολλο για την διαφανή

επικοινωνία μεταξύ του υπολογιστή και των διάχυτων συσκευών.

Στο κεφάλαιο 5 παρουσιάζεται ένα παιχνίδι υλοποιημένο με τις παραπάνω τεχνολογίες που αναφέρθηκαν, αποτελώντας ένα βασικό παράδειγμα για τον συνδυασμό των παραπάνω εργαλείων. Στο κεφάλαιο 6 παρατίθεται το 'social network " των προγραμματιστών, το GitHub και στο τελευταίο κεφάλαιο παρουσιάζονται τα συμπεράσματα και οι προοπτικές που προέκυψαν απο την εκπόνηση της εργασίας αυτής.

Κεφάλαιο 2

Πανταχού παρών υπολογισμός και Παιχνίδια

2.1 Pervasive Computing

2.1.1 What is pervasive computing?

Ο όρος Pervasive Computing [2] συστήνει την νέα εποχή του υπολογισμού περιγράφοντας ένα μοντέλο αλληλεπίδρασης ανθρώπου - υπολογιστή, στο οποίο η επεξεργασία πληροφορίας έχει ενσωματωθεί στο φυσικό και καθημερινό περιβάλλον.

Καποιος μπορεί να χαρακτηρίζει τον διάχυτο υπολογισμό ως το αντίθετο της εικονικής πραγματικότητας. Εκεί που η εικονική πραγματικότητα εισάγει τον άνθρωπο στον ψηφιακό 'κόσμο', ο διάχυτος υπολογισμός αναγκάζει την τεχνολογία να 'ζήσει' με τους ανθρώπους. Ειδικότερα ο ορισμός αυτός θεωρεί ότι ο κοινός υπολογιστής έχει την ικανότητα να λάβει πληροφορίες σχετικά με την συμπεριφορά και την κίνηση των ανθρώπων, και έτσι να υιοθετήσει μια 'ευφυή' δράση μέσα στο χώρο συνύπαρξης τους.

Τα πληροφοριακά συστήματα θα είναι πανταχού παρόντα και μη ορατά, ώστε οι άνθρωποι να χρησιμοποιούν τις υπηρεσίες τους χωρίς να αντιλαμβάνονται την φυσική τους ύπαρξη. Ταυτόχρονα θα διαθέτουν στους χρήστες πληροφορίες και υπηρεσίες ανα πάσα στιγμή.

2.1.2 Pervasive computing technologies

Ο διάχυτος υπολογισμός συμπεριλαμβάνει τρεις συγκλίνουσες περιοχές της τεχνολογίας πληροφοριών και επικοινωνιών [3] : υπολογισμός (συσκευές), επικοινωνίες ('συνδεσιμότητα') και 'διεπαφές χρήστη'.

* **Υπολογισμός / Συσκευές:** Οι συσκευές μπορεί να είναι διαφόρων ειδών, μεγεθών και δυνατοτήτων. Θα μπορούν να επικοινωνούν μεταξύ τους και να φερθούν 'έξυπνα'. Αυτές οι συσκευές μπορούν να χωριστούν σε τρεις κατηγορίες:

    1. Αισθητήρες: συσκευές που συναισθάνονται το περιβάλλον και τις αλλαγές σε αυτό.
    2. Επεξεργαστές: ηλεκτρονικά συστήματα τα οποία αναλύουν δεδομένα.
    3. Ενεργοποιητές : συσκευές οι οποίες αντιδρούν στις επεξεργασμένες πληροφορίες επηρεάζοντας το περιβάλλον.

* **Συνδεσιμότητα:** Τα συστήματα του διάχυτου υπολογισμού θα βασίζονται στην σύνδεση των ανεξάρτητων συσκευών με μεγαλύτερα δίκτυα και φυσικά και το διαδίκτυο. Αυτό μπορεί να επιτευχθεί μέσω ενσύρματων τεχνολογιών, π.χ Ethernet, όσο και μέσω ασύρματων, π.χ Bluetooth και IEEE 802.15.4 standar.

* **'Διεπαφές χρήστη':** Οι διεπαφές χρήστη αντιπροσωπεύουν το σημείο επαφής ανάμεσα στον χρήστη και της τεχνολογιάς πληροφοριών και επικοινωννιών. Νέες μορφές διεπαφών χρήστη δημιουργούνται, που θα είναι ικανά να παρέχουν περισσότερες πληροφορίες για το περιβάλλον, στον προσωπικό υπολογιστή προς επεξεργασία.

2.1.3 Development

Ο διάχυτος υπολογισμός μπορεί να έχει μια πληθώρα εφαρμογών, πολλές από τις οποίες μπορεί να μην έχουν αναγνωριστεί ακόμα. Οι πιο συζητημένες είναι εφαρμογές στην υγεία, στις οικιακές λειτουργίες, την παρακολούθηση του περιβάλλοντος και τον αθλητισμό.

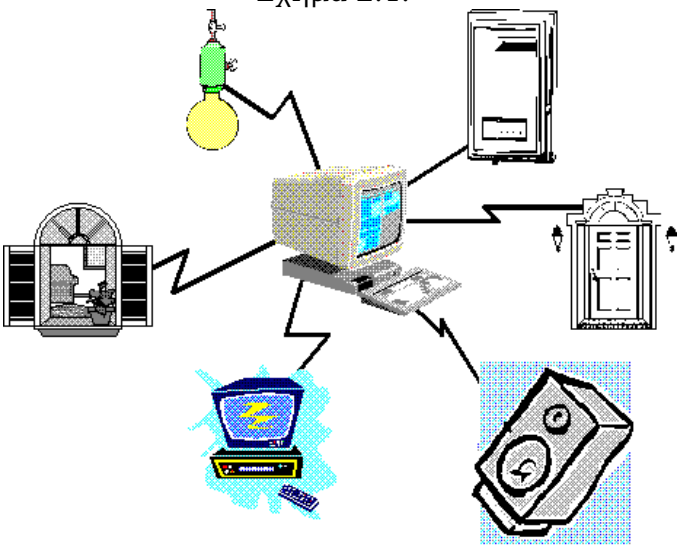
Αθλητισμός

Ο διάχυτος υπολογισμός έχει επηρεάσει και τον τομέα του αθλητισμού [4]. Αισθητήρες, συνδεδεμένοι είτε στον αθλητή είτε στα όργανα άθλησης, συλλέγουν πληροφορίες για την πρόοδο και την φυσική κατάσταση των ακούμενων. Η συλλογή αυτών των πληροφορίων και η ανάλυση τους συνδράμει στον τρόπο προπόνησης, απόδοσης αλλά και ψυχαγωγίας μέσα από την δραστηριότητα. Για παράδειγμα κάποια γυμναστήρια ενσωματώνουν διαδραστικές εφαρμογές κατά την εκτέλεση διαφόρων ασκήσεων.

Έξυπνα σπίτια [5]

Ένα έξυπνο σπίτι είναι μια συλλογή από δικτυωμένους αισθητήρες στο οικιακό περιβάλλον με σκοπό την αυτοματοποιήση της διαχείρησης του οικίματος. Το έξυπνο σπίτι αποτελείται από διάφορους αισθητήρες οι οποίοι έχουν διαλεχθεί ώστε να ελέγχουν συγκεκριμένα σημεία του σπιτιού, όπως για παράδειγμα τα παράθυρα ή τις διάφορες ηλεκτρικές συσκευές. Βασικός σκοπός της δημιουργίας των σπιτιών αυτών είναι η διευκόλυνση της ανθρώπινης καθημερινότητας καθώς και η διασκέδαση και η βελτίωση της ποιότητας της ζωής.

Σχήμα 2.1:



2.2 Pervasive Games

Τα παιχνίδια διάχυτου υπολογισμού έχουν γίνει πολύ δημοφιλή τον τελευταίο καιρό και είναι ένας ακόμα τομέας μελέτης. Στόχος τους είναι να φέρουν τον υπολογιστή στον πραγματικό κόσμο, ώστε να επαυξήσουν τον υπάρχοντα τομέα των ηλετρονικών παιχνιδιών και να εισάγουν εναλλακτικές μορφές ψυχαγωγίας.

Το παιχνίδι είναι ένας τρόπος έκφρασης και δημιουργίας μέσα από την αλληλεπίδραση με το σύνολο ή την ομάδα που συμμετέχει, ενώ ταυτόχρονα βελτιώνει και αναδεικνύει ατομικές ικανότητες και δεξιότητες. Τα παιχνίδια

διάχυτου υπολογισμού προσπαθούν να διατηρήσουν ακριβώς αυτά τα χαρακτηριστικά αποδεσμεύοντας τον παίχτη απο την κονσόλα και τοποθετώντας τον πάλι στο φυσικό περιβάλλο.

2.2.1 Σχεδιαστικές προσεγγίσεις παιχνιδιών διάχυτου υπολογισμού

Ένα από τα βασικά ερωτήμα στα παιχνίδια διάχυτου υπολογισμού είναι το πως τα σχεδιάζεις. Υπάρχουν διάφορες αντιλήψεις πάνω σε αυτή την ερώτηση.

- Σχεδιασμός Παιχνιδιού διάχυτου υπολογισμού σαν Κλασσικό Παιχνίδι

Η πρώτη και η πιο προφανής προσέγγιση σχεδιάσης παιχνιδιών διάχυτου υπολογισμού είναι η χρήση των κλασσικών παιχνιδιών ως μεταφορά. Αυτό σημαίνει την μετατροπή του φυσικού περιβάλλοντος σε επιτραπέζιο παιχνίδι, και μεταφράζοντας/μεταβάλλοντας τους κανόνες ώστε να ταιριάζουν με το νέο πλαίσιο. Αυτή η προσέγγιζει ταιριάζει σε μια ευρύτερη ανάπτυξη όπου το παιχνίδι γίνεται μια φυσική δραστηριότητα.

- Σχεδιασμός Παιχνιδιού διάχυτου υπολογισμού ως μέσο για την παραγωγή Τέχνης

Μια άλλη σημαντική προσέγγιση των παιχνιδιών διάχυτου υπολογισμού είναι η προσέγγιση και η αντίληψη τους ώς θέατρο. Πολλά παιχνίδια διάχυτου υπολογισμού βασίζονται και εκτελλίσονται γύρω από ένα γεγονός και πολλές φορές περιλαμβάνουν και ηθοποιούς. Η βασική διαφορά μεταξύ αυτών των παιχνιδιών και του κλασσικού θεάτρου είναι ότι στα πρώτα το κοινό έχει άμεση συμμετοχή, και αντί για ένα σενάριο, υπάρχει ένα σύνολο κανόνων που τόσο και οι ηθοποιοί όσο και το κοινό πρέπει να ακολουθούν.

- Σχεδιασμός Παιχνιδιού διάχυτου υπολογισμού ως προέκταση αστικού πολιτισμού

Κάποιος θα μπορούσε να αντιληφθεί τα παιχνίδια διάχυτου υπολογισμού ως ένα μέσο για να προστεθεί περισσότερη διασκέδαση σε καθημερινές αστικές περιστάσεις.

2.2.2 Παραδείγματα Παιχνιδιών Διάχυτου Υπολογισμού

Στην ενότητα αυτή παρουσιάζονται μια σειρά απο αλληλεπιδριστικά παιχνίδια.

- Human Pacman [6]

Σχήμα 2.2:

Είναι μια διαδραστική εναλλακτική μορφή του γνωστού ηλεκτρονικού παιχνιδιού Pacman. Αναπτύχθηκε από το εργαστήριο 'Mixed Reality" του Εθνικού Πανεπιστημίου της Σιγκαπούρης. Οι παίχτες αλληλεπιδρούν μεταξύ τους και με το ψηφιακό περιβάλλον "PacWorld". Χρησιμοποιώντας ειδικούς φορέσιμους υπολογιστές, ενισχύτεται η οπτική πραγματικότητα των παιχτών. Ένας παίχτης δρα ως PacMan ενώ οι άλλοι δρουν ως Ghosts, ανιχνεύοντας ο ενας τον άλλον στον πραγματικό κόσμο.

- Uncle Roy All Around you [7]

Είναι ένα παιχνίδι που εξελλίσεται τόσο στον πραγματικό κόσμο - στους δρόμους μιας πόλης - όσο και στον εικονικό κόσμο. Οι παίχτες που βρίσκονται

στο εξωτερικό περιβάλλον χρησιμοποιούν ηλεκτρινικούς υπολογιστές χειρός, και ψάχνουν μέσα στην πόλη τον Uncle Roy με την καθοδήγηση τόσο ενός χάρτη, όσο και μηνυμάτων που ανταλλάσονται με τους συμπαίχτες. Αντίστοιχα οι παίχτες που παίζουν μεσω του διαδικτύου, περιηγούνται σε έναν εικονικό χάρτη της ίδιας περιοχής , ωστε να παρέχουν στους υπόλοιπους διάφορα μυστικά μονοπάτια.

- Hot Potato [8]

Κάθε παίχτης κατέχει μια συκευή, η οποία έχει την δυνατότητα να παράγει μια Hot Potato η οποία μπορεί ανα πάσα στιγμή να εκραγεί και να αποκλίσει τον αντίστοιχο παίχτη. Στόχος κάθε παίχτη είναι να μεταβιβάσει την Hot Potato του σε κάποιον γειτονικό παίχτη ωστε να επιβιώσει ο ίδιος και να παραμείνει ενεργός.

Σχήμα 2.3: Στιγμιότυπο από το παιχνίδι Ήot Potato"

- Sensors Enabling Automatic Generation of Electronic music [9]

Το τμήμα της επιστήμης των υπολογιστών του Cincinatti πανεπιστημίου δημιούργησαν μια εφαρμογή, όπου η μουσική καθορίζεται και αλλάζει από τις κινήσεις ενός χορευτή στο χώρο. Τοποθετήθηκαν 23 αισθητήρες σε έναν χορευτικό χώρο ενώ δύο αισθητήρες συνδέονται στα χέρια κάθε χορευτή. Μέσα από αυτό το σύστημα των αισθητήρων, συλλέγονται πληροφορίες για τις κινήσεις στο χώρο, και στέλνονται σε ένα κεντρικό υπολογιστικό σύστημα, το οποίο διαχειρίζεται την μουσική που παίζεται.

- Aarhus by Light

Είναι ουσιαστικά μια αλληλεπιδραστική προσωψη ενός κτιρίου - του Concert Hall Aarhus - . Συγκεκριμένα ‘ μέσα’ στην πρόσωψη ζουν φανταστικά ψηφιακά πλάσματα που αλληλεπιδρουν με τους περαστικούς. Τα πλάσματα αυτά χαρακτηρίζονται απο κοινωνικότητα και τις περισσότερες φορές χαίρονται να αλληλεπιδρουν με τον κόσμο

Σχήμα 2.4: Στιγμιότυπο από το Aarhus by Light

Σχήμα 2.5: Στιγμιότυπο από το Aarhus by Light

Κεφάλαιο 3

Εργαλεία

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιαστούν οι συσκευές/αισθητήρες που μελετήθηκαν και χρησιμοποιήθηκαν για την ανάπτυξη της βιβλιοθήκης που παρουσιάζεται στο επόμενο κεφάλαιο, όπως επίσης και το περιβάλλον γραφικών Processing.org .

3.2 SunSPOTs

Figure 3.1: SunSPOT

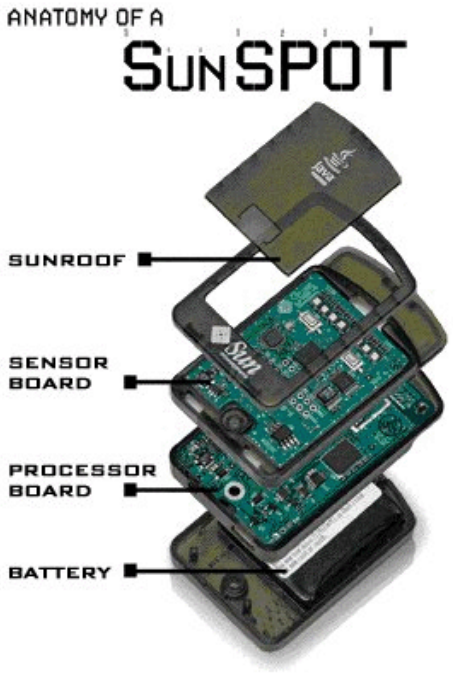
Το SunSPOT [10] είναι μια μικρή ασύρματη, πειραματική πλατφόρμα. Προγραμματίζεται σχεδόν εξόλοκληρου σε Java επιτρέποντας έτσι την χρήση της στην ανάπτυξη εφαρμογών, χωρίς την ανάγκη για ιδιαίτερες γνώσεις στον προγραμματισμό ενσωματωμένων συστημάτων. Η ευελιξία που το χαρακτηρίζει, δίνει την δυνατότητα για πειραματισμό και καινοτομία.

Κάθε SPOT kit περιέχει δύο ασύρματα SunSPOTs και ένα Basestation SunSPOT, όπως και επίσης διάφορα εργαλεία για την ανάπτυξη εφαρμογών.

Το BaseStation SunSPOT συνδεέται με τον υπολογιστή μέσω ενός καλωδίου USB, εφοδιάζοντας τον με IEEE 802.15.4 radio ώστε να μπορεί να επικοινωνήσει με τα remote SunSPOTs και γενικά συσκευές που υποστηρίζουν αυτό το πρότυπο.

