ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΕΛΕΤΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΥΛΙΚΟΥ ΣΥΣΤΗΜΑΤΩΝ ΑΣΥΡΜΑΤΩΝ ΑΙΣΘΗΤΗΡΩΝ ΓΙΑ ΤΟ INTERNET OF THINGS

ΓΕΩΡΓΙΤΖΙΚΗΣ ΒΑΣΙΛΕΙΟΣ

ΑΜ. 3906

ΕΠΙΒΛΕΠΩΝ: ΚΑΘΗΓΗΤΗΣ ΠΑΥΛΟΣ ΣΠΥΡΑΚΗΣ
ΣΥΝΕΠΙΒΛΕΠΩΝ: ΔΡ. ΙΩΑΝΝΗΣ ΧΑΤΖΗΓΙΑΝΝΑΚΗΣ

ΠΑΤΡΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2013

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Σπυράκη Παύλο, καθηγητή του Τμήματος Μηχανικών Η/Υ και Πληροφορικής και Διευθυντή του ΙΤΥΕ «Διόφαντος» για την εμπιστοσύνη που μου έδωσε στην ανάθεση της διπλωματικής, και στην καθοδήγηση του κατά την εργασία μου ως ερευνητής στην Ερευνητική Μονάδα 1 του ΙΤΥΕ. Επίσης, θα ήθελα να ευχαριστήσω θερμά τον Γιάννη Χατζηγιαννάκη, για την καθοδήγηση του, την συμπαράσταση του, και την αμέριστη υπομονή όλα αυτά τα χρόνια, καθώς και τους μεταπτυχιακούς και διδακτορικούς της ΕΜ1 (Ορέστη, Δημήτρη, Μάριο, Έφη, Γιώργο, και όλους όσους με ανέχτηκαν όσο καιρό πέρασα εκεί).

Επίσης θα ήθελα να ευχαριστήσω την οικογένεια μου, την μητέρα μου Ελένη, τον πατέρα μου Γιώργο, και τις αδερφές μου Ελπίδα και Άννα για όλα τα εφόδια, την συμπαράσταση (και την απαραίτητη πίεση) που μου έδωσαν για να φτάσω εδώ που είμαι.

Τέλος, όλους τους φίλους που με υποστήριξαν, τους “συγκατοίκους μου στην τρέλα” στο P-Space και στο codebender, τους φίλους στο υπολογιστικό κέντρο του τμήματος Μηχανικών Η/Υ και Πληροφορικής, και όλους όσους με στήριξαν και το αχάριστο μυαλό μου ξεχνάει.

Βασίλης Γεωργιτζίκης
Σεπτέμβριος 2013

Μελέτη και Ανάπτυξη Λογισμικού και Υλικού Συστημάτων Αισθητήρων για το Internet of Things

Εισαγωγή

Κίνητρο και Σημασία

Κίνητρο της Διπλωματικής Εργασίας είναι η επίλυση των προβλημάτων που συναντούνται συχνά στα υπάρχοντα και αναπτυσσόμενα Ασύρματα Δίκτυα Αισθητήρων, τα οποία αφορούν την διασυνδεσιμότητα των υπάρχοντων ασύρματων συσκευών μεταξύ τους καθώς και με το Διαδίκτυο (Internet). Αυτά τα προβλήματα αποτελούν και τροχοπέδη για την ανάπτυξη συσκευών και εφαρμογών για το Internet of Things, ο οποίος είναι ένας από τους πιο ραγδαία αναπτυσσόμενους κλάδους σε επίπεδο έρευνας αλλά και της αγοράς ηλεκτρονικών συσκευών.

Στόχοι της Διπλωματικής Εργασίας

Πρωταρχικός στόχος της εργασίας είναι η ανάπτυξη τεχνολογίας που επιτρέπει την δημιουργία ετερογενών δικτύων ασύρματων αισθητήρων και ελεγκτών, το οποίο αποτελείται από πολλές διαφορετικές πλατφόρμες και συσκευές, και οι οποίες έχουν την δυνατότητα να επικοινωνούν μεταξύ τους χωρίς ενδιάμεσους κόμβους-μεταφραστές, χρησιμοποιώντας το πρωτόκολλο ασύρματης επικοινωνίας για δίκτυα χαμηλής κατανάλωσης IEEE 802.15.4.

Δευτερεύον στόχος είναι αυτές οι συσκευές να μπορούν να αναγνωρίζονται και να παραμετροποιούνται αυτόματα χωρίς την ανάγκη επιπλέον προγραμματισμού για κάθε αλλαγή που γίνεται στην προσθήκη ή αφαίρεση συσκευών στο δίκτυο, ή αισθητήρων και ελεγκτών σε μια μεμονωμένη συσκευή.

Συνεισφορά της Διπλωματικής Εργασίας

Στα πλαίσια της εργασίας αυτής, σχεδιάστηκε και αναπτύχθηκε η βιβλιοθήκη mkSense, η οποία δρα ως ενδιάμεσο λογισμικό στο επίπεδο δικτύωσης (network stack) των συσκευών, και επιτρέπει στις συσκευές του δικτύου να επικοινωνούν μεταξύ τους χρησιμοποιώντας ένα ενιαίο πρωτόκολλο επικοινωνίας.

Στη συνέχεια, αναπτύχθηκε ένα λογισμικό που μας επιτρέπει να προσθέτουμε και να αφαιρούμε συσκευές στο δίκτυο με αυτόματη αναγνώριση και προσαρμογή του δικτύου. Τέλος, αναπτύχθηκε ένας συνδυασμός λογισμικού και υλικού, ο οποίος μας δίνει την δυνατότητα να συνδέσουμε οποιονδήποτε συνδυασμό αισθητήρων και ελεγκτών επιθυμούμε στις συσκευές μας σε πραγματικό χρόνο και χωρίς επανεκκίνηση των συσκευών (hot-swap και plug-n-play), καθώς οι συσκευές μπορούν να αναγνωρίζουν τις αλλαγές και να παραμετροποιούνται αντίστοιχα.

Δομή της Διπλωματικής Εργασίας

Στα Κεφάλαια 1 & 2 παρουσιάζεται μια εισαγωγή των προβλημάτων που επιλύονται, καθώς και οι τεχνολογίες και συσκευές που χρησιμοποιήθηκαν. Στο Κεφάλαιο 3 παρουσιάζεται η βιβλιοθήκη ασύρματης ετερογενής δικτύωσης που αναπτύχθηκε στα πλαίσια της εργασίας. Στα Κεφάλαια 4 & 5 παρουσιάζονται δυο βιβλιοθήκες που αναπτύχθηκαν για την ευκολότερη ανάπτυξη εφαρμογών και συσκευών, και την επικοινωνία με τυποποιημένα πρωτόκολλα και συστήματα για το Internet of Things (π.χ. CoAP, Uberdust). Στο Κεφάλαιο 6 παρουσιάζεται ο συνδυασμός υλικού και λογισμικού που δίνει την δυνατότητα αυτόματης αναγνώρισης και παραμετροποίησης σε επίπεδο κόμβου και δικτύου. Τέλος, στο Κεφάλαιο 7 αναλύονται τα συμπεράσματα της εργασίας, και στο Κεφάλαιο 8 η Βιβλιογραφία.

Συσκευές και Εφαρμογές

Wireless Sensor Networks

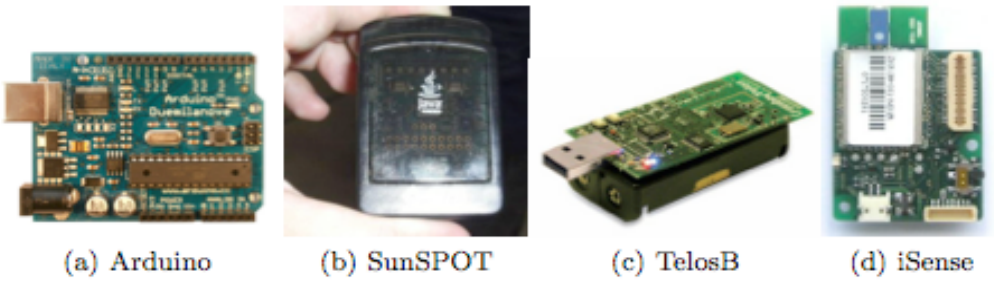
Ένα ασύρματο δίκτυο αισθητήρων (WSN - Wireless Sensor Network) είναι ένα δίκτυο που αποτελείται από κατανεμημένες συσκευές, οι οποίες παρέχουν δυνατότητες αισθητήρων όπως μέτρηση θερμοκρασίας, ήχου, δόνησης, πίεσης, υγρασίας κοκ. Τα τελευταία χρόνια, τα συστήματα που χρησιμοποιούν ασύρματα δίκτυα αισθητήρων (WSN) είναι ολοένα και περισσότερα.

Το Arduino[1], που είναι μια από τις πιο κοινές πλατφόρμες για ανάπτυξη πρωτοτύπων και χρησιμοποιείται διεθνώς (όχι μόνο ως πλατφόρμα ανάπτυξης υλικού - hardware - αλλά και για την δημιουργία διαδραστικών αντικειμένων ή περιβάλλοντων) θα ήταν ιδανικός κόμβος σε ένα δίκτυο ασύρματων αισθητήρων, λόγω του μικρού του μεγέθους, του χαμηλού κόστους και της επεκτασιμότητας του.

Παρ' όλα αυτά, το Arduino δεν έχει κανένα τύπο ασύρματης επικοινωνίας, κάτι που το καθιστά αδύνατο να χρησιμοποιηθεί σε ένα WSN. Επομένως, ο πρωταρχικός / σκοπός αυτής της εργασίας ήταν η επέκταση των δυνατοτήτων του Arduino με την σύνδεση ενός συστήματος ασύρματης επικοινωνίας συμβατής με το πρωτόκολλο IEEE 802.15.4[2], ώστε να δοθεί η δυνατότητα στο Arduino για αμφίδρομη ασύρματη επικοινωνία. Για αυτό χρησιμοποιήθηκε η συσκευή ασύρματης δικτύωσης XBee[3], στην οποία θα αναφερθούμε στο μέλλον.

Με αυτό τον τρόπο, το Arduino μπορεί να δράσει ως Smart Object (Έξυπνο Αντικείμενο) και να εκθέσει τις δυνατότητες του στο Internet of Things (το Διαδίκτυο των Αντικειμένων).

Έτσι, είναι δυνατή η σχεδίαση ενός ετερογενούς δικτύου ασύρματων αισθητήρων, που αποτελείται από διαφορετικές συσκευές. Στην συγκεκριμένη περίπτωση, αναφερόμαστε συγκεκριμένα σε 4 διαφορετικές πλατφόρμες (Arduino, SunSPOT[4], TelosB[5], iSense[6]), οι οποίες αποτελούν τις ποιο διαδεδομένες και ενεργές στον ακαδημαϊκό και ερευνητικό τομέα.



(a) Arduino (b) SunSPOT (c) TelosB (d) iSense

Οι 4 πλατφόρμες που υποστηρίζονται

Ο συνδυασμός αυτών των τεσσάρων διαφορετικών πλατφορμών έχει το πλεονέκτημα της χρήσης όλων των χαρακτηριστικών της κάθε μιας σε ένα ενιαίο δίκτυο. Ο συνδυασμός Arduino & XBee είναι ιδανικός για το χαμηλό κόστος, την ανοιχτή αρχιτεκτονική, και για τον έλεγχο ηλεκτρικών κυκλωμάτων (π.χ. κλιματισμός, θέρμανση, φωτισμός), το SunSPOT λόγω της δυνατής επεξεργαστικής ισχύος του, και τα iSense και TelosB για την χαμηλή κατανάλωση τους.

Κάποια από τα βασικά προβλήματα που προκύπτουν είναι η ασυμβατότητα στο software αλλά και στο hardware μεταξύ διαφόρων συσκευών, η οποία έχει ως αποτέλεσμα την ανύπαρκτη ή, στην καλύτερη περίπτωση, ελλειπή συνδεσιμότητα μεταξύ των διαφορετικών πλατφορμών ασύρματων δικτύων αισθητήρων, λόγω της ελλιπούς υλοποίησης του πρωτοκόλλου 802.15.4 στην κάθε πλατφόρμα.

Όλες οι πλατφόρμες έχουν υποστήριξη για ασύρματη επικοινωνία σύμφωνα με το πρωτόκολλο 802.15.4, αλλά διαφέρουν ως προς τον βαθμό υποστήριξης, τον αριθμό των διαφορετικών υπό-πρωτοκόλλων που υποστηρίζουν, αλλά και τις παραδοχές που κάνει η κάθε πλατφόρμα. Έτσι, είναι αδύνατη η επικοινωνία μεταξύ διαφορετικών συσκευών.

Τη λύση σε αυτό το πρόβλημα έρχεται να δώσει μια βιβλιοθήκη η οποία λειτουργεί σε επίπεδο network stack (στο επίπεδο δικτύωσης μεταξύ των συσκευών), και αντιμετωπίζει τα παραπάνω προβλήματα, επιτρέποντας την επικοινωνία μεταξύ διαφορετικών πλατφορμών hardware αλλά και software.

Η βιβλιοθήκη αυτή ονομάζεται mkSense[7], και η υλοποίηση της για την πλατφόρμα του Arduino αναπτύχθηκε ως μέρος της εργασίας αυτής, σε συνεργασία με μεταπτυχιακούς φοιτητές του τμήματος οι οποίοι ανέπτυξαν τις υλοποιήσεις για τις υπόλοιπες πλατφόρμες. Η ανάπτυξη αφορά τον λεπτομερή σχεδιασμό των αναγκών ενός ετερογενούς συστήματος, και την εύρεση ενός κοινού σημείου μεταξύ των διαφορετικών υλοποιήσεων του πρωτοκόλλου 802.15.4, το οποίο να υποστηρίζεται από όλες τις συσκευές. Τέλος, αναπτύχθηκε ο κώδικας που εφαρμόζει αυτή την κοινή υλοποίηση του πρωτοκόλλου, καθώς και η λεπτομερής πειραματική αξιολόγηση του, αναλύοντας τις δυνατότητες της κάθε συσκευής και των συνολικών δυνατοτήτων του δικτύου.

Internet of Things

Το Διαδίκτυο των Αντικειμένων (Internet of Things - IoT) αναφέρεται στην διασύνδεση έξυπνων αντικειμένων (Smart Objects) που μας περιβάλλουν με το διαδίκτυο και μεταξύ τους, και την διαχείριση τους μέσω διαδικτυακών υπηρεσιών. Τελικώς, το Internet of Things θα προσφέρει την δυνατότητα σε προγραμματιστές εφαρμογών να αξιοποιήσουν το hardware και software αισθητήρων και ελεγκτών σε όλα τα αντικείμενα που μας περιβάλλουν, και χωρίς να απασχολούνται με τους περιορισμούς της κάθε συσκευής ή της κάθε υλοποίησης. Παρόλο που υπάρχει εκτεταμένη έρευνα προς αυτό τον τομέα, όσον αφορά την δημιουργία αφαιρετικών μοντέλων χρήσης και ανάπτυξης αισθητήρων ως έξυπνα αντικείμενα, δεν υπάρχει καμία τυποποιημένη ή εύχρηστη τεχνολογία προς το παρόν. Γιαυτό τον λόγο, η ανάπτυξη εφαρμογών για το Internet of Things είναι ακόμα συνυφασμένη με τα αρνητικά χαρακτηριστικά των ασύρματων δικτύων αισθητήρων (WSNs - Wireless Sensor Networks). Γιαυτό και είναι σημαντική η επίλυση βασικών προβλημάτων στα ασύρματα δίκτυα αισθητήρων και στο internet of things, όπως:

- ασυμβατότητες στο hardware, το software και την δικτύωση
- ελλιπής συνδεσιμότητα
- κλιμάκωση (scaling) των εφαρμογών
- απλοποίηση του κύκλου ανάπτυξης
- απουσία τυποποίησης εφαρμογών
- διαχείριση μεγάλου αριθμού δεδομένων (big data)

Μία από τις κύριες δυσκολίες που δεν έχει επιλυθεί σε αποδεκτό βαθμό είναι η παροχή αμφίδρομης επικοινωνίας μεταξύ εφαρμογών και αντικειμένων με εύκολο και αφαιρετικό τρόπο. Στις εφαρμογές του Internet of Things, προβλήματα όπως ο έλεγχος των συσκευών και της κατάστασης τους απαιτούν την αλληλεπίδραση πολλών συσκευών με δεδομένα και υπηρεσίες που βρίσκονται στο διαδίκτυο. Η ικανότητα ανάληψης δράσεων ως ανταπόκριση σε συγκεκριμένα γεγονότα ή καταστάσεις είναι σημαντική για την επίτευξη στόχων όπως η ελαχιστοποίηση της κατανάλωσης ενέργειας και η προσαρμογή του περιβάλλοντος στις προτιμήσεις του χρήστη.

Ενιαίο επίπεδο επικοινωνίας συσκευών και διαδικτύου - CoAP
Πέρα από την επικοινωνία μεταξύ των συσκευών χρησιμοποιώντας ένα ενιαίο πρωτόκολλο, είναι απαραίτητη και η επικοινωνία μεταξύ συσκευών και διαδικτυακών εφαρμογών χρησιμοποιώντας ένα κοινό πρωτόκολλο επικοινωνίας. Για αυτό το λόγο χρησιμοποιήθηκε το CoAP[8] (Constrained Application Protocol), ένα πρωτόκολλο σχεδιασμένο για να φέρει τους κόμβους ενός ασύρματου δικτύου αισθητήρων πιο κοντά στο όραμα του Internet of Things. Καλύπτει όλες τις ανάγκες και τους περιορισμούς των συσκευών χαμηλής κατανάλωσης και λειτουργικότητας που χρησιμοποιούνται στα WSN, και αυτή τη στιγμή βρίσκεται κάτω από τυποποίηση. Προσφέρει μια λύση η οποία είναι αρκετά ανοιχτή για να υποστηριχθεί από ένα μεγάλο εύρος συσκευών, υπηρεσιών και εφαρμογών, και είναι ιδανικό για τις ανάγκες μας.

