ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Υλοποίηση Συστημάτων Ενδιαμέσου Λογισμικού για Ασύρματα Δίκτυα Αισθητήρων (Wireless Sensor Networks – WSN)

Ιωάννης Η. Αγγελόπουλος

Επιβλέπων: Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΗΣ 2005

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη και Υλοποίηση Συστημάτων Ενδιαμέσου Λογισμικού για Ασύρματα Δίκτυα Αισθητήρων

(Wireless Sensor Networks – WSN)

Ιωάννης Η. Αγγελόπουλος

A.M.:018200000164

ΕΠΙΒΛΕΠΟΝΤΕΣ:

Ευστάθιος Χατζηευθυμιάδης, Επίκουρος Καθηγητής ΕΚΠΑ

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της πτυχιακής εργασίας είναι η υλοποίηση ενός οδηγού (driver) για εξαγωγή μετρήσεων από ασύρματα δίκτυα αισθητήρων. Ο οδηγός αυτός, επικοινωνεί με το δίκτυο μέσω ενός ενδιάμεσου λογισμικού. Στην παρούσα εργασία, έγινε μελέτη για το ενδιάμεσο λογισμικό TinyDB. Ο απώτερος στόχος του οδηγού είναι να αποκρύψει τις τεχνικές λεπτομέρειες του υποκείμενου δικτύου αισθητήρων, ώστε οι ζητούμενες πληροφορίες να προσπελαύνονται με όσο το δυνατό ομοιόμορφο και εύκολο τρόπο. Ο οδηγός που υλοποιήθηκε δέχεται αιτήσεις που έχουν μια συγκεκριμένη δομή σύμφωνα με τα πρότυπα μιας προκαθορισμένης, βασισμένης σε XML, γλώσσας, της Unified Sensor Language. Οι αιτήσεις αυτές υφίστανται μία σειρά από ελέγχους (έλεγχο εγκυρότητας του αιτήματος, έλεγχο υποστήριξης των επιθυμητών τύπων δεδομένων και σημασιολογικό έλεγχο) και στη συνέχεια εισάγονται στο δίκτυο υπό μορφή επερώτησης μέσω του TinyDB. Ο τύπος του κάθε αιτήματος διαφέρει αναλόγως με το αν περιέχει την επιθυμία για τις μετρήσεις των αισθητήρων εκείνη τη στιγμή, την παρακολούθηση των τιμών των αισθητήρων μέσα σε κάποιο χρονικό διάστημα, την αναμονή κάποιου γεγονότος ή παλιότερες μετρήσεις. Αναλόγως διαμορφώνεται και η απάντηση που επιστρέφεται κάθε φορά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ασύρματα δίκτυα

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: αισθητήρες, ενδιάμεσο λογισμικό, ασύρματα δίκτυα

ABSTRACT

In this thesis, a driver for interfacing a wireless sensor network was implemented. In the context of this thesis, the TinyDB middleware is studied. The responsibility of the driver, termed WSN Driver, is to handle requests that follow a specific XML-based format, according to the Unified Sensor Language (USL). Before a new request is served, a series of tests is performed in order to ensure that such a request is valid. Firstly the request is validated syntactically (i.e., it is checked whether the request syntax complies with the allowed, predefined USL syntax). In the case that syntactic validity is verified, the information carried by the request is extracted and checked for semantic-level errors. Specifically, conflicts and lack of support are considered. For example, the nodes in the underlying network might not be equipped with the sensors that are required for replying to the request in question(e.g., if no node possesses a temperature sensor, a request for temperature data is useless to be injected into the network). According to the type of the request, the processing proceeds differently. In the case of a simple request, a specific query is constructed and therefore injected in the network. Once the responses are received, any requested processing takes place and the response if formed according to the USL response syntax. In the case of a monitor request, a similar query is injected in the network, but the responses are sent back, in regular time intervals, for a given time period defined by a given beginning and ending time. In the case of an event-driven request, an response is returned only if a predefined event takes place.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

1

Εισαγωγή

Η πρόσφατη πρόοδος στην τεχνολογία των μικροηλεκτρομηχανικών συστημάτων (micro-electro-mechanical systems - MEMS), των ασύρματων επικοινωνιών και της ψηφιακής ηλεκτρονικής, επέτρεψαν τη δημιουργία χαμηλού κόστους και χαμηλής κατανάλωσης ισχύος πολυλειτουργικών κόμβων αισθητήρων, που είναι μικροί σε μέγεθος και μπορούν να επικοινωνούν σε κοντινές μεταξύ τους αποστάσεις βασιζόμενοι σε μια ad-hoc συμπεριφορά. Αυτοί οι κόμβοι συνήθως χαρακτηρίζονται από περιορισμένες υπολογιστικές και επικοινωνιακές δυνατότητες. Ένας μεγάλος αριθμός από τέτοιους αισθητήρες που επικοινωνούν μεταξύ τους ασύρματα, συνιστούν ένα ασύρματο δίκτυο αισθητήρων (Wireless Sensor Network - WSN). Η μεγαλύτερη ανάπτυξη των δικτύων αυτών αφορά σε εξειδικευμένες εφαρμογές, όπως η παρακολούθηση του περιβάλλοντος, habitat monitoring, παρακολούθηση κυκλοφορίας, στρατιωτικά συστήματα, supply chain management, συστήματα υγείας και άλλα.

Ο προγραμματισμός των ασύρματων δικτύων αισθητήρων, γίνεται κατά βάση με τη χρήση ενδιάμεσου λογισμικού (Middleware), που συνήθως συνεργάζεται με συγκεκριμένες δομές υλικού (Hardware) για WSN. Αυτού του

είδους το ενδιάμεσο λογισμικό, εν αντιθέσει με τη συνήθη ερμηνεία του όρου, δεν αποσκοπεί στο να παρέχει μια γενική και υψηλού επιπέδου προγραμματιστική πλατφόρμα για ασύρματα δίκτυα αισθητήρων, αλλά ένα βασικό σύνολο εργαλείων και βιβλιοθηκών για τον χαμηλού επιπέδου χειρισμό των κόμβων αισθητήρων. Ο προγραμματισμός και η ρύθμιση του δικτύου αισθητήρων είναι εργασία δύσκολη και ευάλωτη σε λάθη, αφού αυτός που αναπτύσσει την εφαρμογή πρέπει στις περισσότερες περιπτώσεις να προγραμματίσει ξεχωριστούς κόμβους αισθητήρων, χρησιμοποιώντας χαμηλού επιπέδου προγραμματιστικές γλώσσες.

Όσο τα ασύρματα δίκτυα αισθητήρων χρησιμοποιούνται σε συγκεκριμένα πεδία εφαρμογών, αυτή η προγραμματιστική προσέγγιση είναι αποδεκτή. Παρόλα αυτά προκειμένου να υιοθετηθούν οι υπάρχουσες WSN υποδομές από άλλα υπολογιστικά παραδείγματα, κάποιες βελτιώσεις στο υπάρχον ενδιάμεσο λογισμικό είναι απαραίτητες. Στις μέρες μας, ένα από τα πιο υποσχόμενα υπολογιστικά παραδείγματα, είναι ο «διάχυτος υπολογισμός» (pervasive computing) που περιλαμβάνει έξυπνα δικτυακά περιβάλλοντα, στα οποία οι χρήστες μπορούν αδιάκοπα να χρησιμοποιούν προδραστικές (proactive) υπηρεσίες περιεχομένου. Προκειμένου ο «διάχυτος υπολογισμός» να έρθει ένα βήμα πιο κοντά, είναι απαραίτητο οι «διάχυτες» εφαρμογές να μπορούν αδιάκοπα να έχουν πρόσβαση σε πληροφορίες πλαισίου, οι οποίες πηγάζουν από δομές αισθητήρων που βρίσκονται σε χαμηλότερο επίπεδο. Επιπλέον είναι πολύ σημαντικό οι εφαρμογές να μην χρειάζεται να γνωρίζουν τις διαφορετικές τεχνολογίες ενδιάμεσου λογισμικού που χρησιμοποιούνται, καθώς και τα χαρακτηριστικά τους, Επιπλέον, προκειμένου να γίνει ο προγραμματισμός σε αυτά τα περιβάλλοντα όσο περισσότερο φιλικός προς το χρήστη γίνεται, χρειάζεται να χρησιμοποιηθούν κάποιες προγραμματιστικές αφαιρέσεις (abstractions) που απλοποιούν τη προγραμματιστική διαδικασία ενώ πρέπει παράλληλα το ενδιάμεσο λογισμικό υποστηρίζει αυτές τις αφαιρέσεις. Με άλλα λόγια οι πρόσφατες και οι μελλοντικές διάχυτες εφαρμογές προσανατολίζονται

περισσότερο προς την ίδια τη πληροφορία και όχι τις τεχνολογίες και το υλικό που χρησιμοποιείται.

1.1 Στόχος

Από τα παραπάνω γίνεται εύκολα αντιληπτό πως ένα περιβάλλον ανάπτυξης προγραμματιστικών εφαρμογών προκειμένου να ικανοποιεί διαφορετικούς χρήστες με διαφορετικές ανάγκες (χαρακτηριστικό ενός «διάχυτου» περιβάλλοντος) πρέπει να χτιστεί βάσει των ακόλουθων αρχών:

1. Πρέπει να μην αποκαλύπτει διαφοροποιήσεις μεταξύ διαφορετικών δομών δικτύων αισθητήρων, αναφορικά με τις τεχνολογίες των αισθητήρων, του δικτύου και του middleware που χρησιμοποιούνται. Αυτό είναι απαραίτητο καθώς θεωρείται σχετικά απίθανο μια συγκεκριμένη δομή να ικανοποιήσει το μεγαλύτερο μέρος του τομέα των δικτύων αισθητήρων για τα επόμενα χρόνια και να γίνει πρότυπο.
2. Πρέπει να παρέχει ένα κατάλληλο προγραμματιστικό μοντέλο (ένα φιλικό προς το χρήστη σύνολο εντολών, ευκολίες επεξεργασίας πληροφορίας, καθώς και ρουτίνες επαναπροσδιορισμού και ελέγχου) που θα προωθήσει ακόμα περισσότερο την ανάπτυξη διάχυτων εφαρμογών του πραγματικού κόσμου. Πιστεύεται πως αυτές οι εφαρμογές θα αξιοποιήσουν πλήρως τις δυνατότητες των WSN.
3. Τέλος πρέπει να λαμβάνει υπόψη τις βασικές αρχές των ασύρματων δικτύων αισθητήρων (εξοικονόμηση ενέργειας, επεκτασιμότητα και ευρωστία).

Το σχήμα που αναπτύσσουμε βασίζεται ακριβώς σε αυτές τις αρχές καθώς και στο ODBC/JDBC πλαίσιο που χρησιμοποιείται ευρέως στο τομέα των βάσεων δεδομένων. Το δίκτυο αισθητήρων σαν σύνολο, αντιμετωπίζεται σαν μια πηγή πληροφοριών, ακριβώς όπως μια βάση δεδομένων στο κόσμο του ODBC/JDBC.

1.2 Οργάνωση του τόμου

Στην δεύτερη ενότητα του τόμου αυτού, παρατίθενται πληροφορίες σχετικά με τις εφαρμογές των ασύρματων δικτύων αισθητήρων, δίνοντας έτσι μια εικόνα των μελλοντικών στόχων της επιστημονικής κοινότητας, αναφορικά με την νέα αυτή τεχνολογία που αναμένεται να χρησιμοποιηθεί ευρέως στα επόμενα χρόνια. Στη τρίτη ενότητα γίνεται αναφορά στους κυριότερους παράγοντες που αφορούν στην σχεδίαση ενός ασύρματου δικτύου αισθητήρων, ενώ στην τέταρτη περιγράφεται η γενική αρχιτεκτονική δικτύων αυτών και γίνεται μια αναφορά στα υπάρχοντα συστήματα ενδιάμεσου λογισμικού. Η πέμπτη ενότητα αφορά την πλατφόρμα που αναπτύσσουμε, μέρος της οποίας αποτελεί και ο WSN Driver, ή υλοποίηση του οποίου είναι το αντικείμενο της τρέχουσας πτυχιακής εργασίας και περιγράφεται αναλυτικά στην έκτη ενότητα. Όλες οι ενότητες του τόμου συνοδεύονται από αναλυτικά διαγράμματα και πίνακες παραδειγμάτων, που αποσκοπούν στην καλύτερη και σαφέστερη κατανόηση του περιεχομένου τους από τον αναγνώστη. Στην έβδομη, όγδοη και ένατη ενότητα περιλαμβάνονται χρήσιμα παραρτήματα με που αφορούν στις λεπτομέρειες της υλοποίησης, τη περιγραφή του TinyDB (το οποίο θα συζητηθεί παρακάτω) καθώς και την παρουσίαση σειράς ελέγχων του συστήματος, αντίστοιχα.

2

Εφαρμογές των Ασύρματων Δικτύων Αισθητήρων

Τα ασύρματα δίκτυα αισθητήρων έχουν πολλές εφαρμογές στην καθημερινή μας ζωή και αναμένεται να αποτελέσουν μια από τις σημαντικότερες νέες τεχνολογίες του 21^{ου} αιώνα.

2.1 Στρατιωτικές Εφαρμογές

Τα WSN μπορούν να αποτελέσουν ένα αναπόσπαστο κομμάτι των ελέγχων, των επικοινωνιών, της παρακολούθησης, της νοημοσύνης και της στόχευσης για μια στρατιωτική επιχείρηση. Η πολύ γρήγορη ανάπτυξη, αυτοοργάνωση, και ανεκτικότητα σε λάθη κάνουν τα WSNs μια πολλά υποσχόμενη λύση για τις στρατιωτικές επιχειρήσεις του μέλλοντος. Αφού τέτοια δίκτυα βασίζονται στον πυκνό διασκορπισμό χαμηλού κόστους κόμβων, η καταστροφή ορισμένων από αυτούς από εχθρικές ενέργειες δεν επηρεάζει την στρατιωτική επιχείρηση όσο η καταστροφή ενός παραδοσιακού αισθητήρα, και κάνει τέτοια δίκτυα πιο εύχρηστα σε τέτοιες επιχειρήσεις. Μερικές από τις στρατιωτικές εφαρμογές αυτών των δικτύων, είναι η παρακολούθηση συμμαχικών δυνάμεων, εξοπλισμών και πυρομαχικών, η παρακολούθηση του πεδίου μάχης και ο συμπερασμός εχθρικών ενεργειών, η ανίχνευση στόχων, η καταγραφή ζημιών και τέλος η ανίχνευση πυρηνικής ή βιολογικής και χημικής επίθεσης.

Παρακολούθηση συμμαχικών δυνάμεων, εξοπλισμών και πυρομαχικών

Οι αρχηγοί και αυτοί που δίνουν εντολές, μπορούν κάθε στιγμή να παρακολουθούν τις κινήσεις φιλικών προς αυτούς στρατευμάτων, τη κατάσταση και την διαθεσιμότητα του εξοπλισμού και των πυρομαχικών σε ένα πεδίο μάχης, με τη χρήση δικτύων αισθητήρων. Κάθε στρατός, όχημα, εξοπλισμός καθώς και τα κρίσιμα πυρομαχικά μπορεί να φέρει μικρούς αισθητήρες που δίνουν αναφορά για την κατάστασή του. Αυτές οι αναφορές μαζεύονται σε κεντρικούς κόμβους και στέλνονται στους αρχηγούς του εκάστοτε στρατού. Τα δεδομένα μπορούν επίσης να προωθηθούν σε ανώτερα επίπεδα της στρατιωτικής ιεραρχίας ενώ συγκρίνονται με δεδομένα από άλλες μονάδες.

Παρακολούθηση του πεδίου μάχης

Τα κρίσιμα πεδία, μονοπάτια και στενά, μπορούν πολύ γρήγορα να καλυφθούν από αισθητήρες, ώστε να παρακολουθούνται στενά οι ενέργειες των αντίπαλων στρατευμάτων. Όσο οι επιχειρήσεις εξελίσσονται και ετοιμάζονται καινούργια στρατηγικά σχέδια, καινούργιοι κόμβοι μπορούν να διασκορπιστούν για να ενισχύσουν τη παρακολούθηση.