3.2.1 Hardware

Ένα ολοκληρωμένο ασύρματο SunSPOT θα πρέπει να χωράει στην παλάμη ενός χεριού. Έχει κοινά χαρακτηριστικά με τα κινητά τηλέφωνα και συγκεκριμένα αποτελείται απο έναν processor board με radio, έναν sensor board και μια μπαταρία. Η πλήρης ανατομία του φαίνεται στο παρακάτω σχήμα.



Σχήμα 3.2: Τα διαφορετικά επίπεδα Hardware ενός remote SunSPOT

Processor Board

Η Processor Board απαρτίζεται από :

* Κεντρικό επεξεργαστή
* Μνήμη
* 802.15.4 radio με ενσωματωμένη κεραία
* σύνδεση USB

Ο επεξεργαστής είναι ένας ATMEL ARM920T των 32-bit και με συχνότητα λειτουργίας στα 180MHz. Η μνήμη αποτελείται από μια RAM των 512K και μια flash των 4M και τα περιεχόμενα της διατηρούνται μόνο για όσο η συσκευή τροφοδοτείται μέσω της μπαταρίας ή της σύνδεσης με το PC. Με τον επεξεργαστή και το radio ενεργά η μπαταρία μπορεί να υποστηρίξει μέχρι και 7 ώρες λειτουργίες. Η διάρκεια ζωής της μπαταρίας μπορεί να επιμηκινθεί με το μηχανισμό διατήρησης ενέργειας που διαθέτει κάθε SPOT. Τέλος η ασύρματη επικοινωνία υποστηρίζεται από τον TICC2420 πομποδέκτη, ο οποίος είναι και 802.15.4 συμβατός.

Κάθε SunSPOT έχει μια μοναδική IEEE 64-bit διεύθυνση, η οποία εκφράζεται με 4 σύνολα των 4 δεκαεξαδικών ψηφίων. Τα πρώτα 8 ψηφία είναι πάντα 0014.4F01 και τα υπόλοιπα 4 αναγράφονται πάνω σε κάθε SunSPOT.

Sensor Board

Η eDemo πλακέτα παρέχει τους παρακάτω ενεργοποιητές/αισθητήρες:

* επιταχυνσιόμετρο τριών αξόνων
* αισθητήρα θερμότητας
* αισθητήρα φωτός
* 8 RGB LEDs
* 6 αναλογικές εισόδους
* 2 διακόπτες
* 5 γενικής χρήσης pins
* 4 pins υψηλού δυναμικού

3.2.2 Software

Το SunSPOT χρησιμοποιεί μια έκδοση της Java Micro Edition Virtual Machine και συγκεκριμένα την έκδοση Squawk [11]. Το Squawk παρέχει τα βασικά ενός λειτουργικού συστήματος.

Οι εφαρμογές που αναπτύσσονται για τα SunSPOTs ακολουθούν τις προδιαγραφές MIDP που χτίζεται πάνω στο CLDC. Τα προγράμματα που ακολουθούν αυτές τις προδιαγραφές ονομάζονται MIDlets και έχουν συγκεκριμένη δομή. Έτσι κάθε εφαρμογή θα πρέπει να κληρονομεί τα στοιχεία της κλάσης MIDlet και να υλοποιεί τις παρακάτω μεθόδους:

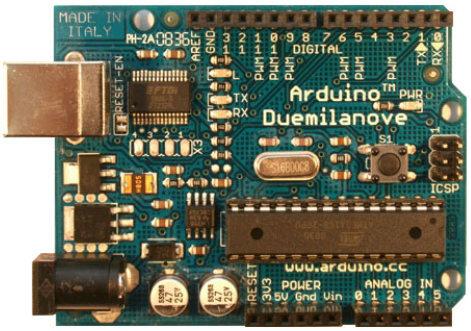
- startApp()
- PauseApp()
- destroyApp()

Για τον προγραμματισμό των SunSPOTs χρησιμοποιείται το Sun SPOT SDK που προσφέρει ένα σύνολο βιβλιοθηκών για τον πλήρη χειρισμό των συσκευών. Ονομαστικά οι βιβλιοθήκες είναι:

* Device Library : βρίσκεται στο spotlib_ device.jar και spotlib_ common.jar, και περιέχει drivers για:
  - τα On-Board LEDs
  - τους διαύλους PCI, USART και τους μετρητές του επεξεργαστή
  - τον πομποδέκτη
  - την flash μνήμη

* Radio Communication Library:πρόκειται για την βιβλιοθήκη που είναι υπεύθυνη για την δημιουργία και τον έλεγχο όλων των συνδέσεων και πρωτοκόλλων και βρίσκεται στο multihoplib_ rt.jar.

* Sensor Board Library: πρόκειται για την βιβλιοθήκη που περιέχει όλες τις κλάσεις και τα interfaces που χρειάζονται για την διαχείρηση των αισθητήρων και βρίσκεται στο transducerlib_ rt.jar



Σχήμα 3.3: Arduino board

3.3 Πλατφόρμα Arduino

Το Arduino ([12], [13]) είναι μια open-source πλατφόρμα βασισμένη σε έναν απλό μικροεπεξεργαστή , και ένα απλό περιβάλλον ανάπτυξης λογισμικού. Μπορεί να χρησιμοποιηθεί για την κατασκευή διαδραστικών εφαρμογών, λαμβάνοντας εισόδους από μια ποικιλία αισθητήρων και επηρεάζοντας το περιβάλλον του ελέγχοντας διάφορους ενεργοποιητές.

Hardware

Ένα Arduino αποτελείται από έναν μικροεπεξεργαστή ATMEL AVR των 8-bit. Συγκεκριμένα το Arduino Duemilanove το οποίο εξετάστηκε σε αυτή την διπλωματική εργασία έχει έναν ATmega168 ή έναν ATmega328 με συχνότητα λειτουργίας στα 16MHz. Επίσης έχεις 14 digital-pins, 6 analog-pins, μία USB σύνδεση και ένα διακόπτη επαναφοράς. Η μνήμη του ειναι μια Flash των 16KB, μία SRAM του 1KB και μία EEPROM των 512 bytes.

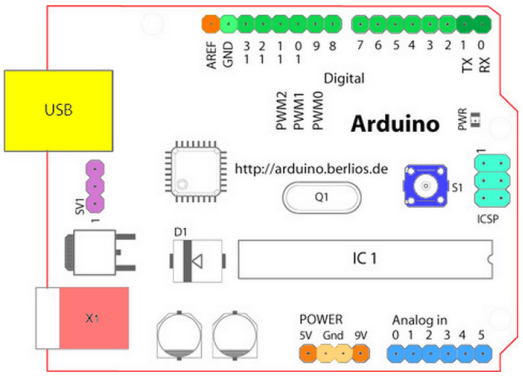
Μπορεί να τροφοδοτηθεί είτε από USB σύνδεση, είτε από εξωτερική πηγή και η προτεινόμενη τάση για την σωστή λειτουργία είναι 7-12V.

Pins : Στο σχήμα 3.4 φαίνονται όλα τα pins της πλατφόρμας των οποίων δίνουμε μια συνοπτική περιγραφή.

+ Pins τροφοδοσίας: **VIN, 5V, 3V3, GND** (πορτοκαλί χρώμα)
+ I/O Pins: **Digital**, κάθε ένα μπορεί να χρησιμοποιηθεί ως είσοδος ή έξοδος, ενώ κάποια από αυτά προσφέρουν και κάποιες ιδιαίτερες λειτουργίες(πράσινο χρώμα)

+ Αναλογικά Pins: **AnalogIn**, λειτουργούν σαν είσοδοι και έχουν εύρος ανάλυσης μέχρι και 1024 διαφορετικές τιμές(μπλέ χρώμα)

+ Ειδικά Pins: **IC, AREF, Reset**(κόκκινο χρώμα)



Σχήμα 3.4: Χρωματική σημείωση των Pins του Arduino

Software

Η Arduino πλατφόρμα δεν διαθέτει κάποια έκδοση λειτουργικού συστήματος όπως τα SunSPOT που αναφέραμε, αλλά προσφέρει διάφορες βιβλιοθήκες που κάνουν τον προγραμματισμό του εξίσου εύκολο. Για την ανάπτυξη προγραμμάτων για την πλατφόρμα παρέχεται το Arduino IDEA. Περιλαμβάνει έναν editor και χρησιμοποιείται και για την μεταγλώττιση και το ανέβασμα προγραμμάτων στον μικροεπεξεργαστή. Τα Arduino προγράμματα είναι γραμμένα σε C/C++, αλλά οι χρήστες χρειάζεται να όρισουν μόνο δυο μεθόδους:

+ setup(): τρέχει μόνο μια φορά στην αρχή του προγράμματος και αρχικοποιεί μεταβλητές περιβάλλοντος

+ loop(): καλείται επαναλαμβανόμενα μέσα στο πρόγραμμα μέχρι να κοπεί η τροφοδοσία στην πλατφόρμα

Πέρα απο αυτές τις δυο βασικές μεθόδους, η διασύνδεση για τον προγραμματισμό του Arduino board περιέχει και άλλες που κατηγοριοποιούνται σε structure, values, functions. Συγκεκριμενα:

- **Structure** Παρέχονται όλες οι δομες ελεγχου ροής, αριθμητικών και λογικών πράξεων όπως για παραδειγμα: **if, if..else,**
- **Values** Παρέχονται όλες οι δομες δεδομένων όπως για παραδειγμα: **boolean, char, int, byte, long**
- **Functions** Παρέχονται όλες οι μέθοδοι για την διαχείρηση των pin και γενικα των εσωτερικών λειτουργιών του μικροεπεξεργαστή, όπως για παράδειγμα: **pinMode(), digitalWrite(), digitalRead(), analogRead(), tone(), interrupts()**, κ.α

Το Arduino δεν έχει ενσωματομένο radio για την υλοποιήση ασύρματης επικοινωνίας. Συνδέοντας την πλατφόρμα με το XBee Radio, μπορούμε να προσθέσουμε ασύρματη επικοινωνία βασισμένη στο IEEE 802.15.4 πρωτόκολλο.

XBee Radio

Σχήμα 3.5: XBee Modem

Το ΩBee Radio ([14], [15]) βασίζεται στο πρότυπο IEEE 802.15.4. και υποστηρίζει την ανάγκη για low-cost και low-power δικτύωση. Απαιτούν ελάχιστη τάση λειτουργίας (3.3V) και παρέχουν αξιόπιστη παράδοση δεδομένων μεταξύ των συσκευών και μπορούν να υποστηρίξουν multi-point δίκτυα.

Μπορούν να λειτουργήσουν με δύο τρόπους, είτε σε transparent-data-mode όπου τα δεδομένα στέλνονται στους δέκτες σειριακά, είτε σε API-mode όπου τα δεδομένα πακετάρονται σε frames για να αποσταλλούν. Ο δεύτερος τρόπος λειτουργίας υποστηρίζει και broadcast όσο και unicast. Τέλος για να ελεχθούν και να τεθούν οι ρυθμίσεις της συσκευής χρησιμοποιούνται AT-commands.

Για να μπορέσει ένα Arduino board επικοινωνήσει με ένα XBee Radio θα πρέπει να συνδεθεί με ένα XBeeShield [16] (βλ. εικόνα 3.6)

Σχήμα 3.6: XBee Modem συνδεδεμένο σε ένα XBeeShield

3.4 Σύγκριση

Παρατίθεται μια σύγκριση ανάμεσα στις δύο πλατφόρμες που παρουσιάστηκαν παραπάνω.

	Arduino	SunSPOT
Επεξεργαστής	Atmega328	ARM920T
Συχνότητα Λειτουργίας	16	180
Μνήμη Flash	32KB	4MB
Μνήμη RAM	16KB	512KB
Προγραμματισμός	C++	J2ME

Πίνακας 3.1: Σύγκριση Πλατφορμών σύμφωνα με το υλικό

Κάποιες από τις βασικές δυνατότητες κάθε συσκευής παρατίθενται στον παρακάτω πίνακα. Το Arduino προσφέρει εύκολη διασύνδεση με μια ποικιλιά αισθητήρων και γενικα ηλεκτρονικών κυκλωμάτων γεγονός που το κάνει μια καλή επιλογή ως actuator. Όμως προσφέρει από την πλευρά λογισμικού προσφέρει μόνο μια απλή διεπαφή για την καταμέτρηση των τιμών των αισθητήρων. Από την άλλη μεριά το SunSPOT έχει ήδη κάποια συνδεδεμένα στοιχεία, αλλά δεν είναι ευκολα επεκτάσιμο με άλλα ηλεκτρονικά κυκλώματα. Το γεγονός οτι έχει δικο του λειτουργικό του παρέχει την δυνατότητα να αντιλαμβάνεται ειδικές καταστάσεις στο περιβάλλον, όπως για παράδειγμα ότι η θερμοκρασία ξεπέρασε κάποιο προκαθορισμένο κατώφλι.

	τροφοδοσία	υπηρεσίες	επεκτασιμότητα
SunSPOT	ενσωματωμένη μπαταρία	καταγραφή μέτρησης/ events	μικρή
Arduino	εξωτερική μπαταρία	καταγραφή μέτρησης	υψηλή

Πίνακας 3.2: Σύγκριση σε δυνατότητες

3.5 Processing.org

Σχήμα 3.7: Processing Logo

Η Processing ([17], [18]) είναι μια γλώσσα ανοιχτού λογισμικού για την δημιουργία εικόνων και διαδραστικών γραφικών, που βασίζεται στην Java. Περιλαμβάνει ένα βασικό περιβάλλον ανάπτυξης προγραμμάτων (IDEA). Κάθε πρόγραμμα ουσιαστικά είναι μια υποκλάση της PApplet, η οποία υλοποιεί και τα περισσότερα από τα γνωρίσματα της γλώσσας, και ονομάζεται sketch.

Κάθε Processing εφαρμογή πρέπει να υλοποιεί δυό βασικές μεθόδους:

* setup(): καλείται μόνο μια φορά μέσα στο πρόγραμμα και αρχικοποιεί μεταβλητές περιβάλλοντος

* draw(): καλείται επανειλλημένα αμέσως μετα την setup() μέχρι να σταματήσει το πρόγραμμα

Η γλώσσα προσφέρει επίσης μια συλλογή από μεθόδους για τον σχηματισμό βασικών γεωμετρικών σχημάτων, για την χρήση και τον μετασχηματισμό εικόνων και βασικές τριγωνομετρικές συναρτήσεις. Από τα πιο αντιπροσωπευτικά μέρη αυτής της διασύνδεσης για τον προγραμματισμό με την Processing.org παρατίθενται αμέσως:

- **Data**

  * Παρέχονται οι βασικοί τύποι δεδομένων: **boolean, byte, char, double**, ενώ έχει οριστεί ένας ακομα **color** που είναι τύπος δεδομένων που αποθηκεύει τιμές χρωμάτων.
  * Τύποι δεδομένων σύνθεσης: **ArrayList, Array, String, Hashmap**
  * Επίσης παρέχονται και μέθοδοι για την διαχείρηση και τον μετασχηματισμό των δεδομένων σύνθεσης όπως: **match(), matchAll(), splitTokens(), append(), sort(), subset()**, κ.α.

- **Control**

  * Παρέχονται ολοι οι λογικοί, αριθμητικοί operators όπως: **for, while, == (ισότητα), switch(),** , κ.α

- **Shape**

  * Παρέχονται μέθοδοι για την δημιουργία 2D σχημάτων όπως: **ellipse(), triangle(), rect(), bezier()**, κ.α
  * Παρέχονται μέθοδοι για την δημιουργία 3D σχημάτων οπως: **box(), sphere()**, κ.α

- **Input**

  * Παρέχονται μέθοδοι για την εισοδο πληροφορίας είτε μέσω πληκτρολογίου, ποντικιού, αρχείων ή και το web. Τυπικά παραδείγματα: **mouseClicked(), mousePressed(), keyPressed(), keyReleased(), BufferedReader(), loadStrings(), link()**, κ.α

- **Color**

  * Παρέχονται συναρτήσεις για το 'γέμισμα' των σχημάτων με χρώμα όπως για παράδειγμα: **fill(), colorMode(), stroke()**, κ.α
  * Παρέχονται μέθοδοι για την δημιουργία χρωμάτων όπως: **blendColor(), Color(), saturation()**, κ.α

- **Image**

  * Παρέχονται συναρτήσεις για την χρήση εικόνων: **image(), loadImage(), createImage()**, κ.α

* Παρέχονται μέθοδοι για την διαχείρηση και τροποποιήση των φωτογραφιών: **blend(), loadPixels(), filter(),** , κ.α

Επιπρόσθετα έχουν αναπτυχθεί μια ποικιλία βοηθητικών βιβλιοθηκών για τη υποστήριξη προηγμένων χαρακτηριστικών, όπως για παράδειγμα τον σχεδιασμό με OpenGL.