Συγκεκριμένα, χρησιμοποιούμε τις δυνατότητες που μας δίνει το CoAP για να συνδέσουμε συσκευές με διαφορετικές δυνατότητες (επεξεργαστική ισχύς, μνήμη, αισθητήρια, ελεγκτές) με υπάρχοντα συστήματα. Αυτό είναι απαραίτητο αφού δεν έχουν όλες οι συσκευές τη δυνατότητα να αντεπεξέλθουν σε όλα τα περιβάλλοντα, ούτε και έχουν τις ίδιες ικανότητες όπως μνήμη, διάρκεια μπαταρίας και εύρος επικοινωνίας.

Ως αποτέλεσμα, σε ένα ρεαλιστικό σενάριο, συνήθως έχουμε ένα ετερογενές ασύρματο δίκτυο αισθητήρων με θέματα συνδεσιμότητα και ιδιαιτέρων χαρακτηριστικών της κάθε πλατφόρμας τα οποία πρέπει να επιλυθούν από ένα αφαιρετικό επίπεδο, το οποίο ανταλλάσσει αιτήσεις και απαντήσεις συμβατές με το CoAP. Έτσι, μπορεί να χρησιμοποιηθεί όλο το εύρος των διαφορετικών πλατφορμών hardware (iSense, Arduino, XBee), με την αντίστοιχη υλοποίηση του CoAP στην κάθε μία, ως το κυρίως πρωτόκολλο επικοινωνίας του συνολικού συστήματος. Πράγμα που καθιστά εύκολη την επικοινωνία και με υπόλοιπα συστήματα στα οποία μπορεί να αναπτυχθεί η αντίστοιχη υλοποίηση εύκολα και με μικρό κόστος.

Αυτόματη Αναγνώριση και Προσαρμοστικότητα Συσκευών
Ένα πολύ σημαντικό κομμάτι του ασύρματου δικτύου αισθητήρων μας, που το ξεχωρίζει σημαντικά από όλα τα υπάρχοντα, είναι ο ενσωματωμένος μηχανισμός αυτόματης προσαρμογής των ασύρματων συσκευών με αναγνώριση των δυνατοτήτων του κάθε κόμβου ατομικά, αναγνώριση των αισθητήρων που είναι συνδεδεμένες στον καθένα ανά δεδομένη στιγμή, και αντίστοιχη προσαρμογή τους.

Η κάθε συσκευή ατομικά έχει τη δυνατότητα αναγνώρισης των αισθητήρων, ελεγκτών, και ανεξάρτητων συστημάτων που είναι συνδεδεμένες σε αυτή, την προσαρμογή σε αυτές της συσκευές, και την καταχώριση τους στο δίκτυο ως

διαθέσιμους πόρους του εκάστοτε κόμβου. Αντίστοιχα, οι επεκτάσεις αισθητήρων και ελεγκτών που συνδέονται στους κόμβους αυτούς παρέχουν σημασιολογικές πληροφορίες που βρίσκονται αποθηκευμένες στο σχεδιασμό του hardware τους, και τις ταυτοποιούν στον εκάστοτε κόμβο όπου συνδέονται, χωρίς καμία επέμβαση ανθρώπινου παράγοντα, πέρα από την αρχική εγκατάσταση της συσκευής.

Home Automation

Τα τελευταία χρόνια η εκτεταμένη έρευνα στον τομέα των ασύρματων δικτύων αισθητήρων (WSN) είχε ως αποτέλεσμα την ανάπτυξη και χρήση μεγάλων εγκαταστάσεων, κυρίως για πειραματικούς και ακαδημαϊκούς σκοπούς. Η χρήση αυτών των εγκαταστάσεων έκανε εμφανή τα πλεονεκτήματα της χρήσης των WSN σε διαχείριση κτηρίων και σενάρια αυτοματισμού, κυρίως λόγω των χαμηλών προϋποθέσεων και αναγκών για την εγκατάσταση του δικτύου. Είναι πλέον ευρέως αποδεκτό ότι τα WSN μπορούν να παρέχουν συστήματα διαχείρισης κτιρίων (BMS - Building Management System) για καταγραφή και έλεγχο υπηρεσιών. Ο λόγος είναι επειδή τα WSN έχουν από τη φύση του πολλά πλεονεκτήματα έναντι των υπαρχόντων ενσύρματων λύσεων BMS.

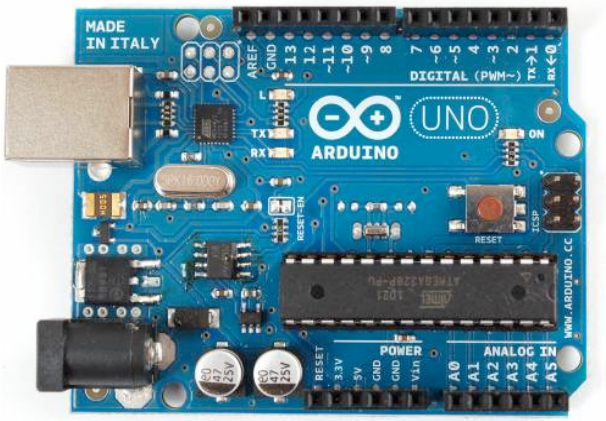
Οι μικρές ασύρματες συσκευές μπορούν να συνδεθούν στις υπάρχοντες λύσεις χωρίς την ανάγκη ειδικής καλωδίωσης πέρα από την τροφοδοσία ρεύματος. Η επικοινωνία γίνεται ασύρματα και δεν απαιτεί έξτρα κόστος ή αλλαγές στο εσωτερικό των κτιρίων. Στα επαγγελματικά κτίρια, οι συσκευές που ελέγχουν τους αισθητήρες και τους ελεγκτές μπορούν να τοποθετηθούν στην ψευδοροφή, όπου η τροφοδοσία ρεύματος υπάρχει ήδη για τον εξαερισμό και τον φωτισμό.

Ένα ακόμα προτέρημα των WSN έχει να κάνει με την ευκολία επέκτασης του συστήματος για την παροχή επιπλέον υπηρεσιών. Μετά την αρχική εγκατάσταση τους, τα υπάρχοντα συστήματα χρειάζονται ρύθμιση από ειδικούς και οποιαδήποτε επέκταση του συστήματος απαιτεί επιπλέον ρύθμιση, κάτι που έχει ως αποτέλεσμα μεγάλο κόστος διαχείρισης τους. Σε αντίθεση, τα ασύρματα δίκτυα αισθητήρων αποτελούνται από έξυπνες συσκευές, ικανές να εκτελούν τοπικό, κατανεμημένο κώδικα. Αυτή η επεξεργαστική ισχύς επιτρέπει την ανάπτυξη πρωτοκόλλων τα οποία, σε συνδυασμό την δυνατότητα της απευθείας προσθήκης ή αφαίρεσης συσκευών, επιτρέπει την εγκατάσταση νέων συσκευών και την αυτόματη ρύθμιση τους και ενεργοποίηση τους στο BMS με ελάχιστο κόστος.

Τα υπάρχοντα συστήματα BMS δεν υποστηρίζουν πλήρως τα ευρέως διαδεδομένα ανοιχτά πρότυπα, κάνοντας την διασύνδεση με άλλα συστήματα στην ίδια περιοχή δύσκολη αν όχι ανέφικτη. Αντίθετα, τα WSN έχουν ένα κοινό φυσικό επίπεδο, καθώς και ένα κοινό επίπεδο δεδομένων και επικοινωνίας (ακολουθώντας το πρωτόκολλο επικοινωνίας 802.15.4), αλλά πρότυπα επικοινωνίας σε επίπεδο δικτύου (ZigBee[9], 6lowpan[10]). Αυτό επιτρέπει σε συσκευές με διαφορετικά χαρακτηριστικά στο hardware ή/και στο software να επικοινωνούν με τις κοντινές τους συσκευές με ελάχιστες ρυθμίσεις.

Συσκευές
Ακολουθεί μια αναλυτική περιγραφή της κάθε πλατφόρμας που χρησιμοποιήθηκε, καθώς και των δυνατοτήτων και των περιορισμών τους.

Arduino
Το Arduino είναι μια open-source πλατφόρμα σχεδίασης πρωτότυπων ηλεκτρονικών κυκλωμάτων και συσκευών. Η πιο διαδεδομένη έκδοση του, το Arduino Uno, είναι ένας μικροϋπολογιστής με chipset ATMega3228 της εταιρίας ATMEL. Έχει 14 ψηφιακές εισόδους/εξόδους, 6 αναλογικές εισόδους, έναν ταλαντωτή 16MHz, μια θύρα USB, μία θύρα τροφοδοσίας, μια θύρα ICSP (In-Circuit Serial Programing), και ένα κουμπί επανεκκίνησης. Οι βασικές βιβλιοθήκες του Arduino είναι γραμμένες σε C και C++ και μεταγλωττίζονται χρησιμοποιώντας τον μεταγλωττιστή avr-gcc και την βιβλιοθήκη AVR Libc. Το hardware του Arduino προγραμματίζεται χρησιμοποιώντας μια γλώσσα βασισμένη στην γλώσσα Wiring[11], η οποία είναι αντίστοιχη της C++, με κάποιες απλοποιήσεις και τροποποιήσεις, και ένα περιβάλλον προγραμματισμού (IDE - Integrated Development Environment) βασισμένο στο προγραμματιστικό περιβάλλον Processing[12].



Arduino Uno

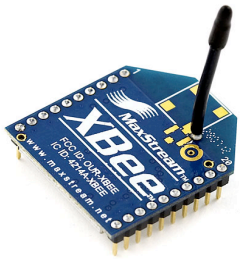
Το Arduino έχει δυο ιδιαίτερα χαρακτηριστικά. Καταρχάς, η γλώσσα προγραμματισμού του είναι εξαιρετικά εύχρηστη και διευκολύνει τον προγραμματισμό, τον πειραματισμό, και την γρήγορη δοκιμή διαφορετικών προσεγγίσεων και εναλλακτικών, και την γρήγορη ανάπτυξη εύκολα και αποδοτικά. Το Arduino είναι επίσης σημαντικά επεκτάσιμο λόγω των προγραμματιζόμενων γραμμών εισόδων/εξόδων του, και την ύπαρξη ενός μεγάλου αριθμού επεκτάσεων (ονόματι Arduino Shields) που είναι συμβατές με αυτό.

Παρόλα αυτά, το Arduino δεν έχει ενσωματωμένη ασύρματη επικοινωνία. Ανταυτού, μπορεί να προστεθεί ασύρματη επικοινωνία με την σύνδεση μιας συσκευής ασύρματης επικοινωνίας, συγκεκριμένα συσκευές της πλατφόρμα XBee, ρυθμισμένες ώστε να δέχονται εντολές από το Arduino. Πιο συγκεκριμένα, το Arduino συνδέεται με το XBee Series 1 Chip Antenna, το οποίο του παρέχει την ασύρματη επικοινωνία ακολουθώντας το πρωτόκολλο 802.15.4. Για να μεγιστοποιήσουμε τη ασύρματη συνδεσιμότητα του Arduino, αναπτύξαμε και την δυνατότητα ασύρματου προγραμματισμού του χρησιμοποιώντας το XBee, πράγμα που μας επιτρέπει τον γρήγορο πειραματισμό στο σύστημα μας.

Xbee
Το XBee είναι μια ασύρματη συσκευή επικοινωνίας, που υλοποιεί το πρωτόκολλο ασύρματης επικοινωνίας 802.15.4, σχεδιασμένη για να συνδέεται με υπάρχοντα συστήματα και πλατφόρμες, επεκτείνοντας τις δυνατότητες τους με την παροχή ασύρματης δικτύωσης.

Το XBee συνδέεται με το Arduino μέσω μιας σειριακής θύρας, και προσθέτει ασύρματες δυνατότητες με το να προωθεί οποιαδήποτε δεδομένα λαμβάνει από το Arduino στο ασύρματο δίκτυο, και οποιαδήποτε δεδομένα λαμβάνει από το ασύρματο δίκτυο στο Arduino αντίστοιχα. Σύμφωνα με τις εργοστασιακές ρυθμίσεις, τα δεδομένα που λαμβάνονται από το Arduino μεταδίδονται σε όλες τις συσκευές του δικτύου, πράγμα που δημιουργεί προβλήματα όταν χρησιμοποιείται σε ένα ασύρματο δίκτυο αισθητήρων.

Επομένως, χρησιμοποιούμε το XBee ρυθμίζοντας το σε λειτουργία API, δηλαδή χρησιμοποιώντας συγκεκριμένες εντολές ελέγχου, οι οποίες μας επιτρέπουν να χρησιμοποιήσουμε το XBee ώστε να επικοινωνήσουμε με συγκεκριμένες συσκευές του δικτύου, αλλά και να αναγνωρίσουμε ποια συσκευή μεταδίδει ανά πάσα στιγμή.



XBee Series 1

iSense
Το iSense Sensor1 της Coalesenses είναι μια επεκτάσιμη πλατφόρμα hardware για χρήση σε διάφορες εφαρμογές ασύρματων δικτύων αισθητήρων. Οι συσκευές iSense (Core Module 2) είναι βασισμένες στον μικροελεγκτή Jennic 32Bit RISC, με 128kB RAM, 512kB Flash και ασύρματη επικοινωνία συμβατή με το πρωτόκολλο IEEE 802.15.4, με ενσωματωμένη και εξωτερική κεραία, ανάλογα με τις ανάγκες της κάθε εφαρμογής, όλα ενσωματωμένα σε ένα chip, αποκαλούμενο Core Module.

Οι επεκτάσεις (i.e., Environmental Module για θερμοκρασία και φωτεινότητα) μπορούν να συνδεθούν πάνω στο Core Module, και εξυπηρετούν ένα μεγάλο εύρος αναγκών που μπορεί να προκύψει σε εγκαταστάσεις συστημάτων ελέγχου κτιρίων (BMS - Building Management Systems). Τα iSense προγραμματίζονται σε C++ μέσω του μεταγλωττιστή ba-elf και ενός ειδικευμένου κιτ ανάπτυξης λογισμικού (SDK - Software Development Kit) το οποίο καλύπτει όλες τις λειτουργίες του Core Module αλλά και των υπόλοιπων Modules αισθητήρων. Τέλος, υποστηρίζεται ασύρματος (Over-The-Air) προγραμματισμός, δίνοντας την δυνατότητα εγκατάστασης νέων “εφαρμογών” στο ασύρματο δίκτυο αισθητήρων, με τον προγραμματισμό των υπάρχοντων κόμβων χωρίς την ανάγκη φυσικής παρουσίας σε κάθε μια από αυτές.

iSense Core Module

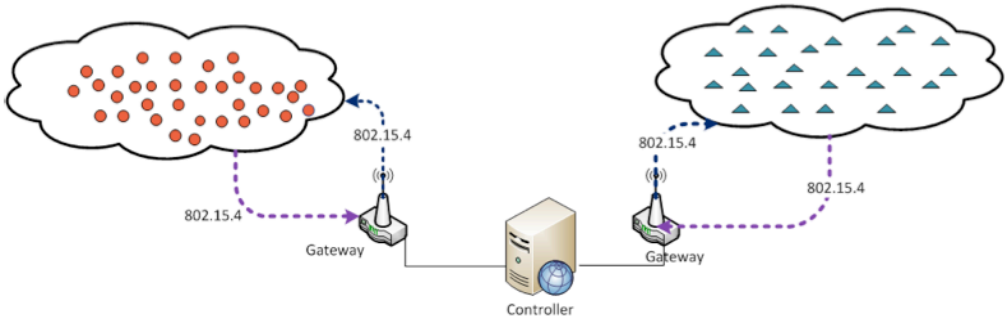
SunSPOT
Το Sun SPOT (Sun Small Programmable Object Technology) είναι μια πλατφόρμα ασύρματων δικτύων αισθητήρων (WSN) που αναπτύχθηκε από την Sun Microsystems. Σε αντίθεση με άλλα διαθέσιμα συστήματα, το Sun SPOT είναι χτισμένο πάνω στην Java Virtual Machine ονόματι Squawk . Χρησιμοποιεί έναν πυρήνα 32 bit ARM920T στα 180 MHz με 512K RAM, 4M μνήμης Flash καθώς και ασύρματη δικτύωση 2.4 GHz IEEE 802.15.4 με ενσωματωμένη κεραία και θύρα USB.