Έλεγχος επίθεσης από αντίπαλα στρατεύματα

Τα δίκτυα αισθητήρων μπορούν να χρησιμοποιηθούν σε κρίσιμα πεδία με αποτέλεσμα να μπορούν να εξαχθούν πολύ χρήσιμες πληροφορίες σχετικά με το που, πως και πότε επιτίθενται τα αντίπαλα στρατεύματα, ώστε να μην υπάρξει περίπτωση αιφνιδιασμού.

Εύρεση στόχου

Με ειδική επεξεργασία των πληροφοριών που συλλέγονται από αυτά τα δίκτυα, μπορούν να εξαχθούν συμπεράσματα για την εύρεση του σωστού στόχου.

Καταγραφή ζημιών

Πριν και μετά από μια επίθεση, μπορούν να χρησιμοποιηθούν τα δίκτυα αισθητήρων για να μας δώσουν ποιοτικά και ποσοτικά συμπεράσματα αναφορικά με το μέγεθος της ζημιάς.

Διαπίστωση πυρηνικής ή βιολογικής ή χημικής επίθεσης

Σε περίπτωση χημικού ή βιολογικού πολέμου, είναι πάρα πολύ σημαντική η έγκαιρη ενημέρωση από κάποιον ή κάτι που βρίσκεται στο σημείο μηδέν. Η χρήση δικτύων αισθητήρων θα ήταν πολύ σημαντική για την γρήγορη ενημέρωση σχετικά με μια τέτοια επίθεση, καθώς και για τα στάδια της μόλυνσης σε διάφορες περιοχές και θα συνέβαλλε στην μείωση των θυμάτων τέτοιων

επιθέσεων δραστικά. Επιπλέον δεν θα χρειαζόταν η αποστολή ειδικής ομάδας σε σημεία μολυσμένα από ραδιενέργεια για να κάνουν μετρήσεις, αφού αυτές θα μπορούν να γίνονται αυτόματα.

2.2 Περιβαλλοντολογικές Εφαρμογές

Μερικές περιβαλλοντολογικές εφαρμογές των ασύρματων δικτύων αισθητήρων περιλαμβάνουν την καταγραφή των μετακινήσεων πουλιών, μικρών ζώων και εντόμων, την παρακολούθηση περιβαλλοντολογικών συνθηκών που επηρεάζουν τη χλωρίδα και την πανίδα, τους υδροφόρους ορίζοντες, την ανίχνευση βιολογικών και χημικών στοιχείων, την ανίχνευση πυρκαγιών σε δασικές εκτάσεις, μετεωρολογικές ή γεωφυσικές έρευνες, την ανίχνευση πλημμύρων, τη χαρτογράφηση της βιοποικιλότητας του περιβάλλοντος καθώς και τη μελέτη της μόλυνσης του περιβάλλοντος.

Ανίχνευση πυρκαγιών

Από τη στιγμή που οι κόμβοι του δικτύου μπορούν να διασκορπιστούν τυχαία ή με βάση κάποια στρατηγική και με μεγάλη πυκνότητα μέσα σε κάποιο δάσος, οι αισθητήρες μπορούν να αποκαλύψουν την ακριβή εστία της φωτιάς πριν η διάδοσή της είναι ανεξέλεγκτη. Εκατομμύρια αισθητήρες-κόμβοι μπορούν να χρησιμοποιηθούν μέσω ραδιοσυχνοτήτων / οπτικών συστημάτων. Επίσης, μπορούν να εξοπλιστούν με αποτελεσματικές μεθόδους για την εξοικονόμηση ενέργειας, όπως ηλιακά κελιά, επειδή οι αισθητήρες μπορεί να παραμείνουν χωρίς επιτήρηση για μήνες ή ακόμα και για χρόνια. Οι αισθητήρες θα αλληλεπιδρούν μεταξύ τους προκειμένου να πραγματοποιούν κατανεμημένες λειτουργίες αισθήσεων και προκειμένου να παρακάμψουν εμπόδια, όπως δέντρα και πέτρες που θα εμπόδιζαν την επικοινωνία μεταξύ τους.

Χαρτογράφηση της βιοποικιλότητας του περιβάλλοντος

Η χαρτογράφηση της βιοποικιλότητας του περιβάλλοντος προϋποθέτει ανεπτυγμένες προσεγγίσεις προκειμένου να συνδυάσουν τις πληροφορίες μεταξύ χωρικής και χρονικής κλίμακας. Η πρόοδος της τεχνολογίας στον τομέα της απομακρυσμένης αίσθησης και της αυτοματοποιημένης συλλογής δεδομένων έχει δώσει τη δυνατότητα για μεγαλύτερη χωρική και χρονική ανάλυση, με ένα γεωμετρικά φθίνον κόστος ανά κόμβο. Παράλληλα με αυτή την τεχνολογική πρόοδο, οι αισθητήρες έχουν τη δυνατότητα να συνδέονται με το internet, το οποίο επιτρέπει σε απομακρυσμένους χρήστες να ελέγχουν, να παρακολουθούν και να παρατηρούν τη βιοποικιλότητα του περιβάλλοντος.

Παρόλο που οι δορυφορικοί και αεροπορικοί αισθητήρες είναι χρήσιμοι στην παρατήρηση μεγάλης κλίμακας βιοποικιλότητας, π.χ. χωρική πολυπλοκότητα κάποιων κυρίαρχων ειδών φυτών, δεν είναι αρκετά ικανοποιητικοί στην παρατήρηση μικρής κλίμακας βιοποικιλότητας που ουσιαστικά αποτελεί και το βασικό κομμάτι ενός οικοσυστήματος. Αυτό έχει ως αποτέλεσμα να υπάρχει ανάγκη για χρήση ασύρματου δικτύου αισθητήρων στην επιφάνεια του εδάφους για την πιο αποτελεσματική παρατήρηση της βιοποικιλότητας.

Ανίχνευση πλημμύρων

Ένα χαρακτηριστικό παράδειγμα ανίχνευσης πλημμύρων είναι το σύστημα ALERT [10] εφαρμόστηκε στις Ηνωμένες Πολιτείες της Αμερικής. Μερικοί τύποι αισθητήρων που χρησιμοποιήθηκαν στο σύστημα ALERT [10] είναι αισθητήρες βροχής, ύδατος και κλίματος. Αυτοί οι αισθητήρες παρέχουν πληροφορίες σε μία κεντρική βάση δεδομένων με έναν προδιαγεγραμμένο τρόπο. Μερικά ερευνητικά προγράμματα, όπως το COUGAR [11], εξετάζουν κατανεμημένες προσεγγίσεις αναφορικά με την αλληλεπίδραση με τους αισθητήρες σε ένα πεδίο αισθητήρων προκειμένου να παρέχουν άμεσες αλλά μακροχρόνιες επερωτήσεις.

Προβλέψεις χρήσιμες για τη γεωργία

Μερικά από τα πλεονεκτήματα των δικτύων αισθητήρων είναι η παρακολούθηση των επιπέδων του πόσιμου νερού, της διάβρωσης του εδάφους και της μόλυνσης του αέρα σε πραγματικό χρόνο.

2.3 Εφαρμογές στον κλάδο Υγείας

Μερικές από τις εφαρμογές των δικτύων αισθητήρων στο κλάδο της υγείας είναι η παροχή διεπαφών προς τους ανάπηρους, η συνεχής παρακολούθηση ασθενών, η διάγνωση, η σύσταση συγκεκριμένων φαρμάκων στα νοσοκομεία, η παρακολούθηση των κινήσεων και των εσωτερικών διεργασιών εντόμων και άλλων μικρών ζώων, η τηλεπαρακολούθηση δεδομένων της ανθρώπινης φυσιολογίας καθώς και ο εντοπισμός και η παρακολούθηση γιατρών και ασθενών σε ένα νοσοκομείο.

Τηλεπερακολούθηση των δεδομένων της ανθρώπινης φυσιολογίας

Τα δεδομένα της φυσιολογίας που συλλέγονται από τους αισθητήρες μπορούν να αποθηκεύονται για ένα μεγάλο χρονικό διάστημα και να χρησιμοποιούνται για λόγους ιατρικής έρευνας. Οι αισθητήρες μπορούν επίσης να ανιχνεύουν τη συμπεριφορά ηλικιωμένων ανθρώπων όπως για παράδειγμα κάποιο πέσιμο. Αυτοί οι μικροί κόμβοι δίνουν μια μεγάλη ελευθερία κινήσεων στους εξεταζόμενους ενώ παράλληλα δίνουν στους γιατρούς τη δυνατότητα να ανιχνεύουν προδιαγεγραμμένα για το άτομο συμπτώματα αρκετά νωρίς. Επιπλέον εξασφαλίζουν μια υψηλότερης ποιότητας ζωή για κάποιον εξεταζόμενο, σε σύγκριση με την αντίστοιχη ενός κέντρου νοσηλείας.

Ανίχνευση και παρακολούθηση των κινήσεων των γιατρών και των ασθενών στο νοσοκομείο

Κάθε ασθενής έχει προσαρτημένους επάνω του μικρούς και ελαφρούς αισθητήρες. Κάθε αισθητήρας έχει το δικό του ξεχωριστό ρόλο. Για παράδειγμα ένας αισθητήρας μπορεί να παρακολουθεί τους χτύπους της καρδιάς, ενώ κάποιος άλλος ελέγχει την πίεση του αίματος. Οι γιατροί μπορούν επίσης να μεταφέρουν τέτοιους αισθητήρες, κάτι που επιτρέπει σε άλλους γιατρούς να τους εντοπίζουν μέσα στο νοσοκομείο.

Σύσταση συγκεκριμένων φαρμάκων στα νοσοκομεία

Αν ένας ασθενής φέρει κάποιους αισθητήρες, τότε είναι εύκολο να ανιχνευθούν οι αλλεργίες του και αντίστοιχα οι κατάλληλες φαρμακευτικές αγωγές για την αντιμετώπιση κάποιου προβλήματος υγείας, αποφεύγοντας έτσι την χορήγηση φαρμάκων στα οποία τελικά ο ασθενής είναι αλλεργικός.

2.4 Οικιακές Εφαρμογές

Οικιακός αυτοματισμός

Καθώς η τεχνολογία προοδεύει, έξυπνοι αισθητήρες μπορούν να τοποθετηθούν πάνω σε οικιακές συσκευές, όπως ηλεκτρικές σκούπες, φούρνοι μικροκυμάτων, ψυγεία κ.α. Αυτοί οι αισθητήρες μπορούν να επικοινωνούν μεταξύ τους, καθώς και με εξωτερικά δίκτυα μέσω Internet η δορυφόρου. Έτσι ο χρήστης μπορεί πολύ εύκολα να χρησιμοποιεί και να ελέγχει αυτές τις συσκευές όντας απομακρυσμένος από αυτές.

Έξυπνο περιβάλλον

Ο σχεδιασμός του έξυπνου περιβάλλοντος μπορεί να έχει δύο διαφορετικές οπτικές. Την ανθρωποκεντρική και την τεχνοκεκτρική. Στη περίπτωση της ανθρωποκεντρικής, ένα έξυπνο περιβάλλον πρέπει να

προσαρμόζεται άριστα στις ανάγκες του χρήστη αναφορικά με τις δυνατότητες εισόδου εξόδου. Στη περίπτωση της τεχνοκεντρικής, πρέπει να αναπτυχθούν καινούργιες τεχνολογίες υλικού, δικτυακές λύσεις καθώς και υπηρεσίες ενδιάμεσου λογισμικού. Ένα σενάριο του πως οι αισθητήρες μπορούν να χρησιμοποιηθούν για την ανάπτυξη ενός έξυπνου περιβάλλοντος είναι το εξής: Οι αισθητήρες εφάπτονται σε έπιπλα και συσκευές και επικοινωνούν μεταξύ τους, καθώς και με έναν server του δωματίου. Κάθε server δωματίου μπορεί να επικοινωνεί με άλλους παρόμοιους servers και να μαθαίνει τι υπηρεσίες προσφέρουν (π.χ. scanning, printing και faxing).Άλλες Εμπορικές Εφαρμογές

Κάποιες επιπλέον εμπορικές εφαρμογές είναι η ανίχνευση αδυναμιών σε υλικά, η δημιουργία εικονικών πληκτρολογίων, η κατηγοριοποίηση αγαθών, ο έλεγχος ποιότητας αγαθών, η κατασκευή έξυπνων χώρων σε γραφεία, ο έλεγχος του περιβάλλοντος σε κτίρια γραφείων, ο έλεγχος και η καθοδήγηση ρομπότ σε αυτοματοποιημένα κατασκευαστικά περιβάλλοντα, η κατασκευή διαδραστικών παιχνιδιών, η κατασκευή διαδραστικών μουσείων, ο έλεγχος διεργασιών στη βιομηχανία και η αυτοματοποίηση τους, η παρακολούθηση κατεστραμμένων περιοχών, η κατασκευή έξυπνων δομών με ενσωματωμένους αισθητήρες, η διάγνωση μηχανημάτων, η μετακίνηση, η ανίχνευση κλεμμένων αυτοκινήτων, ο εντοπισμός κινούμενων οχημάτων, καθώς και άλλες πολλές.

Έλεγχος περιβάλλοντος σε κτίρια γραφείων

Ο κλιματισμός και η θέρμανση στα περισσότερα κτίρια ελέγχεται από κάποιο κεντρικό σημείο. Συνεπώς η θερμοκρασία μέσα σε ένα δωμάτιο μπορεί να διαφέρει μερικούς βαθμούς. Για παράδειγμα αν υπάρχει μόνο ένα σώμα θέρμανσης και η κατανομή της θερμότητας δεν γίνεται ισομερώς, η μια πλευρά θα είναι πιο ζεστή από την άλλη. Ένα κατανεμημένο ασύρματο δίκτυο αισθητήρων θα μπορούσε να χρησιμοποιηθεί για να ελέγχει τη ροή του αέρα και της θερμοκρασίας σε διαφορετικά μέρη του δωματίου.

Κατασκευή διαδραστικών μουσείων

Μελλοντικά τα παιδιά θα μπορούν να αλληλεπιδρούν με αντικείμενα σε μουσεία προκειμένου να μάθουν περισσότερα για αυτά. Αυτά τα αντικείμενα θα μπορούν να ανταποκρίνονται στο άγγιγμα τους και στο λόγο τους. Επίσης τα παιδιά θα μπορούν να συμμετέχουν σε πραγματικού χρόνου πειράματα που δικαιολογούν την αιτία και το αποτέλεσμα του πειράματος, το οποίο θα τους διδάξει πολλά για την επιστήμη.

Ανίχνευση και παρακολούθηση κλοπής αυτοκινήτων

Οι αισθητήρες που εφάπτονται σε ένα αυτοκίνητο μπορούν να προσδιορίσουν την ακριβή του θέση κάθε στιγμή. Έτσι αν το αυτοκίνητο κλαπεί, θα είναι εύκολο για την αστυνομία να εξακριβώσει γρήγορα και αποτελεσματικά που το έχουν μεταφέρει οι κλέφτες.

Κατηγοριοποίηση και έλεγχος λίστας αντικειμένων

Έστω μια αποθήκη στην οποία σε κάθε αντικείμενο εφάπτεται και ένας αισθητήρας. ¨Ένας τελικός χρήστης μπορεί πολύ εύκολα να μάθει ποιά αντικείμενα και πόσα περιέχονται στην αποθήκη αυτή. Αν θέλει να εισάγει μια νέα λίστα από τέτοια αντικείμενα, αρκεί να τους προσάψει από ένα αισθητήρα.