Στην συνέχεια παρατίθεται ο κώδικας ενός απλού sketch που δημιουργήθηκε με την βοήθεια της Processing και στην συνέχεια το οπτικό αποτέλεσμα.

```
1  public class weather extends PApplet {
2
3      Tree myTree;
4      int hour = hour();
5      Thread tree;
6
7     public void setup() {
8
9           hour = hour();
10          /*defines the size of the window*/
11          size(1000, 700);
12          /*sets the background color according to the time*/
13          if ((hour < 7) || (hour > 17))
14              background(13, 57, 75);
15          else
16              background(71, 163, 201);
17          /*draw smooth lines*/
18          smooth();
19
20          (new Tree(770, height, height - 300, this)).start();
21
22   /*draw only once*/
23          noLoop();
24     }
25
26      public void draw() {
27      /*do nothing*/
28   }
29  }
```

```
1  public class Tree extends PApplet extends Thread{
2      /*coordinates to create branch*/
3      float x, y;
4      /*the height of the tree*/
5      float h;
6      PApplet parent;
7
8      public Tree(float x_, float y_, float h_, PApplet thisParent) {
```

```
 9         x = x_;
10         y = y_;
11         parent = thisParent;
12         h = map(h_, 0, (float) screen.getHeight(), 0, 250);
13     }
14
15     public void run()
16     {
17         display();
18     }
19
20     public void display() {
21         parent.stroke(55);
22         branch(x, y, -PApplet.HALF_PI, h);
23     }
24
25     public void branch(float x_, float y_, float a_, float s_) {
26
27         parent.strokeWeight(s_ / 16);
28         float a = random(-PApplet.PI / 16, PApplet.PI / 16) + a_;
29         float nx = PApplet.cos(a) * s_ + x_;
30         float ny = PApplet.sin(a) * s_ + y_;
31
32         parent.stroke(32, 16 * s_);
33
34         int ay = (int) (y_ + ny) / 2;
35
36         parent.smooth();
37         parent.noFill();
38         parent.bezier(x_, y_, x_, ay, nx, ay, nx, ny);
39
40         if (s_ > 10) {
41             branch(nx, ny, a_ - random(PI / 4), s_ * random((float) 0.6,
                       (float) 0.8));
42             branch(nx, ny, a_, s_ * random((float) 0.6, (float) 0.8));
43             branch(nx, ny, a_ + random(PI / 4), s_ * random((float) 0.6,
                       (float) 0.8));
44         } else {
45             float w = random(155, 255);
46             parent.stroke(255, w, w, random(32, 192));
47             parent.strokeWeight(random(0, 8));
48             parent.point(nx + random(-2, 2), ny + random(-2, 2));
49         }
50     }
51 }
```

Figure 3.8: Sketch που δημιουργήθηκε με την βοήθεια της Processing

Chapter 4

Σχεδιασμός βιβλιοθήκης για τον έλεγχο ετερογενών συσκευών

4.1 Εισαγωγή

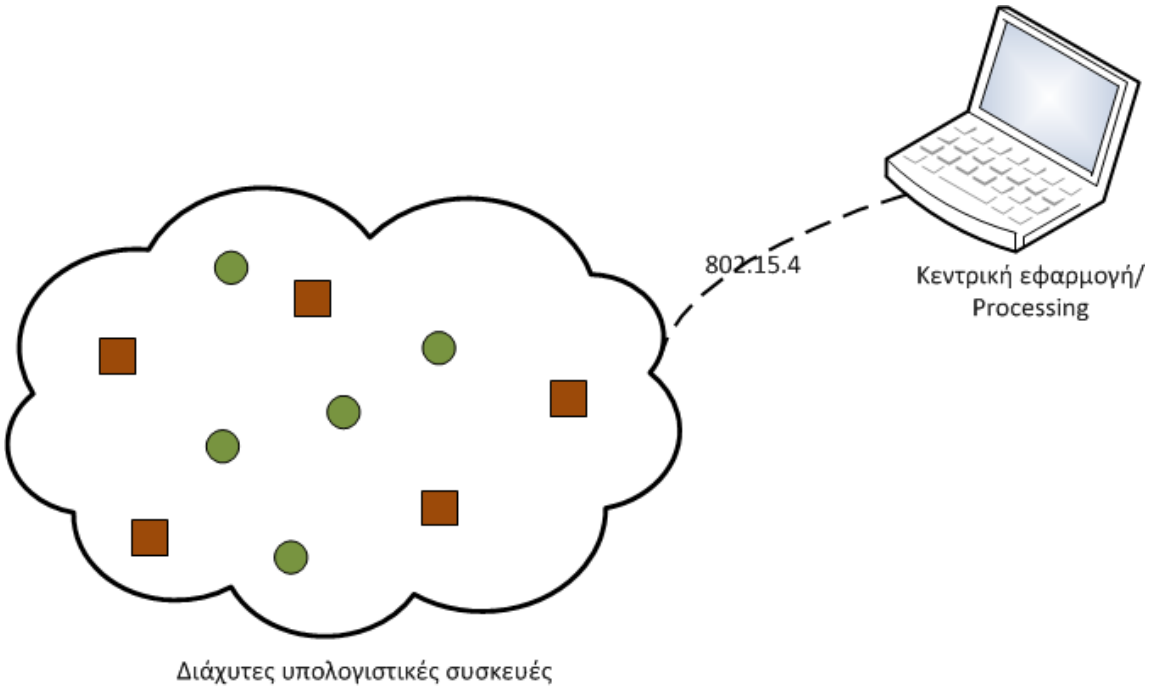


Figure 4.1: Γενική Αρχιτεκτονική Συστήματος Υλοποιησης Διαδραστικών Εφαρμογών με αισθητήρες

Η συνεχής ανάπτυξη της τεχνολογίας και η εμφάνιση της σε όλο και περισσότερες πτυχές της ζωής του ανθρώπου έχουν ως αποτέλεσμα την δημιουργία απαιτητικών και πολύπλοκων εφαρμογών. Για να καλυφθούν επαρκώς οι απαιτήσεις αυτές, γίνεται αναγκαία η χρήση διάφορων συσκευών ανόμοιων μεταξύ τους. Η ετερογένεια έγγειται όπως έχουμε ήδη αναφέρει στο προηγούμενο κεφάλαιο στις δυνατότητες τους, στον τρόπο έκθεσης αυτών όπως και επίσης στα αναγκαία εξαρτήματα για την διασύνδεση τους με τον κεντρικό υπολογιστή.

Στο Σχήμα: 4.1 φαίνεται η βασική αρχιτεκτονική υλοποιήσης διαδραστικών εφαρμογών, η οποία βασίζεται σε μια απλή ιεραρχία.Στο πρώτο επίπεδο είναι οι αισθητήρες που επηρεάζονται από το περιβάλλον και τον χρήστη, και στέλνουν τις πληροφορίες στο κεντρικό υπολογιστικό σύστημα. Το δεύτερο επίπεδο είναι ο ηλεκτρονικός υπολογιστής / laptop, στον οποίο εκτελείται η εφαρμογή. Η εφαρμογή αυτή,δέχεται τα δεδομένα τα οποία έχει ζητήσει από τις συσκευές, τα επεξεργάζεται και παράγει την επιθυμητή έξοδο, είτε στο περιβάλλον γραφικών, είτε δίνοντας εντολές στις συσκευές να δράσουν ανάλογα.

Στην ενότητα αυτή παρουσιάζεται η υλοποιήση μιας βιβλιοθήκης για τον κοινό προγραμματισμό και έλεγχο αυτών των συσκευών.Στο επίπεδο του κεντρικού υπολογιστικού συστήματος αναπτύχθηκε μια ενιαία διεπαφή για την πρόσβαση στην προς χρήση συσκευή, ανεξάρτητα της τεχνολογίας της, με σκοπό να απλοποιήσει τις απαιτούμενες ενέργειες για την επικοινωνία μεταξύ των δύο επιπέδων. Αντίστοιχα αναπτύχθηκε λογισμικο και για κάθε αισθητήρα. Ο σχεδιασμός έγινε με βασικούς άξονες την ευκολία χρήσης αλλά και την ευκολία στην επεκτασιμότητα, τόσο από μεριά λειτουργιών όσο και από την μεριά της προσθήκης νέων συσκευών.

Στη συνέχεια στην ενότητα 4.2 παρουσιάζεται το γενικό πρωτόκολλο που αναπτύχθηκε για την επικοινωνία των δύο επιπέδων. Στην ενότητα 4.3 αναλύεται το λογισμικό για κάθε συσκευή και τέλος στην 4.4 παρουσιάζεται η δομή της διεπαφής χρήστη.

4.2 Πρωτόκολλο

Για να μπορέσουμε να έχουμε αμφίδρομη επικοινωνία μεταξύ του κεντρικού υπολογιστικού συστήματος και των κατανεμειμένων αισθητήρων στο δίκτυο, αναπτύχθηκε ένα γενικό πρωτόκολλο επικοινωνίας.

Οι κανόνες στελέχωσης του πρωτοκόλλου δημιουργήθηκαν με βάση τα εξαρτήματα που διαθέτει η κάθε συσκευή και τις δυνατότητες αυτων. Η βασική δομή του μηνύματος που ανταλλάσουν τα δύο επίπεδα μεταξύ τους παρουσιάζεται παρακάτω.

4.2.1 Δομή Μήνυματος απο την εφαρμογή προς τις συσκευές

Η δομή των μηνυμάτων από τον κεντρικό υπολογιστή προς τις συσκευές φαινεται παρακάτω:

Byte 0	Byte 1	Byte 2-6

Το πρώτο byte (byte 0) σε κάθε μήνυμα υποδηλώνει το εξάρτημα κάθε αισθητήρα για το οποίο ζητάει να ενημερώνεται το κεντρικό υπολογιστικό σύστημα. Συγκεκριμένα έχουν δωθεί οι παρακάτω τιμές.

Εξάρτημα	Τιμή Byte 0
LEDs	0
SWITCH	1
LIGHT_VALUE	2
LIGHT_THRESHOLD	3
TEMPERATURE_VALUE	4
TEMPERATURE_THRESHOLD	5

Πίνακας 4.1: Τιμές για το πρώτο byte

Αντίστοιχα **το δεύτερο byte** (byte 1) υποδηλώνει ποια δυνατότητα του κάθε εξαρτήματος ζητάται. Επειδή κάποια από τα εξαρτήματα μπορούν να προσφέρουν μόνο απλή καταμέτρηση τιμών ενώ κάποια άλλα την ανίχνευση γεγονότων στο περιβάλλον οι τιμές που ανατίθενται σε αυτό καθορίζονται απο την τιμή του πρώτου. Στους παρακάτω πίνακες δίνεται η σημασία και η τιμή του byte 1, ανάλογα με την τιμή που έχει δωθεί στο προηγούμενο.

byte 0	byte 1	Σημασία
LEDs	0	άναψε ένα LED
	1	άναψε μια σειρά απο LEDs
	2	σβήσε ένα LED
	3	σβήσε μια σειρά από LED

Πίνακας 4.2: Τιμές δεύτερου byte όταν το πρώτο ισούται με LEDs

byte 0	byte 1	Σημασία
SWITCH / LIGHT_THRESHOLD / TEMPERATURE_TRESHOLD	0	ενεργοποιήσε την ενημέρωση για όταν ένα διακόπτης χρησιμοποιείται
	1	απενεργοποιήσε την ενημέρωση για όταν ένα διακόπτης χρησιμοποιείται

Πίνακας 4.3: Τιμές δεύτερου byte όταν το πρώτο ισούται με Switch/Light_Threshold / Temperature_Threshold

byte 0	byte 1	Σημασία
LIGHT_VALUE / TEMPERATURE_VALUE	pin	ζητά την τιμή του αντίστοιχου αισθητήρα στο συνδεδεμένο pin

Πίνακας 4.4: Τιμές δεύτερου byte όταν το πρώτο ισούται με Light_Value / Temperature_Value

Η επόμενη ομάδα από bytes καθορίζει κάποια ιδιαίτερα στοιχεία για τις ενέργειες που είναι να γίνουν. Οι τιμές σε αυτά δίνονται και πάλι ανάλογα με τις τιμές που έχουν ανατεθεί στα προηγούμενα. Αναφορικά περιέχουν πληροφορίες για το ποια LEDs θα ανάψουν ή θα σβήσουν, τι χρώμα θα έχουν και ποια θα είναι τα κατώφλια στους αισθητήρες θερμότητας και φωτός για τα οποία θα ενημερώνεται ο κεντρικός υπολογιστής.

4.2.2 Δομή μηνύματος από τις συσκευές προς την εφαρμογή

Η δομή των μηνυμάτων από τις συσκευές προν τον κεντρικό υπολογιστή/εφαρμογή φαινεται παρακάτω:

Byte 0	Byte 1	Byte 3

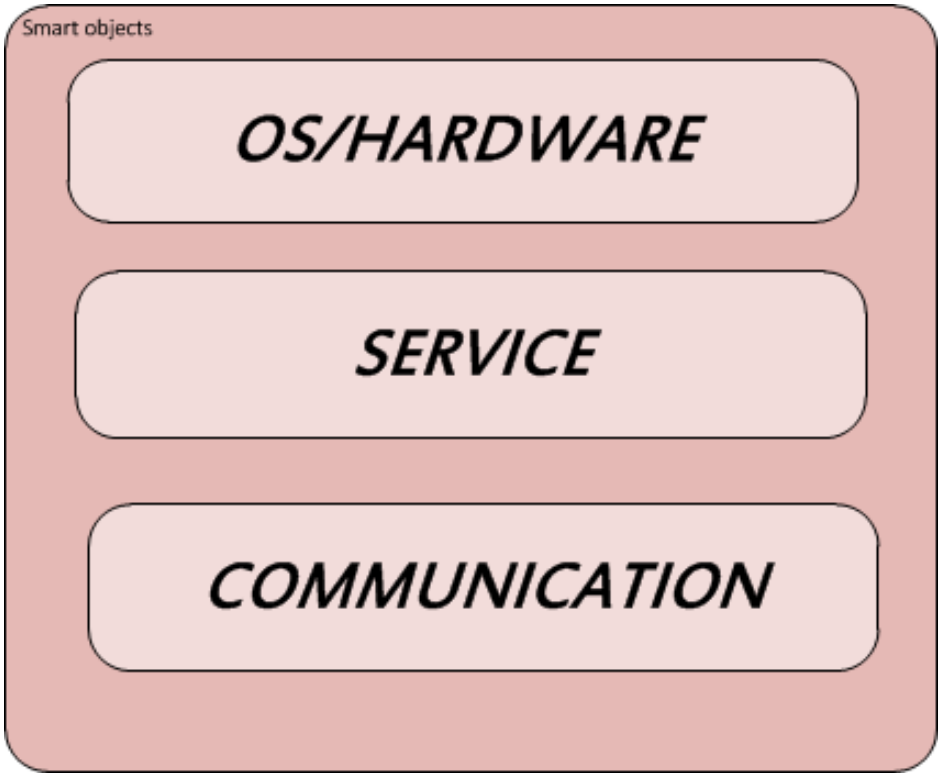
Πίνακας 4.5: Μήνυμα Συσκευής

Το πρώτο byte (byte 0) σε κάθε μήνυμα υποδηλώνει το εξάρτημα για το οποίο ενημερώνει η συσκευή το κεντρικό υπολογιστικό σύστημα. Οι τιμές είναι αντίστοιχες με αυτές που δίνονται στον πίνακα 4.1.

Αντίστοιχα **το δεύτερο byte** (byte 1) υποδηλώνει το pin που είναι συνδεδεμένο το κάθε εξαρτήμα. Και τέλος το **το τρίτο byte** περιέχει την ακέραια τιμή που έχει καταμετρηθεί από τον αντίστοιχο αισθητήρα ή αντίστοιχα την τιμή που ξεπερασε το ορισμένο κατώφλι.

4.3 Λογισμικό Συσκευών

Για κάθε συσκευή αναπτύχθηκε και το αντίστοιχο λογισμικό για την διαφανή επικοινωνία με το κεντρικό υπολογιστικό σύστημα, χρησιμοποιώντας το πρωτόκολλο που αναπτύξαμε παραπάνω. Η αρχιτεκτονική αυτού φαίνεται στο παρακάτω σχήμα.



Σχήμα 4.2: Αρχιτεκτονική Λογισμικού συσκευών

4.3.1 SunSPOT

Το πρόγραμμα στα SunSPOT αποτελειταί από τέσσερις βασικές κλάσεις: DataSender, Transmitter, Message, Receiver, Worker. Κάθε κλάση επι-

τελεί κάποιον συγκεκριμένο σκοπό και τρέχει στο δικό της νήμα (thread). Στη συνέχεια περιγράφεται ο σκοπός κάθε layer και των αντιστοιχων κλάσεων και παρατίθενται τμήματα κώδικα που θεωρούνται τα πιο σημαντικά.

Communication Layer

Η αποστολή και η λήψη πακέτων γίνεται χρησιμοποιώντας την κλάση radiogram από την Radio Communication Library.

- **Receiver**: είναι η κλάση που βασικό της σκοπός είναι να λαμβάνει μηνύματα με προορισμό αυτή την συσκευή. Το Port το οποίο έχει επιλεγεί για την λήψη των μηνυμάτων είναι το 37. Η Receiver τρέχει στο δικό της Thread. Κάθε φορά που λαμβάνει ένα μήνυμα το βάζει σε μια ουρά προτεραιότητας, ωστέ αυτό στην συνέχεια να επεξεργαστεί από την Worker. Ο διαχωρισμός της λήψης από την επεξεργασία των εισερχόμενων μηνυματων έγινε με σκοπό τον περιορισμό των πιθανών πακέτων που θα μπορούσαν να χαθούν εάν η επεξεργασία γινόταν αμέσως μετά την λήψη.