SunSPOT

Σχεδιασμός και Υλοποίηση βιβλιοθήκης για Επικοινωνία σε Ετερογενή Ασύρματα Δίκτυα Αισθητήρων

Ομογενή δίκτυα 802.15.4

Υπάρχουν δυο διακριτές κατηγορίες έξυπνων αντικειμένων. Τα παθητικά, τα οποία έχουν την δυνατότητα να αισθάνονται και καταγράφουν το περιβάλλον, και τα ενεργητικά, τα οποία λαμβάνουν ατομικές αποφάσεις και δρουν αναλόγως. Αυτές οι δύο συμπεριφορές υλοποιούνται αντίστοιχα από διαφορετικές πλατφόρμες, η κάθε μια χρησιμοποιώντας την δική της τεχνολογία. Για να χρησιμοποιηθούν ταυτόχρονα για τη δημιουργία ενός διαδραστικού συστήματος, είναι απαραίτητη η χρήση μιας αρχιτεκτονικής όπως η παρακάτω:

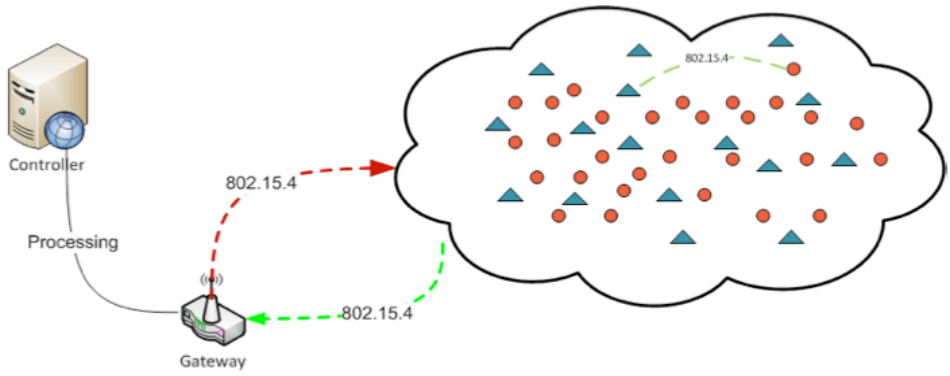


Αρχιτεκτονική Συστήματος ενός Ετερογενούς Δικτύου

Όπως είναι εμφανές, η αρχιτεκτονική αποτελείται από δυο ή περισσότερα κυρίως στοιχεία, τα οποία είναι τα δίκτυα που αποτελούνται από συσκευές μιας πλατφόρμας, είτε ενεργητικές είτε παθητικές, οι οποίες είναι απομονωμένες αναμεταξύ τους. Για να είναι εφικτή η επικοινωνία ανάμεσα στα δίκτυα, υπάρχει και ένας κεντρικός κόμβος (Gateway) για κάθε δίκτυο που συνδέεται με τα υπόλοιπα δίκτυα μέσω ενός ελεγκτή (Controller), ο οποίος συλλέγει και αναλύει δεδομένα, και συντονίζει τα επιμέρους δίκτυα με έναν ενιαίο τρόπο.

Παρόλα αυτά, υπάρχουν σημαντικοί περιορισμοί σε αυτή την αρχιτεκτονική. Καταρχάς, για να μπορέσει ο ελεγκτής να έχει πρόσβαση σε κάθε δίκτυο, χρειάζεται η παρουσία του αντίστοιχου ελεγκτή, λόγω της ασυμβατότητας στο hardware. Επιπλέον, χρειάζονται το αντίστοιχο λογισμικό για την συλλογή και επεξεργασία των δεδομένων για κάθε δίκτυο, την λήψη αποφάσεων, και την ανάληψη δράσης. Το πιο σημαντικό πρόβλημα όμως είναι η δυσκολία προγραμματισμού και διατήρησης ενός τέτοιου πολύπλοκου συστήματος, κάτι που το κάνει δύσκολο ως προς την σχεδίαση, εγκατάσταση, και επέκταση ενός τέτοιου συστήματος. Ενώ η αποσφαλμάτωση καθίσταται πρακτικά αδύνατη.

Λόγω των περιορισμών της παραπάνω αρχιτεκτονικής, προτείνεται μια νέα αρχιτεκτονική, ενός ετερογενούς δικτύου, το οποίο αποτελείται από παθητικά αλλά και ενεργητικά στοιχεία, ανεξαρτήτως τεχνολογίας. Υπάρχει επίσης η ανάγκη για μόνο ένα κεντρικό κόμβο (Gateway) για την επικοινωνία με τις συσκευές. Πρακτικά, παύει να ισχύει η ανάγκη του Ελεγκτή για την επικοινωνία μεταξύ συσκευών και τον συντονισμό τους, με την χρήση της στοίβας δικτύωσης (network stack) σε μορφή βιβλιοθήκης ονόματι mkSense, το οποίο αναλύεται παρακάτω.



Αρχιτεκτονική Συστήματος ενός Ετερογενούς Δικτύου

Ετερογενή δίκτυα 802.15.4 - mkSense στην πράξη

Καθεμία από τις πλατφόρμες που υποστηρίζονται από το mkSense έχει τη δυνατότητα ασύρματης επικοινωνίας με το πρωτόκολλο IEEE std. 802.15.4 2003. Παρόλα αυτά, κάθε πλατφόρμα υλοποιεί μόνο ένα υποσύνολο του IEEE 802.15.4, το οποίο είναι ασύμβατο με τις υπόλοιπες πλατφόρμες. Επομένως είναι αδύνατη η επικοινωνία μεταξύ των τεσσάρων διαφορετικών συσκευών.

Το IEEE 802.15.4 παρέχει δύο τρόπους διευθυνσιοδότησης, την διευθυνσιοδότηση με 16-bit και με 64-bit. Η πλατφόρμα SunSPOT radio stack υποστηρίζει μόνο την 64-bit διευθυνσιοδότηση ενώ το TelosB μόνο την 16-bit. Οι υλοποιήσεις των XBee και iSense παρέχουν και τους δύο τρόπους.

Το πρώτο βήμα για ένα ετερογενές ασύρματο δίκτυο αισθητήρων είναι η επιλογή ενός κοινού τρόπου διευθυνσιοδότησης (π.χ. η υλοποίηση του mkSense για SunSPOT υλοποιεί την 16-bit διευθυνσιοδότηση σε επίπεδο λογισμικού). Για να επιτευχθεί αυτό, η υλοποίηση του mkSense για Arduino & XBee που αναπτύχθηκε ως μέρος αυτής της εργασίας ρυθμίζει το XBee σε καθαρή επικοινωνία με βάση το πρωτόκολλο 802.15.4, με αυτόματη αποστολή handshaking μηνυμάτων (ACKs).

Επιπλέον, η στοίβα δικτύου mkSense δεν υπόστηρίζει δρομολόγηση πλέγματος (mesh routing) ούτε κατακερματισμό μηνυμάτων, πράγμα που υποστηρίζεται από τις συσκευές SunSPOT. Γιαυτό το λόγο, η υλοποίηση της στοίβας σε όλες τις υπόλοιπες συσκευές προσθέτει δυο σταθερά byte στην αρχή του πακέτου δεδομένων για να επιτευχθεί η συμβατότητα με SunSPOT. Αυτά τα σταθερά byte ενημερώνουν τις συσκευές SunSPOT ότι το δίκτυο δεν χρησιμοποιεί δρομολόγηση, και απενεργοποιούν την στοίβα δρομολόγησης/δικτύου LowPAN που χρησιμοποιούν από προεπιλογή τα SunSPOT. Έτσι επιτυγχάνεται η επικοινωνία μεταξύ και των τεσσάρων συσκευών.

Platform	Max Payload Size	Addressing Mode		Incompatibilities
		16-bit	64-bit	
Arduino XBee	100 bytes	YES	YES	Extra Headers (MaxStream Headers)
SunSPOT	113 bytes	NO	YES	Extra Headers (LowPan)
TelosB	128 bytes	YES	NO	Auto Ack is Disabled
iSense	116 bytes	YES	YES	

Συνοπτικη περιγραφή της κάθε πλατφόρμας και των διαφορών τους

Αναλυτικότερα, το mkSense για SunSPOT έχει υλοποιηθεί σε Java J2ME. Η αντίστοιχη υλοποίηση για iSense είναι γραμμένη σε C++, ενώ το TelosB τρέχουν το λειτουργικό TinyOS[13] v2.1.0, και επομένως η συγκεκριμένη υλοποίηση έχει γραφτεί σε nesC. Αντιστοίχως, η έκδοση της mkSense για Arduino & XBee στα πλαίσια της διπλωματικής γράφτηκε σε C++, χρησιμοποιώντας τις βιβλιοθήκες προγραμματισμού του Arduino SDK, και την βιβλιοθήκη προγραμματισμού Arduino XBee API (arduino-xbee).

Ασύρματος Προγραμματισμός
Μία ακόμα από τις σημαντικές δυνατότητες του XBee είναι ο ασύρματος απομακρισμένος έλεγχος των εισόδων/εξόδων του. Με τις κατάλληλες ρυθμίσεις, τα XBee μπορούν να αποστείλουν την κατάσταση των ψηφιακών εισόδων/εξόδων τους στο ασύρματο δίκτυο, και αντιστοίχως να λαμβάνουν τις καταστάσεις των εισόδων/εξόδων από τους υπόλοιπους κόμβους του ασύρματου δικτύου και να θέτουν τις δικές τους εξόδους στην ίδια κατάσταση.

Χάρις σε αυτή τη λειτουργεία, είναι δυνατή η ρύθμιση του απομακρυσμένου ελέγχου των εισόδων/εξόδων με τέτοιο τρόπο ώστε να δώσουμε την δυνατότητα ασύρματου προγραμματισμού ενός Arduino, χρησιμοποιόντας το XBee που είναι συνδεδεμένο σε αυτό, και ένα XBee συνδεδεμένο σε έναν υπολογιστή. Ξεκινάμε με την απαραίτητη συνδεσμολογία για τον απομακρυσμένο έλεγχο του reset pin του Arduino που είναι χρειάζεται για τον προγραμματισμό του. Η υπόλοιπη διαδικασία προγραμματισμού παραμένει ως έχει, επειδή εκμεταλευόμαστε την σειριακή επικοινωνία που έχουμε ήδη συνδέσει μεταξύ του Arduino και του XBee, και προωθώντας τα δεδομένα του προγράμματος μέσω του ασύρματου δικτύου.

Ένα από τα προβλήματα που προκύπτουν με αυτή την προσέγγιση είναι ότι τα Xbee χρησιμοποιούν έξτρα bytes για την προώθηση της κατάστασης των εισόδων/εξόδων τους στο ασύρματο δίκτυο, πράγμα που τα κάνει ασύμβατα με το καθαρό πρωτόκολλο 802.15.4, και με τις υπόλοιπες συσκευές του δικτύου μας. Η υλοποίηση του mkSense επιλύει αυτό το πρόβλημα με το να προσθέτει διασυνδεσιμότητα με τις υπόλοιπες συσκευές, αλλά υλοποιημένη με τέτοιο τρόπο ώστε να μπορούμε να χρησιμοποιήσουμε την δυνατότητα προώθησης της κατάστασης των εισόδων/εξόδων του XBee μέσω δικτύου χωρίς την ανάγκη της χρήσης των έξτρα byte που χρειάζονται για αυτήν συνεχώς. Αυτό επιτυγχάνεται με την ενεργοποίηση αυτών των έξτρα byte κατά παραγγελία, μόνο όταν χρειάζεται να προγραμματίσουμε το Arduino.

Υλοποίηση βιβλιοθήκης mkSense για Arduino

XBeeRadio.h

```
/*
  XBeeRadio.h Library for communicating with heterogenous 802.15.4 networks.
  Created by Vasileios Georgitzikis, November 23, 2010.
*/

#ifndef XbeeRadio_h
#define XbeeRadio_h

#include <Xbee.h>
#include "Arduino.h"

#ifdef DEBUG
  #define DBG(X) X
#else
  #define DBG(X)
#endif
class XBeeRadioResponse : public XBeeResponse
{
public:
      // const static uint8_t LP1  = 0x7f;
      // const static uint8_t LP2  = 0x69;
      // const static uint8_t DEFAULT_PORT = 110;
      XBeeRadioResponse() : XBeeResponse(){}
      uint8_t getDataLength();
      uint8_t getData(int index);
      uint8_t* getData();
      uint8_t validPacket();
      uint8_t validPacket(uint8_t valid_port);
      uint8_t validPacket(uint8_t lp1,uint8_t lp2,uint8_t port);
      uint8_t getRssi();
};

class XBeeRadio: public XBee
{
public:
      const static uint8_t LP1  = 0x7f;
```

```
        const static uint8_t LP2  = 0x69;
        const static uint8_t DEFAULT_PORT = 110;
        const static uint8_t PROGRAMMING_PORT = 42;
        const static uint8_t LED_PIN = 13;
        XBeeRadio() : XBee(){}
        void getResponse(XBeeRadioResponse &response);
        XBeeRadioResponse& getResponse();
        uint16_t myAddress;
        uint16_t getMyAddress();
        bool setChannel(uint8_t channel);
        void send(Tx16Request &request);
        void send(Tx16Request &request, uint8_t port);
        void send(AtCommandRequest);
        bool sendAndCheck(Tx16Request &request);
        bool sendAndCheck(Tx16Request &request, uint8_t port);
        void sendAtCommand(uint8_t command[], uint8_t reply[]);
        void sendAtCommand(uint8_t command[], uint8_t value[], uint8_t length);
        uint8_t init(/*NewSoftSerial mySerial*/void);
        uint8_t init(uint8_t channel);
        bool checkForData(void);
        bool checkForData(uint8_t valid_port);
        bool checkForData(uint8_t lp1,uint8_t lp2,uint8_t port);

        void initialize_xbee_module();
        void initialize_xbee_module(long baudrate);
private:
        uint8_t trySendingCommand(uint8_t buffer[2], AtCommandRequest
atRequest,AtCommandResponse atResponse);
        void getReadyForProgramming(uint16_t programmer_address);

        bool setup_command(char* command);
        long setup_baudrate(void);
        long setup_baudrate(long starting_baud);
        bool setup_xbee(long baudrate);
        bool wait_for_response(unsigned long milliseconds);
        bool check_for_response(void);
};

#endif
```