3

Σημαντικοί παράγοντες που αφορούν στη σχεδίαση ενός WSN

Η σχεδίαση ενός ασύρματου δικτύου αισθητήρων επηρεάζεται από πάρα πολλούς παράγοντες, μερικοί εκ των οποίων είναι η ανεκτικότητα σε λάθη, η επεκτασιμότητα, το κόστος παραγωγής των κόμβων, το περιβάλλον στο οποίο οι κόμβοι θα λειτουργούν, η τοπολογία του δικτύου, οι περιορισμοί του υλικού, τα μέσα μετάδοσης που χρησιμοποιούνται και η κατανάλωση ενέργειας. Όλοι αυτοί παράγοντες είναι πολύ σημαντικοί καθώς σχηματίζουν μια κατεύθυνση βάση της οποίας σχεδιάζεται ένας αλγόριθμος ή ένα πρωτόκολλο για τα δίκτυα αισθητήρων. Παρακάτω περιγράφεται συνοπτικά κάθε ένας από τους προαναφερθέντες παράγοντες

3.1 Ανεκτικότητα σε λάθη

Μερικοί κόμβοι μέσα στο δίκτυο μπορεί να αποτύχουν ή να μπλοκαριστούν λόγω της έλλειψης αποθέματος ενέργειας ή γιατί υπέστησαν κάποια ζημιά. Η αποτυχία ενός κόμβου ή ενός συνόλου κόμβων δεν θα πρέπει να επηρεάζει την συνολική εργασία του δικτύου.

3.2 Επεκτασιμότητα

Ο αριθμός των κόμβων ενός τέτοιου δικτύου αναλόγως την εφαρμογή μπορεί να είναι από μερικές δεκάδες ή εκατοντάδες, μέχρι και μερικές χιλιάδες, ίσως και εκατομμύρια μελλοντικά. Τα σχήματα τα οποία υιοθετούνται πρέπει να μπορούν να ανταποκρίνονται σε τέτοιες τάξεις μεγέθους κόμβων, καθώς και στην υψηλή πυκνότητα των κόμβων ενός τέτοιου δικτύου.

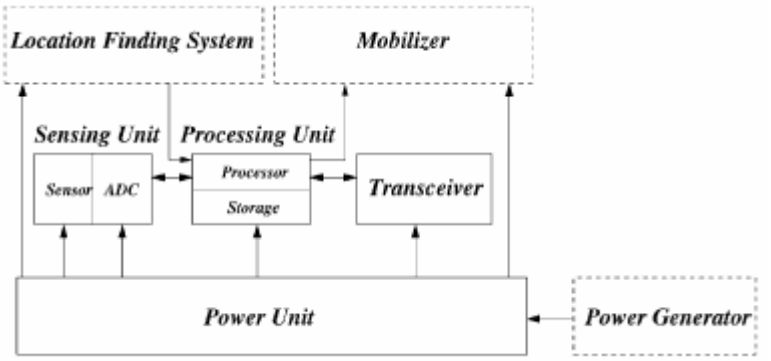
3.3 Κόστος Παραγωγής

Από τη στιγμή που θα χρησιμοποιούνται πάρα πολλοί κόμβοι σε ένα τέτοιο δίκτυο, το κόστος παραγωγής ενός μόνο κόμβού πρέπει να δικαιολογεί το κόστος του συνολικού δικτύου. Αν το τελικό κόστος του δικτύου είναι πιο μεγάλο από τους παραδοσιακούς αισθητήρες, τότε κάνοντας απολογισμό κόστους και απόδοσης μπορεί να μην συμφέρει η παραγωγή των κόμβων. Ένα ραδιοσύστημα Bluetooth [2] κοστίζει λιγότερο από 10$. Το κόστος ενός PicoNode [2] αναμένεται να ανέρχεται σε 1$. Το κόστος ενός κόμβου ενός WSN θα πρέπει να είναι αρκετά χαμηλότερο από 1$.

3.4 Περιορισμοί Υλικού

Ένας κόμβος ενός τέτοιου δικτύου αποτελείται κατά βάση από τα εξής τέσσερα μέρη: Τη μονάδα αίσθησης (αισθητήρας), τη μονάδα επεξεργασίας, τη μονάδα μετάδοσης και τη μονάδα ενέργειας. Αναλόγως την εφαρμογή μπορεί να

διαθέτει και επιπλέον μέρη, όπως ένα σύστημα εύρεσης θέσης σε περίπτωση που απαιτείται εύρεση του γεωγραφικού στίγματος του αισθητήρα με μεγάλη ακρίβεια, μια μονάδα παροχής ενέργειας προς τη μονάδα ενέργειας (π.χ. από ηλιακά κελιά), και έναν κινητήρα στη περίπτωση που απαιτείται μετακίνηση του αισθητήρα. Όλα αυτά θα πρέπει να χωρούν σε μια επιφάνεια μεγέθους ενός σπιρτόκουτου η οποία μελλοντικά μπορεί να μικρύνει ακόμα περισσότερο, στο μέγεθος ενός κυβικού εκατοστού.



Σχήμα 3.1: Τμήματα ενός κόμβου αισθητήρα [2].

3.5 Τοπολογία του Δικτύου

Ένας τεράστιος αριθμός κόμβων στους οποίους δεν υπάρχει πρόσβαση και παραμένουν ως έχουν, ενώ παράλληλα είναι ευάλωτοι σε λάθη, καθιστούν τη διατήρηση της τοπολογίας του δικτύου μια πρόκληση. Εκατοντάδες, ως χιλιάδες κόμβοι διασκορπίζονται σε κάποιο πεδίο και η πυκνότητά τους μπορεί να φτάνει και τους 20 κόμβους ανά κυβικό μέτρο. Ο χειρισμός αυτής της κατάστασης απαιτεί μεγάλη προσοχή ώστε να διατηρείται η τοπολογία. Θα μπορούσαμε να χωρίσουμε τον χειρισμό της τοπολογίας σε τρεις φάσεις.

1. <u>Φάση αρχικής τοποθέτησης</u>

   Σε αυτό το σημείο οι κόμβοι ρίχνονται από κάποιο αεροπλάνο ή από κάποιο πύραυλο, ή από κάποιο καταπέλτη ή τοποθετούνται από κάποιο ρομπότ. Ο τρόπος με τον οποίο θα τοποθετηθούν οι κόμβοι πρέπει να

ελαχιστοποιεί το κόστος εγκατάστασης και τις ανάγκες για προσχεδιασμό, να αυξάνει την ευελιξία της τακτοποίησης των κόμβων και να προωθεί την αυτοοργάνωση και την ανεκτικότητα σε λάθη για το δίκτυο.

2. Φάση επαναπροσδιορισμού τοπολογίας

   Μπορεί να χρειαστεί να οριστούν ξανά η θέση ή οι οδηγίες σχετικά με την εργασία κάποιου κόμβου, εξαιτίας κάποιας δυσλειτουργίας στο δίκτυο ή της έλλειψης ενέργειας.

3. Φάση επανατοποθέτησης κόμβων

   Κάποιοι κόμβοι μπορεί να χρειαστεί να αντικατασταθούν από καινούργιους λόγω δυσλειτουργίας. Η προσθήκη νέων κόμβων θα έχει ως αποτέλεσμα τον επαναπροσδιορισμό της τοπολογίας.

3.6 Περιβάλλον Λειτουργίας

Οι κόμβοι μπορεί να ριχθούν μέσα στο βυθό της θάλασσας, να εφάπτονται σε κάποια ζώα ή σε κάποια κινούμενα οχήματα, μπορεί να τοποθετηθούν σε κάποιο μεγάλο κτίριο ή σε μια βιολογικά μολυσμένη περιοχή, καθώς και σε πολλά άλλα μέρη. Τα παραπάνω μας δίνουν μια μικρή εικόνα σχετικά με τις συνθήκες που θα πρέπει να μπορούν να αντιμετωπίσουν οι αισθητήρες (π.χ. υψηλή πίεση, ακραίες θερμοκρασίες, θόρυβο κ.α.)

3.7 Μέσα Μετάδοσης

Στα δίκτυα πολλών κόμβων η επικοινωνία μεταξύ δύο απομακρυσμένων αισθητήρων γίνεται μέσω κάποιου ενδιάμεσου. Αυτές οι επικοινωνίες μπορούν να πραγματοποιηθούν μέσω RF (Radio Frequency), οπτικών ή υπέρυθρων μέσων.

3.8 Κατανάλωση Ενέργειας

Ένας κόμβος, ως μια μικροηλεκτρονική συσκευή, μπορεί να φέρει μόνο μια περιορισμένη πηγή ενέργειας (<0.5Ah,1.2V). Σε μερικές περιπτώσεις η επανατροφοδοσία δεν είναι δυνατή, οπότε η ζωή του αισθητήρα εξαρτάται από τη διάρκεια ζωής της μπαταρίας. Σε περίπτωση μη λειτουργίας κάποιων κόμβων, μπορεί να χρειαστεί επαναδρομολόγηση κάποιων πακέτων και προώθηση κάποιων αλλαγών στο δίκτυο, που σαν σύνολο συνεπάγονται περισσότερη κατανάλωση ενέργειας. Αυτός είναι και ο λόγος που τα πρωτόκολλα και οι αλγόριθμοι που χρησιμοποιούνται πρέπει να λαμβάνουν υπόψη την κατανάλωση ενέργειας. Η κατανάλωση ενέργειας καταμερίζεται σε τρεις εργασίες. Την αίσθηση, την επεξεργασία που καταναλώνει και το μικρότερο ποσοστό ενέργειας και τέλος την ασύρματη επικοινωνία που καταναλώνει το περισσότερο.

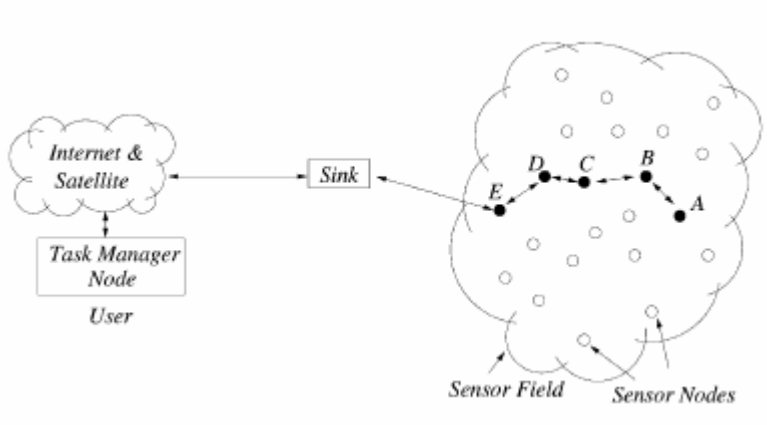
4

Γενική Αρχιτεκτονική των WSN και Ενδιάμεσο Λογισμικό

Οι κόμβοι ενός ασύρματου δικτύου αισθητήρων υπάγονται σε μια γενική αρχιτεκτονική, βάση του τρόπου με τον οποίο έχουν σχεδιαστεί να λειτουργούν και λαμβάνοντας υπόψη τις παραμέτρους που αναφέρθηκαν στην προηγούμενη ενότητα.

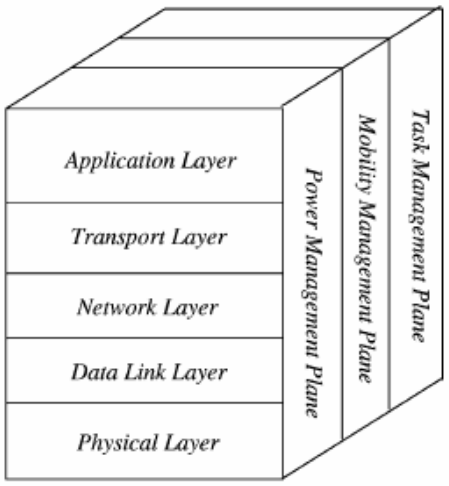
4.1 Στρωμάτωση Αρχιτεκτονικής

Οι αισθητήρες συνήθως διασκορπίζονται σε ένα πεδίο όπως δείχνει η παρακάτω εικόνα:



Σχήμα 4.1: Γενικός τρόπος οργάνωσης των κόμβων των WSN [2].

Κάθε ένας από τους διασκορπισμένους αισθητήρες έχει τη δυνατότητα να συλλέγει δεδομένα, και να τα στέλνει πίσω σε ένα κεντρικό κόμβο (στο σχήμα 4.1 έχει την ονομασία sink) ο οποίος στη συνέχεια τα προωθεί στους τελικούς χρήστες. Το sink μπορεί να επικοινωνεί με το κόμβο Task Manager (ο υπολογιστής του τελικού χρήστη) μέσω του Internet ή μέσω δορυφόρου. Η στρωμάτωση του δικτύου φαίνεται στη παρακάτω εικόνα:



Σχήμα 4.2: Στρωμάτωση των WSN [2].

Αυτή η αρχιτεκτονική καλύπτει τα χαρακτηριστικά της δρομολόγησης και της παροχής ενέργειας, συνδυάζει τα δεδομένα με τα πρωτόκολλα δικτύωσης και προωθεί τη σωστή συνεργασία μεταξύ των κόμβων. Αποτελείται από το επίπεδο εφαρμογής (application layer), το επίπεδο μεταφοράς (transport layer), το επίπεδο δικτύου (network layer), το επίπεδο δεδομένων (data link layer), το φυσικό επίπεδο (physical layer), το επίπεδο διαχείρισης ενέργειας (power management plane), το επίπεδο κινητικότητας (mobility plane) και το επίπεδο διαχείρισης των εργασιών (task management plane). Με βάση τις διαφορετικές εργασίες αίσθησης, διαφορετικοί τύποι λογισμικού μπορεί να χτιστούν στο επίπεδο εφαρμογής. Το επίπεδο μεταφοράς επιτρέπει τη διατήρηση της ροής των δεδομένων αν το απαιτεί η εφαρμογή του δικτύου αισθητήρων. Το επίπεδο δικτύου, φροντίζει για την δρομολόγηση των δεδομένων που παρέχονται από το επίπεδο μεταφοράς. Από τη στιγμή που το περιβάλλον μπορεί να εμπεριέχει θόρυβο και οι αισθητήρες μπορούν να είναι κινητοί, το πρωτόκολλο MAC πρέπει να λαμβάνει υπόψη τους περιορισμούς ενέργειας και να ελαχιστοποιεί τα συγκρούσεις (collisions) μεταξύ των γειτονικών μεταδόσεων. Το φυσικό επίπεδο καθορίζει τις ανάγκες για απλές αλλά αποτελεσματικές τεχνικές διαμορφώσεις καθώς και μετάδοσης και λήψης δεδομένων. Επιπλέον τα επίπεδα διαχείρισης ενέργειας, κινητικότητας και διαχείρισης των εργασιών καθορίζουν τη κατανομή

ενέργειας, κινητικότητας και εργασιών στο δίκτυο αντίστοιχα. Αυτά τα επίπεδα βοηθούν τους αισθητήρες να χωρίσουν την εργασία της αίσθησης και να μειώσουν τη συνολική κατανάλωση ενέργειας.

4.2 Υπάρχοντα Συστήματα Ενδιάμεσου Λογισμικού

Είναι πολύ σημαντική η παρουσίαση των βασικών εργασιών που έχουν πραγματοποιηθεί τα τελευταία χρόνια αναφορικά με την ανάπτυξη ενδιάμεσου λογισμικού για ασύρματα δίκτυα αισθητήρων.