  Στη συνέχεια παρατίθεται το τμήμα της μεθόδου run(), η οποία ανοίγει μια σύνδεση και λαμβάνει περιοδικά πακέτα:

```
1  public void run() {
2
3   DatagramConnection dgConnection = null;
4   try {
5   dgConnection = (DatagramConnection) Connector.open("radiogram
        ://:37");
6   } catch (IOException e) {
7    System.out.println("Could not open radiogram receiver connection
         ");
8    e.printStackTrace();
9    return;
10   }
11   while (true) {
12   try {
13    Datagram dg = dgConnection.newDatagram(dgConnection.
          getMaximumLength());
14    dg.reset();
15    dgConnection.receive(dg);
16    myQueue.put(dg);
17    } catch (IOException e) {
18    System.out.println("Nothing received");
19    }
20   }
21  }
```

- **Transmitter**: είναι η κλάση που βασικός της σκοπός είναι να δημιουργεί τα datagrams με τα δεδομένα που θέλει να στείλει η συσκευή, και να ανοίγει τις αντίστοιχες συνδέσεις. Τα μηνύματα, που είναι τύπου Message και θέλει να ενσωματώσει μέσα σε κάθε datagram, τα παίρνει από μια ουρά προτεραιότητας. Τις συνδέσεις, την δημιουργία και την στελέχωση των πακέτων με τα κατάλληλα δεδομένα τις υλοποιεί χρησιμοποιώντας την κλάση DataSender. Η κλάση DataSender ακολουθεί το Singleton pattern ώστε να εξασφαλιστεί οτι δημιουργείται μόνο ένα αντικείμενο αυτής και να αποφευχθούν διάφορα σφάλματα.

  Στη συνέχεια παρατίθεται ο κώδικας της κλάσης Transmitter:

```
1  public class Transmitter extends Thread {
2      private Queue messageQueue;
3      private Queue fMessageQueue;
4
5      public Transmitter(Queue q, Queue fq) {
6          this.messageQueue = q;
7          fMessageQueue = fq;
8      }
9
10     public void run() {
11
12         while (true) {
13             if (!(messageQueue.isEmpty())) {
14                 Message msg = (Message) messageQueue.get();
15                 DataSender.getInstance().send(msg.getAddress(),
                       msg.getID(), msg.getAction(), msg.getValue());
16                 msg.flashMsg();
17                 fMessageQueue.put(msg);
18             }
19         }
20     }
21 }
```

  και η βασική συνάρτηση από την κλάση DataSender:

```
1  public void send(final String targetAddress, final int id, final
       int action, final int s) {
2  try {
3   final  DatagramConnection dgConnection = (DatagramConnection)
        Connector.open("radiogram://" + targetAddress + ":37");
4   final Datagram dg = dgConnection.newDatagram(dgConnection.
        getMaximumLength());
5
6   dg.reset();
7
```

```
 8   dg.write(id);
 9   dg.write(action);
10   dg.writeInt(s);
11
12   dgConnection.send(dg);
13   dgConnection.close();
14
15   } catch (IOException ex) {
16   System.out.println("Could not open radiogram connection");
17   ex.printStackTrace();
18    }
19   }
20 }
```

Service Layer

Ουσιαστικά αυτό το επίπεδο επιτελεί την παροχή των υπηρεσιών στον κεντρικό υπολογιστή, εκμεταλλευόμενο τις διεπαφές για τον χειρισμό των αισθητήρων. Η κλάση που αντιπροσωπεύει αυτό το επίπεδο είναι η Worker.

- **Worker**: Παίρνει από την ουρά προτεραιότητας τα μηνύματα που έχει τοποθετήσει η κλάση Receiver και στην συνέχεια τα επεξεργάζεται byte προς byte για να ορίσει την υπηρεσία που ζητάτε. Για να έχει την δυνατότητα πρόσβασης στα συνδεδεμένα εξαρτήματα υλοποιεί και τα προσφερόμενα απο το λειτουργικό σύστημα interfaces: ISwitchListener, ILightSensorThresholdListener, ITemperatureInputThresholdListener. Παρακάτω παρουσιάζεται η συνάρτηση που κάνει την επεξεργασία.

```
 1 public void run() {
 2         (new Transmitter(msgQueue, freeMsgQueue)).start();
 3         while (true) {
 4             Datagram MSG = (Datagram) queueProcess.get();
 5             try {
 6                 int id = MSG.readUnsignedByte();
 7                 if (id == LED) {
 8                     int action = MSG.readUnsignedByte();
 9                     switch (action) {
10                         case SET_ON_O: {
11                             onOne(MSG.readUnsignedByte(), MSG.
                                   readUnsignedByte(), MSG.
                                   readUnsignedByte(), MSG.
                                   readUnsignedByte());
12                             break;
13                         }
```

```
14                          case SET_ON_M: {
15                              onMany(MSG.readUnsignedByte(), MSG.
                                    readUnsignedByte(), MSG.
                                    readUnsignedByte(), MSG.
                                    readUnsignedByte(), MSG.
                                    readUnsignedByte());
16                              break;
17                          }
18                          case SET_OFF_O: {
19                              offOne(MSG.readUnsignedByte());
20                              break;
21                          }
22                          case SET_OFF_M: {
23                              offMany(MSG.readUnsignedByte(), MSG.
                                    readUnsignedByte());
24                              break;
25                          }
26                      }
27
28                  } else if (id == SWITCH) {
29                      int action = MSG.readUnsignedByte();
30                      switch (action) {
31                          case ON: {
32                              int sw = MSG.readUnsignedByte();
33                              switch (sw) {
34                                  case 1: {
35                                      add = MSG.getAddress();
36                                      sw1.addISwitchListener(this);
37                                      break;
38                                  }
39                                  case 2: {
40                                      add = MSG.getAddress();
41                                      sw2.addISwitchListener(this);
42                                      break;
43                                  }
44                              }
45                              break;
46                          }
47                          case OFF: {
48                              int sw = MSG.readUnsignedByte();
49                              switch (sw) {
50                                  case 1: {
51                                      sw1.removeISwitchListener(this
                                            );
52                                      break;
53                                  }
54
55                                  case 2: {
56                                      sw2.removeISwitchListener(this
```

```
                                        );
57                                      break;
58                                  }
59                              }
60                              break;
61                          }
62                      }
63                  } else if (id == LIGHT_T) {
64                      int action = MSG.readUnsignedByte();
65                      switch (action) {
66                          case ON: {
67                              add = MSG.getAddress();
68                              LThreshA = MSG.readInt();
69                              LThreshB = MSG.readInt();
70                              LTime = MSG.readInt();
71                              lightSensorRun(LThreshA, LThreshB,
                                    this);
72                              break;
73                          }
74                          case OFF: {
75                              lightSensorReset(this);
76                              break;
77                          }
78                      }
79                  } else if (id == LIGHT_V) {
80                      try {
81                          add = MSG.getAddress();
82                          lightLevel = lightSensor.getValue();
83                          Message msg = getMessage();
84                          msg.set(add, LIGHT_V, 1, lightLevel);
85                          msgQueue.put(msg);
86                      } catch (IOException e) {
87                          e.printStackTrace();
88                      }
89                  } else if (id == TEMP_V) {
90                      add = MSG.getAddress();
91                      tempValue = tempSensor.getValue();
92                      DataSender.getInstance().send(add, TEMP_V, 3,
                            tempValue);
93                  } else if (id == TEMP_T) {
94                      int action = MSG.readUnsignedByte();
95                      switch (action) {
96                          case ON: {
97                              add = MSG.getAddress();
98                              TThreshA = (double) MSG.readInt();
99                              TThreshB = (double) MSG.readInt();
100                             TTime = MSG.readInt();
101                             tempSensorRun(TThreshA, TThreshB, this
                                    );
```

```
102                             break;
103                         }
104                         case OFF: {
105                             tempSensorReset(this);
106                             break;
107                         }
108                     }
109                 }
110
111             } catch (IOException e) {
112                 e.printStackTrace();
113             }
114         }
115 }
```

Παρακάτω παρουσιάζεται ο κώδικας για την προσπέλαση κάθε εξαρτήματος, υλοποιώντας τις συναρτήσεις που παρέχει η κάθε διεπαφή.

* **LEDs**:

```
1  private void onOne(final int led, final int red, final int
       green, final int blue) {
2   leds[led].setRGB(red, green, blue);
3           leds[led].setOn();
4  }
5
6  private void onMany(final int ledA, final int ledB, final int
       red, final int green, final int blue) {
7   for (int i = ledA; i <= ledB; i++) {
8    leds[i].setRGB(red, green, blue);
9    leds[i].setOn();
10   }
11 }
12
13 private void offOne(final int led) {
14  leds[led].setOff();
15  }
16
17 private void offMany(final int ledA, final int ledB) {
18  for (int i = ledA; i <= ledB; i++) {
19   leds[i].setOff();
20   }
21 }
```

* **Διακόπτες**:

```
1  public void switchPressed(ISwitch iSwitch) {
```

```
2  int switchNum = (iSwitch == sw1) ? 1 : 2;
3          Message msg = getMessage();
4          msg.set(add, SWITCH, switchNum, 1);
5          msgQueue.put(msg);
6  }
```

* **Αισθητήρας Φωτός**:

```
1  private void lightSensorRun(final int threshA, final int
       threshB, final Worker w) throws IOException {
2   lightSensor.addILightSensorThresholdListener(w);
3          lightSensor.setThresholds(threshA, threshB);
4          lightSensor.enableThresholdEvents(true);
5  }
6  private void lightSensorReset(final Worker w) {
7          lightSensor.removeILightSensorThresholdListener(w);
8          lightSensor.enableThresholdEvents(false);
9  }
10 public void thresholdExceeded(ILightSensor iLightSensor, int
       i) {
11  Message msg = null;
12         if (i >= LThreshB) {
13             msg = getMessage();
14             msg.set(add, LIGHT_T, 2, i);
15             msgQueue.put(msg);
16             Utils.sleep(LTime);
17             lightSensor.enableThresholdEvents(true);
18         } else if (i <= LThreshA) {
19             msg = getMessage();
20             msg.set(add, LIGHT_T, 2, i);
21             msgQueue.put(msg);
22             Utils.sleep(LTime);
23             lightSensor.enableThresholdEvents(true);
24         } else {
25             lightSensor.enableThresholdEvents(true);
26         }
27 }
28 public void thresholdChanged(ILightSensor iLightSensor, int i
       , int i1) {
29         //ignore
30 }
```

* **Αισθητήρας Θερμότητας**:

```
1  private void tempSensorRun(final double threshA, final double
       threshB, final Worker w) {
2          tempSensor.addITemperatureInputThresholdListener(w);
```

```
3           tempSensor.setThresholds(threshA, threshB, true);
4           tempSensor.enableThresholdEvents(true);
5   }
6   private void tempSensorReset(final Worker w) {
7           tempSensor.removeITemperatureInputThresholdListener(w
                );
8           tempSensor.enableThresholdEvents(false);
9   }
10  public void thresholdExceeded(ITemperatureInput
        iTemperatureInput, double v, boolean b) {
11          Message msg = null;
12          if (v >= TThreshB) {
13              msg = getMessage();
14              msg.set(add, TEMP_T, 4, (int) v);
15              msgQueue.put(msg);
16              Utils.sleep(TTime);
17              tempSensor.enableThresholdEvents(true);
18          } else if (v <= TThreshA) {
19              msg = getMessage();
20              msg.set(add, TEMP_T, 4, (int) v);
21              msgQueue.put(msg);
22              Utils.sleep(TTime);
23              tempSensor.enableThresholdEvents(true);
24          } else {
25              tempSensor.enableThresholdEvents(true);
26          }
27
28  }
29  public void thresholdChanged(ITemperatureInput
        iTemperatureInput, double v, double v1, boolean b) {
30          //ignore
31  }
```

Το Service Layer 'πακετάρει' τα δεδομένα που θέλει να στείλει σε αντικείμενα τύπου Message και τα τοποθετεί σε μια ουρά προτεραιότητας από όπου θα τα πάρει το Communication Layer για να τα ενσωματώσει σε datagrams και να τα στείλει.

4.3.2 Arduino

Το πρόγραμμα που δημιουργήθηκε για το Arduino board αποτελείται από τέσσερις βασικές συναρτήσεις: available(), processPacket(), checkDigital() και checkAnalog(). Επίσης έχουν δημιουργηθεί και κάποιοι πίνακες για την διαχείρηση των pin της πλατφόρμας. Στη συνέχεια περιγράφονται αντίστοιχα

τα layer της βασικής αρχιτεκτονικής μαζί με τις συναρτήσεις που ανήκουν σε αυτό.

Communication Layer

- **available()**: Αυτή η συνάρτηση ελέγχει έαν υπάρχει κάποιο διαθέσιμο πακέτο στο Port που έχει οριστεί να ακούει το συγκεκριμένο Arduino Board και ειδοποιεί το Service Layer. Παρακάτω παρατίθεται ο κώδικας:

```
1  boolean overAir::available()
2  {
3   xbee.readPacket();
4   if(xbee.getResponse().isAvailable() && xbee.getResponse().
        validPacket(100))
5   {
6    return true;
7   }
8   else
9    return false;
10 }
```

Service Layer

Και σε αυτή την συσκευή είναι το επίπεδο που ουσιαστικά προσφέρει τις υπηρεσίες προσπελαύνοντας τα συνδεδεμένα εξαρτήματα. Για την σωστή διαχείρηση τόσο των αναλογικών όσο και των ψηφιακών pin της συσκευής έχουν οριστεί κάποιοι πίνακες:

1. **digitalPinValue[]**: Κρατάει τις τιμές που διαβάζει από τα εξαρτήματα που είναι συνδεδεμένα σε ψηφιακά pin και θέλει να στείλει στον κεντρικό υπολογιστή.

2. **digitalPinPreviousValue[]**: Κάθε φορά που είναι να αποσταλεί μια ενημέρωση για ένα εξάρτημα, ελέγχεται πρώτα εάν έχει αλλάξει η τιμή χρησιμοποιώντας τον πίνακα αυτό. Σκοπός είναι η μείωση των αποστελλόμενων μηνυματών καθώς διαφορετικά το κόστος θα ηταν υπερβολικό και ο κεντρικός υπολογιστής θα πλημμύριζε απο αχρειάστες ενημερώσεις.

3. **digitalPinConfig[]**: σε αυτό τον πίνακα αποθηκεύεται η λειτουργία του κάθε pin. Δηλαδή ορίζεται αν λειτουργεί σαν είσοδος (τιμή = 1) ή σαν έξοδος (τιμή = 0).

4. **digitalPinFunction[]**: κρατάει το είδος του εξαρτήματος που είναι συνδεδεμένο στο αντίστοιχο pin. Συγκεκριμένα έχουν οριστεί: Διακόπτης=1,

5. **analogPinValue[]**: κρατάει την στιγμιαία τιμή του εξαρτήματος και η οποία θα αποσταλλεί στον κεντρικο υπολογιστή.

6. **analogPinFunction[]**: έχει ίδιο ρόλο με την digitalPinFunction[] αλλά για τα αναλογικά pin.

7. **analogPinToReport[]**: αυτός ο πίνακας έχει άσσο στο αντίστοιχο pin ωστέ να ελεγχθεί η τιμή του από την checkAnalog().

Στην συνέχεια περιγράφονται οι συναρτήσεις του επιπέδου αυτού.

- **processPacket()**: Επεξεργάζεται το εισερχόμενο πακέτο για να ορίσει την υπηρεσία που ζητάται και θέτει ανάλογα τους πίνακες για την προσπέλαση των pin.

```
1  boolean overAir::available()
2  {
3   xbee.readPacket();
4   if(xbee.getResponse().isAvailable() && xbee.getResponse().
        validPacket(100))
5   {
6    return true;
7   }
8   else
9    return false;
10 }
```

- **checkDigital()**: η συνάρτηση αυτή χρησιμοποιώντας τους πίνακες για τα ψηφιακά pin που αναλύθηκαν ελέγχει τις τιμές τους και ενημερώνει ανάλογα τον κεντρικό υπολογιστή. Ο κώδικας αυτής:

```
1  void overAir::checkDigital()
2  {
3   for(int i=2;i<14;i++)
4    {
5      if(digitalPinConfig[i]==1)
6      {
7        digitalPinValue[i]= digitalRead(i);
8      }
9    }
10   for(int i=2;i<14;i++)
11   {
```

```
12      if(digitalPinValue[i]!=digitalPinPreviousValue[i])
13      {
14        payload[0]=digitalPinFunction[i]&0xff;
15        payload[1]=i&0xff;
16        payload[5]=digitalPinValue[i]&0xff;
17
18        tx.setAddress16(address);
19        xbee.send(tx,37);
20
21        digitalPinPreviousValue[i]=digitalPinValue[i];
22
23      }
24    }
25 }
```

- **checkAnalog()**: ανάλογη δουλειά με την checkDigital αλλά για τα αναλογικά pin. Και αντίστοιχα έχουμε:

```
1  void overAir::checkAnalog()
2  {
3   for(int i=0;i<6;i++){
4    if(analogPinToReport[i]==1)
5    {
6     analogRead(i);
7     delay(50);
8     analogPinValue[i]=analogRead(i);
9    }
10  }
11  for(int i=0;i<6;i++)
12  {
13   if(analogPinToReport[i]==1)
14   {
15   payload[0]=analogPinFunction[i]&0xff;
16   payload[1]=i&0xff;
17   payload[2]=0&0xff;
18   payload[3]=0&0xff;
19   payload[4]=analogPinValue[i]>>8&0xff;
20   payload[5]=analogPinValue[i]&0xff;
21
22   tx.setAddress16(address);
23   xbee.send(tx,37);
24
25   analogPinToReport[i]=0;
26   analogPinValue[i]=0;
27   }
28  }
29 }
```

4.4 Λογισμικό Κεντρικού Υπολογιστή

Η αρχιτεκτονική του λογισμικού που αναπτύχθηκε για τον κεντρικό υπολογιστή φαίνεται στο σχήμα 4.3. Έχει υλοποιηθεί σε Java Standard Edition έκδοση 6. Ακολουθεί η επεξήγηση και ο κώδικας για το κάθε επίπεδο.