XBeeRadio.cpp

```
/*
XBeeRadio.h Library for communicating with heterogenous 802.15.4 networks.
      Created by Vasileios Georgitzikis, November 23, 2010.
*/

#include "XbeeRadio.h"

//XBeeRadioResponse::XBeeRadioResponse() : XBeeResponse(){}
uint8_t XBeeRadioResponse::validPacket(uint8_t lp1,uint8_t lp2,uint8_t port)
{
      Rx16Response response = Rx16Response();
      uint8_t ApiId = XBeeResponse::getApiId();
      if(ApiId == RX_16_RESPONSE)
      {
            getRx16Response(response);
            //uint8_t myData = response.getData(0);
            if(response.getData(0) == lp1 && response.getData(1) == lp2)
                  return response.getData(2);
            return false;
      }
}

uint8_t XBeeRadioResponse::getDataLength()
{
      Rx16Response response = Rx16Response();
      getRx16Response(response);
      return response.getDataLength()-3;
}
uint8_t XBeeRadioResponse::getData(int index)
{
      Rx16Response response = Rx16Response();
      getRx16Response(response);
      return response.getData(index+3);
}
uint8_t* XBeeRadioResponse::getData()
{
      Rx16Response response = Rx16Response();
```

```
	getRx16Response(response);
	return response.getData()+3;
}
// uint8_t XBeeRadioResponse::validPacket()
// {
//    return validPacket(LP1, LP2, DEFAULT_PORT);
// }
// uint8_t XBeeRadioResponse::validPacket(uint8_t valid_port)
// {
//    return validPacket(LP1, LP2, valid_port);
// }

bool XBeeRadio::checkForData(void)
{
	return checkForData(LP1, LP2, DEFAULT_PORT);
}
bool XBeeRadio::checkForData(uint8_t valid_port)
{
	checkForData(LP1, LP2, valid_port);
}

bool XBeeRadio::checkForData(uint8_t lp1,uint8_t lp2,uint8_t port)
{
	this->readPacket();
	if(this->getResponse().isAvailable())
	{
		uint8_t port_validity = this->getResponse().validPacket(lp1, lp2,
port);
		if(port_validity == port)
			return true;
		else if(port_validity == PROGRAMMING_PORT)
		{
			pinMode(LED_PIN, OUTPUT);
			digitalWrite(LED_PIN, HIGH);
			delay(200);
			digitalWrite(LED_PIN, LOW);
			delay(200);
			digitalWrite(LED_PIN, HIGH);
			delay(200);
			digitalWrite(LED_PIN, LOW);
```

```
                delay(200);
                digitalWrite(LED_PIN, HIGH);
                delay(200);
                digitalWrite(LED_PIN, LOW);
                delay(200);

                Rx16Response rx = Rx16Response();
                this->getResponse().getRx16Response(rx);
            uint16_t sender = rx.getRemoteAddress16();
                getReadyForProgramming(sender);
            }
      }
      return false;
}

uint8_t XBeeRadioResponse::getRssi()
{
      return getFrameData()[2];
}
//XBeeRadio::XBeeRadio() : XBee(){}
void  XBeeRadio::getResponse(XBeeRadioResponse &response)
{
      XBee::getResponse(response);
}
XBeeRadioResponse& XBeeRadio::getResponse()
{
      return (XBeeRadioResponse&) XBee::getResponse();
}
void XBeeRadio::send(Tx16Request &request)
{
      send(request, DEFAULT_PORT);
}

void XBeeRadio::send(Tx16Request &request, uint8_t port)
{
      uint8_t *temp_payload = request.getPayload();
      uint8_t payloadLength = request.getPayloadLength();
      uint8_t *new_payload = (uint8_t*) malloc(sizeof(uint8_t) * (payloadLength
+3));
```

```
	for(int i=0;i<payloadLength;i++)
	{
		new_payload[i+3] = temp_payload[i];
	}
	new_payload[0] = LP1;
	new_payload[1] = LP2;
	new_payload[2] = port;

	request.setPayload(new_payload);
	request.setPayloadLength(payloadLength+3);

	XBee::send(request);

	delay(10);

	request.setPayload(temp_payload);
	request.setPayloadLength(payloadLength);

	free(new_payload);
}

void XBeeRadio::send(AtCommandRequest request)
{
	XBee::send(request);
}

bool XBeeRadio::sendAndCheck(Tx16Request &request)
{
	return sendAndCheck(request, DEFAULT_PORT);
}

bool XBeeRadio::sendAndCheck(Tx16Request &request, uint8_t port)
{
	bool retVal = false;
	send(request, port);
	TxStatusResponse txStatus = TxStatusResponse();

	 // after sending a tx request, we expect a status response
    // wait up to 5 seconds for the status response
    if (readPacket(5000))
```

```
    {
        // got a response! should be a znet tx status
      if (getResponse().getApiId() == TX_STATUS_RESPONSE)
        {
         getResponse().getZBTxStatusResponse(txStatus);

         // get the delivery status, the fifth byte
           if (txStatus.getStatus() == SUCCESS)
           {
                  // success.  time to celebrate
                        retVal = true;
           }
           else
           {
                  // the remote XBee did not receive our packet. is it powered
on?
           }
        }
    }
    else
    {
      // local XBee did not provide a timely TX Status Response -should not
happen
    }
}

uint8_t XBeeRadio::init(/*NewSoftSerial mySerial*/void)
{
      pinMode(LED_PIN, OUTPUT);
      digitalWrite(LED_PIN, HIGH);

      delay(1000);

      //XBeeRadio temp_xbee = XBeeRadio();
// serial low
      uint8_t slCmd[] = {'S','L'};

      uint8_t chCmd[] = {'C','H'};
      uint8_t chValue[] = {0x0C};
```

```
      uint8_t pidCmd[] = {'I','D'};
      uint8_t pidValue[] = {0x01};

      uint8_t apCmd[] = {'A','P'};
      uint8_t apValue[] = {0x02};

      uint8_t mmCmd[] = {'M','M'};
      uint8_t mmValue[] = {0x02};

      uint8_t myCmd[] = {'M','Y'};
      uint8_t myValue[] = {0x00, 0x0A};

      uint8_t acCmd[] = {'A', 'C'};
      uint8_t acValue[] = {};


// get SL
      uint8_t buffer[] = {0x00, 0x0B};
      sendAtCommand(slCmd, buffer);

//set MY
      buffer[0] &= 0x0f;

      myAddress = buffer[1];
      myAddress <<= 8;
      myAddress += buffer[0];

      myValue[0] = buffer[0];
      myValue[1] = buffer[1];

      // mySerial.print("My address should be: ");
      // mySerial.print(myValue[0], HEX);
      // mySerial.println(myValue[1], HEX);

      sendAtCommand(myCmd, myValue, sizeof(myValue));
      // mySerial.println("Set MY");

      sendAtCommand(chCmd, chValue, sizeof(chValue));
      // mySerial.println("Set CH");
```

```
	sendAtCommand(pidCmd, pidValue, sizeof(pidValue));
	// mySerial.println("Set PID");

	sendAtCommand(apCmd, apValue, sizeof(apValue));
	// mySerial.println("Set AP");

	sendAtCommand(mmCmd, mmValue, sizeof(mmValue));
	// mySerial.println("Set MM");

// set AC (Enable the values we sent)
	sendAtCommand(acCmd, buffer);
	// mySerial.println("Set AC");
	digitalWrite(LED_PIN, LOW);
}
uint8_t XBeeRadio::trySendingCommand(uint8_t buffer[2], AtCommandRequest
atRequest,AtCommandResponse atResponse)
{
	uint8_t error=0;

// send the command
	this->send(atRequest);

// wait up to 5 seconds for the status response
	if (this->readPacket(5000))
	{
	// should be an AT command response
		if (this->getResponse().getApiId() == AT_COMMAND_RESPONSE)
		{
			this->getResponse().getAtCommandResponse(atResponse);

			if (atResponse.isOk())
			{
				if (atResponse.getValueLength() > 0)
				{
					buffer[0] = atResponse.getValue()[2];
					buffer[1] = atResponse.getValue()[3];
				}
			}
			else
			{
```

```
                        error=1;
                  }
            }
            else
            {
                  error=1;
            }
      }
      else
      {
            error=1;
      }

      return error;
}
uint16_t XBeeRadio::getMyAddress()
{
      return this->myAddress;
}

uint8_t XBeeRadio::init(uint8_t channel)
{
      uint8_t return_val = init();
      setChannel(channel);
      return return_val;
}

bool XBeeRadio::setChannel(uint8_t channel)
{
      digitalWrite(LED_PIN, HIGH);
      delay(100);
      if(channel<0x0b || channel>0x17)
      {
            digitalWrite(LED_PIN, LOW);
            return false;
      }

      uint8_t chCmd[] = {'C','H'};
      uint8_t chValue[] = {channel};
      AtCommandRequest atRequest = AtCommandRequest(chCmd);
```

```
        atRequest.setCommandValue(chValue);
        atRequest.setCommandValueLength(sizeof(chValue));

        AtCommandResponse atResponse = AtCommandResponse();

        uint8_t buffer[] = {0x00, 0x0B};

        // set CH
        trySendingCommand(buffer, atRequest, atResponse);
        atRequest.clearCommandValue();
        digitalWrite(LED_PIN, LOW);
        return true;
}

void XBeeRadio::getReadyForProgramming(uint16_t programmer_address)
{
        pinMode(LED_PIN, OUTPUT);

        uint8_t reCmd[] = {'R','E'};

        uint8_t slCmd[] = {'S','L'};

        uint8_t myCmd[] = {'M','Y'};
        uint8_t myValue[] = {0x00, 0x0A};

        uint8_t dhCmd[] = {'D','H'};
        uint8_t dhValue[] = {0x00, 0x00};

        uint8_t dlCmd[] = {'D','L'};

        uint8_t chCmd[] = {'C','H'};
        uint8_t chValue[] = {0x0C};

        uint8_t pidCmd[] = {'I','D'};
        uint8_t pidValue[] = {0x12, 0x34};

        uint8_t apCmd[] = {'A','P'};
        uint8_t apValue[] = {0x00};

        uint8_t mmCmd[] = {'M','M'};
```

```
      uint8_t mmValue[] = {0x00};

      uint8_t bdCmd[] = {'B','D'};
      uint8_t bdValue[] = {0x06};

      uint8_t rrCmd[] = {'R','R'};
      uint8_t rrValue[] = {0x03};

      uint8_t roCmd[] = {'R','O'};
      uint8_t roValue[] = {0x10};

      uint8_t d3Cmd[] = {'D','3'};
      uint8_t d3Value[] = {0x05};

      uint8_t iuCmd[] = {'I','U'};
      uint8_t iuValue[] = {0x00};

      uint8_t iaCmd[] = {'I','A'};
      uint8_t iaValue[] = {0xFF, 0xFF};

      uint8_t wrCmd[] = {'W','R'};

      uint8_t acCmd[] = {'A', 'C'};

// get SL
      uint8_t buffer[] = {0x00, 0x0B};
      sendAtCommand(slCmd, buffer);

//set MY
      buffer[0] &= 0x0f;
      myAddress = buffer[1];
      myAddress <<= 8;
      myAddress += buffer[0];

      myValue[0] = buffer[0];
      myValue[1] = buffer[1];

      // mySerial.print("My address should be: ");
      // mySerial.print(myValue[0], HEX);
      // mySerial.println(myValue[1], HEX);
```

```
sendAtCommand(myCmd, myValue, sizeof(myValue));
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

// mySerial.println("Set MY");

sendAtCommand(dhCmd, dhValue, sizeof(dhValue));
// mySerial.println("Set DH");

/***** THIS MUST GO AWAY!!!! ******/
// programmer_address = 0x664f;
uint8_t dlValue[2];
dlValue[0] = programmer_address >> 8;
dlValue[1] = (uint8_t) (programmer_address % 0x100);
sendAtCommand(dlCmd, dlValue, sizeof(dlValue));
// mySerial.println("Set DL");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(chCmd, chValue, sizeof(chValue));
// mySerial.println("Set CH");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(pidCmd, pidValue, sizeof(pidValue));
// mySerial.println("Set PID");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);
```

```
sendAtCommand(mmCmd, mmValue, sizeof(mmValue));
// mySerial.println("Set MM");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(rrCmd, rrValue, sizeof(rrValue));
// mySerial.println("Set RR");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(roCmd, roValue, sizeof(roValue));
// mySerial.println("Set RO");

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(d3Cmd, d3Value, sizeof(d3Value));
// mySerial.println("Set D3");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(iuCmd, iuValue, sizeof(iuValue));
// mySerial.println("Set IU");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(iaCmd, iaValue, sizeof(iaValue));
```

```
// mySerial.println("Set IA");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

// sendAtCommand(wrCmd, buffer);
sendAtCommand(acCmd, buffer);
// mySerial.println("Set wr");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(bdCmd, bdValue, sizeof(bdValue));
// mySerial.println("Set BD");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

Serial.end();
Serial.begin(57600);

digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

// sendAtCommand(wrCmd, buffer);
sendAtCommand(acCmd, buffer);
// mySerial.println("Set wr");
digitalWrite(LED_PIN, HIGH);
delay(200);
digitalWrite(LED_PIN, LOW);
delay(500);

sendAtCommand(apCmd, apValue, sizeof(apValue));
// mySerial.println("Set AP");
digitalWrite(LED_PIN, HIGH);
```

```
	delay(200);
	digitalWrite(LED_PIN, LOW);
	delay(500);


// set AC (Enable the values we sent)
	// sendAtCommand(acCmd, buffer);
	// mySerial.println("Set AC");



	unsigned long timestamp = millis();
	while(millis() timestamp < 30000)
	{
		digitalWrite(LED_PIN, HIGH);
		delay(200);
		digitalWrite(LED_PIN, LOW);
		delay(200);
	}
}

void XBeeRadio::sendAtCommand(uint8_t command[], uint8_t reply[])
{
	AtCommandRequest atRequest = AtCommandRequest(command);
	AtCommandResponse atResponse = AtCommandResponse();
	while(true)
	{
		// mySerial.println("Getting SL");
		if(trySendingCommand(reply, atRequest, atResponse) != 1)
			break;
	}
}
void XBeeRadio::sendAtCommand(uint8_t command[], uint8_t value[], uint8_t
length)
{
	AtCommandRequest atRequest = AtCommandRequest(command);
	AtCommandResponse atResponse = AtCommandResponse();
	atRequest.setCommandValue(value);
	atRequest.setCommandValueLength(length);
```

```
	uint8_t buffer[2];
	while(true)
	{
		// mySerial.println("Setting MY");
		if(trySendingCommand(buffer, atRequest, atResponse) != 1)
			break;
	}
	atRequest.clearCommandValue();
}

void XBeeRadio::initialize_xbee_module()
{
	initialize_xbee_module(38400);
}

void XBeeRadio::initialize_xbee_module(long baudrate)
{
	pinMode(13, OUTPUT);
	unsigned long timestamp = millis();
	while(millis() timestamp < 10000)
	{
		digitalWrite(13, HIGH);
		delay(100);
		digitalWrite(13, LOW);
		delay(100);
	}
	digitalWrite(13, HIGH);

	uint8_t atvr[] = {'V', 'R'};
	AtCommandRequest atRequest = AtCommandRequest(atvr);
	AtCommandResponse atResponse = AtCommandResponse();

	uint8_t buffer[2];
	begin(baudrate);

	// NewSoftSerial mySerial(4, 5);
	// mySerial.begin(9600);
	// mySerial.println("Starting!");
	uint8_t api_mode = trySendingCommand(buffer, atRequest, atResponse);
	Serial.end();
```

```
      if(api_mode != 1)
      {
            // mySerial.println("ALREADY IN API MODE");
            return;
      }
      // mySerial.println("Trying to make everything right");

  long cur_baud = setup_baudrate();
  DBG(mySerial.print("Baudrate: ");)
  DBG(mySerial.println(cur_baud);)
  setup_command("ATAP2");
  setup_command("ATBD5");
  setup_command("ATAC");
  Serial.end();
  digitalWrite(13, LOW);
}


//Send an AT command
bool XBeeRadio::setup_command(char* command)
{
  char bla[10];
  strcpy(bla, command);
  int cmd_length = strlen(bla);
  bla[cmd_length++] = '\r';
  bla[cmd_length] = '\0';
//  = "ATMM2\r";
  bool ret_val;
  for(int i = 0; i < 5; i++)
  {
    for(int k = 0; k < 6; k++)
    {
      Serial.print(bla[k]);
      delay(200);
    }

    bool response = wait_for_response(3000);
    DBG(
    if(response)
```

```
      mySerial.println("Got response");
    else
      mySerial.println("No response");
    )

    ret_val = check_for_response();
    if(ret_val) break;
  }
  return ret_val;
}


long XBeeRadio::setup_baudrate(void)
{
  long ret_val = setup_baudrate(9600);
  return ret_val;
}

//find the correct baudrate to talk to the XBee
long XBeeRadio::setup_baudrate(long starting_baud)
{
  long bla[] = {9600, 19200, 38400, 57600, 115200};
  int k;
  for(int i = 0 ; i < 5; i++)
    if(bla[i] == starting_baud)
      k = i;

  while(true)
  {
    DBG(mySerial.print("Starting Baudrate: ");)
    DBG(mySerial.println(bla[k]);)
    bool ret_val = setup_xbee(bla[k++]);
    //we found our baudrate
    if(ret_val) break;
    if(k == 5)
    {
      //wait a while before starting from 9600 baudrate
      delay(3000);
      k = 0;
    }
```

```
  }
  return bla[k-1];
}
//Ping the XBee
bool XBeeRadio::setup_xbee(long baudrate)
{
    Serial.begin(baudrate);
    //for some reason, the first message doesn't go to
    //the XBee, so we sent some dummy data first.
    Serial.print("X");
    delay(1100);
    //Go into API mode
    for(int k = 0; k < 3; k++)
    {
      Serial.print("+");
      delay(200);
    }

    //Wait for a respsonse (OK) from the XBee.
    bool response = wait_for_response(2000);
    DBG(
    if(response)
      mySerial.println("Got response");
    else
      mySerial.println("No response");
    )
    bool ret_val = check_for_response();
    if(!ret_val)
    {
      Serial.end();
    }
    return ret_val;
}

//Wait for a response by the XBee after sending a command
bool XBeeRadio::wait_for_response(unsigned long milliseconds)
{
  unsigned long timestamp = millis();
  bool return_value = false;
  while(millis() timestamp < milliseconds)
```

```
  {
    if(Serial.available() > 2)
    {
      return_value = true;
      break;
    }
  }
  return return_value;
}

//Check if the response was 'OK'
bool XBeeRadio::check_for_response(void)
{
  #define BUFFER_SIZE 10
  char buffer[BUFFER_SIZE];
  for(int i = 0 ; i< BUFFER_SIZE; i++)
    buffer[i] = 0;
  int j = 0;
  while(Serial.available() && j < (BUFFER_SIZE-1))
  {
      int tmpChar = Serial.read();
      buffer[j++] = (char) tmpChar;
      buffer[j]='\0';
//      mySerial.println(tmpChar, BYTE);
  }
  DBG(mySerial.print(buffer);)
  buffer[2] = '\0';
  bool return_value = false;
  if(!strcmp(buffer,"OK"))
    return_value = true;

  DBG(
  if(return_value)
    mySerial.println("YAY");
  else
    mySerial.println("NAY");
  )
  return return_value;
}
```

Διασύνδεση Arduino με Σύστημα Ελέγχου Uberdust

Εκτός από την βιβλιοθήκη mksense, στα πλαίσια της διπλωματικής υλοποιήθηκε επίσης μια βιβλιοθήκη για Arduino η οποία διευκολύνει την επικοινωνία μεταξύ Arduino και των υπόλοιπων συσκευών και υπολογιστών του δικτύου σε ένα υψηλότερο επίπεδο αφαίρεσης, κάνοντας κάποιες βασικές παραδοχές όσον αφορά τον τύπο των αισθητήρων. Παρακάτω παρατίθεται ο κώδικας της.