Το TinyDB [3] (το οποίο χρησιμοποιήσαμε), όπως και το Cougar [11] αντιμετωπίζουν το δίκτυο σαν μια κατανεμημένη βάση δεδομένων. Θεωρούν την ύπαρξη ενός εικονικού πίνακα (Sensors) μιας βάσης δεδομένων, η κάθε στήλη του οποίου είναι και ένας συγκεκριμένος υποστηριζόμενος τύπος (π.χ. Θερμοκρασία ή Υγρασία κ.α.). Οι τιμές κάθε αισθητήρα καταγράφονται σε μια γραμμή του πίνακα αυτού. Οι χρήστες επιλέγουν τα δεδομένα που επιθυμούν μέσα από συγκεκριμένες επερωτήσεις, παρόμοιες με αυτές της SQL (για την ακρίβεια είναι ένα υποσύνολό τους με κάποιους περιορισμούς, (π.χ. δεν υποστηρίζεται το λογικό “or”) και κάποιες επεκτάσεις που επιτρέπουν σε μια επερώτηση να είναι περιοδικά επαναλαμβανόμενη). Και το TinyDB και το Cougar [11] χρησιμοποιούν μια αποκεντρωμένη προσέγγιση, όπου ο κάθε κόμβος έχει το δικό του επεξεργαστή επερωτήσεων που επεξεργάζεται και αποστέλλει τα δεδομένα που συλλέγει κατευθείαν στο χρήστη. Παρόλο που το TinyDB θεωρείται μια “de-facto” λύση, η SQL-like φύση των επερωτήσεων που δέχεται δεν είναι πάντοτε ευέλικτη. Επίσης δεν υποστηρίζει τον “on-the-fly” ορισμό συμβάντων (events), αλλά μόνο ορισμένα από πριν συμβάντα τα οποία έχουν προγραμματιστεί στο υλικό του αισθητήρα κατά τη διάρκεια της μεταγλώττισης (compilation).

Το SINA [13] είναι ένα ενδιάμεσο λογισμικό που επιτρέπει σε εφαρμογές των αισθητήρων να πραγματοποιούν επερωτήσεις και εργασίες, να συλλέγουν απαντήσεις και αποτελέσματα και να παρακολουθούν αλλαγές μέσα στο δίκτυο.

Τα βασικά χαρακτηριστικά του περιλαμβάνουν την ιεραρχική ομαδοποίηση των κόμβων των αισθητήρων, την ονοματοδοσία βάση μεταβλητών των κόμβων, καθώς και ένα παράδειγμα οργάνωσης των δεδομένων στους κόμβους. Το SINA [13] χρησιμοποιεί SQL-Like επερωτήσεις καθώς και SQTL (Sensor Query and Tasking Language) διαδικαστικά scripts. Οι SQL-Like επερωτήσεις χρησιμοποιούν τα προαναφερθέντα χαρακτηριστικά για να εκτελέσουν απλές εργασίες επερωτήσεων και παρακολούθησης, ενώ για πιο προχωρημένες εφαρμογές η SQTL παίζει το ρόλο της διεπαφής μεταξύ των αισθητήρων και του SINA [13]. Ένα SQTL μήνυμα που περιέχει ένα script, βρίσκεται μέσα σε ένα XML-Like SQTL Wrapper και προορίζεται ώστε να διαβαστεί και να εκτελεστεί από ένα περιβάλλον εκτέλεσης των αισθητήρων (Sensor Execution Environment - SEE), που τρέχει σε κάθε αισθητήρα. Εν αντιθέση με το TinyDB [3] και το Cougar [11], επειδή παρέχει μια εναλλακτική scripting διεπαφή, θεωρείται πιο ευέλικτη προσέγγιση. Παρόλα αυτά ο ουσιαστικός προγραμματισμός των εργασιών μπορεί να αποδειχθεί αρκετά μεγάλος φόρτος.

Το SensorWare [14] είναι ένα ακόμα σύστημα ενδιάμεσου λογισμικού που γενικεύει το περιβάλλον εκτέλεσης από το κόμβο αισθητήρα, χρησιμοποιώντας μια σειρά από υπηρεσίες και μια scripting γλώσσα προκειμένου να συνθέσει εργασίες μέσα από αυτές τις υπηρεσίες. Ο scripting κώδικας μπορεί να μεταναστεύει από κόμβο σε κόμβο αυτόνομα. Αυτή η agent-based προσσέγγιση διευκολύνει τη παρακολούθηση της κίνησης καθώς και δυναμικά φαινόμενα, όπως μια αγέλη ζώων που μετακινείται μέσα σε ένα δάσος. Βέβαια αυτή η προσέγγιση, λόγω της πολυπλοκότητάς της, απαιτεί υπολογιστικά δυνατούς κόμβους αισθητήρων. Τέλος η «μετανάστευση» κώδικα μέσα στο δίκτυο καταναλώνει αρκετή ενέργεια και συμβάλλει στην μικρής διάρκειας ζωή του δικτύου.

Πέρα από τις προσεγγίσεις που αναφέρθηκαν πιο πάνω, υπάρχουν και άλλες παρόμοιες εκδοχές ενδιάμεσου λογισμικού για ασύρματα δίκτυα αισθητήρων. Για παράδειγμα το Impala [15] εκμεταλλεύεται τις τεχνικές κινητού κώδικα, προκειμένου να αλλάζει τη λειτουργικότητα του ενδιάμεσου λογισμικού

που εκτελείται σε κάποιο απομακρυσμένο αισθητήρα. Το MiLAN [16] είναι μια ακόμη προσέγγιση ενδιάμεσου λογισμικού που επιτρέπει τη δυναμική ρύθμιση του δικτύου (π.χ. εύρεση και οργάνωση των πόρων του δικτύου) προκειμένου να εξασφαλίσει εγγύηση καλής ποιότητας υπηρεσιών στις εφαρμογές. Τέλος το DSWare [20] είναι μια λύση που βασίζεται αποκλειστικά στην ιδέα των συμβάντων (events). Η εφαρμογή δηλώνει ενδιαφέρον για μια συγκεκριμένη κατάσταση αλλαγής του πραγματικού κόσμου και προσδιορίζει την αντίδραση που θα πρέπει να την ακολουθήσει.

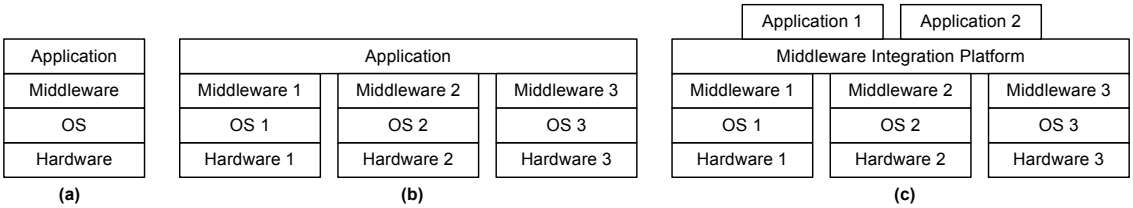
5

SENSATION: Μια Πλατφόρμα Ενδιάμεσου Λογισμικού για Διάχυτες Εφαρμογές στα WSN.

Οι περισσότερες εφαρμογές είναι δέσμιες μια συγκεκριμένης κάθε φορά τεχνολογίας WSN (συνήθως δηλαδή έχουν ρυθμιστεί κατάλληλα ώστε να λειτουργούν με το δίκτυο που τους ενδιαφέρει), με αποτέλεσμα να μην είναι μεταφέρσιμες και εύκολα χρησιμοποιήσιμες από άλλες εφαρμογές. Φυσικά αυτό δημιουργεί πολλές δυσκολίες στον προγραμματιστή ο οποίος πρέπει να γνωρίζει κάθε φορά όλες τις λεπτομέρειες της χρησιμοποιούμενης τεχνολογίας.

Όπως αναφέρθηκε και στην ενότητα 4, όλα αυτά τα συστήματα ενδιάμεσου λογισμικού ακολουθούν διαφορετικές προγραμματιστικές προσεγγίσεις (π.χ. database προσέγγιση, agent-based προσέγγιση και event-based προσέγγιση) και διαφέρουν μεταξύ τους όσον αφορά στην ευκολία χρήση τους, την εκφραστικότητα, την επεκτασιμότητα και το overhead. Αυτό που αξίζει να τονιστεί είναι πως αυτός που αναπτύσσει το σύστημα είναι στενά εξαρτημένος από την προσέγγιση που χρησιμοποιεί και ως αποτέλεσμα πρέπει να γνωρίζει τα συγκεκριμένα χαρακτηριστικά, τις δυνατότητες και τους περιορισμούς της, ενώ

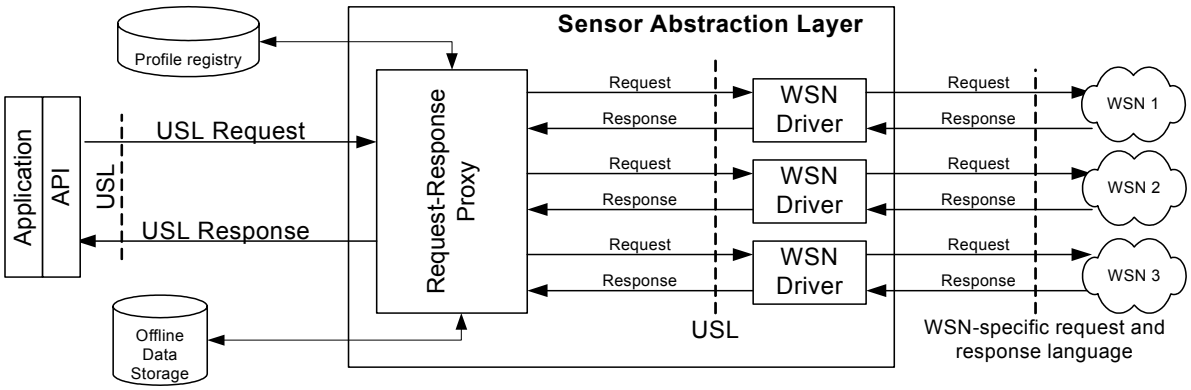
πρέπει παράλληλα να γνωρίζει τις συγκεκριμένες προγραμματιστικές διεπαφές. Παρόλα αυτά, προκειμένου η ανάπτυξη εφαρμογών βασισμένων στα ασύρματα δίκτυα αισθητήρων να γίνει πραγματικότητα, πρέπει αυτός που αναπτύσσει το σύστημα να μην χρειάζεται να γνωρίζει τις τεχνολογίες ενδιάμεσου λογισμικού που χρησιμοποιούνται καθώς και τα χαρακτηριστικά τους. Προκειμένου να επιτευχθούν τέτοιοι στόχοι, μια αρχιτεκτονική του σχήματος 5.1(c) είναι απαραίτητη. Εν αντιθέσει με τις 5.1(a) και 5.1(b), οι εφαρμογές δεν γνωρίζουν τις χρησιμοποιούμενες τεχνολογίες και έχουν πρόσβαση στα δεδομένα των αισθητήρων με μια ενιαία λογική.



Σχήμα 5.1: Σενάρια χρήσης των WSN. [9] (a) Μια απλή εφαρμογή λειτουργεί πάνω από ένα WSN (WSN-specific application). Αυτή είναι η περίπτωση για τις πιο σύγχρονες εφαρμογές (b) Μια απλή εφαρμογή χρησιμοποιεί διάφορα WSNs. Η εφαρμογή πρέπει να κρύβει την ετερογένεια. (c) Διαφορετικές εφαρμογές χρησιμοποιούν διάφορα WSN’s με μια ενοποιημένη συμπεριφορά αναφορικά με την χρήση μιας πλατφόρμας χρήσης ενδιάμεσου λογισμικού. Οι εφαρμογές δεν γνωρίζουν τις χρησιμοποιούμενες τεχνολογίες (εφαρμογές middleware)

5.1 Η Αρχιτεκτονική της Πλατφόρμας

Σε αυτήν την ενότητα περιγράφεται πως οι στόχοι που αναφέρθηκαν προηγουμένως μπορούν να πραγματοποιηθούν. Προκειμένου να κρυφτεί η ετερογένεια μεταξύ των διαφόρων υλοποιήσεων του δικτύων αισθητήρων, πρέπει να προστεθεί ένα ακόμα επίπεδο το οποίο θα διευκολύνει την επικοινωνία εφαρμογών παρακολούθησης (και διαφόρων άλλων), με το δίκτυο αισθητήρων.



Σχήμα 5.2: Η αρχιτεκτονική της πλατφόρμας [1]

Αυτό το επίπεδο θα χρησιμεύει σαν ένα επίπεδο αφαίρεσης, κρύβοντας τις διάφορες υλοποιήσεις του δικτύου από τις εφαρμογές που θα χρησιμοποιούν οι χρήστες. Αιτήσεις και απαντήσεις από διάφορα ετερογενή δίκτυα θα χειρίζονται με τον ίδιο τρόπο, με τη βοήθεια μιας καλά ορισμένης γλώσσας. Τη γλώσσα αυτή την αποκαλούμε USL (από το Unified Sensor Language). Οι εφαρμογές θα πρέπει να γνωρίζουν τη γλώσσα αυτή και να είναι συμβατές μαζί της. Οι αιτήσεις που θα φτάνουν στο Sensor Abstraction Layer (SAL), θα αναλύονται, θα μεταφράζονται στις συγκεκριμένες WSN γλώσσες και θα προωθούνται στα υποκείμενα δίκτυα. Αντιστρόφως, οι απαντήσεις, έχοντας μια συγκεκριμένη δομή, θα συλλέγονται, θα μεταφράζονται σε USL και στη συνέχεια θα προωθούνται σε αυτόν που έκανε την αίτηση. Αυτό το κομμάτι μπορεί να θεωρηθεί ο κορμός όλου του συστήματος. Επιλέγουμε XML για την εφαρμογή της USL, καθώς είναι μια

standard και ευρέως χρησιμοποιούμενη μεταγλώσσα, ικανή να περιγράφει δομές δεδομένων με έναν επίσημο και καλά δομημένο χαρακτήρα. Επιπλέον ο αυτοματοποιημένος μετασχηματισμός και η επεξεργασία της σύνταξης που χρησιμοποιεί η XML προσδίδουν στην USL τα χαρακτηριστικά που επιθυμούμε.

Αναφορικά με το στόχο της παροχής στους προγραμματιστές ενός υψηλού επιπέδου και διαισθητικού προγραμματιστικού μοντέλου, μια αποδοτική προγραμματιστική διεπαφή πρέπει να χρησιμοποιηθεί, όπως δείχνει και το σχήμα 5.2. Αυτό θα αποτελεί και το API (Application Programming Interface) που θα χρησιμοποιούν οι χρήστες, έχοντας έτσι μια πληθώρα εκφραστικών και ευέλικτων εντολών, προκειμένου να πραγματοποιούν αιτήσεις προς το δίκτυο. Αυτό το API θα εφαρμοστεί σε διαφορετικά επίπεδα αφαίρεσης και υπό διαφορετικές οπτικές γωνίες του περιβάλλοντος που θα μπορούσε ο προγραμματιστής να χρησιμοποιήσει βασιζόμενος στις συγκεκριμένες ανάγκες. του (π.χ. μια χαμηλού επιπέδου έκδοση του API θα προσέδιδε στον προγραμματιστή μεγάλη ευελιξία, αλλά θα απαιτούσε πιο προσεκτικό προγραμματισμό, ενώ μια υψηλού επιπέδου έκδοσή του, μπορεί μεν να είχε περιορισμένη ευελιξία, που προϋπέθετε δε ευκολότερο προγραμματισμό και φιλικότερο περιβάλλον προς το χρήστη).

Η σχεδίασή μας για το API βασίζεται σε μια επιλογή περιοχών (locations) και συσκευών (device). Ο προγραμματιστής αντιλαμβάνεται το περιβάλλον σαν έναν συνδυασμό από διαφορετικές περιοχές και συσκευές και βάση αυτών κατευθύνει τις επερωτήσεις του προς το δίκτυο. Αντιθέτως με τις περιοχές, οι συσκευές δεν θεωρούνται στάσιμες, αλλά μπορούν να είναι φορητές (π.χ. PDA) έχοντας παράλληλα δυνατότητες αισθητήρων. Άλλες οντότητες, όπως άνθρωποι, ρομπότ, αυτοκίνητα κ.α μπορούν να κουβαλούν αυτές τις συσκευές, οι οποίες σημασιολογικά σχετίζονται με αυτές. Με αυτές τις δυνατότητες, οι εφαρμογές μπορούν να ζητούν πληροφορίες από τους αισθητήρες που βρίσκονται σε συγκεκριμένες περιοχές, ή που σχετίζονται με συγκεκριμένες οντότητες.