Σχήμα 4.3: Αρχιτεκτονική Λογισμικού κεντρικού υπολογιστή

Registration

Είναι το επίπεδο που εντοπίζει και αναγνωρίζει τις συσκευές που υπάρχουν στο περιβάλλον και επικοινωνούν με τον κεντρικό υπολογιστή. Ο χρήστης εγγράφει κάθε συσκευή που θέλει να ελέγξει, να προγραμματίσει και να χρησιμοποιήσει τις ενημερώσεις της στην εφαρμογή του. Η εγγραφή πραγματοποιείται παρέχοντας την μοναδική 16-bit MAC-address των συσκευών

και το Port στο οποίο ακούν. Για αυτό το σκοπό έχει δημιουργηθεί η κλάση Sensors.

- **Sensors** Η κλάση αυτή είναι Singleton. Έχει δυο HashTables που ο καθένας κρατάει αντίστοιχα τα SunSPOT και τα Arduino Boards που υπάρχουν στο δίκτυο

```
1 private static Hashtable<String, SunSPOT> sunSPOT;
2
3 private static Hashtable<String, Arduino> arduino;
```

και προσφέρει τις εξής συναρτήσεις:

- **public synchronized void addSPOT(String address, int port)** : Η συγκεκριμένη συνάρτηση προσθέτει ένα νέο SunSPOT στον αντίστοιχο HashTable.
- **public synchronized void addArduino(String address, int port)**: Προσθέτει ένα νέο Arduino Board στον αντίστοιχο HashTable.
- **public synchronized SunSPOT getSPOT(String address)**: επιστρέφει το SunSPOT με την αντίστοιχη 16-bit διεύθυνση.
- **public synchronized Arduino getArduino(String address)**: επιστρέφει το Arduino Board με την αντίστοιχη 16-bit διεύθυνση.

Από την στιγμή που μια συσκευή εγγραφεί, ο χρήστης μπορεί να έχει πρόσβαση στο Service Layer. Ουσιαστικά αυτό το επίπεδο είναι ενα interface για την διαχείρηση των λειτουργιών των συσκευών. Αναλυτικά περιγράφεται αμέσως παρακάτω.

Service Layer

Αυτό το επίπεδο παρέχει στον χρήστη την δυνατότητα προγραμματισμού κάθε εξαρτήματος των συσκευών όπως και επίσης και την δυνατότητα να ενημερώνεται για τις καταστάσεις και τις τιμές αυτών.

Για κάθε είδος συσκευής έχουν δημιουργηθεί αντίστοιχα δυο Facade κλάσεις, οι οποίες δίνουν πρόσβαση στα εξαρτήματα που υποστηρίζει η κάθε συσκευή. Αυτές οι κλάσεις είναι οι: SunSPOT και Arduino. Και οι δύο κληρονομούν απο την κλάση SmartObject που αντιπροσωπεύει τις συσκευές στο σύνολο τους. Παρακάτω δίνεται ο κώδικας της υπερκλάσης.

```
1  public class SmartObject {
2      public String address;
3      public int port;
4
5      public SmartObject(String add, int p) {
6          address = add;
7          port = p;
8      }
9
10     public String getAddress() {
11         return address;
12     }
13
14     public int getPort() {
15         return port;
16     }
17 }
```

Κάθε Facade προσφέρει τις παρακάτω Accessor μεθόδους:

- **public IColorLed getLEDs()** :επιστρέφει ένα αντικείμενο IColorLed που αντιπροσωπεύει τα συνδεδεμένα LEDs και τις δράσεις πάνω σε αυτά.

- **public Switch getSwitch(int i)** : επιστρέφει ένα αντικείμενο Switch που αντιπροσωπεύει τον συνδεδεμένο διακόπτη. Σαν όρισμα παίρνει έναν ακέραιο αριθμό που είναι είτε το id του διακόπτη (για τις SunSPOT συσκευές), είτε το pin που είναι συνδεδεμένος ο διακόπτης (για τις Arduino συσκευές).

- **public LightThresholdSensor getLightSensor()**:
  Συνάρτηση που υλοποιείται μόνο από την κλάση SunSPOT. Επιστρέφει ένα αντικείμενο LightThresholdSensor που αντιπροσωπεύει τον αισθητήρα φωτός που είναι build-in στα SunSPOT.

- **public LightSensor getLightSensor(int i)**: Συνάρτηση που υλοποιείται μόνο από την κλάση Arduino. Επιστρέφει ένα αντικείμενο LightSensor που αντιπροσωπεύει αναλογικούς αισθητήρες φωτός που μπορούν να συνδεθούν στο Arduino. Παίρνει σαν όρισμα έναν ακέραιο αριθμό, ο οποίος είναι το pin σύνδεσης του αισθητήρα.

- **public TThresholdSensor getTemperatureSensor()**: Υλοποιείται μόνο από την κλάση SunSPOT. Επιστρέφει ένα αντικείμενο TThresholdSensor που αντιπροσωπεύει τον build-in αισθητήρα θερμότητας της αντίστοιχης συσκευής.

- **public TemperatureSensor getTemperatureSensor(int i)**: Υλοποιείται μόνο από την κλάση Arduino. Επιστρέφει ένα αντικείμενο TemperatureSensor που αντιπροσωπεύει τον αναλογικό αισθητήρα θερμότητας που είναι συνδεδεμένος στο pin i του Arduino.

Για κάθε εξάρτημα δημιουργήθηκε και το αντίστοιχο αντικείμενο για την προσπέλαση και τον έλεγχο του. Πληροφορίες για το κάθε ένα δίνονται παρακάτω.

- ***LEDs***

  Τα LEDs αντιπροσωπεύονται από την κλάση ColorLed. Αυτή υλοποιεί το IColorLed interface, ο κώδικας του οποίου φαίνεται παρακάτω και το οποίο παρέχει τις βασικές δράσεις.

```
1  public interface IColorLed {
2      /**
3       * turn the led on
4       *
5       * @param led which led to turn on and later with which color
6       * @param clr color
7       */
8      void setON(int led, LColor clr);
9
10     void setON(int led);
11
12     /**
13      * turn the led ON
14      *
15      * @param ledA start led
16      * @param ledB last led
17      * @param clr  color
18      */
19     void setON(int ledA, int ledB, LColor clr);
20
21     void setON(int ledA, int ledB);
22
23     /**
24      * turn the led off
25      *
26      * @param led which led to turn off
27      */
28     void setOFF(int led);
29
30     /**
31      * turn led off
32      *
33      * @param ledA start led
```

```
34        * @param ledB stop led
35        */
36       void setOFF(int ledA, int ledB);
37
38       /*return the state of the led true or false*/
39
40       public boolean getMode();
41   }
```

Αναλυτικότερα είναι:

- **void setON(int led, LColor clr)**: δίνει εντολή στην συσκευή να ανάψει το LED που καθορίζεται από το ακέραιο όρισμα και με χρώμα το όρισμα LColor.
- **void setON(int led)**: δίνει εντολή να ανάψει ένα LED.
- **void setON(int ledA, int ledB, LColor clr)**: δίνει εντολή να ανάψουν μια σειρά απο LEDs με το αντίστοιχο προσδιοριζόμενο χρώμα.
- **void setON(int ledA, int ledB)**: δίνει εντολή να ανάψουν μια σειρά απο LEDs
- **void setOFF(int led)**: δίνει εντολή να σβίσει το συγκεκριμένο LED.
- **void setOFF(int ledA, int ledB)**: δίνει εντολή να σβήσουν μια σειρά από LEDs.

- ***Διακόπτης***

Αυτό το εξάρτημα αντιπροσωπεύεται από την κλάση Switch. Παρακάτω φαίνεται ο κώδικας αυτής.

```
1  public class Switch implements IMsgSListener, PacketTypes {
2
3   ...
4
5       public Switch(SmartObject o, int i) {
6         ...
7       }
8
9       /**
10       * function to handle incoming msg interesting for
11       *
12       * @param event the kind of msg /event
13       */
14      public void HandleSEvent(Event event) {
15
```

```
16          if ((Switcher == event.getEvent()) && (event.
                getParentAddress().contains(parent.getAddress()))) {
17              final Switch sw = this;
18              for (int i = 0; i < listeners.size(); i++) {
19                  final ISListener l = listeners.get(i);
20                  new Thread("Switch is Pressed") {
21                      public void run() {
22                          l.SwitchPressed(sw);
23                      }
24                  }.start();
25              }
26          }
27      }
28
29      /**
30       * method to ask for particular switch
31       *
32       * @param S switches id
33       */
34      private void askSwitch(int S) {
35        ...
36      }
37
38      /**
39       * method to reset msg for particular switch
40       *
41       * @param S switches id
42       */
43      private void resetSwitch(int S) {
44        ...
45      }
46
47      /**
48       * method to add listener
49       *
50       * @param who the listener
51       */
52      public synchronized void addListener(ISListener who) {
53          if (!listeners.contains(who))
54              listeners.add(who);
55          if (listeners.size() == 1)
56              askSwitch(this.Switcher);
57      }
58
59      /**
60       * method to remove listener
61       *
62       * @param who the listener
63       */
```

```
64      public synchronized void removeListener(ISListener who) {
65          listeners.remove(who);
66          if (listeners.size() == 0) {
67              resetSwitch(this.Switcher);
68          }
69      }
70  }
```

Η βασική λειτουργία αυτού του εξαρτήματος είναι η ενημέρωση για την αλλαγή κατάστασης του διακόπτη. Η κλάση αυτή ακολουθεί το Observer pattern [19] και είναι Observable για όλα τα αντικείμενα που υλοποιούν την παρακάτω διασύνδεση.

```
1  /**
2   * interface for listening the Switches
3   */
4  public interface ISListener {
5      /**
6       * @param sw the switch that was pressed
7       */
8      public void SwitchPressed(Switch sw);
9  }
```

Κάθε Observer αντικείμενο εγγράφεται σε μια λίστα για να ειδοποιείται για events. Για αυτό το σκοπό παρέχονται οι παρακάτω μέθοδοι:

- **public void addListener(ISListener who)**: προσθέτει τον Observer στην λίστα
- **public void removeListener(ISListener who)**: αφαιρεί τον αντίστοιχο Observer από την λίστα.

• *Αισθητήρας Φωτός*

Οι δυνατότητες που προσφέρει ο αισθητήρας φωτός διαφέρουν ανάμεσα στις δύο συσκευές, καθώς είναι διαφορετικές και οι δυνατότητες των συσκευών. Γι αυτό το λόγο έχουν δημιουργηθεί δύο κλάσεις. Η βασική κλάση είναι η LightSensor που φαίνεται παρακάτω:

```
1  public class LightSensor implements IMsgLListener, PacketTypes {
2    ...
3
4      public LightSensor(SmartObject p, int pin) {
5    ...
6      }
```

```
7
8       public void askValue() {
9     ...
10      }
11
12      public void HandleLEvent(Event ev) {
13     ...
14      }
15
16      public synchronized void addListener(ILValueListener who) {
17          if (!Listeners.contains(who))
18              Listeners.add(who);
19      }
20
21      public synchronized void removeListener(ILValueListener who) {
22          Listeners.remove(who);
23      }
24
25  }
```

Η βασική λειτουργία που παρέχεται είναι η αίτηση προς την αντίστοιχη συσκευή για την ενημέρωση της παρούσας τιμής που έχει ο αισθητήρας. Αυτή η λειτουργία υλοποιείται από την μέθοδο **public void askValue()**.

Και αυτή η κλάση ακολουθεί το Observer pattern και είναι Observable για τα αντικείμενα που υλοποιούν την διεπαφή ILValueListener.

```
1  /**
2   * interface implemented by classes to listen for light values
3   */
4  public interface ILValueListener {
5      /**
6       * @param light the sensor
7       * @param val   the value
8       */
9      public void lightSensorValue(LightSensor light, int val);
10 }
```

Κάθε αντικείμενο που θέλει να ενημερώνεται για την στιγμιαία τιμή του αισθητήρα φωτός εγγράφεται σε μια λίστα με τους Observers. Παρέχονται οι παρακάτω μέθοδοι για αυτόν τον σκοπό.

- **public void addListener(ILValueListener who)**: προσθέτει τον Observer στην λίστα
- **public void removeListener(ILValueListener who)**: αφαιρεί τον αντίστοιχο Observer από την λίστα.

Ο αισθητήρας φωτός που είναι build-in στην συσκευή SunSPOT παρέχει τόσο την δυνατότητα να διαβάζει απλά την τιμή του αισθητήρα, όσο και να αντιλαμβάνεται πότε η τιμή του φωτός έχει ξεπεράσει κάποιο προκαθορισμένο κατώφλι. Έτσι δημιουργήθηκε η κλάση LightThresholdSensor η οποία κληρονομεί την βασική LightSensor:

```
1  public class LightThresholdSensor extends LightSensor {
2    ...
3
4      public LightThresholdSensor(SmartObject p, int pin) {
5    ...
6      }
7
8
9      public void askValue() {
10         super.askValue();
11     }
12
13     private void askLightThresholdSensor() {
14   ...
15     }
16
17     private void resetLightThresholdSensor() {
18   ...
19     }
20
21     public synchronized void setThresholdsTime(final int a, final
           int b, final int interval) {
22         if (((threshA != -1) && (threshB != -1))) {
23             return;
24         }
25         threshA = a;
26         threshB = b;
27         time = interval;
28     }
29
30     public void HandleLEvent(Event ev) {
31   ...
32     }
33
34     public synchronized void addListener(ILValueListener who) {
35         super.addListener(who);
36     }
37
38     public synchronized void removeListener(ILValueListener who) {
39         super.removeListener(who);
40     }
41
```

```
42      public synchronized void addThresholdListener(
            ILThresholdListener who) {
43          if (!TListeners.contains(who))
44              TListeners.add(who);
45          if (TListeners.size() == 1) {
46              askLightThresholdSensor();
47          }
48      }
49
50      public synchronized void removeThresholdListener(
            ILThresholdListener who) {
51          TListeners.remove(who);
52          if (TListeners.size() == 0) {
53              resetLightThresholdSensor();
54          }
55      }
56  }
```

Οι λειτουργίες που παρέχονται από αυτήν την κλάση είναι αυτές που παρέχει η LightSensor και έχουν ήδη περιγρφεί, και η δυνατότητα ενημέρωσης όταν η τιμη του φωτός ξεπεράσει τα κατώφλια που έχει ορίσει ο ίδιος ο χρήστης.

- **public void setThresholdsTime(final int a, final int b, final int interval)**: αυτή η συνάρτηση θέτει το άνω και το κάτω κατώφλι για τα οποία πρέπει να ελέγχει η συσκευή οπως και επίσης το χρονικό διάστημα επανάληψης του ελέγχου.

Και η LightThresholdSensor ακολουθεί το Observer pattern και είναι Observable για κάθε ενδιαφερόμενο αντικείμενο. Συγκεκριμένα όταν κάποιο αντικείμενο ενδιαφέρεται για την στιγμιαία τιμή του αισθητήρα υλοποιεί την διεπαφή ILValueListener και εγγράφεται στις λίστες αντιστοίχως, ενώ εάν ενδιαφέρεται για το αν ξεπεράστηκαν τα κατώφλια υλοποιεί την διεπαφη ILThresholdListener:

```
1  public interface ILThresholdListener {
2      /**
3       * @param sensor the sensor from which the value comes
4       * @param val    the value that exceeded the thresholds
5       */
6      public void thresholdExceed(LightThresholdSensor sensor, int
           val);
7  }
```

Αντίστοιχα η εγγραφή στην λίστα με τους Observers πραγματοποιείται με τις παρακάτω μεθόδους:

- **public void addThresholdListener(ILThresholdListener who)**
- **public void removeThresholdListener(ILThresholdistener who)**

- ***Αισθητήρας Θερμότητας***

  Και στην περίπτωση του αισθητήρα θερμότητας προσφερόμενες λειτουργίες διαφέρουν ανάλογα με την συσκευή και έτσι ο σχεδιασμός τους είναι πανομοιότυπος με αυτόν του αισθητήρα φωτός. Συγκεκριμένα η βασική κλάση είναι η TemperatureSensor:

```
1  public class TemperatureSensor implements IMsgTListener,
       PacketTypes {
2    ...
3      public TemperatureSensor(SmartObject p, int pin) {
4    ...
5      }
6
7      public void askValue() {
8    ...
9      }
10
11
12     public void HandleTEvent(Event ev) {
13   ....
14     }
15
16     public synchronized void addListener(ITValueListener who) {
17         if (!Listeners.contains(who))
18             Listeners.add(who);
19     }
20
21     public synchronized void removeListener(ITValueListener who) {
22         Listeners.remove(who);
23     }
24 }
```

  Και εδώ η βασική λειτουργία που παρέχεται είναι η αίτηση για ενημέρωση της στιγμιαίας τιμής του αισθητήρα και υλοποιείται από την μέθοδο **public void askValue()**.