Uberdust.h

```
#ifndef Uberdust_h
#define Uberdust_h
#include <Arduino.h>
#include <XbeeRadio.h>
#include <String.h>
#include "RelaySensor.h"

#define PAYLOAD_SIZE 30
class Uberdust
{
public:
	//the period we use when broadcasting our status. currently 1 min.
	const static unsigned broadcastPeriod = 60000;
	//the default pins for our switch, and the relay
	const static long DEFAULT_BAUDRATE = 38400;
	//setting the default LED pin
	const static int ledPin = 13;
	uint8_t payload[PAYLOAD_SIZE];
	char* name;
	Uberdust();
	// HomeAuto(char name[]);
	void setup(XBeeRadio* xbee, Tx16Request* tx);
	void blinkLED(int times, int milliseconds);
	void sendValue(String capability, int value);
	void sendValue(String capability, float value);
	void sendValue(String capability, double value);
	void sendValue(String capability, String value);
	// void sendValue(String capability, char value[]);
private:
	XBeeRadio* xbee;
	Tx16Request* tx;
	void init(char name[]);
	void send_data();
	void send_data(int sensor_type, int sensor_val);
};

#endif
```

Uberdust.cpp

```
#include "Uberdust.h"

Uberdust::Uberdust()
{
	init("unknown");
}

void Uberdust::init(char name[])
{
	this->name = (char*) malloc(sizeof(char) * (strlen(name)+1));
	strcpy(this->name, name);
}

void Uberdust::setup(XBeeRadio* xbee, Tx16Request* tx)
{
	pinMode(ledPin, OUTPUT);
	digitalWrite(ledPin, LOW);
	this->xbee = xbee;
	this->tx = tx;
}

void Uberdust::blinkLED(int times, int milliseconds)
{
	for(int i = 0; i < times; i++)
	{
	digitalWrite(ledPin, HIGH);
	delay(milliseconds/times/2);
	digitalWrite(ledPin, LOW);
	delay(milliseconds/times/2);
	}
}

#define CAPAB_LENGTH 10
#define VALUE_LENGTH 10

void Uberdust::sendValue(String capability, int value)
{
	char val[VALUE_LENGTH];
```

```
	memset(val, '\0', VALUE_LENGTH);
	itoa(value, val, VALUE_LENGTH);
	sendValue(capability, String(val));
}

void Uberdust::sendValue(String capability, float value)
{
	sendValue(capability, (double) value);
}

void Uberdust::sendValue(String capability, double value)
{
	char val[VALUE_LENGTH];
	memset(val, '\0', VALUE_LENGTH);
	dtostrf(value, 6, 3, val);
	sendValue(capability, String(val));
}

void Uberdust::sendValue(String capability, String value)
{
	char capab[CAPAB_LENGTH];
	memset(capab, '\0', CAPAB_LENGTH);
	capability.toCharArray(capab, CAPAB_LENGTH);

	char val[VALUE_LENGTH];
	memset(val, '\0', VALUE_LENGTH);
	value.toCharArray(val, VALUE_LENGTH);

	uint16_t myAddr = xbee->getMyAddress();

	payload[0] = 103;
	payload[1] = strlen(capab)+1;
	payload[2] = strlen(val)+1;
	memcpy(payload+3, &myAddr, 2);
	payload[5] = 0xff;
	payload[6] = 0xff;
	memcpy(payload+7, capab, strlen(capab)+1);
	memcpy(payload+7+strlen(capab)+1, val, strlen(val)+1);
	send_data();
}
```

```
//Subroutine, which sends sensor's data
void Uberdust::send_data(int sensor_type, int sensor_val)
{
	// initialize bla_pointer pointer for reading 32 bit sensors values
	uint8_t * bla_pointer;

	payload[0] = 103;
	payload[1] = sensor_type;
	bla_pointer = (uint8_t*) &sensor_val; //ldr_val
	payload[2] = *bla_pointer;
	bla_pointer++;
	payload[3] = *bla_pointer;
	bla_pointer++;
	payload[4] = 0; //*bla_pointer;
	bla_pointer++;
	payload[5] = 0; //*bla_pointer;
	send_data();
}

void Uberdust::send_data(void)
{
	uint8_t* oldPayload = tx->getPayload();
	uint8_t oldPayloadLength = tx->getPayloadLength();

	tx->setPayload(payload);
	tx->setPayloadLength(PAYLOAD_SIZE);
	xbee->sendAndCheck(*tx,112);
	delay(100);

	tx->setPayload(oldPayload);
	tx->setPayloadLength(oldPayloadLength);
}
```

Uberdust Library Example

Παρατίθενται επίσης δυο παραδείγματα χρήσης της βιβλιοθήκης αυτής, τα οποία ελέγχουν δύο από τα πρωτότυπα συσκευών που αναπτύχθηκαν στα πλαίσια της διπλωματικής. Το πρώτο κύκλωμα αφορά έναν ασύρματο κόμβο δικτύου με την δυνατότητα ελέγχου του φωτισμού και της παροχής ρεύματος ενός δωματίου με 1-6 διακριτές ζώνες, καθώς και την καταγραφή του περιβάλλοντος με αισθητηρες φωτεινότητας, μεθανίου, μονοξειδίου του άνθρακα, θερμοκρασίας και κίνησης, ενώ το δεύτερο αφορά τον απομακρυσμένο έλεγχο της βρύσης και του φωτισμού ενός μπάνιου στην οποία θα αναφερθούμε περαιτέρω σε επόμενο κεφάλαιο.

Κώδικας Ελεγκτή Φωτισμού, Παροχής Ρεύματος και Αισθητήρων

```
#include <XbeeRadio.h>
#include <XBee.h>

#include <Uberdust.h>

// Create the xbee object
XBeeRadio xbee = XBeeRadio();

// Create reusable response objects for responses we expect to handle
XBeeRadioResponse response = XBeeRadioResponse();

// Allocate two bytes for to hold a 32-bit analog reading
uint8_t payload[] = {
  102, 0,  0, 0, 0, 0};

// 16-bit addressing: Enter address of remote XBee, typically the coordinator
Tx16Request tx = Tx16Request(0xffff, payload, sizeof(payload));

TxStatusResponse txStatus = TxStatusResponse();

uint8_t lampPins[] = { 2, 3, 4, 5, 6};
uint8_t relayCheckPin = A4;
uint8_t numOfRelays = 0;

uint8_t lampStatuses[5] = { 0, 0, 0, 0, 0};

uint8_t pirPin = 9;
uint8_t heaterPin = 10;
uint8_t securityPin = 11;
uint8_t sensorsCheckPin = 12;
```

```
uint8_t ledPin = 13;
uint8_t tempPin = A0;
uint8_t lightPin = A1;
uint8_t methanePin = A2;
uint8_t carbonPin = A3;
bool sensorsExist = false;

uint8_t pirStatus;
uint8_t securityStatus;
int tempValue=0;
int lightValue=0;
int methaneValue=0;
int carbonValue=0;


Uberdust uber = Uberdust();

void setup()
{

  xbee.initialize_xbee_module();
  // setup xbee
  xbee.begin(38400);
  xbee.init();

  uber.setup(&xbee, &tx);

  numOfRelays = getNumOfRelays();
  for(int i=0; i< numOfRelays; i++)
  {
    pinMode(lampPins[i], OUTPUT);
    setLamp(i, LOW);
  }
  delay(1000);
  uber.blinkLED(numOfRelays, 200*numOfRelays);
  delay(1000);

  pinMode(sensorsCheckPin, INPUT);
  digitalWrite(sensorsCheckPin, HIGH);
  sensorsExist = !digitalRead(sensorsCheckPin);
```

```
  if(sensorsExist)
  {
    pinMode(pirPin, INPUT);
    digitalWrite(pirPin, HIGH);
    pinMode(heaterPin, OUTPUT);
    pinMode(securityPin, INPUT);
    digitalWrite(securityPin, HIGH);
    uber.blinkLED(1, 500);
  }

    sendCapabilities();
}

void loop()
{
  static unsigned long ledTimestamp = 0;
  if(millis() - ledTimestamp > 5000)
  {
    uber.blinkLED(1,100);
    ledTimestamp = millis();
  }

  if(numOfRelays)
    checkLamps();

  if(sensorsExist)
    checkSensors();

  periodicCapabilities();
}

void periodicCapabilities()
{
  static unsigned long capabTimestamp = 0;
  if(millis() - capabTimestamp > 60000)
  {
    digitalWrite(13, HIGH);
    sendCapabilities();
    digitalWrite(13, LOW);
```

```
    delay(100);
    capabTimestamp = millis();
  }
}
void sendCapabilities(void)
{

  for(int i = 0; i < numOfRelays; i++)
  {
    uber.sendValue("report", String("light")+(i+1));
  }

  if(sensorsExist)
  {
    uber.sendValue("report", "light");
    uber.sendValue("report", "temperature");
    uber.sendValue("report", "pir");
    uber.sendValue("report", "ch4");
  }

}

uint8_t getNumOfRelays(void)
{
  uint8_t relays[] ={0, 0, 0, 0, 0, 0};
  for(int i = 0; i < 10; i++)
  {
    relays[getNumOfRels()]++;
  }
  int num = 0;

  for(int i = 1 ; i < 6; i++)
  {
    if(relays[i] > relays[i-1])
      num = i;
  }
  return num;
}

uint8_t getNumOfRels(void)
```

```
{
  int value = analogRead(relayCheckPin);
  delay(10);
  int relNum = 0;
  int distance[5];
  int thresholds[] = {
    0, 342, 512, 614, 683, 732   };
  for(int i = 0; i< 6; i++)
  {
    thresholds[i] < value? distance[i] = value - thresholds[i] : distance[i] =
thresholds[i] - value;
    //    Serial.print(thresholds[i], DEC);
    //    Serial.print("\t");
    //    Serial.println(distance[i], DEC);
  }

  for(int i = 1; i< 6; i++)
    if(distance[i] < distance[i-1]) relNum = i;

  return relNum;
}

void checkLamps(void)
{
  const int inactiveMins = 5;
  static unsigned long lastTimestamp = 0;
  if(xbee.checkForData(112))
  {

    xbee.getResponse(response);
    if(response.getData(0) == 1)
    {
      lastTimestamp = millis();
      int lamp = response.getData(1);
      int value = response.getData(2);
      if(lamp > 0 && lamp <= numOfRelays)
      {
        setLamp(lamp-1, value);
        reportLamp(lamp-1);
      }
```

```
      else if(lamp == 0xff)
      {
        setAllLamps(value);
        reportAllLamps();
      }
      uber.blinkLED(2,100);
    }
  }

  if(millis() - lastTimestamp > inactiveMins * 60000)
  {
    setAllLamps(LOW);
    reportAllLamps();
    lastTimestamp = millis();
  }

  static unsigned long reportTimestamp = 0;
  if(millis() - reportTimestamp > 60000)
  {
    reportAllLamps();
    reportTimestamp = millis();
  }

}

void setLamp(int lamp, int value)
{
  lampStatuses[lamp] = value;
  digitalWrite(lampPins[lamp], lampStatuses[lamp]);
}

void setAllLamps(int value)
{
  for(int i = 0; i < numOfRelays ; i++)
    setLamp(i, value);
}

void reportLamp(int lamp)
{
  uber.sendValue(String("light")+(lamp+1), lampStatuses[lamp]);
```

```
}

void reportAllLamps(void)
{
  for(int i = 0; i < numOfRelays; i++)
    reportLamp(i);
}

void checkSensors(void)
{
  checkPir();
  checkLight();
  checkTemp();
  checkMethane();
}

void checkPir(void)
{
  static unsigned long pirTimestamp = 0;
  if(millis() - pirTimestamp > 500)
  {

    int newPirStatus = digitalRead(pirPin); // read the value from the sensor
    if(newPirStatus != pirStatus || !newPirStatus)
    {

      uber.sendValue("pir", !newPirStatus);
      if(newPirStatus)
      {
        uber.sendValue("pir", !newPirStatus);
        uber.sendValue("pir", !newPirStatus);
      }

    }
    pirStatus = newPirStatus;
    pirTimestamp = millis();
  }
}

void checkLight(void)
```

```
{
  // for light sensor

  static unsigned long lightTimestamp = 0;
  if(millis() - lightTimestamp > 3 * 60000)
  {
    lightValue = analogRead(lightPin);  // read the value from the sensor
    uber.sendValue("light", lightValue);
    lightTimestamp = millis();
  }

}

void checkTemp(void)
{
  // for temp sensor

  static unsigned long tempTimestamp = 0;
  if(millis() - tempTimestamp > 3 * 60000)
  {
    uint8_t value = analogRead(tempPin);  // read the value from the sensor
    tempValue = map(value, 0, 1024, 0, 5000)/10;
    uber.sendValue("temperature", tempValue);
    tempTimestamp = millis();
  }
}

void checkMethane(void)
{
  // for temp sensor

  static unsigned long methaneTimestamp = 0;
  if(millis() - methaneTimestamp > 3 * 60000)
  {
    methaneValue = analogRead(methanePin);  // read the value from the sensor
    uber.sendValue("ch4", methaneValue);
    methaneTimestamp = millis();
  }
}
```

Κώδικας Ελεγκτή Βρύσης και Αυτόματου Φωτισμού Μπάνιου

```
#include <XbeeRadio.h>
#include <XBee.h>

#include <Uberdust.h>

#define ZONE_NAME "zone"
#define PROX_DISTANCE 35.0

// Create the xbee object
XBeeRadio xbee = XBeeRadio();

// Create reusable response objects for responses we expect to handle
XBeeRadioResponse response = XBeeRadioResponse();

// Allocate two bytes for to hold a 32-bit analog reading
uint8_t payload[] = {
  102, 0,  0, 0, 0, 0};

// 16-bit addressing: Enter address of remote XBee, typically the coordinator
Tx16Request tx = Tx16Request(0xffff, payload, sizeof(payload));

TxStatusResponse txStatus = TxStatusResponse();

uint8_t lampPins[] = { 2, 3};
uint8_t lampStatuses[] = { 0, 0};

uint8_t numOfRelays = 2;

uint8_t pirPin = 9;
uint8_t pirStatus = LOW;

uint8_t proxPin = A3;
uint8_t proxStatus = LOW;

uint8_t floodPin = 11;
uint8_t floodStatus = LOW;

uint8_t modePin = 8;
```

```
uint8_t modeStatus = LOW;

uint8_t ledPin = 13;

Uberdust uber = Uberdust();

void setup()
{

  xbee.initialize_xbee_module();
  // setup xbee
  xbee.begin(38400);
  xbee.init();

  uber.setup(&xbee, &tx);

  setupRelays();
  initializePins();

  delay(1000);
  uber.blinkLED(numOfRelays, 200*numOfRelays);
  delay(1000);

  sendCapabilities();
}

void loop()
{
  periodicBlink();
  doCheckSensors();

  doCheckMode();
  if(modeStatus)
  {
    if(numOfRelays)
      checkLamps();
    checkSensors();
  }
  else
  {
```

```
    static unsigned long valveTimestamp = 0;
    if(proxStatus && !floodStatus)
    {
      digitalWrite(lampPins[1], HIGH);
      valveTimestamp = millis();
    }
    else
    {
      if(millis() - valveTimestamp > 1000)
        digitalWrite(lampPins[1], LOW);
    }

    static unsigned long moveTimestamp = 0;
    if(pirStatus)
    {
      digitalWrite(lampPins[0], HIGH);
      moveTimestamp = millis();
    }
    else
    {
      if(millis() - moveTimestamp > 60000)
        digitalWrite(lampPins[0], LOW);
    }
  }
  periodicCapabilities();
}

void initializePins(void)
{
  pinMode(ledPin, OUTPUT);

  pinMode(pirPin, INPUT);
  digitalWrite(pirPin, HIGH);

  pinMode(floodPin, INPUT);
  digitalWrite(floodPin, HIGH);

  pinMode(modePin, INPUT);
  digitalWrite(modePin, HIGH);
}
```

```
void periodicBlink()
{
  static unsigned long ledTimestamp = 0;
  if(millis() - ledTimestamp > 5000)
  {
    uber.blinkLED(1,100);
    ledTimestamp = millis();
  }
}

void periodicCapabilities()
{
  static unsigned long capabTimestamp = 0;
  if(millis() - capabTimestamp > 60000)
  {
    digitalWrite(13, HIGH);
    sendCapabilities();
    digitalWrite(13, LOW);
    delay(100);
    capabTimestamp = millis();
  }
}
void sendCapabilities(void)
{

  for(int i = 0; i < numOfRelays; i++)
  {
    uber.sendValue("report", String(ZONE_NAME)+(i+1));
  }
  uber.sendValue("report", "proximity");
  uber.sendValue("report", "pir");
  uber.sendValue("report", "flood");
}

void setupRelays(void)
{
//  numOfRelays = getNumOfRelays();
  for(int i=0; i< numOfRelays; i++)
  {
```

```
    pinMode(lampPins[i], OUTPUT);
    setLamp(i, LOW);
  }
}

void checkLamps(void)
{
  const int inactiveMins = 5;
  static unsigned long lastTimestamp = 0;
  if(xbee.checkForData(112))
  {

    xbee.getResponse(response);
    if(response.getData(0) == 1)
    {
      lastTimestamp = millis();
      int lamp = response.getData(1);
      int value = response.getData(2);
      if(lamp > 0 && lamp <= numOfRelays)
      {
        setLamp(lamp-1, value);
        reportLamp(lamp-1);
      }
      else if(lamp == 0xff)
      {
        setAllLamps(value);
        reportAllLamps();
      }
      uber.blinkLED(2,100);
    }
  }

  if(millis() - lastTimestamp > inactiveMins * 60000)
  {
    setAllLamps(LOW);
    reportAllLamps();
    lastTimestamp = millis();
  }

  static unsigned long reportTimestamp = 0;
```

```
  if(millis() - reportTimestamp > 60000)
  {
    reportAllLamps();
    reportTimestamp = millis();
  }
}

void setLamp(int lamp, int value)
{
  lampStatuses[lamp] = value;
  digitalWrite(lampPins[lamp], lampStatuses[lamp]);
}

void setAllLamps(int value)
{
  for(int i = 0; i < numOfRelays ; i++)
    setLamp(i, value);
}

void reportLamp(int lamp)
{
  uber.sendValue(String(ZONE_NAME)+(lamp+1), lampStatuses[lamp]);
}

void reportAllLamps(void)
{
  for(int i = 0; i < numOfRelays; i++)
    reportLamp(i);
}

void doCheckSensors(void)
{
  doCheckFlood();
  doCheckPir();
  doCheckProximity();
}

void checkSensors(void)
{
  checkFlood();
```

```
  checkPir();
  checkProximity();
}

void doCheckPir(void)
{
  static unsigned long pirTimestamp = 0;
  if(millis() - pirTimestamp > 500)
  {
    pirStatus = !digitalRead(pirPin); // read the value from the sensor
    pirTimestamp = millis();
  }
}

void checkPir(void)
{
  static uint8_t oldPirStatus = 0;
  if(oldPirStatus != pirStatus)
  {
    uber.sendValue("pir", pirStatus);
    if(oldPirStatus)
    {
      uber.sendValue("pir", pirStatus);
      uber.sendValue("pir", pirStatus);
    }
    oldPirStatus = pirStatus;
  }

  static unsigned long pirTimestamp = 0;
  if(millis() - pirTimestamp > 1000)
  {
    if(pirStatus)
      uber.sendValue("pir", pirStatus);
    pirTimestamp = millis();
  }
}

void doCheckProximity()
{
  static unsigned long proxTimestamp = 0;
```

```
  if(millis() - proxTimestamp > 30)
  {
    static uint8_t count = 0;
    static float data[5];
    float volts = analogRead(proxPin)*0.0048828125;  // value from sensor *
(5/1024) - if running 3.3.volts then change 5 to 3.3
    data[count++] = 65*pow(volts, -1.10)*0.254;
    if(count == 5) count = 0;

    int value_count = 0;
    for(int i = 0; i < 5; i++)
    {
      if(data[i] > 3.0 && data[i] < PROX_DISTANCE)
        value_count++;
    }
    if(value_count == 5)
      proxStatus = HIGH;
    else
      proxStatus = LOW;

      proxTimestamp = millis();
  }
}

void checkProximity(void)
{
  static uint8_t oldProxStatus = 0;
  if(proxStatus != oldProxStatus)
  {
    uber.sendValue("proximity", proxStatus);
    oldProxStatus = proxStatus;
  }

  static unsigned long proxTimestamp = 0;
  if(millis() - proxTimestamp > 60000)
  {
    uber.sendValue("proximity", proxStatus);
    proxTimestamp = millis();
  }
}
```

```
void doCheckFlood(void)
{
  static unsigned long floodTimestamp = 0;
  if(millis() - floodTimestamp > 500)
  {
    floodStatus = !digitalRead(floodPin); // read the value from the sensor
    floodTimestamp = millis();
  }
}

void checkFlood(void)
{
  static uint8_t oldFloodStatus = 0;
  if(floodStatus != oldFloodStatus)
  {
    uber.sendValue("flood", floodStatus);
    oldFloodStatus = floodStatus;
  }

  static unsigned long floodTimestamp = 0;
  if(millis() - floodTimestamp > 60000)
  {
    uber.sendValue("flood", floodStatus);
    floodTimestamp = millis();
  }
}

void doCheckMode(void)
{
  static unsigned long modeTimestamp = 0;
  if(millis() - modeTimestamp > 500)
  {
    modeStatus = digitalRead(modePin); // read the value from the sensor
    modeTimestamp = millis();
  }
}
```