Όπως φαίνεται στο σχήμα 5.2, το σύστημα υποστηρίζει επίσης τα «Profile Registry» και «Offline - Data Storage». Το πρώτο περιλαμβάνει διάφορους

πίνακες βάσης δεδομένων και configuration αρχεία και κρατάει πληροφορίες για τις ρυθμίσεις, τις δυνατότητες υποστήριξης και τα χαρακτηριστικά του συγκεκριμένου δικτύου αισθητήρων με το οποίο σχετίζεται. (π.χ. τους τύπους που υποστηρίζουν οι αισθητήρες όπως θερμοκρασία, υγρασία κ.α., διάφορες μονάδες μέτρησης όπως Celsius, Fahrenheit, Volt κ.α, βασικές συναρτήσεις που μετασχηματίζουν ακατέργαστα δεδομένα από τους αισθητήρες σε πιο χρήσιμες μορφές όπως μετασχηματισμός από Volt σε θερμοκρασία καθώς και πληροφορίες σχετικά με συγκεκριμένες WSN διεπαφές. Όλη αυτή η πληροφορία είναι χρήσιμη και βοηθάει στο να επιλεγούν τα WSN που ικανοποιούν συγκεκριμένα requests (αιτήματα). Επιπλέον το Profile Registry αποθηκεύει πληροφορίες σχετικά με τις φορητές συσκευές που αναφέρθηκαν παραπάνω. Όποτε υπάρχει κάποια αλλαγή στο σύστημα, ο διαχειριστής του συστήματος πρέπει να ενημερώνει τις ρυθμίσεις του Profile Registry κατάλληλα.

Το ODS βασίζεται σε ένα κοινό σύστημα βάσεων δεδομένων (π.χ. μια σχεσιακή βάση) και είναι υπεύθυνο για την συγκέντρωση και την αποθήκευση όλων των απαντήσεων που περνούν από το SAL, προκειμένου να δώσουν τη ευκαιρία στους χρήστες που ενδιαφέρονται για τα ιστορικά στοιχεία καθώς και στατιστικές ιδιότητες του φαινομένου που παρακολουθούν να αποκτήσουν αυτή τη δυνατότητα. Η χρήση του ODS θα επιτρέψει την εφαρμογή πολύπλοκης επεξεργασίας πληροφορίας (π.χ. post-processing για εξόρυξη δεδομένων και caching) που αποτελεί ενεργό πεδίο έρευνας, σε αυτήν την εποχή.

5.2 Η UNIFIED SENSOR LANGUAGE (USL)

Το Sensor Abstraction Layer (SAL), όπως περιγράφηκε στη προηγούμενη ενότητα, βασίζεται στην USL, μια generic γλώσσα για το χειρισμό αισθητήρων. Αυτή η γλώσσα έχει οριστεί σαφώς, μετά από εκτεταμένη ανάλυση απαιτήσεων, (απαιτήσεις που αφορούν τόσο στην αλληλεπίδραση μεταξύ του προγραμματιστή με το περιβάλλον των αισθητήρων, όσο και στην ουσιαστική λειτουργικότητα των πρόσφατα διαθέσιμων ενδιάμεσων λογισμικών για

ασύρματα δίκτυα αισθητήρων). Η ανάλυση αυτή οδήγησε στα ακόλουθα απαραίτητα χαρακτηριστικά της γλώσσας προκειμένου να ικανοποιούνται πλήρως όλες οι απαιτήσεις των χρηστών:

1. Υποστήριξη για σύγχρονες αιτήσεις (επερωτήσεις) που παίρνουν τα επιθυμητά δεδομένα, είτε από το WSN, είτε από το ODS και επιστρέφουν τις απαντήσεις σε πραγματικό χρόνο.
2. Υποστήριξη για προγραμματισμό βάση συμβάντων (events). Πολλές εφαρμογές επίγνωσης πλαισίου (context-aware) πρέπει να πυροδοτούν διάφορες ενέργειες όταν συμβούν κάποια συμβάντα στο δίκτυο. Οι χρήστες πρέπει να μπορούν να εγκαθιστούν listeners και χειριστές (handlers) τέτοιων συμβάντων (π.χ. σε περίπτωση που η θερμοκρασία είναι αρκετά υψηλή, μεγαλύτερη από 42 βαθμούς Κελσίου, μέσα σε ένα δωμάτιο και η φωτεινότητα είναι αρκετά δυνατή, τότε θα πρέπει να σημάνει συναγερμός για φωτιά).
3. Υποστήριξη για περιοδική παρακολούθηση των τιμών των αισθητήρων (π.χ. μέτρηση της θερμοκρασίας κάθε 5 λεπτά, ξεκινώντας από τη Δευτέρα 2 Ιουλίου στις 15:00 μ.μ. και σταματώντας ακριβώς μια βδομάδα μετά).
4. Εύκολη και δυναμική αλλαγή των τύπων που μετρούν οι αισθητήρες, καθώς και των συναρτήσεων που υποστηρίζουν. Καθώς νέα WSN ή αισθητήρες χρησιμοποιούνται, ο προγραμματιστής πρέπει να έχει τη δυνατότητα να συμπεριλάβει τη νέα αυτή λειτουργικότητά στη λογική του προγράμματός του. Επιπλέον, σε περίπτωση σφάλματος στους αισθητήρες, κάτι που συμβαίνει αρκετά συχνά, πρέπει να υπάρχει ειδικός χειρισμός του λάθους που να συνοδεύεται με σαφή περιγραφή.

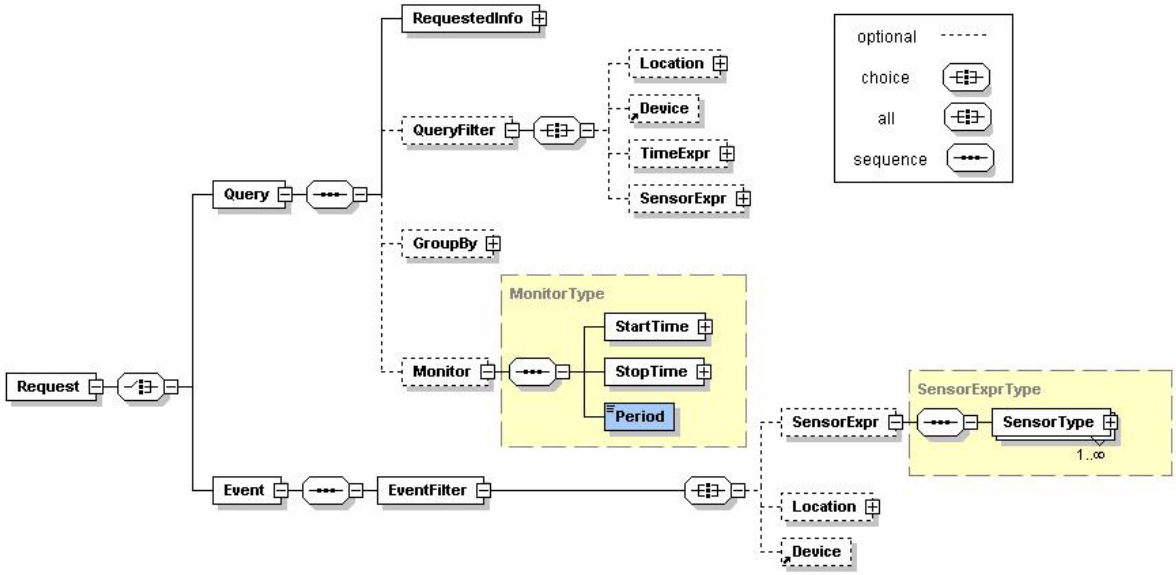
Το αποτέλεσμα από την ανάλυση απαιτήσεων είναι ο προσδιορισμός της USL γλώσσας. Η USL περιγράφεται σε XML format, αφού η XML παρέχει μια ανεξαρτησία από την πλατφόρμα που μπορεί εύκολα να αναλυθεί και να επεξεργαστεί. Δεδομένης της πρόσφατης απήχησης που έχει η database-like

επεξεργασία πληροφορίας για δεδομένα δικτύων αισθητήρων, κάποιος θα μπορούσε να υποστηρίξει πως μια SQL-like γλώσσα θα ήταν πιο κατάλληλη.

Πιστεύουμε πως η σχεσιακή φύση των γλωσσών αυτών, δημιουργεί κάποιους περιορισμούς, αναφορικά με τον γενικό χειρισμό των WSN υποδομών, καθώς είναι εξαρτημένη από τη μορφή των σχεσιακών επερωτήσεων (select - from - where) και δεν υποστηρίζει την declarative και event-driven λογική που θέλουμε να παρέχεται στο χρήστη. Παράλληλα, έχουμε σκοπό να βελτιώσουμε την USL, προσδίδοντάς της διαχειριστική λειτουργικότητα (π.χ. υποστήριξη για πολιτικές, διαχείριση και επαναπροσδιορισμό του WSN), κάτι που θα καθυστερούσε πάρα πολύ με τη χρήση μιας SQL-like γραμματικής. Η USL ορίζει δύο βασικές οντότητες, την αίτηση και την απάντηση. Αφηρημένες εκδοχές (λόγω του περιορισμού χώρου) των XML Schema αυτών των δύο οντοτήτων φαίνονται στα σχήματα 5.3 και 5.4

5.3 Η USL Αίτηση (Request)

Η USL αίτηση αντιπροσωπεύει τις αιτήσεις (σύγχρονες, event-driven και περιοδικές) του χρήστη προς το WSN ή το ODS. Το στοιχείο που βρίσκεται στη ρίζα του δένδρου, το request δηλαδή, περιέχει μια μοναδική ID μεταβλητή. Αυτή η μεταβλητή γενικεύεται από το API και προσδιορίζει μοναδικά ένα αίτημα από τον χρήστη.



Σχήμα 5.3: Η USL Αίτηση [1]

Στη περίπτωση της σύγχρονης επερώτησης, ο χρήστης προσδιορίζει τις πληροφορίες για τις οποίες ενδιαφέρεται, μέσα στο στοιχείο RequestedInfo. Αυτή η πληροφορία μπορεί να περιλαμβάνει μια ανάγνωση από έναν ή περισσότερους αισθητήρες, την τοποθεσία στην οποία ισχύει αυτή η συνθήκη, τη χρονική στιγμή ή τη διάρκεια ενός συγκεκριμένου φαινομένου, το ID της συσκευής που ικανοποιεί κάποια κριτήρια, ή ένα συνδυασμό των παραπάνω. Φυσικά δεν είναι όλοι οι συνδυασμοί έγκυροι, γι'αυτό διενεργείται ένας σημασιολογικός έλεγχος. Οι περιορισμοί του request περιγράφονται στο QueryFilter στοιχείο. Σε αυτό το στοιχείο, κάποιος μπορεί να δηλώσει συνθήκες χρόνου (στοιχείο TimeExpr), να δηλώσει συνθήκες για τον αισθητήρα (στοιχείο SensorExpr), καθώς και να περιορίσει την επερώτηση σε μια συγκεκριμένη περιοχή ή σε μια συγκεκριμένη συσκευή (στοιχεία Location και Device αντίστοιχα). Το SensorExprType (βλέπε σχήμα 5.3), είναι η XML παρουσίαση της παρακάτω BNF[17] γραμματικής:

Πίνακας 5.1: Το SensorExpression κομμάτι της αίτησης

SensorExpr *= [Function,] SensorType, Conditional, Value;*
Function *= 'tempAverage' | 'tempMinimum' | 'tempCount' |'tempMaximum' | 'spatialAverage' | 'spatialSum';*
SensorType *= 'Temperature' | 'Humidity' | 'Acceleration';*
Conditional *= 'Greater' | 'Less' | 'Equals' | 'WithinRange';*
Value *= Alphanumeric | RealNumber | Integer;*

Φυσικά οι ορισμοί των Function και SensorType δεν είναι ολοκληρωμένοι. Οι δυνατές τιμές τους προσδιορίζονται σε ένα ξεχωριστό XML Schema. Το στοιχείο TimeExpr έχει παρόμοια σύνταξη.

Επιπλέον υπάρχει το στοιχείο GroupBy, το οποίο μπορεί να ταξινομεί τις απαντήσεις με τρόπο αντίστοιχο με το γνωστό GroupBy της SQL, ενώ η σύνταξη του είναι παρόμοια με του RequestedInfo. Το στοιχείο Monitor, στη περίπτωση που υπάρχει, δηλώνει πως η επερώτηση θα πρέπει να εκτελεστεί περιοδικά όπως περιγράφεται από τα StartTime, StopTime και Period στοιχεία. Από εδώ και πέρα θα αναφερόμαστε στις περιοδικές επερωτήσεις χρησιμοποιώντας τον όρο monitors. Παρόμοια με τα events, απαιτούν ο προγραμματιστής να εγκαταστήσει τους κατάλληλους listeners.

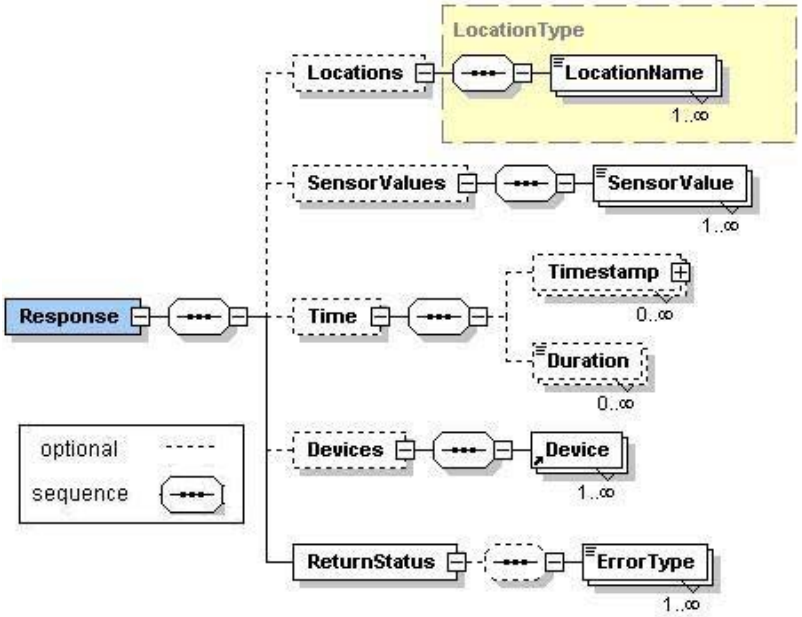
Από την άλλη, στη περίπτωση της χρήσης event-driven προγραμματισμού, το μόνο υποστοιχείο του Request, είναι το Event. Αυτό το στοιχείο περιλαμβάνει μια σειρά από συνθήκες (στοιχείο EventFilter), που όταν ικανοποιούνται, πυροδοτούν κάποια events στα ανώτερα στρώματα του ενδιάμεσου λογισμικού. Αυτά τα ανώτερα επίπεδα, έχουν εγκαταστήσει από πριν listeners για αυτά τα events, και (μέσω της οντότητας USL Response), διενεργούν κάποιες προσδιορισμένες από πριν (από τους προγραμματιστές), ενέργειες.

Το EventFilter στοιχείο μοιάζει πάρα πολύ με το QueryFilter, εκτός από το γεγονός ότι δεν περιέχει καθόλου συνθήκες χρόνου. Αυτό είναι λογικό για αυτήν την πρώτη έκδοση της USL, καθώς events που γενικεύονται από συνθήκες χρόνου, μπορούν να εφαρμοστούν μόνο με στόχο το ODS και περιλαμβάνουν πολύπλοκες τεχνικές επεξεργασίας πληροφορίας. Καθώς υπάρχει μεγάλη ερευνητική δραστηριότητα σε αυτές τις περιοχές, μια μελλοντική έκδοση της USL, θα μπορούσε να συμπεριλαμβάνει events βασισμένα στο χρόνο (π.χ. σε περίπτωση που η θερμοκρασία αλλάζει σε ένα δωμάτιο με υπολογιστές με ρυθμό +3 βαθμούς Κελσίου την ώρα, ενημέρωσε τον θυρωρό προκειμένου να ενεργοποιήσει τον κλιματισμό).