  Και πάλι ακολουθείται το Observer pattern και η TemperatureSensor είναι Observable για όσα αντικείμενα υλοποιούν την διεπαφή ITValueListener:

```
1  public interface ITValueListener {
2      /**
```

```
3       * @param s    the temperature sensor
4       * @param val the incoming value
5       */
6      public void TemperatureValue(TemperatureSensor s, int val);
7  }
```

Κάθε αντικείμενο που θέλει να ενημερώνεται για την στιγμιαία τιμή του αισθητήρα θερμότητας εγγράφεται σε μια λίστα με τους ITValueListener Observers. Παρέχονται οι παρακάτω μέθοδοι για αυτόν τον σκοπό.

- **public void addListener(ITValueListener who)**: προσθέτει τον Observer στην λίστα
- **public void removeListener(ITValueListener who)**: αφαιρεί τον αντίστοιχο Observer από την λίστα.

Η κλάση TThresholdSensor κληρονομεί από την TemperatureSensor:

```
1  public class TThresholdSensor extends TemperatureSensor {
2    ....
3      public TThresholdSensor(SmartObject p, int pin) {
4    ....
5      }
6
7      public void askValue() {
8          super.askValue();
9      }
10
11     private void askTemperatureThresholdSensor() {
12   ....
13     }
14
15     private void resetTemperatureThresholdSensor() {
16   ....
17     }
18
19     public synchronized void setThresholdsTime(final int a, final
           int b, final int time) {
20         if (((threshA != -1) && (threshB != -1))) {
21             return;
22         }
23         threshA = a;
24         threshB = b;
25         this.time = time;
26     }
27
28     public void HandleTEvent(Event ev) {
```

```
29    ....
30      }
31
32      public synchronized void addListener(ITValueListener who) {
33          super.addListener(who);
34      }
35
36      public synchronized void removeListener(ITValueListener who) {
37          super.removeListener(who);
38      }
39
40      public synchronized void addThresholdListener(
            ITThresholdListener who) {
41          if (!TListeners.contains(who))
42              TListeners.add(who);
43          if (TListeners.size() == 1) {
44              askTemperatureThresholdSensor();
45          }
46      }
47
48      public synchronized void removeThresholdListener(
            ITThresholdListener who) {
49          TListeners.remove(who);
50          if (TListeners.size() == 0) {
51              resetTemperatureThresholdSensor();
52          }
53      }
54  }
```

Και στην περίπτωση του αισθητήρα θερμότητας η TThresholdSensor παρέχει τόσο τις βασικές λειτουργίες που κληρονομεί, όσο και τις επιπλέον για την ενημέρωση των ειδικών γεγονότων. Για να τεθούν τα κατώφλια χρησιμοποιείται η μέθοδος:

- **public void setThresholdsTime(final int a, final int b, final int interval)**: αυτή η συνάρτηση θέτει το άνω και το κάτω κατώφλι για τα οποία πρέπει να ελέγχει η συσκευή οπως και επίσης το χρονικό διάστημα επανάληψης του ελέγχου.

Κάθε αντικείμενο που ενδιαφέρεται να ενημερώνεται για αυτές τις ειδικές καταστάσεις υλοποιεί την διεπαφή ITThresholdListener:

```
1  public interface ITThresholdListener {
2      /**
3       * @param sensor the temperature sensor
4       * @param val    the incoming threshold/value
5       */
```

```
6     public void TempThresholdExceed(TThresholdSensor sensor,
          double val);
7 }
```

και εγγράφεται στην αντίστοιχη λίστα με τους Observers χρησιμοποιώντας τις μεθόδους:

- **public void addThresholdListener(ITThresholdListener who)**
- **public void removeThresholdListener(ITThresholdListener who)**

Communication Layer

Αυτό το επίπεδο είναι υπευθυνό για την εγκαθίδρυση της επικοινωνίας. Από την μια μερά στέλνει τα μηνύματα στις συσκευές και από την άλλη λαμβάνει μηνύματα και τα επεξεργάζεται. Οι κλάσεις που στοιχειοθετούν αυτό το επίπεδο είναι οι: Receiver, DataSender και Worker. Η κάθε μία τρέχει στο δικό της Thread.

- **Receiver**: είναι κλάση που λαμβάνει τα μηνύματα από τις συσκευές και τα τοποθετεί σε μια ουρά προτεραιότητας για να τα πάρει η Worker και να τα επεξεργαστεί.
- **DataSender**:είναι μια Singleton κλάση η οποία ανοίγει την σύνδεση με την εκάστοτε συσκευή προορισμού και στέλνει το μήνυμα σε μορφή datagram.
- **Worker**:είναι η πιο σημαντική κλάση σε αυτό το επίπεδο. Παίρνει από την ουρά προτεραιότητας τα μηνύματα που έχει τοποθετήσει η κλάση Receiver και τα αναλύει byte προς byte. Η βασική συνάρτηση λειτουργίας είναι:

```
1  public void run() {
2
3          while (true) {
4              Datagram MSG = null;
5              try {
6                  MSG = queueTo.take();
7                  int id = 0;
8                  int event = 0;
9                  int value = 0;
10                 if (MSG.getLength() > 0) {
11                     try {
```

```
12                          id = MSG.readUnsignedByte();
13                          if (id == SWITCH) {
14                              event = MSG.readUnsignedByte();
15                              if (MSG.readInt() != 0) {
16                                  notifySListeners(new Event(this,
                                        event, MSG.getAddress()));
17                              }
18                          } else if (id == LIGHT_T) {
19                              event = MSG.readUnsignedByte();
20                              value = MSG.readInt();
21                              notifyLListeners(new Event(this, event
                                    , value, MSG.getAddress()));
22                          } else if (id == LIGHT_V) {
23                              event = MSG.readUnsignedByte();
24                              value = MSG.readInt();
25                              notifyLListeners(new Event(this, event
                                    , value, MSG.getAddress()));
26                          } else if (id == TEMP_V) {
27                              event=MSG.readUnsignedByte();
28                              value=MSG.readInt();
29                              notifyTListeners(new Event(this,event,
                                    value, MSG.getAddress()));
30                          } else if(id==TEMP_T) {
31                                event=MSG.readUnsignedByte();
32                               value=MSG.readInt();
33                              notifyTListeners(new Event(this,event,
                                    value,MSG.getAddress()));
34                          }
35                      } catch (IOException e) {
36                          e.printStackTrace();
37                      }
38                  }
39              } catch (InterruptedException e) {
40                  e.printStackTrace();
41              }
42          }
43      }
```

Η Worker ακολουθεί το Observer pattern και είναι Observable για το Service Layer. Οι Observers έχουν χωριστεί σε τρείς κατηγορίες οι οποίες δημιουργήθηκαν αντιστοίχως με τα είδη των εξαρτημάτων. Ανάλογα με το εισερχόμενο μήνυμα η Worker ειδοποιεί και τους αντίστοιχους Observers:

- **public static void notifySListeners(final Event ev)**: μέθοδος που ειδοποιεί το Service Layer για εισερχόμενα μηνύματα σε σχέση με κάποιον διακόπτη.

- **public static void notifyLListeners(final Event ev)**: μέθοδος που ειδοποιεί το Service Layer για εισερχόμενα μηνύματα σε σχέση με κάποιον αισθητήρα φωτός.
- **public static void notifyTListeners(final Event ev)**: μέθοδος που ειδοποιεί το Service Layer για εισερχόμενα μηνύματα σε σχέση με κάποιον αισθητήρα θερμότητας.

Για να ειδοποιήσει τους Observers η Worker στέλνει ένα μήνυμα τύπου Event που περιέχει την τιμή του εκάστοτε αισθητήρα όπως και επίσης την μοναδική MAC-address της συσκευής από την οποία ήρθε η ειδοποιήση. Ο κώδικας της Event φαίνεται πσρακάτω:

```
1  //class to transfer messages to observers
2  public class Event extends EventObject {
3
4      private int event;
5      private int iValue;
6      private String parentAddress;
7
8      public Event(Object source, int ev, String add) {
9          super(source);
10         event = ev;
11         parentAddress = add;
12     }
13
14     public Event(Object source, int ev, int val, String add) {
15         super(source);
16         event = ev;
17         iValue = val;
18         parentAddress = add;
19     }
20
21     public int getEvent() {
22         return event;
23     }
24
25     public int getIValue() {
26         return iValue;
27     }
28
29     public String getParentAddress() {
30         return parentAddress;
31     }
32 }
```

Κάθε Observer στο Service Layer αφού αποδεχθεί την ειδοποιήση, ειδοποιεί με την σειρά του τους Observers στο επίπεδο του χρήστη(όπως έχει ήδη περιγραφεί).

Chapter 5

Παράδειγμα Διαδραστικής Εφαρμογής

5.1 Εισαγωγή

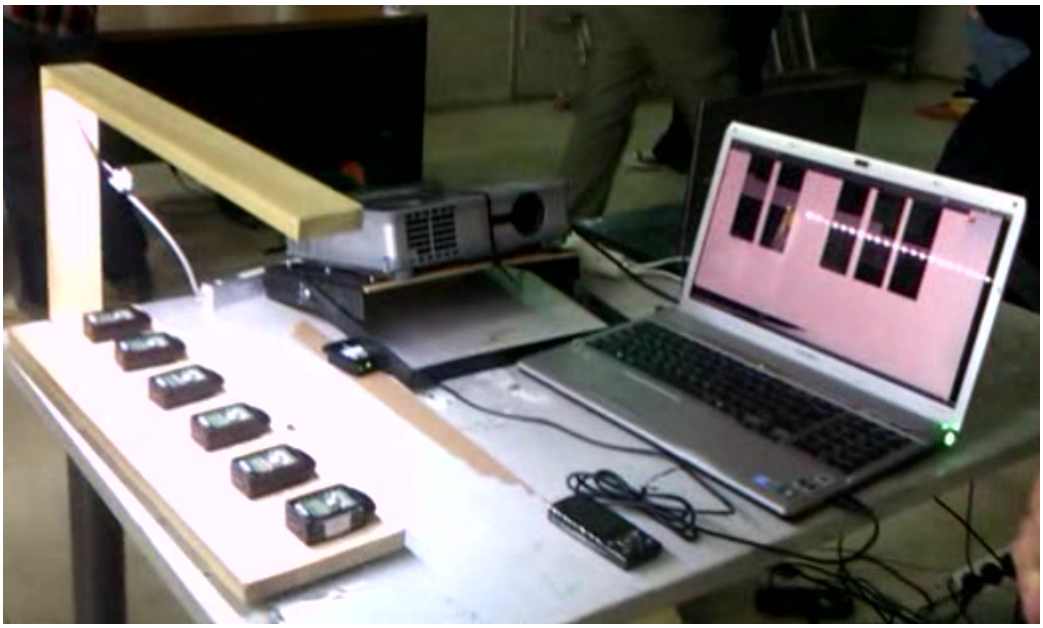
Σε αυτό το κεφάλαιο παρουσιάζεται μια ψυχαγωγική αλληλεπιδραστική εφαρμογή, η οποία δημιουργήθηκε χρησιμοποιώντας τα εργαλεία που εκτέθηκαν στις προηγούμενες ενότητες. Η εφαρμογή που θα περιγραφεί παρακάτω εστιάζει τόσο στην παραγωγή ήχου μέσω της αλληλεπίδρασης με τον χρήστη, όσο και στην δημιουργία του αντίστοιχου γραφικού περιβάλλοντος και ονομάστηκε 'Έξυπνο Πιάνο'

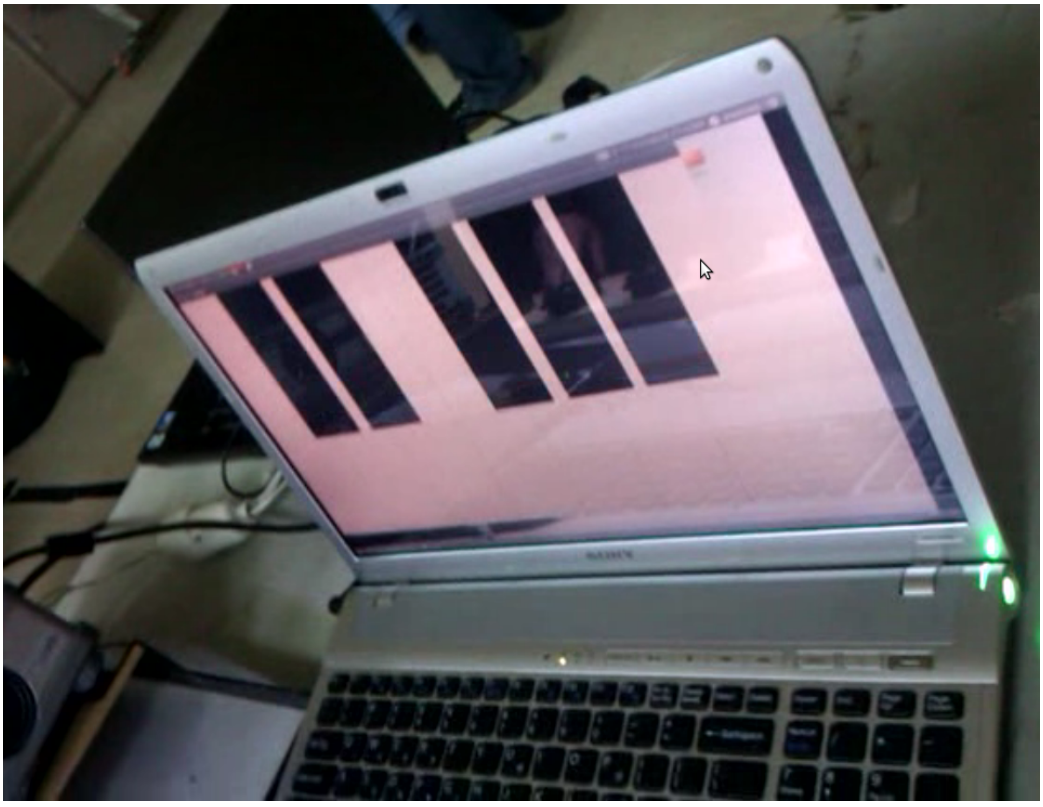
5.2 Έξυπνο Πιάνο

Για την υλοποιήση του χρησιμοποιήθηκαν οι αισθητήρες φωτός απο τις συσκευές SunSPOT με σκοπό να προσομοιωθούν τα δώδεκα βασικά πλήκτρα του πιάνου.

Ο παίχτης τοποθετεί τα χέρια του πάνω από τους αισθητήρες με σκοπό να αλλάξει το έυρος της τιμής του φωτός που αντιλαμβάνεται το εξάρτημα και έτσι να αλλάξει και η νότα που ακούγεται. Οι τιμές αυτές που επηρεάζει ο χρήστης έχουν χωριστεί σε τρία διαστήματα / σκάλες. Η πρώτη σκάλα αντιπροσωπευει τα εφτά άσπρα πλήκτρα ενώ η δεύτερη σκάλα αντιπροσωπεύει τα αντίστοιχα πέντε μαύρα πλήκτρα. Το τρίτο διάστημα αντιπροσωπεύει την έλλειψη αλληλεπίδρασης.

Στην συνέχεια φαίνονται κάποιες φωτογραφίες από την εφαρμογή ενώ στην επόμενη ενότητα σχολιάζεται η υλοποιηση της.





5.3 Υλοποιήση

Για την υλοποιήση της παραπάνω εφαρμογής χρησιμοποιήθηκαν οι συσκευές SunSPOT το γραφικό περιβάλλον Processing και η βιβλιοθήκη Minim όπως και επίσης και η βιβλιοθήκη που παρουσιάστηκε στο τέταρτο κεφάλαιο αυτής της διπλωματικής εργασίας. Ο κώδικας στον κετρικό υπολογιστή αποτελείται από τις κλάσεις: Frame, Processing, Between, SoundCreator, PianoGraphics. Οι πιο σημαντικές από αυτές είναι η Processing και η SoundCreator και οι οποίες παρουσιάζονται και αναλύονται παρακάτω. Κάθε συσκευή SunSPOT τρέχει το λογισμικό που παρουσιάστηκε στην ενότητα 4.3.1.