Απομακρυσμένος Έλεγχος Arduino με χρήση Πρωτοκόλλου CoAP

Όπως αναφέρθηκε προηγουμένος, για την επικοινωνία των συσκευών χρησιμοποιήθηκε και το πρωτόκολλο CoAP. Για λόγους σύγκρισης, παρατίθεται επίσης το πρώτο από τα παραπάνω παραδείγματα, αλλά με χρήση της βιβλιοθήκης CoAP, η οποία απλοποιεί την επικοινωνία με εναν τυποποιημένο τρόπο.

Χρησιμοποιούμε το ίδιο κύκλωμα ελέγχου του φωτισμού και της παροχής ρεύματος ενός δωματίου με 1-6 διακριτές ζώνες και καταγραφής του περιβάλλοντος με αισθητηρες φωτεινότητας, μεθανίου, μονοξειδίου του άνθρακα, θερμοκρασίας και κίνησης.

Κώδικας Ελεγκτή Φωτισμού, Παροχής Ρεύματος και Αισθητήρων
CoAP_Controller.ino

```
/**
 * Arduino Coap Example Application.
 *
 * This Example creates a CoAP server with 2 resources.
 * resGET : A resource that contains an integer and the GET method is
only available.
 *          A GET request returns the value of the value_get variable.
 * resGET-POST : A resource that contains an integer and the GET-POST
methods are available.
 *               A GET request returns the value of the value_post
variable.
 *               A POST request sets the value of value_post to the sent
integer.
 * Both resources are of TEXT_PLAIN content type.
 */

//Include XBEE Libraries
#include <XBee.h>
#include <XbeeRadio.h>

//Include CoAP Libraries
#include <coap.h>
#include "mySensor.h"

//Create the XbeeRadio object we'll be using
XBeeRadio xbee = XBeeRadio();
//Create a reusable response object for responses we expect to handle
XBeeRadioResponse response = XBeeRadioResponse();
//Create a reusable rx16 response object to get the address
```

```
Rx16Response rx = Rx16Response();

//CoAP object
Coap coap;

//CoapSensor aSensor = CoapSensor();

//Runs only once
void setup()
{
  // comment out for debuging
  xbee.initialize_xbee_module();
  //start our XbeeRadio object and set our baudrate to 38400.
  xbee.begin( 38400 );
  //Initialize our XBee module with the correct values (using the
default channel, channel 12)h
  xbee.init(12);

  // init coap service
  coap.init( &xbee, &response, &rx );

  add_relays();
  add_sensors();
}

void loop()
{
  //run the handler on each loop to respond to incoming requests
  coap.handler();
}


uint8_t getNumOfRelays(int relayCheckPin)
{
  uint8_t relays[] ={0, 0, 0, 0, 0, 0};
  for(int i = 0; i < 10; i++)
  {
    relays[getNumOfRels(relayCheckPin)]++;
  }
  int num = 0;

  for(int i = 1 ; i < 6; i++)
  {
    if(relays[i] > relays[i-1])
      num = i;
```

```
  }
  return num;
}

uint8_t getNumOfRels(int relayCheckPin)
{
  int value = analogRead(relayCheckPin);
  delay(10);
  int relNum = 0;
  int distance[5];
  int thresholds[] = {
    0, 342, 512, 614, 683, 732   };
  for(int i = 0; i< 6; i++)
  {
    thresholds[i] < value? distance[i] = value - thresholds[i] :
distance[i] = thresholds[i] - value;
  }

  for(int i = 1; i< 6; i++)
    if(distance[i] < distance[i-1]) relNum = i;

  return relNum;
}

void add_relays()
{
  #define RELAY_CHECK_PIN A4
  int numOfRelays = getNumOfRelays(RELAY_CHECK_PIN);
  for(int i=0; i< numOfRelays; i++)
  {
    #define RELAY_START_PIN 2
    zoneSensor* lzSensor = new zoneSensor(String("lz")+(i+1),
RELAY_START_PIN+i);
    coap.add_resource(lzSensor);
  }
}

void add_sensors()
{
  #define SENSORS_CHECK_PIN 12
  #define SECURITY_PIN 11
  #define TEMP_PIN A0
  #define LIGHT_PIN A1
  #define METHANE_PIN A2
  #define CARBON_PIN A3
```

```
  #define HEATER_PIN 10
  #define PIR_PIN 9
  pinMode(SENSORS_CHECK_PIN, INPUT);
  digitalWrite(SENSORS_CHECK_PIN, HIGH);
  bool sensorsExist = !digitalRead(SENSORS_CHECK_PIN);

  switchSensor* swSensor = new switchSensor("security", SECURITY_PIN,
HIGH);
  coap.add_resource(swSensor);
  temperatureSensor* tempSensor = new temperatureSensor("temp",
TEMP_PIN);
  coap.add_resource(tempSensor);
  lightSensor* liSensor = new lightSensor("light", LIGHT_PIN);
  coap.add_resource(liSensor);
  methaneSensor* mh4Sensor = new methaneSensor("ch4", METHANE_PIN);
  coap.add_resource(mh4Sensor);
  pirSensor* pSensor = new pirSensor("pir", PIR_PIN);
  coap.add_resource(pSensor);
}
```

---

mySensor.h

```
#include <CoapSensor.h>

class zoneSensor : public CoapSensor
{
public:
int pin, status;
  zoneSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    this->status = 0;
    pinMode(pin, OUTPUT);
    digitalWrite(pin, LOW);
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
  {
    this->status = digitalRead(this->pin);
    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }
  void set_value(uint8_t* input_data, size_t input_data_len, uint8_t*
output_data, size_t* output_data_len)
  {
    this->set(*input_data-0x30);
    output_data[0] = 0x30 + status;
    *output_data_len = 1;
  }
  void set(uint8_t value)
  {
    if(this->status != value)
      this->changed = true;
    this->status = value;
    digitalWrite(pin, status);
  }
};

class switchSensor : public CoapSensor
{
public:
int pin, status;
  switchSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    pinMode(pin, INPUT);
```

```
    this->status = 0;
  }
  switchSensor(String name, int pin, int pullup): CoapSensor(name)
  {
    this->pin = pin;
    pinMode(pin, INPUT);
    digitalWrite(pin, pullup);
    this->status = 0;
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
  {
    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }

  void check()
  {
    int newStatus = digitalRead(this->pin);
    if(newStatus != this->status)
    {
      this->changed = true;
      this->status = newStatus;
    }
  }
};

class lightSensor : public CoapSensor
{
public:
int pin, status;
  lightSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    this->status = 0;
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
  {
    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }
  void check(void)
  {
    static unsigned long timestamp = 0;
    if(millis() - timestamp > 500)
    {
```

```
      int newStatus = analogRead(this->pin);  // read the value from the
sensor
      if(newStatus > this->status + 10 || newStatus < this->status -
10 )
      {
        this->changed = true;
        this->status = newStatus;
      }
      timestamp = millis();
    }
  }
};

class temperatureSensor : public CoapSensor
{
public:
int pin, status;
  temperatureSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    this->status = 0;
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
  {

    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }
  void check(void)
  {
    static unsigned long timestamp = 0;
    if(millis() - timestamp > 500)
    {
      int newStatus = analogRead(this->pin);  // read the value from the
sensor
      newStatus = map(newStatus, 0, 1024, 0, 5000)/10;
      if(newStatus != this->status)
      {
        this->changed = true;
        this->status = newStatus;
      }
      timestamp = millis();
    }
  }
};
```

```
class pirSensor : public CoapSensor
{
public:
int pin, status;
  pirSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    this->status = LOW;
    pinMode(pin, INPUT);
    digitalWrite(pin, HIGH);
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
  {
    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }

  void check(void)
  {
    static unsigned long timestamp = 0;
    if(millis() - timestamp > 500)
    {
      int newStatus = !digitalRead(this->pin); // read the value from
the sensor
      if(newStatus != this->status)
      {
        this->changed = true;
        this->status = newStatus;
      }
      timestamp = millis();
    }
  }
};

class methaneSensor : public CoapSensor
{
public:
int pin, status;
  methaneSensor(String name, int pin): CoapSensor(name)
  {
    this->pin = pin;
    this->status = 0;
  }
  void get_value( uint8_t* output_data, size_t* output_data_len)
```

```
  {
    *output_data_len = sprintf( (char*)output_data, "%d", this-
>status );
  }

  void check(void)
  {
    static unsigned long timestamp = 0;
    if(millis() - timestamp > 150000)
    {
      this->status = analogRead(this->pin);  // read the value from the
sensor
      this->changed = true;
      timestamp = millis();
    }
  }
};
```

Σχεδιασμός και Ανάπτυση Υλικού και Λογισμικού για Αυτόματη Αναγνώριση και Παραμετροποίηση Αισθητήρων και Ελεγκτών

Στην πλατφόρμα δοκιμών (testbed) που έχει αναπτυχθεί στις εγκαταστάσεις του ΙΤΥΕ «Διόφαντος», έχουμε ένα μεγάλο αριθμό κόμβων με διάφορους αισθητήρες, που συνδέονται με διάφορους ελεγκτές, και με διαφορετική επεξεργαστική ισχύ και αποθηκευτικών πόρων.

Το κόστος για την ενημέρωση του firmware των συσκευών κάθε φορά που θα πρέπει να εγκατασταθεί ένας νέος αισθητήρας ή να συνδεθεί μια νέα επέκταση με κάποια από τις συσκευές είναι πολύ υψηλό. Αυτό μας οδηγεί στην ανάπτυξη μιας λύσης υλοποιημένης σε hardware για την ανίχνευση των συσκευών που συνδέονται με τους ασύρματους κόμβους που αποτελούνται από Arduino & XBee. Σε επίπεδο λογισμικού, το σύστημα ελέγχει περιοδικά την πλακέτα επέκτασης του κόμβου, και αναγνωρίζει κάθε νέα συσκευή που εντοπίζεται (ή μια υπάρχουσα που αφαιρείται).

Οι εντοπισμένοι πόροι μπορούν να στη συνέχεια αναφερθούν στην κεντρική πύλη (gateway) του δικτύου και να γίνουν διαθέσιμοι για όλους τους πελάτες στο διαδίκτυο. Στο υπόλοιπο αυτής της ενότητας θα περιγράψουμε πώς εφαρμόζεται η αυτόματη ανίχνευση και ρύθμιση για τους κόμβους Arduino & Xbee.

Κόμβοι Arduino

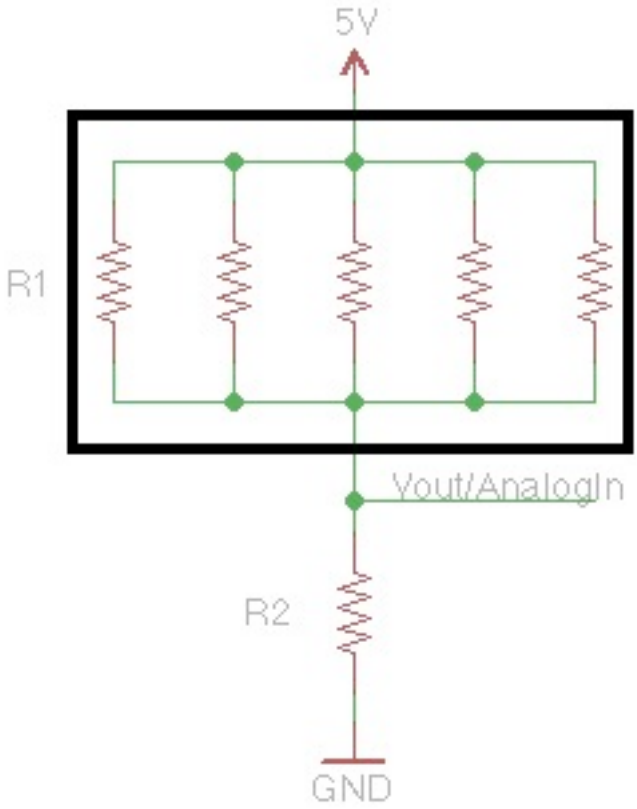
Η πιο κοινή διαφοροποίηση μεταξύ των κόμβων Arduino είναι ο αριθμός των ελεγκτών δυο διακριτών καταστάσεων (on/off) που έχει κάθε συσκευή, στην περίπτωση μας, τα ρελέ που οποία ελέγχουν το φωτισμό του δωματίου. Σχεδιάσαμε μια πλακέτα επέκτασης με ρελέ, η οποία συνδέεται σε ένα Arduino, ή σε σειρά σε μια άλλη πλακέτα επέκτασης.

Η πρώτη πλακέτα επέκτασης ρελέ σε ένα συγκεκριμένο pin του Arduino. Με την προσθήκη μιας δεύτερης πλακέτας επέκτασης σε σειρά στην πρώτη, τα ρελέ της συνδέονται αυτόματα σε επίπεδο hardware στο επόμενο διαθέσιμο pin του Arduino, και ούτω καθεξής. Ως εκ τούτου, όλα τα ρελέ μας είναι συνδεδεμένα σε μια σειρά από pins εισόδου/εξόδου (I/O pins), και τροφοδοτούνται με ρεύμα από την πλακέτα του Arduino. Για τον υπολογισμό του αριθμού των συνδεδεμένων ρελέ, οι πλακέτες επέκταση και το Arduino επίσης εφαρμόζουν έναν διαιρέτη τάσης[14] (όπως αυτός στο επόμενο σχήμα). Η δεύτερη αντίσταση του διαιρέτη τάσεως, που συνήθως αναφέρονται ως R2, είναι σταθερή, ενώ η τιμή της πρώτης, που ονομάζεται R1, είναι ένας συνδυασμός των παράλληλων αντιστάσεων του σχήματος.

Κάθε πλακέτα επέκτασης φέρει μια αντίσταση, η οποία συνδέεται παράλληλα με τις αντιστάσεις των άλλων πλακετών επέκτασης. Η συνολική τιμή του συνδυασμού των αντιστάσεων αλλάζει ανάλογα με τον αριθμό των επεκτάσεων που είναι συνδεδεμένες ανά πάσα χρονική στιγμή, και χρησιμοποιούμε την τάση εξόδου του κυκλώματος διαιρέτη τάσης για να προσδιορίσουμε τον αριθμό των ρελέ που είναι συνδεδεμένα στον εκάστοτε κόμβο. Αυτό μας επιτρέπει επίσης και να σχεδιάσουμε πλακέτες επέκτασης ρελέ με περισσότερα από ένα ρελέ, χωρίς να κάνουμε καμία αλλαγή, εκτός από την χρήση μιας αντίστασης με τιμή που ισοδυναμεί με την χρήση όσων αντιστάσεων όσα και τα ρελέ του πίνακά μας, συνδεδεμένων παράλληλα.