Το USL request, εκτός από το να παρέχει τη δυνατότητα της εγκατάστασης event listeners και τη περιγραφή των επερωτήσεων, υποστηρίζει επίσης τη διευθέτηση ήδη εγκατεστημένων event listeners και monitors. Γι'αυτό το λόγο, τα Event και Monitor στοιχεία, περιλαμβάνουν Boolean μεταβλητές abort. Όταν ένας χρήστης διευθετεί ένα event ή ένα monitor, το γνωστό του ID περνάει στη μεταβλητή ID του Request στοιχείου, και η συγκεκριμένη abort μεταβλητή τίθεται σε true, ενώ όλα τα υπόλοιπα στοιχεία είναι απόντα ή κενά.

5.4 Η USL Απάντηση (Response)

Η USL απάντηση είναι αρκετά πιο απλή από την οντότητα request. Το στοιχείο που βρίσκεται στη ρίζα, περιλαμβάνει επίσης μια μεταβλητή ID και πάντα περιέχει το στοιχείο ReturnStatus.



Σχήμα 5.3: Η USL Απάντηση [1]

Αν η μεταβλητή error αυτού του στοιχείου πάρει τη τιμή true, τότε κάποιο λάθος συνέβη μέσα στο SAL ή το WSN και έτσι ο τύπος του (στοιχείο ErrorType) επιστρέφεται στο API προκειμένου να δημιουργηθεί κάποιο exception (εξαίρεση) στην εφαρμογή. Σε περίπτωση που δεν συνέβη κάποιο λάθος και το request δεν περιέχει κάποιο σφάλμα, τα απαιτούμενα δεδομένα (στο RequestedInfo) επιστρέφονται στον requestor (σε αυτόν πού ζήτησε τα δεδομένα). Εναλλακτικά, σε περίπτωση που το request εγκατέστησε κάποιο event, τότε όλα τα στοιχεία εκτός από το ReturnStatus είναι απόντα (π.χ. η απάντηση είναι ισοδύναμη με ένα flag που δείχνει αν το event έλαβε χώρα).

Κάποιες από τις παραμέτρους στα προαναφερθέντα USL στοιχεία και μεταβλητές, μπορεί να διαφέρουν αναλόγως την περίπτωση (π.χ. εξαιτίας της χρήσης καινούργιων αισθητήρων). Αυτοί είναι οι τύποι των αισθητήρων (π.χ. αισθητήρες θερμοκρασίας), οι συναρτήσεις που μπορούν να εφαρμοστούν σε αυτούς (π.χ. TempAverage, δηλαδή Temporal Average), και οι τύποι ένδειξης λάθους (π.χ. TIMEOUT). Όλα αυτά περιγράφονται σαν απαριθμήσεις σε ένα ξεχωριστό XML Schema το οποίο συμπεριλαμβάνεται από τα Request και

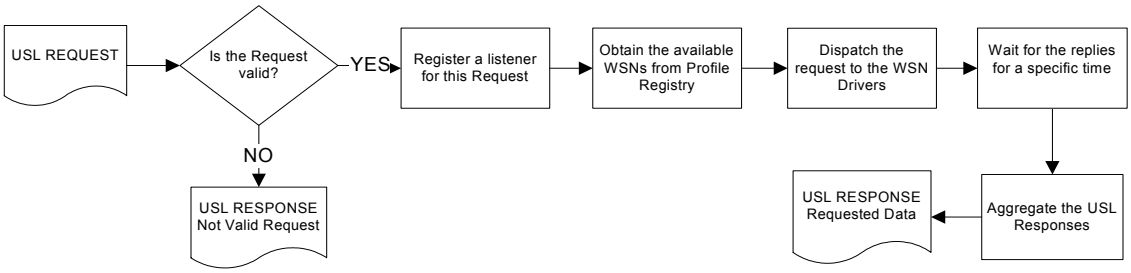
Response Schema, με σκοπό να εφαρμόσουν κάποιους περιορισμούς κατά τη διάρκεια του XML validation των USL αιτήσεων και απαντήσεων από τον RR Proxy. Αυτό το ξεχωριστό XML Schema μπορεί να θεωρηθεί σαν ένα κομμάτι του Profile Registry, επειδή ενημερώνεται κάθε φορά που αλλάζουν οι ρυθμίσεις του WSN.

Ο σχηματισμός του USL request μπορεί να πραγματοποιηθεί από το API που περιγράφεται παρακάτω, στην ενότητα WSN Application Programming Interface (API). Αντιστρόφως, η USL απάντηση γεμίζει με τα δεδομένα που συλλέχθηκαν από το δίκτυο ή από την ένδειξη κάποιου λάθους και τελικώς επιστρέφεται στο χρήστη.

5.5 Το Sensor Abstraction Layer (SAL)

Όπως αναφέρθηκε και προηγουμένως, το SAL είναι το πιο σημαντικό κομμάτι του συστήματος και αποτελείται από τις ακόλουθες λειτουργικές οντότητες:

- Τον RR Proxy (Request – Response Proxy)
- Το WSN Driver (έναν ή περισσότερους, αναλόγως τα διαφορετικά WSN που χρησιμοποιούνται).



Σχήμα 5.4: Η προώθηση της αίτησης προς τον RR Proxy [1]

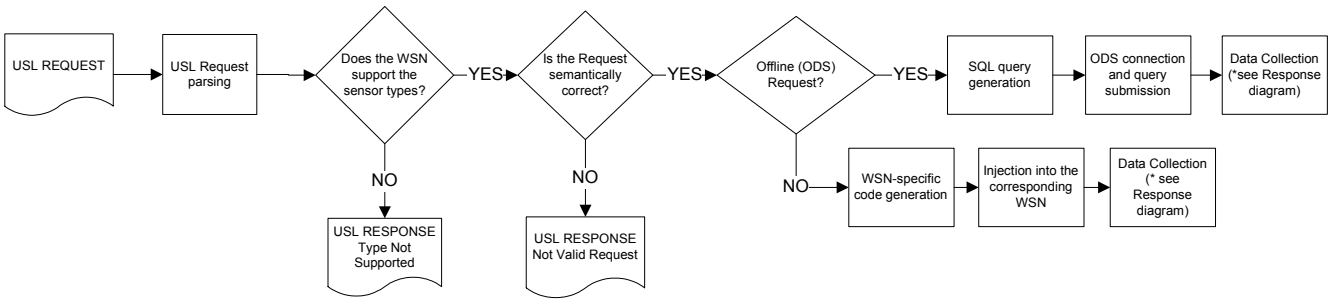
5.6 RR Proxy (Request – Response Proxy)

Ο RR Proxy, όπως φαίνεται και στο σχήμα 5.4, λαμβάνει τις USL αιτήσεις και διενεργεί μια πρώτου επιπέδου επεξεργασία, που περιλαμβάνει τα ακόλουθα βήματα:

- Αρχικά ελέγχει αν το request είναι σωστό γραμματικά (π.χ. αν το XML document που χρησιμοποιείται για το request είναι έγκυρο αναφορικά με το XML Schema που χρησιμοποιείται. Αν το validation επιτύχει, η επεξεργασία του request συνεχίζει, αλλιώς ένα κατάλληλο μήνυμα λάθους (π.χ. Not Valid Request) επιστρέφεται στον χρήστη.
- Μετά από ένα πετυχημένο validation, ο RR Proxy, συλλέγει τα διαθέσιμα WSNs που βρίσκονται στη περιοχή ή στη συσκευή από το Profile Registry. Αν τα διαθέσιμα WSNs είναι περισσότερα από ένα, τότε ο RR Proxy πρέπει να διανείμει το request στους αντίστοιχους WSN Drivers. Επίσης ο RR Proxy εγκαθιστά έναν listener για το request, το οποίο σχετίζεται με ένα μοναδικό request ID που ανατέθηκε στο request από το API του χρήστη. Επίσης διαθέτει έναν σχετιζόμενο Timer που λήγει μετά από μια συγκεκριμένη χρονική περίοδο. Ο listener συγκεντρώνει τις απαντήσεις που σχετίζονται με ένα συγκεκριμένο request ID και προέρχονται από τα υποκείμενα WSN και στα οποία έγιναν οι αιτήσεις και στη συνέχεια επιστρέφεται δεδομένα στον χρήστη. Σε περίπτωση που δεν επιστραφεί τίποτα από τα WSN μέχρι να λήξει ένα συγκεκριμένο χρονικό διάστημα, ο listener επιστρέφει στον χρήστη ένα μήνυμα λάθους (π.χ. Request Timeout). Η βασική επεξεργασία του request γίνεται μέσα στον WSN Driver που περιγράφεται παρακάτω.

5.7 WSN Driver

Όπως φαίνεται στο σχήμα 5.5, ο WSN Driver επεξεργάζεται ένα request που λαμβάνει από τον RR Proxy, ακολουθώντας τα παρακάτω βήματα:



Σχήμα 5.5: Η προώθηση της αίτησης προς το δίκτυο αισθητήρων [1]

1. **USL Request Parsing**

   Ο WSN Driver ελέγχει αν τα δεδομένα που περιέχονται στο USL request, είναι σωστά δομημένα και έγκυρα.

2. **Έλεγχος Υποστήριξης αιτούμενων στοιχείων**

   Στο request περιέχονται διάφορες πληροφορίες που περιλαμβάνουν τους τύπους, τις συναρτήσεις και τις περιοχές που υποστηρίζονται από το δίκτυο με το οποίο συνεργάζεται ο WSN Driver. Αφού γίνει το parsing του USL request, ο WSN Driver πρέπει να ελέγξει αν το ασύρματο δίκτυο αισθητήρων με το οποίο συνεργάζεται υποστηρίζει το request. Αρχικά εξάγει μια λίστα με τύπους αισθητήρων (π.χ. θερμοκρασία) που βρίσκονται στο στοιχείο RequestedInfo και πρόκειται να ενσωματωθούν στην επερώτηση. Το ίδιο κάνει με τους τύπους και τις περιοχές που βρίσκονται στο στοιχείο QueryFilter ή στο στοιχείο EventFilter. Στη συνέχεια συμβουλεύεται το Profile Registry Σε περίπτωση κάποιος από τους τύπους ή τις περιοχές δεν υποστηρίζεται από το δίκτυο, η επεξεργασία τερματίζεται και επιστρέφεται κατάλληλο μήνυμα στον RR Proxy που καταδεικνύει το σφάλμα. Αν υποστηρίζονται όλα τα επιθυμητά δεδομένα, τότε η επεξεργασία προχωράει στο στάδιο της σημασιολογικής ανάλυσης.

3. **Σημασιολογική Ανάλυση**

Σε αυτό το στάδιο, ο WSN Driver πραγματοποιεί σημασιολογικούς ελέγχους στα δεδομένα που περιέχονται στο request που έλαβε. Σε περίπτωση ένδειξης σημασιολογικού σφάλματος, η επεξεργασία σταματά και αποστέλλεται κατάλληλο μήνυμα στον RR Proxy που καταδεικνύει το σφάλμα. Αν όχι, τότε η επεξεργασία προχωράει στο επόμενο στάδιο (αναλόγως αν θέλουμε offline ή real - time επεξεργασία).

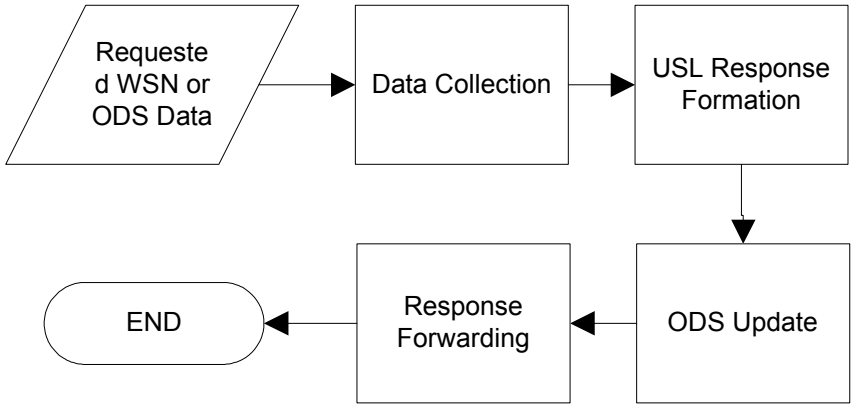
4. **Δρομολόγηση του Request**

   Αυτό το στάδιο βασίζεται στην απόφαση που πάρθηκε στο αμέσως προηγούμενο σχετικά με το αν επιθυμούμε offline (π.χ. πότε η θερμοκρασία ξεπέρασε τους 50 βαθμούς Κελσίου) ή πραγματικού χρόνου (real - time) επεξεργασία (π.χ. Ποιά η μέση τιμή της υγρασίας από όλους τους αισθητήρες). Στη πρώτη περίπτωση, το request μεταφράζεται σε μια SQL επερώτηση, αλλιώς σε ειδικό κώδικά για WSN.

5. **Εισαγωγή του request στο δίκτυο**

   Αν όλα τα προηγούμενα στάδια έχουν ολοκληρωθεί με επιτυχία, τότε το request θα παραδοθεί στο υποκείμενο WSN ή στο ODS και ο Driver θα περιμένει μέχρι να λάβει απάντηση.

Τα βήματα τα οποία ακολουθούνται για το σχηματισμό της απάντησης, όπως φαίνονται και στο σχήμα 5.6, είναι τα ακόλουθα:



Σχήμα 5.6: Η προώθηση της αίτησης προς το δίκτυο αισθητήρων [1]

1. **Συλλογή δεδομένων και σχηματισμός της απάντησης**
   Ο WSN Driver λαμβάνει τα δεδομένα από το δίκτυο ή από τη βάση και στη συνέχεια τα ενσωματώνει σε μια USL απάντηση, σύμφωνα με τη κατάλληλη δομή που περιγράφηκε προηγουμένως.
2. **Ενημέρωση της βάσης (ODS)**
   Πριν επιστραφεί η απάντηση στον RR Proxy, πρέπει να ενημερώνεται η βάση (μόνο στη περίπτωση που το request περιελάμβανε real-time επεξεργασία, που προωθήθηκε δηλαδή στο δίκτυο). Αυτή η εγγραφή στη βάση, θα περιλαμβάνει την απάντηση που επιστράφηκε στον RR Proxy, το request ID καθώς και άλλες πρόσθετες πληροφορίες (π.χ. Timestamp, location, device κ.α.). Με την καταγραφή αυτών των στοιχείων, μπορεί κάποιος αργότερα να πραγματοποιήσει στατιστική ή κάθε άλλου είδους επεξεργασία
3. **Προώθηση της απάντησης**
   Η USL απάντηση προωθείται στον RR Proxy και, πιο συγκεκριμένα, στον listener που έχει εγκατασταθεί για αυτό το request.

Αυτό που αξίζει να σημειωθεί, είναι πως η επικοινωνία μεταξύ του RR Proxy και των WSN Drivers μέσα στο SAL, βασίζεται και αυτή στην USL. Αυτό επιτρέπει στο να χειριζόμαστε το SAL σε ένα ανεξάρτητο γλώσσας επίπεδο (language neutral). Επίσης, με την προτεινόμενη αρχιτεκτονική, ο κατασκευαστής του WSN, δεδομένου ενός αντίστοιχου Driver για το WSN του, (π.χ. ένα συστατικό που να μπορεί να μετατρέπει USL σε κώδικα για WSN και αντίστροφα), θα μπορεί να ενσωματώνει το WSN του στην συγκεκριμένη αρχιτεκτονική και να το κάνει συμβατό με ήδη υπάρχουσες εφαρμογές (υπό αυτή την σκοπιά προέκυψε και ο όρος Driver, παραπέμποντας στους Drivers που χρησιμοποιούνται από εκτυπωτές, περιφερειακά και άλλες συσκευές). Για την πραγματική εφαρμογή της παραπάνω δομής, ο κατασκευαστής του WSN πρέπει να ενημερωθεί επίσης το Profile Registry, με πληροφορίες που αφορούν στις δυνατότητες του WSN και συναρτήσεις, για την μετατροπή των ακατέργαστων

δεδομένων των αισθητήρων, σε κοινά αποδεκτές μορφές (π.χ. Volt σε βαθμούς Κελσίου).