SoundCreator

```
1  public class SoundCreator extends Thread implements ILThresholdListener
       {
2      /**
3       * variables to hold the lightsensors
4       */
5      LightThresholdSensor s1;
6      LightThresholdSensor s2;
7      LightThresholdSensor s3;
8      LightThresholdSensor s4;
9      LightThresholdSensor s5;
10     LightThresholdSensor s6;
11     /**
12      * queues to hold incoming values
13      */
14     private BlockingQueue<Integer> sl1;
15     private BlockingQueue<Integer> sl2;
16     private BlockingQueue<Integer> sl3;
17
18     private BlockingQueue<Integer> sl4;
19     private BlockingQueue<Integer> sl5;
20     private BlockingQueue<Integer> sl6;
21     /**
22      * constructor
23      */
24     public SoundCreator() {
25         init();
26         s1 = Sensors.getInstance().getSPOT("4AD6").getLightSensor();
27         s2 = Sensors.getInstance().getSPOT("5EF0").getLightSensor();
28         s3 = Sensors.getInstance().getSPOT("4884").getLightSensor();
29         s4 = Sensors.getInstance().getSPOT("4AD7").getLightSensor();
30         s5 = Sensors.getInstance().getSPOT("5952").getLightSensor();
31         s6 = Sensors.getInstance().getSPOT("40EA").getLightSensor();
32
33         s1.setThresholdsTime(400, 700, 1000);
34         s2.setThresholdsTime(400, 700, 1000);
35         s3.setThresholdsTime(400, 700, 1000);
36         s4.setThresholdsTime(400, 700, 1000);
37         s5.setThresholdsTime(400, 700, 1000);
38         s6.setThresholdsTime(400, 700, 1000);
39         s1.addThresholdListener(this);
40         s2.addThresholdListener(this);
41         s3.addThresholdListener(this);
42         s4.addThresholdListener(this);
43         s5.addThresholdListener(this);
44         s6.addThresholdListener(this);
45
```

```
46
47      }
48
49      public void init() {
50          /**
51           * add required sunspots
52           */
53          Sensors.getInstance().addSPOT("4AD6", 37);
54          Sensors.getInstance().addSPOT("5EF0", 37);
55          Sensors.getInstance().addSPOT("4884",37);
56          Sensors.getInstance().addSPOT("4AD7", 37);
57          Sensors.getInstance().addSPOT("5952", 37);
58          Sensors.getInstance().addSPOT("40EA",37);
59          /**
60           * initialize queues
61           */
62          sl1 = new LinkedBlockingQueue<Integer>();
63          sl2 = new LinkedBlockingQueue<Integer>();
64          sl3 = new LinkedBlockingQueue<Integer>();
65          sl4 = new LinkedBlockingQueue<Integer>();
66          sl5 = new LinkedBlockingQueue<Integer>();
67          sl6 = new LinkedBlockingQueue<Integer>();
68      }
69
70      public void run() {
71          while (true) {
72              /**
73               * take values and analyze so that processing would choose
                     sound to play
74               */
75              if ((!(sl1.isEmpty())) && (!(sl2.isEmpty())) && (!(sl3.
                 isEmpty())))&&(!(sl1.isEmpty())) && (!(sl2.isEmpty())) &&
                   (!(sl3.isEmpty()))) {
76                  try {
77                      int value1 = sl1.take();
78                      int value2 = sl2.take();
79                      int value3 = sl3.take();
80                      int value4 = sl4.take();
81                      int value5 = sl5.take();
82                      int value6 = sl6.take();
83
84                      final int[] values = new int[6];
85                      for (int i = 0; i < values.length; i++) {
86                          values[i] = -1;
87                      }
88
89                      if ((value1 == 0)) {
90                          values[0] = 0;
91                      }
```

```
92                      if ((value1 > 40) && (value1 < 150)) {
93                          values[0] = 1;
94                      }
95                      if ((value2 == 0)) {
96                          values[1] = 0;
97                      }
98                      if ((value2 > 40) && (value2 < 150)) {
99                          values[1] = 1;
100                     }
101                     if ((value3 == 0)) {
102                         values[2] = 0;
103                     }
104                     if ((value3 > 40) && (value3 < 150)) {
105                         values[2] = 1;
106                     }
107                     if ((value4 == 0)) {
108                         values[3] = 0;
109                     }
110                     if ((value4 > 40) && (value4 < 150)) {
111                         values[3] = 1;
112                     }
113                     if ((value5 == 0)) {
114                         values[4] = 0;
115                     }
116                     if ((value5 > 40) && (value5 < 150)) {
117                         values[4] = 1;
118                     }
119                     if ((value6 == 0)) {
120                         values[5] = 0;
121                     }
122                     if ((value6 > 40) && (value6 < 150)) {
123                         values[5] = 1;
124                     }
125
126
127                     Between.getInstance().setMessage(values);
128                     Between.getInstance().updateStatus();
129
130
131                     Thread.sleep(100);
132
133                     sl1.clear();
134                     sl2.clear();
135                     sl3.clear();
136                     sl4.clear();
137                     sl5.clear();
138                     sl6.clear();
139
140                 } catch (InterruptedException e) {
```

```
141                     e.printStackTrace();
142                 }
143
144             }
145         }
146     }
147
148     /**
149      * callback
150      * @param lightThresholdSensor the sensor
151      * @param i     the threshold value
152      */
153     public void thresholdExceed(LightThresholdSensor
            lightThresholdSensor, int i) {
154         try {
155             if (lightThresholdSensor == s1) {
156                 sl1.put(i);
157             } else if (lightThresholdSensor == s2) {
158                 sl2.put(i);
159             } else if (lightThresholdSensor == s3) {
160                 sl3.put(i);
161             } else if (lightThresholdSensor == s4) {
162                 sl4.put(i);
163             } else if (lightThresholdSensor == s5) {
164                 sl5.put(i);
165             } else if (lightThresholdSensor == s6) {
166                 sl6.put(i);
167             }
168         } catch (InterruptedException e) {
169             e.printStackTrace();
170
171         }
172     }
173 }
```

Η κλάση αυτή τρέχει στο δικό της Thread και υλοποιεί το ILThresholdListener interface. Οι SunSPOT συσκευές που θα χρησιμοποιηθούν στην εφαρμογή δηλώνονται στον δημιουργό της κλάσης παρέχοντας την μοναδική 16-bit MAC-address του καθενός (γραμμές 49-58).

Στην συνέχεια μέσω της μεθόδου **setThresholdsTime()** ορίζονται για κάθε έναν αισθητήρα τα κατώφλια για τα οποία θα ειδοποιεί την εφαρμογή (γραμμές 33-38). Επίσης εγγράφεται στην λίστα με τους Observers του κάθε αισθητήρα (γραμμές 39-44).

Στις γραμμές 153-172 φαίνεται η υλοποιήση της public void thresholdExceed(LightThresholdSensor lightThresholdSensor, int i). Κάθε φορά που έρχεται μια ειδοποιήση από κάποια συσκευή η ακέραια τιμή του φωτός τοπο-

θετείται στην αντίστοιχη ουρά προτεραιότητας. Έπειτα περιοδικά ελέγχονται οι τιμές αυτές για κάθε αισθητήρα και ανάλογα με το εύρος στο οποίο ανήκουν τίθενται και οι τιμές σε έναν byte array, ο οποίος μεταδίδεται στην Processing προς επεξεργασία. Αξίζει να σημειωθεί οτι ο έλεγχος του εύρους γίνεται για να οριστεί το επίπεδο στο οποίο βρίσκεται το χέρι του παίχτη και έτσι να ακουστεί και η αντίστοιχη νότα.

Processing

```
1  public class Processing extends PApplet implements Observer {
2
3      Minim minim;
4      AudioSample[] audioSamples = new AudioSample[6];
5      AudioSample[] audioSamplesUp = new AudioSample[6];
6
7      private int[] player = new int[6];
8
9      public void setup() {
10         size(1400 ,800);
11
12         minim = new Minim(this);
13
14         audioSamples[0] = minim.loadSample("/home/anastasia/Desktop/c.
               mp3", 1024);
15         audioSamples[1] = minim.loadSample("/home/anastasia/Desktop/d.
               mp3", 1024);
16         audioSamples[2] = minim.loadSample("/home/anastasia/Desktop/e.
               mp3", 1024);
17         audioSamples[3] = minim.loadSample("/home/anastasia/Desktop/f.
               mp3", 1024);
18         audioSamples[4] = minim.loadSample("/home/anastasia/Desktop/g.
               mp3", 1024);
19         audioSamples[5] = minim.loadSample("/home/anastasia/Desktop/a.
               mp3", 1024);
20         audioSamplesUp[0] = minim.loadSample("/home/anastasia/Desktop/c
               #.mp3", 1024);
21         audioSamplesUp[1] = minim.loadSample("/home/anastasia/Desktop/d
               #.mp3", 1024);
22         audioSamplesUp[2] = minim.loadSample("/home/anastasia/Desktop/b.
               mp3", 1024);
23         audioSamplesUp[3] = minim.loadSample("/home/anastasia/Desktop/f
               #.mp3", 1024);
24         audioSamplesUp[4] = minim.loadSample("/home/anastasia/Desktop/g
               #.mp3", 1024);
25         audioSamplesUp[5] = minim.loadSample("/home/anastasia/Desktop/a
               #.mp3", 1024);
```

```
26
27          for (int i = 0; i < player.length; i++) {
28              player[i] = -1;
29          }
30      }
31
32      public void draw() {
33          for (int i = 0; i < 6; i++) {
34              if (player[i] == 0) {
35                  audioSamples[i].trigger();
36              }
37              if (player[i] == 1) {
38                  audioSamplesUp[i].trigger();
39              }
40              player[0] = -1;
41          }
42      }
43      public void update(Observable observable, Object o) {
44          if (!(observable instanceof Between)) {
45              return;
46          }
47
48          player = (int[]) o;
49
50          (new PianoGraphics(this, player)).start();
51
52      }
53  }
```

Η κλάση αυτή κληρονομεί από την PApplet. Αρχικά με την βοήθεια της βιβλιοθήκης Minim φορτώνει τα samples για κάθε νότα. Κάθε φορά που έρχεται και μια νέα ειδοποιήση από την SoundCreator ανάλογα με τις τιμές του byte-array (μηδέν για το κάτω επίπεδο , ένα για το πάνω, -1 για την έλλειψη αλληλεπίδρασης με τον αντίστοιχο αισθητήρα) πυροδοτείται και το αντίστοιχο sample όπως φαίνεται στις γραμμές 32-42.

Κεφάλαιο 6

GitHub-Social Coding

6.1 Τι είναι το Git?

Το Git [20] είναι ένα open-source κατανεμειμένο σύστημα ελέγχου εκδόσεων πηγαίου κώδικα, που δημιουργήθηκε από τον Linus Torvalds. Χρησιμοποιείται από πολλά έργα ανοιχτού λογισμικού, με ίσως το πιο ιδιαίτερο από αυτά να είναι ο Linux Kernel. Κάθε Git κατάλογος αρχείων είναι ένα πλήρως ανεπτυγμένο σύστημα αποθήκευσης με δυνατότητες παρακολούθησης της πορείας των εργασιών. Τα πιο σημαντικά χαρακτηριστικά του είναι:

- **Κατανεμημένη ανάπτυξη:** Το Git δίνει σε κάθε προγραμματιστή ένα τοπικό αντίγραφο ολόκληρης της αναπτυξιακής ιστορίας του έργου, και οι αλλαγές αντιγράφονται και συγχωνεύονται μεταξύ διαφορετικών τέτοιων αντιγράφων.

- **Αποδοτικός χειρισμός μεγάλων έργων:** Το Git είναι πολύ γρήγορο και κλιμακώνεται καλά ακόμα και όταν δουλεύει με μεγάλα έργα και μεγάλο ιστορικό ανάπτυξης.

- **Κρυπτογραφική πιστοποιήση της ιστορίας ανάπτυξης:** Η ιστορία ανάπτυξης αποθηκεύται με τέτοιο τρόπο, έτσι ώστε το όνομα μια συγκεκριμένης έκδοσης του έργου να εξαρτάται απο την πορεία ανάπτυξης που οδήγησε σε αυτήν την έκδοση.

- **Συλλογή εργαλέιων:** Το Git είναι μια συλλογή εργαλείων υλοποιημένων σε C, και έναν αριθμό από script. Προσφέρει εργαλεία τόσο για την εύκολη χρήση του όσο και για την επέκταση του με νέες έξυπνες λειτουργίες.

6.2 Τι είναι το GitHub?

Το GitHub [21] είναι μια νέα διαδικτυακή υπηρεσία για την ανάπτυξη λογισμικού, που χρησιμοποιεί το Git. Η βασική φιλοσοφία του είναι να κάνει εύκολη την συνεργασία στη ανάπτυξη ενός έργου λογισμικού. Στοχευει τόσο σε μεμονωμένους προγραμματιστές όσο και σε εταιρείες που απλά χρειάζονται ένα ασφαλές σύστημα ελέγχου εκδόσεων. Η εταιρεία προσφέρει επίσης ένα ελεύθερο πακέτο, που είναι κατάλληλο για open-source έργα.

Κάθε χρήστης του GitHub όπως και επίσης και κάθε έργο έχει το δικό του προφίλ το οποίο την συμμετοχή του προγραμματιστή και την πρόοδο του έργου. Οι χρήστες μπορούν να παρακολουθούν άλλους χρήστες και αλλά έργα, όπως και επίσης να συμμετέχουν στην ανάπτυξή τους. Το GitHub είναι ένα κοινωνικό δίκτυο για του προγραμματιστές και όπως ο Peter Cooper

επισημαίνει :" Αν χρειάζεστε έναν λόγο γιατί το GitHub είναι σχετικό είναι γιατί η διαδικτυακή επαφή συνδέει το φιλοξενούμενο έργο και τις συμμετοχές του σε αυτό. Η διαχείριση του δικού σου προσωπικού Git repository δεν είναι κάτι δύσκολο, αλλά η διαχείρηση των συμμετοχών σε ένα έργο από άτομα που δεν γνωρίζεις προσωπικά είναι κάτι εξαιρετικά δύσκολο. "

Τα ιδιαίτερα χαρακτηριστικά του GitHub είναι:

- **Organizations:** Είτε πρόκειται για open-source έργα, είτε πρόκειται για εταιρείες, οι Organizations απλοποιούν εξαιρετικά την διαχείριση μιας ομάδας. Με τις ομάδες δίνεται σε κάθε συμμετέχοντα προγραμματιστή τόση ελευθερία όση χρειάζετε, από την δυνατότητα να δημιουργεί έργα για την ομάδα μέχρι την read-only πρόσβαση σε έργα που ήδη υπάρχουν. Για τις εταιρείες οι Organizations προσφέρουν την δυνατότητα στα άτομα που διαχειρίζονται τα οικονομικά της ομάδας να συνεργάζοντα απλά και εύκολα με τα άτομα που συντονίζουν το έργο, χρησιμοποιώντας απλά και μόνο τους GitHub λογαριασμούς τους.

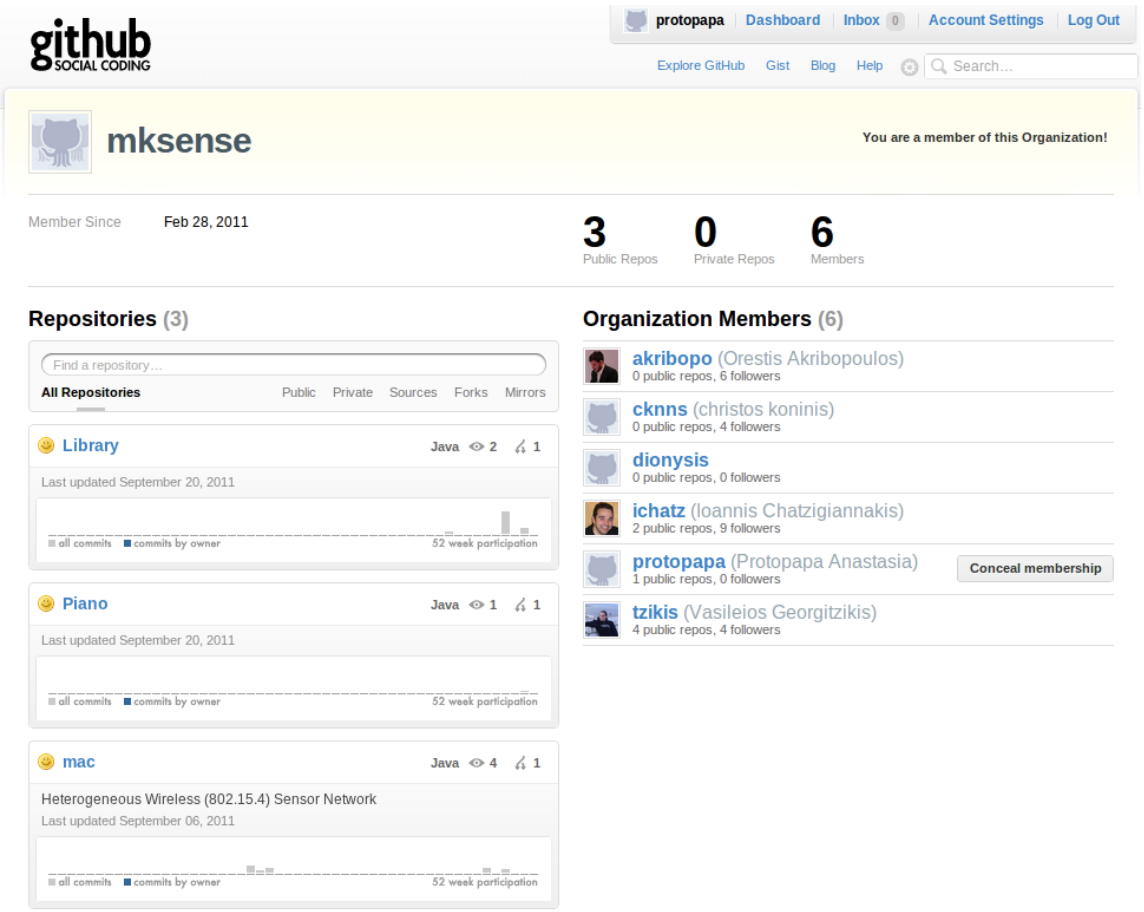
- **Issues Tracking:** Κάθε Git repository είναι εξοπλισμένο με ένα ιστορικό των προβλήμάτων που έχουν εντοπιστεί σε κάθε έργο.