Η δεύτερη πιο κοινή διαφοροποίηση ήταν η ύπαρξη (ή έλλειψη) μιας πλακέτας αισθητήρων, η οποία περιλαμβάνει αισθητήρες για τη μέτρηση της ποιότητας του αέρα, του φωτισμού, της θερμοκρασίας, και μέτρηση κίνησης. Δεδομένου ότι η ύπαρξη της ή μη είναι μια διακριτή δυαδική τιμή, χρησιμοποιείται ένα ψηφιακό pin εισόδου με ένα μια συνδεδεμένη pull-up αντίσταση, οποία θέτει την τιμή της εισόδου σε HIGH ως αρχική τιμή. Σε αντίθεση, πλακέτα αισθητήρων συνδέει την ίδια είσοδο στη γείωση, αναγκάζοντας την είσοδο να γίνει LOW εφόσον υπάρχει συνδεδεμένη πλακέτα αισθητήρων στον κόμβο. Ως εκ τούτου, το μόνο που χρειάζεται είναι να διαβάσουμε την τιμή του ψηφιακού pin, για να διαπιστωθεί αν η πλακέτα αισθητήρων είναι συνδεδεμένη με το Arduino.

Μετά τον καθορισμό του αριθμού των αισθητήρων και ελεγκτών που είναι συνδεδεμένοι στον κόμβο, το λογισμικό μας ειδοποιεί την πλησιέστερη πύλη (gateway) σχετικά με τους περιορισμούς και τις δυνατότητες του κόμβου.



Κύκλωμα Διαιρέτη Τάσης & Αυτόματης Ρύθμισης

Ένα από τα πλεονεκτήματα του σχεδιασμού μας είναι ότι ο αριθμός των ενεργοποιητών και των αισθητήρων που συνδέονται με κάθε κόμβο μπορεί να αλλαχτεί ενώ το σύστημα βρίσκεται σε λειτουργία. Σε μια τέτοια περίπτωση, ο κόμβος θα εντοπίσει αυτόματα τις νέες δυνατότητες του, ή την αφαίρεση κάποιων από αυτών, και ενημερώνει σχετικά το WSN με τη νέα λίστα.

Ωστόσο, όσο επεκτάσιμες και να είναι η παραπάνω προσέγγιση, είμαστε ακόμα περιορισμένοι από τον αριθμό των ψηφιακών εισόδων/εξόδων (I/O pins) του επεξεργαστή του Arduino. Ο ATmega328 συγκεκριμένα έχει 13 ψηφιακές εισόδους/εξόδους, και 5 αναλογικές εισόδους. Από τις ψηφιακές εισόδους/εξόδους, 3 είναι

αχρησιμοποίητα , επειδή καταλαμβάνονται από ένα ενσωματωμένο LED και μια σειριακή θύρα, η οποία είναι ζωτικής σημασίας για την επικοινωνία με το συνημμένο XBee, και μέσω αυτού στο WSN.
Από τα υπόλοιπα pins, χρησιμοποιούμε ένα ψηφιακό I/O pin για την ανίχνευση της πλακέτας αισθητήρων, και μία αναλογική είσοδο για την ανίχνευση των ελεγκτών/ ρελέ. Αυτό μας αφήνει μόνο 9 ψηφιακά I/O pins και 4 αναλογικές εισόδους.

Για τη βελτίωση του παραπάνω συστήματος, χρησιμοποιήθηκε ο προϋπάρχων δίαυλος I2C του Arduino, ώστε να δώσουμε την δυνατότητα σε έναν αριθμό από Arduino να συνδεθούν και να επικοινωνήσουν μέσω ενός ενσύρματου κοινού διαύλου. Ένας Arduino λειτουργεί ως master, και συνδέεται στο WSN με ένα συνημμένο XBee, ενώ τα υπόλοιπα λειτουργούν ως slaves.

Με αυτόν τον τρόπο, έχουμε έναν εξαιρετικά επεκτάσιμο, έξυπνο και χαμηλού κόστους ασύρματο κόμβο, με ένα μεγάλο αριθμό από αισθητήρες και ελεγκτές (θεωρητικά, περισσότερους από εκατό) που μπορούν να συνδεθούν σε αυτόν. Για το σχεδιασμό ενός τέτοιου συστήματος , το κύριο εμπόδιο που έπρεπε να ξεπεραστεί ήταν η έλλειψη αυτόματης διασύνδεσης και αναγνώρισης συσκευών στον δίαυλο I2C, αφού η κάθε συσκευή έχει μια συγκεκριμένη ταυτότητα (ID) στον δίαυλο, η οποία πρέπει να οριστεί πριν συνδεθεί στον δίαυλο, και πρέπει να είναι μοναδική.

Σχεδιάσαμε ένα κύκλωμα το οποίο παρέχει αυτόματα τις διευθύνσεις για τα Arduino slaves του I2C διαύλου τη στιγμή που συνδέονται σε αυτόν . Οι πλακέτες Arduino συνδέονται σε σειρά, και ο Arduino master παρέχει ένα σήμα PWM το οποίο, αφότου περάσει μέσα από ένα χαμηλοπερατό φίλτρο που ενεργεί αντί DAC (Digital-to-Analog-Converter), τροφοδοτείται στην αναλογική είσοδο του δεύτερου Arduino στην αλυσίδα.

Το κάθε Arduino της αλυσίδα λαμβάνει αυτή την είσοδο από το προηγούμενο του, την αποκωδικοποιεί σε ένα διαδοχικό αριθμό, ο οποίος θα είναι η I2C διεύθυνσή του, τον προσαυξάνει και, και το προωθεί μέσω μιας PWM εξόδου του στο επόμενο Arduino της αλυσίδας. Με αυτό τον τρόπο, όλες τα συνδεδεμένα Arduino έχουν διαδοχικές I2C διευθύνσεις , ξεκινώντας από τη διεύθυνση # 1 . Προκειμένου να έχουν τη δυνατότητα να συνδέουν τα Arduino όχι μόνο σε σειρά, αλλά επίσης και παράλληλα, ο σχεδιασμός μας υλοποιεί 3 από αυτές τις αλυσίδες παράλληλα, με διαφορετική διεύθυνση εκκίνησης για το καθένα.

Το μόνο που χρειάζεται να κάνει το master Arduino είναι να αναζητήσει τις συσκευές του διαύλου για να προσδιορίσει τον αριθμό των Arduino που είναι συνδεδεμένα στον δίαυλο, και στη συνέχεια να καταγράψει τις δυνατότητες από κάθε Arduino του διαύλου. Στη συνέχεια, λειτουργεί ως ενδιάμεσος με το WSN, όπου καταχωρεί τις δυνατότητες όλων των Arduino του διαύλου ως τις συνολικές δυνατότητες του κόμβου, και προωθεί τις αιτήσεις του WSN στο εκάστοτε Arduino μέσω του I2C διαύλου.

XBee ως ανεξάρτητοι κόμβοι

Οι συσκευές XBee μπορούν να ενεργήσουν και ως αυτόνομοι κόμβοι, και να ρυθμιστούν ώστε να μεταδίδουν περιοδικά την κατάσταση των 2 αναλογικών εισόδου τους. Προκειμένου να προσθέσουμε αυτόματη αναγνώριση των συνδεδεμένων συσκευών σε κάθε κόμβο XBee, συνδέουμε ένα κύκλωμα διαιρέτη τάσεως σε μία αναλογική είσοδο, και χρησιμοποιούμε προκαθορισμένες τιμές για τις αντιστάσεις, τις οποίες ορίζουμε για να δηλώσουμε τον τον αριθμό των συνδεδεμένων εξόδων στον κόμβο, καθώς και τον τύπο του αισθητήρα που συνδέεται στη δεύτερη αναλογική είσοδο του κόμβου, εφόσον υπάρχει.

Ο κοντινότερος κόμβος του WSN λαμβάνει αυτές τις τιμές, αναγνωρίζει τον αριθμό των εισόδων και εξόδων που είναι συνδεδεμένα στον XBee κόμβο, καθώς και το είδος του αισθητήρα που είναι συνδεδεμένος στην αναλογική είσοδο (εάν υπάρχει), και τα προωθούν στο WSN δίκτυο μας με τρόπο που είναι συμβατός με το mkSense, ως το δικές της δυνατότητες. Στη συνέχεια, λειτουργούν ως ενδιάμεσοι μεταξύ του αυτόνομου κόμβου XBee και του υπόλοιπου WSN, προωθώντας τα μηνύματα μεταξύ τους.

Εγκατάσταση και αξιολόγηση σε πραγματικές συνθήκες - P-Space

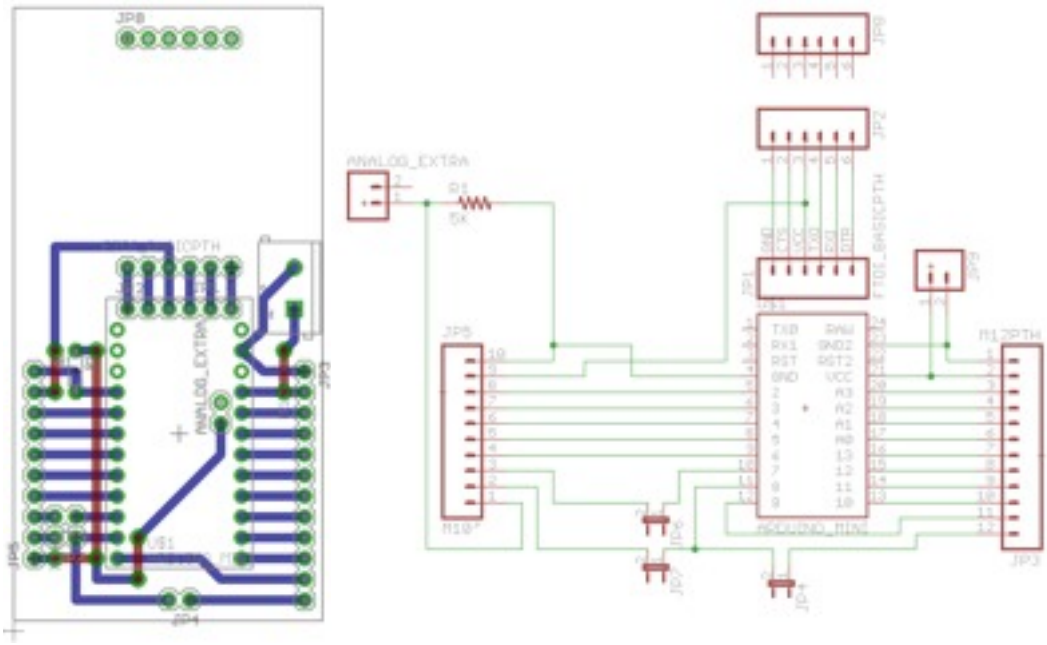
Ως μέρος αυτής της διπλωματικής, αναπτύχθηκε και αξιολογήθηκε και ένα σύστημα πρωτότυπο σύστημα BMS, χρησιμοποιώντας τεχνολογίες Internet of Things για την επίλυση καθημερινών αναγκών σε ένα κτίριο/πολυχώρο στο κέντρο της Πάτρας, ονόματι P-Space. Τα έξυπνα αντικείμενα που εγκαταστάθηκαν στο P-Space σχεδιάστηκαν ως μέρος αυτής της εργασίας, και είναι βασισμένα στην ανοιχτού κώδικα πλατφόρμα Arduino. Η ασύρματη επικοινωνία χρησιμοποιεί τις συσκευές XBee 802.15.4, συνδεδεμένες στα Arduino, και χρησιμοποιώντας την βιβλιοθήκη ασύρματης δικτύωσης mkSense που αναπτύχθηκε επίσης ως μέρος αυτής της εργασίας, και στην οποία αναφέραμε προηγουμένως. Το hardware του κάθε κόμβου συγκεκριμένα αποτελείται από ένα Arduino Pro Mini με επεξεργαστή ATmega328, το οποίο οδηγεί τα κυκλώματα αισθητήρων και ελέγχου που τοποθετήθηκαν σε αντικείμενα όπως γραφεία, φωτιστικά, πόρτες και βρύσες. Τα έξυπνα αντικείμενα που δημιουργήθηκαν ελέγχονται ασύρματα, έχουν χαμηλό κόστος, και είναι συμβατά με ένα μεγάλο εύρος αισθητήρων και ελεγκτών.

Για την επικοινωνία με τους αισθητήρες και τον έλεγχο των έξυπνων αντικειμένων που αναπτύχθηκαν, χρησιμοποιείται το πρωτόκολλο CoAP, ενώ την επικοινωνία μεταξύ συσκευών και εφαρμογών ελέγχου του Internet of Things χρησιμοποιείται το Uberdust[15], μια web εφαρμογή που έχει δυνατότητες αποθήκευσης, καταγραφής, και διαμοιρασμού των δεδομένων από τις έξυπνες συσκευές ή του ιστορικού τους, καθώς και την αυτόματη ρύθμιση νέων συσκευών σε πραγματικό χρόνο, συνδέοντας έτσι τα WSN και των web εφαρμογών που τα ελέγχουν.

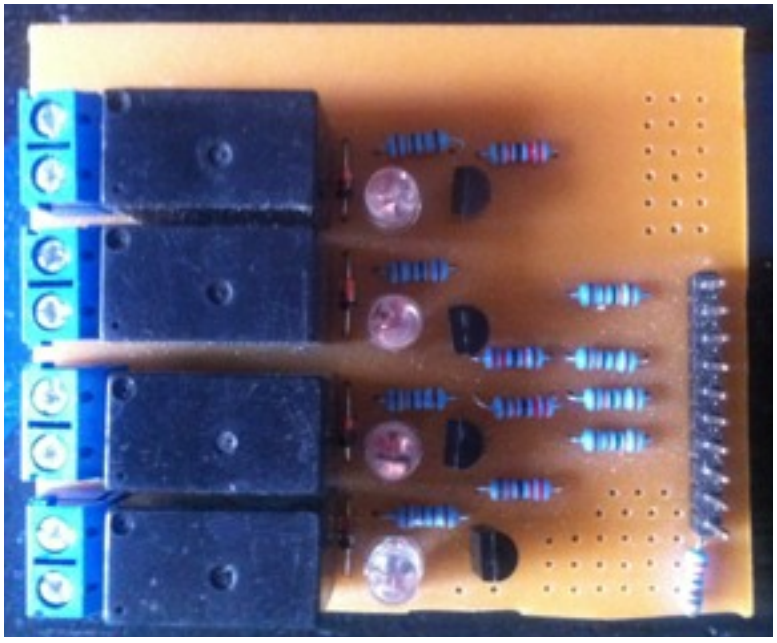
Λόγω της αφαιρετικότητας της αρχιτεκτονικής, οι συσκευές μπορούν να ελεγχθούν από μια ευρεία γλωσσών προγραμματισμού και συστημάτων, όπως π.χ. Python, Java, JavaScript, PHP, ακόμα και Bash Shell scripting. Τέτοιες εφαρμογές χρησιμοποιήθηκαν και για τον έλεγχο των συσκευών που αναπτύχθηκαν ως μέρος της διπλωματικής.

Αυτοματισμός Δωματίου
Ο κύριος κόμβος του δικτύου αφορά τον αυτοματισμό του φωτισμού και του κλιματισμού ενός δωματίου, ελέγχοντας τα φώτα και τον κλιματισμό/θέρμανση με χρήση ρελέ ελέγχου ηλεκτρικών συσκευών 220V, καθώς και καταγραφή της ποιότητας του περιβάλλοντος και της κατάστασης του χώρου.

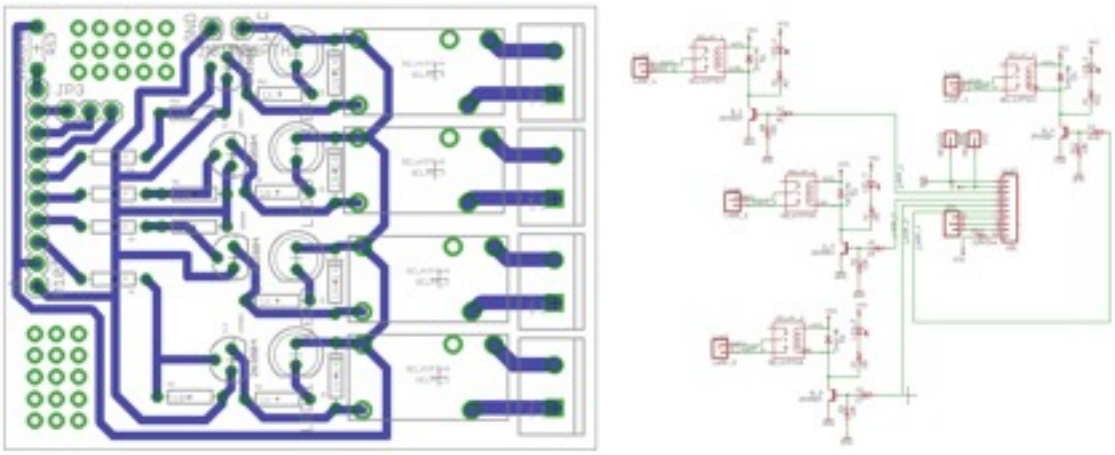
Συγκεκριμένα, ο κόμβος αποτελείται από μία κυρίως πλακέτα, στην οποία βρίσκεται το Arduino και το XBee που αναλαμβάνουν την ασύρματη επικοινωνία και τον έλεγχο των συσκευών και αισθητήρων που είναι συνδεδεμένοι στον κόμβο, από μια πλακέτα επέκτασης ρελέ με αυτόματη αναγνώριση και προσαρμογή στον αριθμό των διαθέσιμων συνδεδεμένων ρελέ/ζωνών προς έλεγχο, και μια πλακέτα επέκτασης αισθητήρων, που παρέχει μέτρηση θερμοκρασίας, φωτεινότητας, κίνησης, ποιότητας αέρα (ποσοστό μεθανίου και μονοξειδίου του άνθρακα στον αέρα), καθώς και αισθητήρας συναγερμού παραθύρου/πόρτας.