5.8 WSN Application Programming Interface (API)

Όπως αναφέρθηκε και προηγουμένως, ένας από τους βασικούς στόχους αναφορικά με το προτεινόμενο σύστημα ολοκλήρωσης ενδιάμεσου λογισμικού για ασύρματα δίκτυα αισθητήρων, είναι η δημιουργία ενός υψηλού επιπέδου Application Programming Interface (API). Ένα τέτοιο API, θα παρέχει στους προγραμματιστές διάχυτων εφαρμογών, τη δυνατότητα να εκμεταλλευτούν τη λειτουργικότητα και τις δυνατότητες των υποκείμενων WSN που χρησιμοποιούν, με το να τους παρέχουν ένα σύνολο από καλά ορισμένες μεθόδους για να πραγματοποιούν αιτήσεις, αλλά και να εκτελούν χαμηλού επιπέδου WSN υπηρεσίες (π.χ. απόκτηση δεδομένων από τους αισθητήρες).

Σχεδιάσαμε δύο διαφορετικά APIs, τα οποία εξυπηρετούν διαφορετικούς σκοπούς. Το ένα (high-level API) βασίζεται στην έννοια της περιοχής (location) και της συσκευής (Device), ενώ το δεύτερο (low-level), βασίζεται στην έννοια των επερωτήσεων. Αυτοί που αναπτύσσουν μια εφαρμογή μπορούν να χρησιμοποιήσουν οποιοδήποτε από τα δύο επιθυμούν, αναλόγως τις ανάγκες τους. Μερικές από τις χαρακτηριστικές μεθόδους των δύο API παρουσιάζονται στον πίνακα 5.2. Το πρώτο API, είναι πιο φιλικό προς τον προγραμματιστή και πιο εύκολα κατανοητό, ενώ το (low-level) είναι πιο ευέλικτο. Η ευελιξία του βασίζεται στο γεγονός ότι οι χρήστες μπορούν να κατασκευάσουν πιο πολύπλοκα requests, αποκομμένα από την ίδια τη περιοχής και τη συσκευή που σε κάποιες περιπτώσεις θα μπορούσαν να αποτελούν απαγορευτικούς παράγοντες. Παρατηρήστε πως το API που προσανατολίζεται στις περιοχές, έχει και κάποιες μεθόδους για το χειρισμό τους.

Ένα πρόβλημα το οποίο προκύπτει κατά τον χειρισμό ετερογενών δικτύων που αλλάζουν δυναμικά, είναι το κατά πόσο ο προγραμματιστής πρέπει να γνωρίζει τους υποστηριζόμενους τύπους αισθητήρων, καθώς και τις συναρτήσεις

τους. Υπάρχει η ανάγκη για ένα μηχανισμό που γνωρίζει τι λειτουργικότητα υποστηρίζεται από το συγκεκριμένο δίκτυο. Προκειμένου να ξεπεραστεί αυτό το εμπόδιο, η χρήση JDBC-like κλάσεων μεταδεδομένων θεωρείται η καταλληλότερη λύση. Με την εξέταση πληροφοριών μεταδεδομένων, από το Profile Registry, οι εφαρμογές μπορούν να ανακαλύψουν και να χρησιμοποιήσουν μόνο τους διαθέσιμους τύπους αισθητήρων και τις συναρτήσεις τους. Ενδεικτικές μέθοδοι μεταδεδομένων παρουσιάζονται στον πίνακα 5.2. Τα μεταδεδομένα μπορούν να ενημερωθούν σε περίπτωση αλλαγής του WSN που χρησιμοποιείται, π.χ. όταν ένας καινούργιος τύπος αισθητήρα (temperature, velocity κ.α.) εγκατασταθεί, ή όταν ένας χρησιμοποιούμενος τύπος πάψει να υποστηρίζεται.

Πίνακας 5.2: Το SensorExpression κομμάτι της αίτησης

	Method
High-Level API	**Location.getSensorValue(SensorType, Function)** Για μια δεδομένη περιοχή και ένα δεδομένο τύπο αισθητήρα, επιστρέφει την τιμή της επιθυμητής συνάρτησης. Η συνάρτηση μπορεί να είναι max, avg κ.α.
	Location.getDeviceWhere (SensorConditions) Επιστρέφει το ID της συσκευής που ικανοποιεί τις συνθήκες του αισθητήρα.
	Location.getTimeWhen (SensorConditions)** Επιστρέφει το χρονόσημο εκείνο που ικανοποιεί τις επιθυμητές συνθήκες.

	Location.addListener(EventFilter) Εγκαθιστά ένα νέο listener για μια δεδομένη περιοχή. Ενεργοποιείται όταν οι συνθήκες που προσδιορίζονται στο στοιχείο EventFilter ικανοποιούνται..
	Device.getSensorValue (SensorType, Function) Παρόμοια με τη πρώτη μέθοδο του πίνακα.
	Device.getLocation () Επιστρέφει τη περιοχή μιας δεδομένης συσκευής
Low-Level API	**Query.setSensorTypes (SensorTypes)** Θέτει τα επιθυμητά sensor types (π.χ. temperature, humidity, κ.α..)
	Query.setFunctions (Functions) Θέτει τις συναρτήσεις (*min ,max, etc.*). Αυτές θα πρέπει να ανταποκρίνονται μια στα sensor types της προηγούμενης μεθόδου.
	Query.setMonitor(Monitor) Σηματοδοτεί ότι μια επερώτηση είναι τύπου monitor. Η μεταβλητή monitor είναι ένα αντικείμενο Monitor που περιλαμβάνει τις χρονικές παραμέτρους.
	Query.abort() Ακυρώνει την επερώτηση που αντιπροσωπεύεται από το αντικείμενο Query.
	Query.send() Στέλνει μια κατασκευασμένη επερώτηση στον RR Proxy.
Metadata Methods	**getSupportedSensorTypes(Location)** Επιστρέφει τα the sensor types που υποστηρίζονται από το σύστημα για μια δεδομένη περιοχή.

	getSupportedTemporalFunctions() Επιστρέφει τις χρονικές συναρτήσεις που υποστηρίζονται (π.χ. min, avg, count κ.α.).

* ΟΙ ΣΥΝΑΡΤΗΣΕΙΣ ΠΟΥ ΜΠΟΡΟΥΝ ΝΑ ΕΦΑΡΜΟΣΤΟΥΝ ΣΕ ΕΝΑ ΤΥΠΟ ΑΙΣΘΗΤΗΡΑ ΚΑΤΗΓΟΡΙΟΠΟΙΟΥΝΤΑΙ ΣΕ ΧΩΡΙΚΕΣ (SPATIAL) ΚΑΙ ΧΡΟΝΙΚΕΣ (TEMPORAL) ΣΥΝΑΡΤΗΣΕΙΣ

** Η ΠΑΡΟΥΣΙΑ ΧΡΟΝΟΥ (TIME) ΣΕ ΜΙΑ ΜΕΘΟΔΟ (ΕΙΤΕ ΜΕ ΤΟ ΟΝΟΜΑ ΕΙΤΕ ΜΕ ΤΟ ATTRIBUTE LIST) ΥΠΟΔΗΛΩΝΕΙ OFFLINE ΕΠΕΞΕΡΓΑΣΙΑ ΑΠΟ ΤΟ ODS

5.9 Ένα παράδειγμα Request-Response

Αυτό το παράδειγμα αποσκοπεί στο να δείξει παραστατικά όλα αυτά που αναφέρθηκαν παραπάνω, παρουσιάζοντας μια τυπική λειτουργία αίτησης προς τους αισθητήρες και συλλογής της απάντησης από αυτούς.

Ας θεωρήσουμε το σενάριο κατά το οποίο οι ασθενείς ενός νοσοκομείου, είναι εξοπλισμένοι με κατάλληλους αισθητήρες από διαφορετικούς προμηθευτές, εκ των οποίων ο ένας μετράει τους παλμούς της καρδιάς, ένας άλλος τη θερμοκρασία του σώματος, ένας τρίτος τη πίεση του αίματος κ.α. Οι γιατροί και οι νοσοκόμες ενδιαφέρονται να παίρνουν πληροφορίες για τη κατάσταση της υγείας κάθε ασθενή και δεν γνωρίζουν τις διαφορετικές τεχνολογίες αισθητήρων που χρησιμοποιούνται. Το μόνο που γνωρίζουν είναι πως οι αισθητήρες εφάπτονται σε κάθε ασθενή. Η αρχιτεκτονική που προτείνουμε τους παρέχει αυτό το επίπεδο αφαίρεσης.

Ας υποθέσουμε πως μια εφαρμογή επόπτευσης της υγείας των ασθενών, πρέπει να γνωρίζει το ρυθμό των χτύπων της καρδιάς και τη θερμοκρασία του σώματος ενός ασθενή με το όνομα “John Malade”. Η εφαρμογή χρειάζεται αρχικά να κάνει μια επερώτηση στην βάση του νοσοκομείου, προκειμένου να βρεί ποιοί

αισθητήρες την ενδιαφέρουν και στη συνέχεια να τους ξεχωρίσει (π.χ. temperature και Heartbeat). Ας θεωρήσουμε πως ο αισθητήρες χτύπων καρδιάς #57 και ο αισθητήρας θερμοκρασίας #309, εφάπτονται στον ασθενή που μας ενδιαφέρει. Οι καταχωρήσεις στο Profile Registry που περιγράφουν αυτές τις δύο διαφορετικές συσκευές, θα έμοιαζαν κάπως έτσι: Heartbeat_57 και Temperature_309. Έχοντας αυτήν τη πληροφορία, η εφαρμογή μπορεί πάρει τα δεδομένα σχετικά με την κατάσταση της υγείας του ασθενή. Ο σχετικός xml κώδικας θα έμοιαζε με τα στοιχεία του πίνακα 5.4.

Πίνακας 5.3: Java κώδικας του προηγούμενου παραδείγματος

```
Device heartBeat = new Device(“Heatbeat_57”);
//creates an instance of the heartbeat sensor node
Device temperature = new Device(“Temperature_309”);
//creates an instance of the body temperature sensor node

String[] heartSupportedSensors = MetaData.getSupportedSensorTypes(heartBeat);
//the programmer is informed about the supported sensor types.
String[] tempSupportedSensors = MetaData.getSupportedSensorTypes(temperature);
//the programmer is informed about the supported sensor types.

SensorType[] sensorTypes = new ArrayList();
sensorTypes.add(heartSupportedSensors[0]);
//we declare interest for the heartbeat rate
String[] results = heartBeat.getSensorValues(sensorTypes);

sensorTypes.clear();
sensorTypes.add(tempSupportedSensors[0]);
//we declare interest for the body temperature
```

```
String[] results = temperature.getSensorValues(sensorTypes);
```

Ο παραπάνω κώδικας παρουσιάζει δύο διαφορετικά requests, ένα για τους χτύπους της καρδιάς και ένα για τη θερμοκρασία του σώματος και αφού αυτοί οι δύο αισθητήρες είναι διαφορετικής τεχνολογίας και συνεπώς δεν μπορούν να ανήκουν στο ίδιο ασύρματο δίκτυο αισθητήρων. Όπως αναφέρθηκε και παραπάνω, αυτά τα requests, εκφρασμένα από το WSN API θα μετασχηματιστούν σε USL (δηλαδή XML Documents) και θα τους ανατεθούν μοναδικά ID's (π.χ. 123456 και 123457). Παρακάτω παρουσιάζεται ένα από τα δύο requests (π.χ. για τη θερμοκρασία σώματος). Το USL Request Document θα έχει την εξής μορφή:

Πίνακας 5.4: USL Request του προηγούμενου παραδείγματος

```
<Request xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" WSNRequest.xsd" ID="123456">
   <Query>
      <RequestedInfo location="false" time-instance="false" device="true" sensor="true"
time-duration="false">
         <SensorList>
            <SensorType function="NoFunction" type="temperature"/>
         </SensorList>
      </RequestedInfo>
      <QueryFilter>
         <Device ID="543"/>
      </QueryFilter>
   </Query>
</Request>
```

Το request σε μορφή XML παραδίδεται στον RR Proxy που ελέγχει αν είναι συντακτικά σωστό, και στη συνέχεια εγκαθιστά έναν listener για το συγκεκριμένο request ID. Στη συνέχεια το USL request παραδίδεται στον WSN Driver (σε κάποιον συμβατό με το TinyDB στη περίπτωσή μας). Εκεί θα μεταφραστεί σε TinyDB (SQL-like) επερώτηση που θα έχει τη μορφή των στοιχείων του πίνακα 5.5.:

Πίνακας 5.5: TinyDB επερώτηση προηγούμενου παραδείγματος

```
SELECT “Temperature”
FROM SENSORS
WHERE MoteID=”534”
```

Στη συνέχεια η επερώτηση θα αποσταλεί στην κατάλληλη TinyDB WSN διεπαφή και θα εισαχθεί στο δίκτυο. Όταν η απάντηση από mode με ID=534 (π.χ. Temperature_309 – Το θερμόμετρο του κυρίου Malade) φτάσει στη διεπαφή, θα αποσταλεί πίσω στον WSN Driver σαν «ακατέργαστα» δεδομένα. Τελικά αυτή η απάντηση θα μεταφραστεί σε USL (θα χτιστεί με XML δηλαδή) και θα μοιάζει κάπως έτσι:

Πίνακας 5.6: TinyDB απάντηση προηγούμενου παραδείγματος

```
<Response xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" WSNResponse.xsd" ID="123456">
   <SensorValues>
      <SensorValue function=”NoFunction” type=”temperature”>37</SensorValue>
   </SensorValues >
   <ReturnStatus error=”false”>   </ReturnStatus>
</Response>
```

Αυτή η USL απάντηση θα φτάσει πίσω στον RR-Proxy, όπου υπάρχει ένας κατάλληλος listener και τελικώς θα επιστρέψει στο WSN API (και συνεπώς και στην εφαρμογή διαχείρισης) αφού δεν έχει συμβεί κάποιο λάθος.

6

Υλοποίηση

Σε αυτήν την ενότητα παρουσιάζεται το κομμάτι της υλοποίησης της πτυχιακής εργασίας. Πρόκειται για τον WSN Driver που παραλαμβάνει μια αίτηση και κατόπιν επεξεργασίας την προωθεί στο υποκείμενο δίκτυο. Αντίστροφα δέχεται τις απαντήσεις από το δίκτυο και τις προωθεί στον χρήστη. Ο driver αυτός, δουλεύει για την ειδική περίπτωση που το υποκείμενο δίκτυο υποστηρίζει το TinyDB.