- **Code Review:** Κάθε Git-pull αίτηση λαμβάνει υπόψιν της όχι μόνο τι θέλει ο χρήστης να κατεβάσει, αλλά και που θέλει να τις εφαρμόσει τις οποιεσδήποτε αλλαγές. Ετσι μια ομάδα μπορεί να συζητά τις αλλαγές χωρίς κάποιο ιδιαίτερο πρόβλημα. Επίσης οι χρήστες μπορούν να αφήνουν και σχόλια για τις αλλαγές που έκαναν. Από τα πιο σημαντικά χαρακτηριστικά που κάνει το GitHub τόσο αγαπητό είναι για την ευκολία ανασκόπησης των διαφορετικών εκδόσεων του έργου, και τις οποίες ο χρήστης μπορεί εύκολα να συγκρίνει μεταξύ τους, είτε μέσω διαδικτύου είτε μέσω του τοπικού του αρχείου.

6.3 Περιήγηση στο Github μέσω της διπλωματικής αυτής εργασίας

6.3.1 Κεντρική σελίδα

Πρόσβαση στην διπλωματική εργασία αυτή μέσω του GitHub παρέχεται μέσω του link: ***https://github.com/mksense***. Στη συνέχεια φαίνεται ένα screenshot από την κεντρική σελίδα:



Σχήμα 6.1: Κεντρική σελίδα

Στο πάνω μέρος της σελίδας φαίνεται το όνομα της ομάδας/organization: "**mkSense**". Στα δεξιά φαίνεται μια λίστα με τα άτομα που συμμετέχουν στην ομάδα, μαζί με κάποια επιπλέον στοιχεία όπως τα public repository που έχουν μέσα στο GitHub, και τους προγραμματιστές που παρακολουθούν τη δουλειά τους. Στα αριστερά φαίνονται τα repositories που διαχειρίζεται η ομάδα. Συγκεκριμένα επιλέχθηκε να δημιουργηθούν τρία repositories.

- **mac**: Το repository αυτό περιέχει την radio stack για την ανάπτυξη ενός ετερογενούς δικτύου μεταξύ τεσσάρων συσκευών (TelosB, SunSPOT, Arduino, iSense) που υποστηρίζουν το IEEE 802.15.4 standard. Η βιβλιοθήκη που παρουσιάστηκε στην διπλωματική αυτή εργασία, αναπτύχθηκε πάνω σε αυτή την radio stack για την υποστηριξη της επικοινωνίας με τον κεντρικό υπολογιστή ανεξάρτητα του συνδεδεμένου radio (XBee module ή SunSPOT BaseStation).

- **Library**: Στο repository αυτό περιέχεται η βιβλιοθήκη που παρουσιάστηκε στο τέταρτο κεφάλαιο αυτής της διπλωματικής εργασίας.
- **Piano**: Εδώ περιέχεται ο κώδικας της αλληλεπιδραστικής εφαρμογής που εκτέθηκε στο πέμπτο κεφάλαιο.

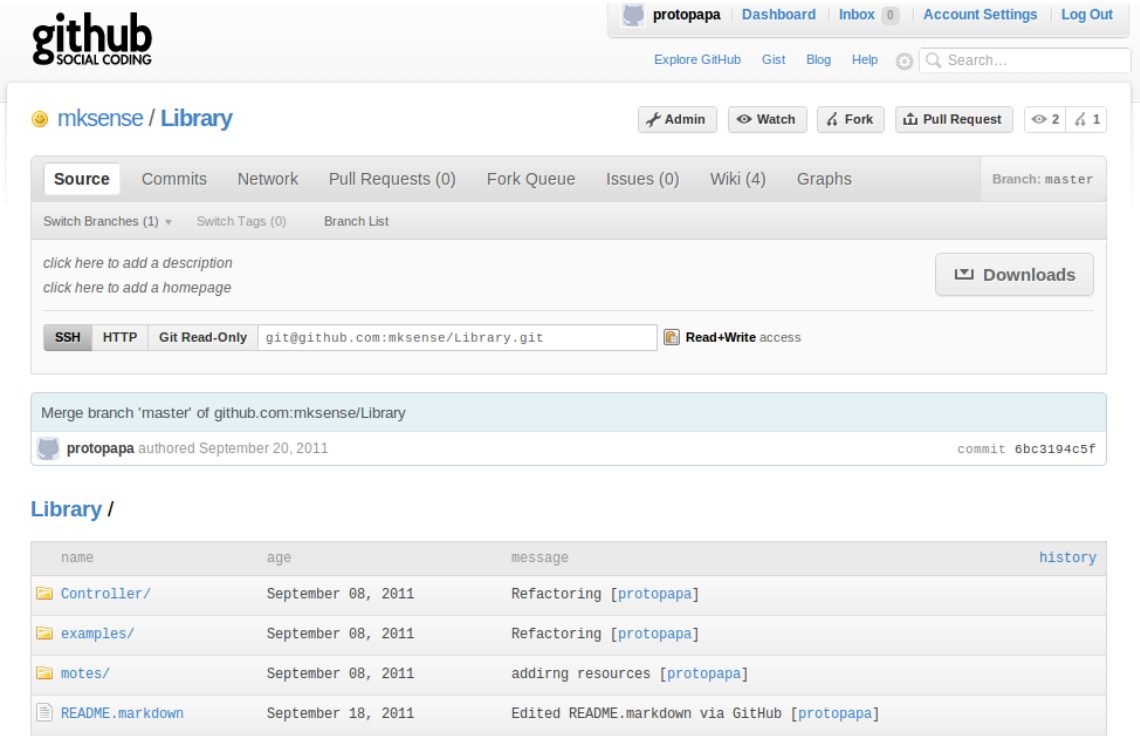
6.3.2 Repository

Στη συνέχεια ρίχνουμε μια πιο λεπτομερή ματιά σε ένα repository.

Figure 6.2: Βασική εικόνα ενός repository

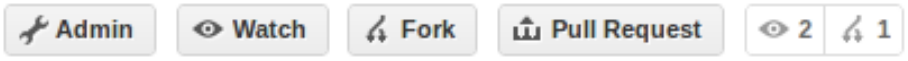
Πάνω αριστερά φαίνεται το όνομα του repository : "**Library**". Αντίστοιχα στα δεξιά υπάρχουν στοιχεία για την γλωσσα προγραμματισμού που χρησιμοποιείται (στην περίπτωση μας: Java) και για τους χρήστες που παρακολουθούν (watchers) το συγκεκριμένο repository. Επίσης δίνονται και πληροφορίες για την τελευταια ημέρα τροποποιήσης των περιεχομένων όπως και επίσης στατιστικά για τα commits που έχουν παραγματοποιηθεί σε διάρκεια 52 βδομάδων.

Η είσοδος στο repository μας βγάζει σε μια σελίδα που έχει γενικώς την μορφή που φαίνεται στην παρακάτω εικόνα και επεξηγείται λεπτομερώς στην συνέχεια:



Σχήμα 6.3: Η εσωτερική δομή της κεντρικής σελίδας ενός repository

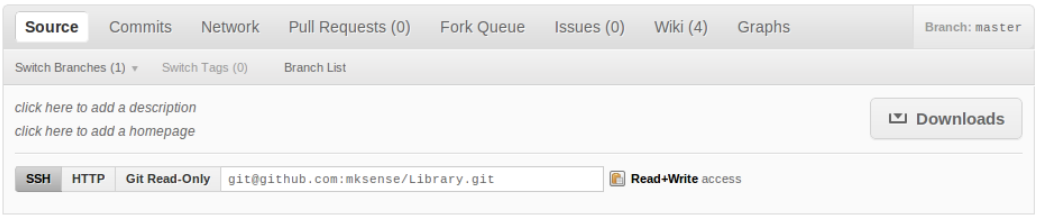
Πάνω δεξιά φαίνονται οι εξής δυνατότητες:



- **Admin**: παρέχει λειτουργίες για την διαχείρηση του repository, όπως για παράδειγμα την αλλαγή του ονόματος.
- **Watch**: μέσω αυτής της λειτουργίας κάθε χρήστης του GitHub μπορεί να παρακολουθεί την πρόοδο της εργασίας που φιλοξενείται.
- **Fork**: Κάθε προγραμματιστής που θέλει να συμβάλλει στην εκάστοτε εργασία κάνει Fork ώστε να κατεβάσει τον κώδικα στον προσωπικό του υπολογιστή ή να δημιουργήσει ένα αντίγραφο σε ένα προσωπικό του repository, και να εργαστεί με αυτόν.

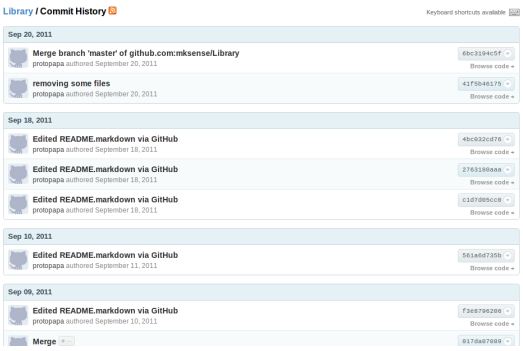
- **Pull Request** : Αυτή η λειτουργία παρέχεται για να ειδοποιούνται οι κάτοχοι του εκάστοτε repository ότι κάποιος εξωτερικός προγραμματιστής συνέβαλλε στην εργασία και να την ελέξουν με σκοπό την αποδοχή ή την απόρριψη.

Ακολούθως επεξηγήτε το βασικό menu πλοήγησης στις διάφορες δυνατότητες που προσφέρει το GitHub :



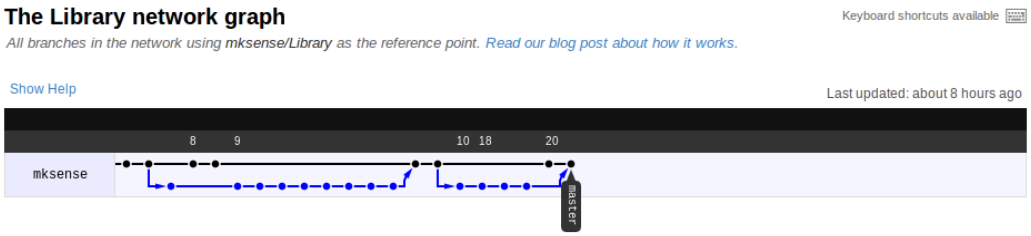
Συγκερκιμένα η επιλογή **Source** παραθέτει τον κώδικα υπο την μορφή που έχει ορίσει ο ιδιοκτήτης, σε διάφορους φακέλους και υποφακέλους, όπως φαίνεται και στο σχήμα 6.2. Παράλληλα φαίνεται και ο χρήστης που τροποποιήσε κάθε φάκελο μαζί με ένα μικρό σχόλειο, το οποίο το καταθέτει ο ίδιος, για τις αλλαγές που έκανε.

Η επιλογή **Commits** παρέχει ολοκληρο το ιστορικό των commits και άρα της προόδου και των αλλαγών που γίνονται στην εκάστοτε εργασία. Παρέχονται πληροφορίες για το ποιος έκανε το commit, την ημερόμηνία όπως και επίσης και μια μικρή περιγραφή για την εκάστοτε ενέργεια. Ενα παράδειγμα για την παρουσιάση αυτών φαίνεται στο σχήμα 6.3.



Σχήμα 6.4: Commits

Η επιλογή **Network** παραθέτει με έναν γράφο το ιστορικό του repository δηλάδή τα commits και τα merges που έγιναν καθ'όλη την πορεία της εργασίας:



Σχήμα 6.5: Παράδειγμα ενός Network Graph

Οι επιλογές **Pull Requests** και **Fork Queue** αντιπροσωπεύουν αντίστοιχα τις λειτουργιες για την ειδοποιήση του ιδιοκτήτη για τις εξωτερικές συμμετοχές που έχει στην εργασία του και την αποδοχή η απόρριψη τους.

Η επιλογή **Issues** περιέχει μια λίστα απο τυχόν επισημάνσεις χρηστών πάνω στον κώδικα με σκοπό την βελτίωση της κάθε εργασίας.

Τέλος η επιλογή **Wiki** προσφέρει τη δυνατότητα για την δημιουργία documentation για το κάθε repository με την μορφή Wiki Pages. Για την δημιουργία αυτών παρέχονται διάφορα formats, όπως MarkDown, Textile και άλλα, ώστε να διαλέξει ο καθένας ότι τον βολεύει περισσότερο.

Το GitHub λοιπόν, όπως φάνηκε και μέσα από την περιήγηση, είναι μια υπηρεσία που ενισχύει τον κοινωνικό προγραμματίσμο, τόσο μεταξύ των ομάδων, όσο και μεταξύ ολόκληρης της προγραμματιστικής κοινότητας. Όπως είδαμε προσφέρει όλα τα εργαλεία για την σωστή διεκπαιρέωση μιας εργασίας, όπως και επίσης την γρήγορη και εύκολη παρακολούθηση της εξέλιξης και πορείας της.

Κεφάλαιο 7

Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Στη διπλωματική αυτή εργασία παρουσιάζεται ο σχεδιασμός και η χρήση μιας βιβλιοθήκης για τον έλεγχο ετερογενών συσκευών διάχυτου υπολογισμού. Παρουσιάστηκαν οι συσκευές για τις οποίες έχει υλοποιηθεί μέχρι στιγμής η βιβλιοθήκη, τα χαρακτηριστικά τους και οι δυνατότητες τους. Επίσης παρουσιαστηκαν η δομή και η αρχιτεκτονική της βιβλιοθήκης όπως και επίσης το γενικό πρωτόκολλο πάνω στο οποίο στηρί. Τέλος παρουσιάζεται η δημιουργία ενός αλληλεπιδρεστικού παιχνιδιού που ονομάστηκε " Smart Piano".

Οι προοπτικές της συγκεκριμένης εργασίας περιλαμβάνουν τις προσπάθειες προσθήκης επιπλέον συσκευών που βασίζονται τόσο στο πρότυπο 802.15.4 όσο και σε άλλες ασύρματες τεχνολογίες, για τον κοινό έλεγχο τους μέσω της βιβλιοθήκης, την επέκταση στην υποστήριξη περισσότερων λειτουργιών για κάθε συσκευή, όπως και επίσης και την δημιουργία και άλλων αλληλεπιδραστικών εφαρμογών.

Βιβλιογραφία

[1] Akribopoulos Orestis, Georgitzikis Vasileios, Protopapa Anastasia, Chatzigiannakis Ioannis: "Building a Platform-agnostic Wireless Network of Interconnected Smart Objects". 15th Panhellenic Conference on Informatics with international participation (PCI 2011). Διαθέσιμο στο σύνδεσμο:

http://ru1.cti.gr/aigaion/?page=publicationkind=singleID=881.

[2] Pervasive Computing: http://en.wikipedia.org/wiki/Ubiquitous_computing

[3] Pervasive Computing Technologies:

http://www.parliament.uk/documents/post/postpn263.pdf

[4] http://www.ulster.ac.uk/scienceinsociety/sporttech.pdf

[5] http://en.wikipedia.org/wiki/Digital$_h ome$

[6] Human Pacman: a sensing-based mobile entertainment system with ubiquitous computing and tangible interaction by Cheok, Adrian David and Fong, Siew Wan and Goh, Kok Hwee and Yang, Xubo and Liu, Wei and Farzbiz, Farzam. Proceedings of the 2nd workshop on Network and system support for games.

[7] Uncle Roy All Around You: Implicating the City in a Location-Based Performance by Steve, Benford and Martin, Flintham and Adam, Drozd and Rob, Anastasi and Duncan, Rowland and Nick, Tandavanitj and Matt, Adams and Ju, Row-Farr and Amanda, Oldroyd and Jon, Sutton

[8] Chatzigiannakis Ioannis, Mylonas Georgios, Akribopoulos Orestis, Logaras Marios, Kokkinos Panagiotis, and Spirakis Paul, The "Hot Potato" Case: Challenges in Multiplayer Pervasive Games Based on Ad hoc Mobile Sensor Networks and the Experimental Evaluation of a Prototype Game, in: arxiv.org (Feb 2010).

[9] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=4660082

[10] sunSPOT: http://www.sunspotworld.com/

[11] Squack VM: http://en.wikipedia.org/wiki/Squawk_virtual_machine

[12] Arduino: http://www.arduino.cc/

[13] Arduino: http://en.wikipedia.org/wiki/Arduino

[14] XBee module: http://en.wikipedia.org/wiki/XBee

[15] XBee module: http://ftp1.digi.com/support/documentation/90000982_A.pdf

[16] XBeeShield: http://www.arduino.cc/en/Main/ArduinoXbeeShield

[17] Processing.org: http://processing.org/

[18] Processing.org: http://en.wikipedia.org/wiki/Processing_(programming_language)

[19] Observer Pattern: http://en.wikipedia.org/wiki/Observer_pattern

[20] Git: http://git-scm.com/

[21] GitHub: https://github.com/