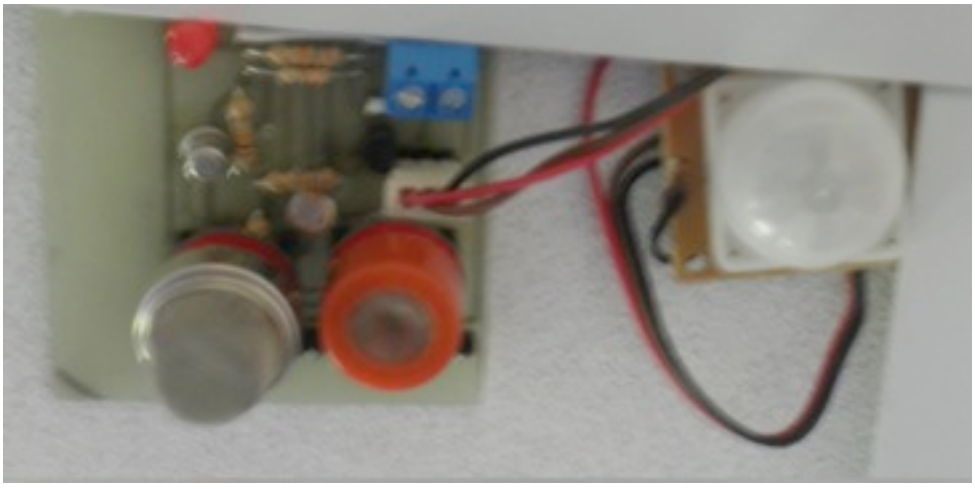
Πλακέτα και Κύκλωμα Κεντρικού Κόμβου



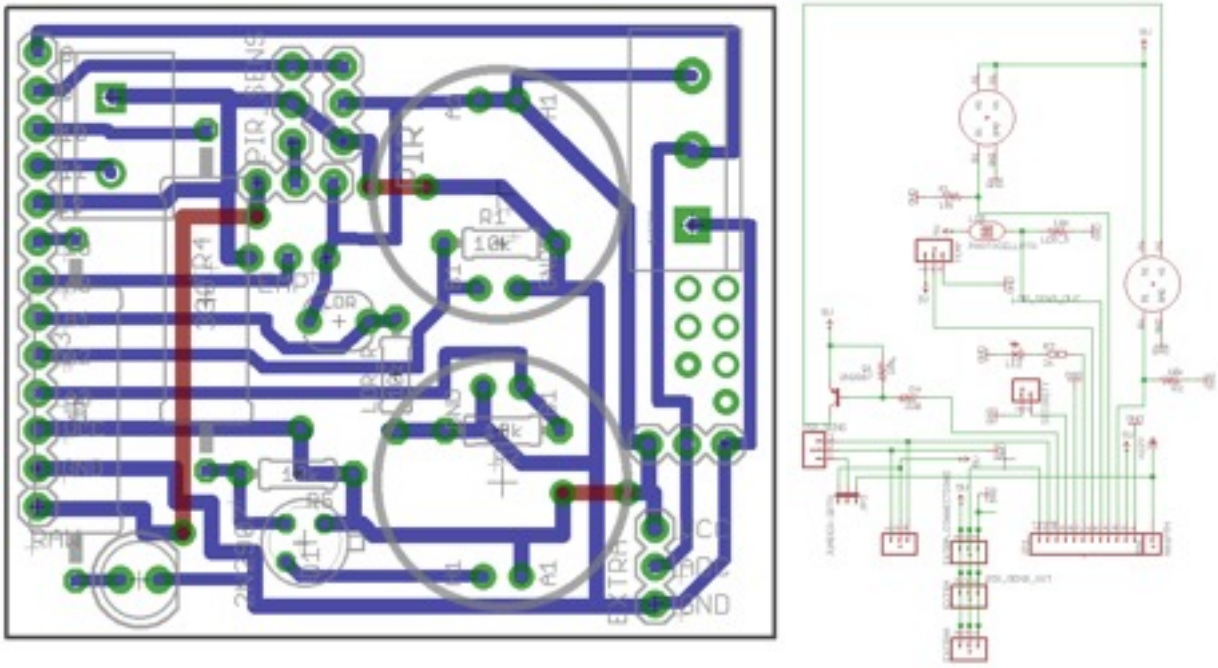
Επέκταση Ελέγχου Ηλεκτρικών Συσκευών (Ρελέ)



Πλακέτα και Κύκλωμα Επέκτασης Ρελέ



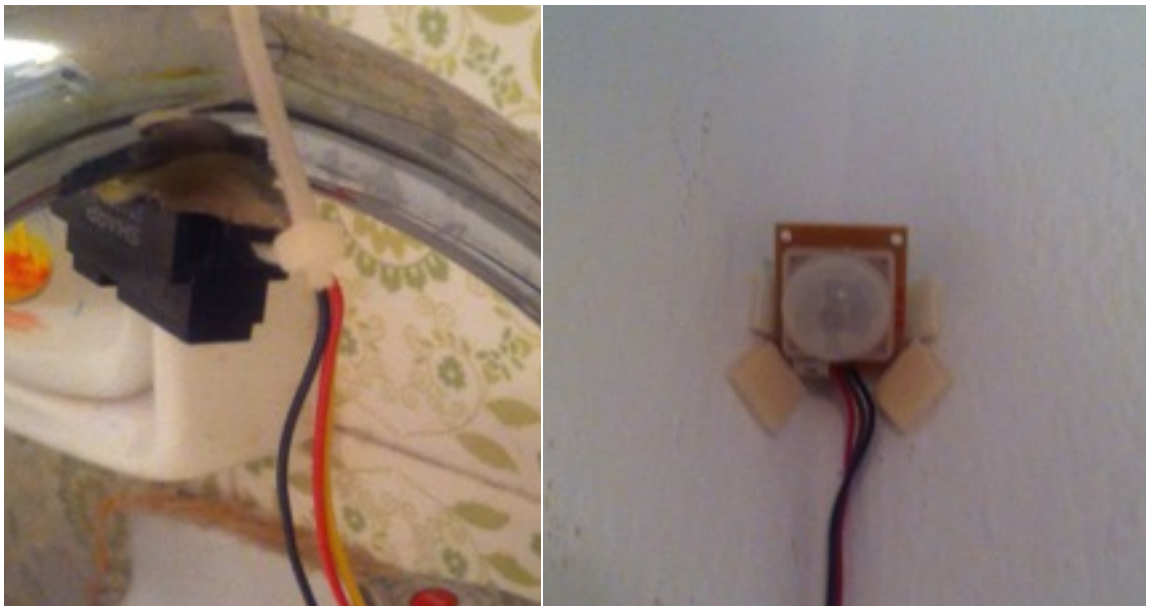
Επέκταση Αισθητήρων



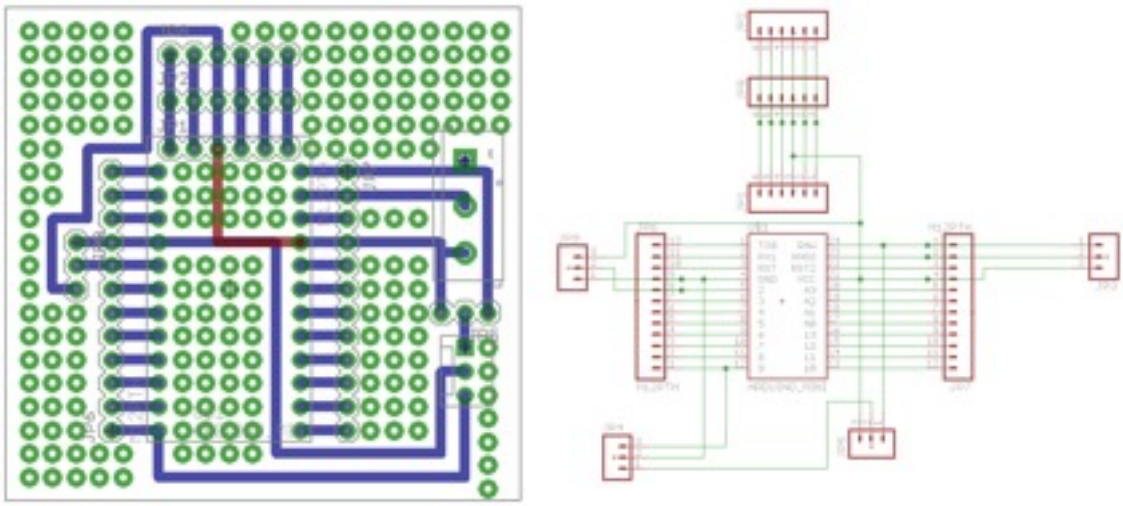
Πλακετα και Κύκλωμα Επέκτασης Αισθητήρων

Έξυπνο Μπάνιο

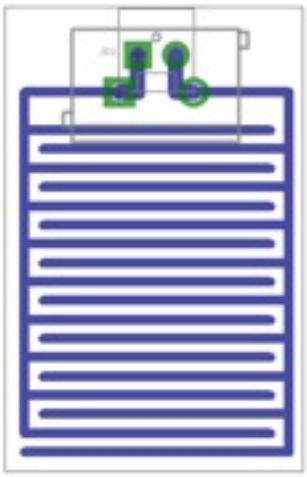
Αυτή η συσκευή παρέχει τον έλεγχο μιας βαλβίδας νερού μέσα από το ασύρματο δίκτυο αισθητήρων. Επιτρέπει τον έλεγχο της παροχής νερού για να αποφευχθούν οι πλημμύρες ή να εντοπισθούν πιθανές διαρροές που μπορεί να έχουν ως αποτέλεσμα την άσκοπη σπατάλη νερού. Η πλακέτα που αναπτύχθηκε λειτουργεί σε 3 επίπεδα. Τοπικά μπορεί να ελέγξει μία βαλβίδα με σκοπό την παροχή νερού από μια βρύση στο μπάνιο. Σε δεύτερο επίπεδο, ανιχνεύει τις πλημμύρες με ένα κύκλωμα το οποίο προκαλεί βραχυκύκλωμα εάν βυθιστεί σε νερό με ύψος περισσότερο από 2 χιλιοστά. Όταν γίνει αυτό, η πλημμύρα ανιχνεύεται και η παροχή νερού σταματά ώστε να αποτραπούν ατυχήματα με τον ηλεκτρικό εξοπλισμό του κτιρίου και πιθανή ζημιά στα έπιπλα ή το ίδιο το κτίριο. Σε τρίτο επίπεδο, η συσκευή ανιχνεύει την κίνηση στο μπάνιο, και ελέγχει αυτόματα τον φωτισμό του δωματίου όταν υπάρχει κάποιο άτομο στον χώρο.



Αισθητήρες Κυκλώματος Έξυπνου Μπάνιου



Πλακέτα και Κύκλωμα Ελέγχου Βρύσης και Φωτισμού



Κύκλωμα Ελέγχου Διαρροής/Πλημμύρας

Για να ανιχνεύσουμε την παρουσία χεριών κάτω από τη βρύση χρησιμοποιήθηκε ένας αισθητήρας απόστασης υπερύθρων (Sharp GP2Y0A02). Ο αισθητήρας μετρά την απόσταση κάθε εμποδίου μέσα σε μία κλίμακα 5-120 εκατοστά. Η απόσταση μετράται μέσω της τάσης μεταβολής στα δύο άκρα του αισθητήρα, χρησιμοποιώντας τις αναλογικές εισόδους του Arduino. Το επίπεδο της τάσης στην αναλογική είσοδο μετράται συνεχώς με περίοδο μερικών χιλιοστών του δευτερολέπτου και οι τιμές τάσης μετατρέπονται στις αντίστοιχες τιμές απόστασης που μετρούνται σε εκατοστά.

Παρόλο που χρησιμοποιούμε αισθητήρα απόστασης, ο υπολογισμός της ακριβής απόστασης δεν είναι τόσο σημαντικός, καθώς αυτό που μας ενδιαφέρει είναι η ανίχνευση των τυχόν εμποδίων (π.χ. χέρια) κάτω από τη βρύση, και όχι η ακριβής απόσταση. Το μόνο που μας ενδιαφέρει είναι εάν η απόσταση που μετράται από τον αισθητήρα είναι λιγότερη από 35 εκατοστά, η οποία είναι η απόσταση μεταξύ του αισθητήρα και του νιπτήρα, και εφόσον αυτό ισχύει, η βαλβίδα του νερού ενεργοποιείται, ανοίγοντας την βρύση.

Τελικά Συμπεράσματα & Παρατηρήσεις

Το Internet of Things είναι μια τεχνολογική επανάσταση που αντιπροσωπεύει το μέλλον της πληροφορικής και των επικοινωνιών, ενώ η ανάπτυξή της υποστηρίζεται από τεχνολογική καινοτομία σε μια σειρά από σημαντικούς τομείς, και ιδιαίτερα στα ασύρματα δίκτυα αισθητήρων. Η σύνδεση ασυρμάτων δικτύων αισθητήρων με το Διαδίκτυο δημιουργεί ατελείωτες ευκαιρίες για εφαρμογές και υπηρεσίες.

Σήμερα, πολλοί ερευνητές εργάζονται για το σχεδιασμό, την ανάπτυξη και την αξιολόγηση των νέων πρωτοκόλλων, ενσωματωμένων IP stacks και λειτουργικών συστημάτων για ασύρματα δίκτυα αισθητήρων. Πολλά έργα αποσκοπούν στην διασύνδεση των ασύρματων δικτύων αισθητήρων με το Διαδίκτυο και τη θέσπιση περιβάλλοντων προγραμματισμού για την ανάπτυξη εφαρμογών.

Στην παρούσα εργασία παρουσιάστηκε η ανάπτυξη λογισμικού που επιτρέπει σε διαφορετικές πλατφόρμες WSN να διασυνδεθούν μεταξύ τους αλλά και με το Διαδίκτυο, η ανάπτυξη επεκτάσεων υλικού για ηλεκτρικές και ηλεκτρονικές συσκευές και η ενσωμάτωση της λειτουργίας τους με το διαδίκτυο.

Η εργασία αυτή δείχνει επίσης πώς IoT συσκευές μπορούν να εφαρμοστούν σε πραγματικά σενάρια και εγκαταστάσεις με τη χρήση προσφάτων τεχνολογιών και ανοικτά πρότυπα. Συγκεκριμένα, πώς μπορούν να επεκταθούν οι δυνατότητες του Arduino με την προσθήκη ενός συστήματος ασύρματης επικοινωνίας 802.15.4 (XBee), προκειμένου να συνδεθεί σε ένα ετερογενές δίκτυο αισθητήρων με την χρήση μιας στοίβα δικτύου και χρησιμοποιηθεί ως κόμβος του Internet of Things.

Σημαντικό ρόλο στην ανάπτυξη των αυτών των συσκευών έπαιξαν καινούργιες μεθοδολογίες και τεχνολογίες, όπως το CoAP, πρωτόκολλα REST[16], WebSockets[17], και η κωδικοποίηση JSON[18], που χρησίμευσαν στην ενσωμάτωση των συστημάτων λογισμικού με τους κόμβους των ασύρματων δικτύων αισθητήρων χρησιμοποιώντας μια τυποποιημένη στοίβα επικοινωνίας.

Περιορισμοί Hardware

Στην προσπάθεια να μειωθεί το κόστος της κάθε συσκευής δοκιμάστηκε η χρήση μόνο XBee για τον έλεγχο των συσκευών και αισθητήρων, μια επιλογή που αποδείχθηκε άσκοπη και λανθασμένη, καθώς αφαιρείται η ικανότητά μας έχουμε εφαρμογή που να τρέχει στον ίδιο τον κόμβο, και προστίθεται επιπλέον πολυπλοκότητα στους υπόλοιπους κόμβους. Το κέρδος από την επιλογή αυτή ήταν επίσης ασύμαντο περιμέναμε (από άποψη κόστους hardware), σε σύγκριση με τη χρήση ενός Arduino, όπου το επιπλέον κόστος είναι ελάχιστο (περίπου $ 15) για την ελευθερία που προσφέρει, σε συνδυασμό με την δυνατότητα over-the-air προγραμματισμού αλλά και της αποσυμφόρησης του WSN από τα συνεχή μηνύματα του XBee.

Βιβλιογραφία

[1] Arduino website, http://www.arduino.cc/
[2] http://www.ieee802.org/15/pub/TG4.html
[3] Xbee module website, http://www.digi.com/
[4] SunSPOT website, http://www.sunspotworld.com/
[5] TelosB website, http://www.xbow.com
[6] iSense website, http://www.coalesenses.com/
[7] mkSense website, https://github.com/
[8] http://tools.ietf.org/html/draft-ietf-core-coap-18
[9] http://www.zigbee.org/
[10] http://tools.ietf.org/wg/6lowpan/
[11] Wiring website, http://wiring.org.co/
[12] Processing website, http://processing.org/
[13] The TinyOS website, http://www.tinyos.net
[14] Voltage Divider - http://en.wikipedia.org/wiki/Voltage_divider
[15] Uberdust - http://uberdust.cti.gr/
[16] REST - http://en.wikipedia.org/wiki/Representational_state_transfer
[17] WebSockets - http://tools.ietf.org/html/rfc6455
[18] JSON - http://www.json.org/