6.1 Ανάλυση Απαιτήσεων

Μια γενική εικόνα του WSN Driver θα ήταν η εξής. Παραλαμβάνει από κάποιον χρήστη κάποιο request το οποίο είναι δομημένο σε μια εύληπτη και αναμενόμενη μορφή, το επεξεργάζεται ώστε να εξάγει κάποια συμπεράσματα σχετικά με το τι ζητάει ο χρήστης και αν αυτά που ζητάει έχουν νόημα και υποστηρίζονται από το δίκτυο, το προωθεί μέσω του TinyDB στους αισθητήρες και συλλέγει τις απαντήσεις. Στη συνέχεια και αφού οι απαντήσεις έχουν συλλεχθεί, τις φιλτράρει, τις επεξεργάζεται και κατασκευάζει μια εύληπτη και αναμενόμενη απάντηση προς τον χρήστη. Αν αυτά που ζητάει ο χρήστης δεν έχει νόημα να προωθηθούν στο δίκτυο, η απάντηση θα δομηθεί πιο γρήγορα και θα περιλαμβάνει μια σαφή ένδειξη για το λάθος. Πιο συγκεκριμένα απαιτείται να ικανοποιούνται τα εξής:

- Χρήση μιας κοινά αποδεκτής γλώσσας για τη μορφοποίηση της αίτησης και της απάντησης. Για τη συγκεκριμένη απαίτηση έχει επιλεγεί η USL (Unified Sensor Language) που είναι βασισμένη στην XML και προδιαγεγραμμένη με XML Schemas. Αυτή η γλώσσα παρέχει μια σειρά από δυνατότητες (όπως η χρήση περιορισμών, ορίων κ.α.) στον προγραμματιστή και καλύπτει πλήρως τις ανάγκες του Driver.
- Σημασιολογική ανάλυση ενός request. Τα requests που δεν ευσταθούν λογικά πρέπει να απορρίπτονται αυτόματα για να μην γίνονται άσκοπες επερωτήσεις προς το δίκτυο ή τη βάση.
- Έλεγχος υποστήριξης των χαρακτηριστικών που εμπεριέχονται σε ένα request. Αν τα συγκεκριμένα δεδομένα δεν υποστηρίζονται από το δίκτυο, το request πρέπει να απορρίπτεται αυτόματα.
- Δυνατότητα εξυπηρέτησης πολλών requests ταυτόχρονα ώστε να μην υπάρχουν άσκοπες καθυστερήσεις
- Επεκτασιμότητα. Το σώμα του WSN Driver είναι χτισμένο και χωρισμένο κατά τέτοιο τρόπο που είναι πολύ εύκολο κάποιος να προσθέσει ή να αφαιρέσει ελέγχους που μελλοντικά μπορεί να

φανούν χρήσιμοι. Αποτελείται από μια σειρά κλάσεων που κάθε μια αναλαμβάνει ένα ρόλο και εμπεριέχει μια σειρά από μεθόδους που κάνουν ακριβώς μια εργασία η κάθε μια.

- Μεταφερσιμότητα (Portability). Για το λόγο αυτό έχει επιλεγεί η αντικειμενοστραφής γλώσσα προγραμματισμού java.
- Σε περίπτωση λάθους, ο χρήστης να λαμβάνει χωρίς καθυστερήσεις μια απάντηση με τη σχετική περιγραφή του λάθους.

Φυσικά υπάρχουν διάφοροι περιορισμοί που προκύπτουν κατά την επεξεργασία της πληροφορίας, των οποίων η ανάλυσή γίνεται λεπτομερώς στο παράρτημα Α «Λεπτομέρειες Υλοποίησης» σε κάθε υποενότητα ξεχωριστά. Όπως φαίνεται και στο γενικό διάγραμμα κλάσεων (Σχήμα 6.1), ο Driver αποτελείται από τις βασικές κλάσεις:

- **Receiver** που αναμένει και παραλαμβάνει τις αιτήσεις τις οποίες προωθεί για επεξεργασία και αντιστρόφως αναμένει απαντήσεις τις οποίες και προωθεί στο χρήστη
- **InfoExtractor** που εξάγει τις πληροφορίες του request και χτίζει μια σειρά χρήσιμων αντικειμένων που διευκολύνουν την επεξεργασία
- **SupportChecker** που εξετάζει αν ζητούνται πληροφορίες που δεν υποστηρίζονται από το συγκεκριμένο δίκτυο
- **SemanticChecker** που αναλύει σημασιολογικά το request
- **QueryInjector** που εισάγει την επερώτηση στο δίκτυο
- **DemoApp** που συνεργάζεται με το TinyDB για την αποστολή της επερώτησης και τη λήψη της απάντησης
- **MonitorHandler** που χειρίζεται περιπτώσεις επαναληπτικής εισαγωγής μιας επερώτησης στο δίκτυο με κάποια συγκεκριμένη περίοδο
- **OdsRequestor** που πραγματοποιεί αιτήσεις προς τη βάση,

- **EventHandler** που χειρίζεται περιπτώσεις που ζητάμε να γίνει κάποια ενέργεια σε περίπτωση που συμβεί κάποιο γεγονός
- **ResponseHandler** που παραλαμβάνει απαντήσεις και φροντίζει ώστε να φτιάχνει μια USL απάντηση και να τη στέλνει πίσω στον receiver για να την αποστείλει στον χρήστη.

Παράλληλα, χρησιμοποιεί μια σειρά βοηθητικών κλάσεων για την μεταφορά και ομαδοποίηση δεδομένων κατά τα διάφορα στάδια της επεξεργασίας. Αυτές είναι οι: **RequestedInfo, QueryFilter, Timestamp, SenExpr, Monitor, Event, MonitorInfo, Response, SensorValues, FinalAnswer, MonitorParts, LocationPlusNodeID, DevIdPlusNodeID.** Οι κλάσεις αυτές δεν περιλαμβάνουν κάποια λειτουργικότητα, γι'αυτό και δεν θα αναλυθούν περαιτέρω. Σε διάφορα σημεία όμως της λεπτομερούς ανάλυσης της υλοποίησης, θα γίνεται αναφορά σε αυτές.

6.2 Requestor

Η κλάση αυτή δεν ανήκει στον WSN Driver, αλλά αποτελεί μια δοκιμαστική έκδοση ενός client που κάνει αιτήσεις προς το δίκτυο και παραλαμβάνει απαντήσεις. Αρχικά συνδέεται με τον Wsn Driver και στη συνέχεια του παραδίδει ένα Usl Request, η μορφή του οποίου περιγράφηκε στην ενότητα 5.3. Όταν η απάντηση είναι έτοιμη, τότε αυτή αποστέλλεται στον Requetor από τον Wsn Driver υπό τη μορφή ενός Usl Response, όπως αυτό περιγράφηκε στην ενότητα 5.4.

6.3 Receiver

Η κλάση αυτή αποτελεί το σημείο επικοινωνίας ενός χρήστη με τον Wsn Driver. Πιο συγκεκριμένα αναμένει αιτήσεις από το χρήστη, τις οποίες όταν τις δεχτεί φροντίζει να τις προωθεί για επεξεργασία στην κλάση InfoExtractor.

Παράλληλα αναμένει απαντήσεις από τη κλάση ResponseHandler, τις οποίες και αποστέλλει στο χρήστη.

6.4 InfoExtractor

Η συγκεκριμένη κλάση δέχεται αιτήσεις (requests) από τον receiver και φροντίζει για την σωστή εξαγωγή της πληροφορίας που περιέχεται σε αυτά. Κάθε request είναι δομημένο κατά τα πρότυπα της USL (Unified Sensor Language), όπως αυτή περιγράφηκε στην ενότητα 5.2. Η εργασία αυτής της κλάσης έχει να κάνει με την εξαγωγή των δεδομένων που περιέχονται μέσα στο request και τη δημιουργία αντικειμένων που περιλαμβάνουν κομμάτια αυτής της πληροφορίας. Ο λόγος που γίνεται αυτό είναι για να γίνει σχετικά εύκολη η επεξεργασία των δεδομένων που εξήχθησαν στα στάδια της επεξεργασίας που ακολουθούν.

6.5 SupportChecker

Ο ρόλος της κλάσης αυτής είναι να ελέγχει αν μέσα στα δεδομένα που περιλαμβάνονται στο request υπάρχουν και κάποια τα οποία δεν υποστηρίζονται από το υποκείμενο δίκτυο και συνεπώς δεν έχουν νόημα για αυτό. Για παράδειγμα αν ο χρήστης έχει ζητήσει μέση θερμοκρασία και αυτό το μέγεθος δεν υποστηρίζεται από το συγκεκριμένο δίκτυο (γιατί για παράδειγμα οι αισθητήρες μετρούν μόνο φωτεινότητα), τότε δεν θα πρέπει να προωθηθεί καν στο δίκτυο. Αυτή είναι και η λογική του Driver. Να πραγματοποιεί τους ελέγχους με γνώμονα το δίκτυο για το οποίο παρέχεται στον χρήστη και αναλόγως να προωθεί ή όχι τις αιτήσεις του. Αν είχαμε 5 Drivers για 5 δίκτυα και ένα εξ'αυτών υποστηρίζει θερμοκρασία, τότε αν το ίδιο request προωθηθεί σε όλους τους drivers, μόνο ο ένας θα επιστρέψει αποτελέσματα. Επίσης η όλη διαδικασία έχει και σημασιολογική χροιά. Αν για παράδειγμα ο requestor παραδώσει ένα αίτημα στο οποίο ζητάει δεδομένα που υποστηρίζονται (μέσα στην οντότητα RequestedInfo), αλλά στην οντότητα QueryFilter περιέχει περιορισμούς για δεδομένα που δεν υποστηρίζονται (π.χ. θέλω θερμοκρασία και υγρασία για τους αισθητήρες στους οποίους η φωτεινότητα είναι 50 και δεν υποστηρίζεται η

φωτεινότητα), τότε δεν έχει νόημα να προωθηθεί το αίτημα, αφού δεν ευσταθεί για το συγκεκριμένο δίκτυο.

Παρόμοια είναι και η διαδικασία με τις περιοχές. Σε περίπτωση που ζητηθεί location=”room10” και το room10 δεν συμπεριλαμβάνεται στο δίκτυο (δεν υπάρχουν αισθητήρες αυτού του δικτύου εκεί), τότε δεν πρέπει να επιτραπεί η προώθηση της αίτησης στο δίκτυο. Σε κάθε περίπτωση, αν παραβιάζεται κάποιος περιορισμός διακόπτεται την επεξεργασία, ενώ αποστέλλεται κατάλληλο μήνυμα τον ResponseHandler, ο οποίος φροντίζει για την σύνταξη της ανάλογης απάντησης προς τον requestor.

Φυσικά υπάρχει το ζήτημα του πως μπορεί να γνωρίζει ο SupportChecker ποιοί τύποι δεδομένων υποστηρίζονται. Στην συγκεκριμένη υλοποίηση έχουμε θεωρήσει μια hardcoded καταγραφή των δεδομένων αυτών. Στη πραγματικότητα όμως, ο SupportChecker ζητά από τα δεδομένα αυτά από το Profile Register, τα οποία στη συνέχεια και φροντίζει να τα τοποθετήσει σε ανάλογες ουρές. Το request, μαζί με τους ξεχωριστούς πίνακες για τα locations, τα functions και τα types, έχουν προωθηθεί στον SupportChecker μέσω της μεθόδου putInfo (η οποία ξυπνάει το Thread του για να ξεκινήσει ο έλεγχος).

Ο έλεγχος για το αν υποστηρίζονται τα functions γίνεται αυτόματα στο προηγούμενο στάδιο κατά τη διάρκεια του parsing, όμως έχει συμπεριληφθεί και σε αυτό το σημείο, παρόλο που δεν έχει ενεργοποιηθεί, για τυχόν μελλοντικές επεκτάσεις που απαιτούν αυτόν τον έλεγχο.

Στη συνέχεια και ενώ η επεξεργασία αυτή έχει τελειώσει, το request προωθείται στο επόμενο στάδιο για σημασιολογική ανάλυση.

6.6 SemanticChecker

Ο SemanticChecker αποτελεί τη καρδιά του WSN Driver καθώς είναι υπεύθυνος για την σημασιολογική επεξεργασία των δεδομένων που περιέχονται στο request. Η σημασία του έγκειται στο γεγονός πως είναι αρκετά εύκολο να

γίνει κάποιο λάθος από τη πλευρά του χρήστη αναφορικά με αυτά που ζητάει, καθώς υπεισέρχεται ο ανθρώπινος παράγοντας.

Αναλυτικότερα, ο SemanticChecker αποτελείται από μια σειρά βημάτων τα οποία εξασφαλίζουν πως κάποιο αίτημα (του οποίου τα εμπλεκόμενα στοιχεία υποστηρίζονται από το δίκτυο καθώς έχουν περάσει από τον έλεγχο υποστήριξης με επιτυχία) έχει νόημα να εισαχθεί στο δίκτυο υπό την μορφή επερώτησης. Παράλληλα, κάνει και μια επιπρόσθετη εργασία. Φροντίζει για την δημιουργία της SQL-Like επερώτησης που θα εισαχθεί στο δίκτυο, κατά τα πρότυπα (και τους περιορισμούς) του TinyDB, όπως περιγράφονται στο παράρτημα Β. Τα βήματα που ακολουθούνται κατά τον σημασιολογικό έλεγχο είναι :

- **Έλεγχος των ορίων** (μέγιστο και ελάχιστο) κάποιων παραμέτρων της αίτησης (π.χ. αν ο χρήστης ζητάει πληροφορίες για κάποιο κόμβο στον οποίο η μέση θερμοκρασία είναι μεγαλύτερη από 20.000 ^{o}C, ενώ ξέρουμε πως οι αισθητήρες μετρούν θερμοκρασίες μέχρι 1000 ^{o}C, τότε θα πρέπει να σταματήσει η επεξεργασία).
- **Έλεγχος αντιθέσεων** στα δεδομένα που ζητούνται. Ελέγχεται δηλαδή αν στους περιορισμούς του request συγκαταλέγονται και περιορισμοί που μπορεί μεν αυτόνομοι να στέκουν, δεν μπορούν όμως να ισχύουν ταυτόχρονα για έναν αισθητήρα. Για παράδειγμα έστω ότι το request απαιτούσε τη μέση θερμοκρασία των αισθητήρων για τους οποίους η επιτάχυνση είναι ταυτόχρονα μεγαλύτερη από 30 και μικρότερη από 10 μονάδες. Όπως είναι λογικό μια τέτοια κατάσταση δεν θα μπορούσε να υπάρξει ποτέ και θα ήταν άσκοπη η προώθηση του συγκεκριμένου request στο δίκτυο.
- **Έλεγχος χρονικών παραμέτρων** των δεδομένων που περιέχονται στα τμήματα που αφορούν σε επερωτήσεις προς τη βάση (TimeStamps) και σε εκείνα που αφορούν επερωτήσεις τύπου monitor. Αυτού του τύπου οι παράμετροι περιγράφονται στην ενότητα 5.3, ενώ ο τρόπος με τον οποίο γίνονται οι έλεγχοι αναλύεται στο παράρτημα Α. Συνοπτικά αξίζει να

αναφερθεί πως οι χρονικές παράμετροι δεν ευσταθούν, όταν ζητάμε ημερομηνίες που δεν υπάρχουν, ή όταν οι ημερομηνίες δεν βγάζουν νόημα (π.χ. όταν ζητάμε η ημερομηνία έναρξης μιας επερώτησης να είναι μετά από την ημερομηνία λήξης της).

- **Έλεγχος Συναρτήσεων.** Δεδομένων κάποιων περιορισμών του TinyDB, δεν επιτρέπεται στο ίδιο request να περιέχονται χρονικές και χωρικές συναρτήσεις ταυτόχρονα.

6.7 QueryInjector

Πρόκειται για την κλάση εκείνη που συνεργάζεται με την DemoApp (η οποία επικοινωνεί με το TinyDB και συνεπώς συνεργάζεται με το υποκείμενο ασύρματο δίκτυο αισθητήρων) και κάνει τα εξής: Παραλαμβάνει μια επερώτηση συμβατή με τα πρότυπα του TinyDB και φροντίζει για την αποστολή του προς το δίκτυο και στη συνέχεια τη συλλογή των απαντήσεων. Όπως έχει αναφερθεί και στα προηγούμενα κεφάλαια, ζητούνται από το δίκτυο τα επιθυμητά στοιχεία από όλους τους κόμβους (motes) και στη συνέχεια πραγματοποιείται φιλτράρισμα των επιθυμητών αποτελεσμάτων και εφαρμογή της επιθυμητής συνάρτησης πάνω στα δεδομένα αυτά. Όταν όλα τα δεδομένα έχουν επιστραφεί από το δίκτυο και έχουν γίνει όλες οι πράξεις πάνω σε αυτά (φιλτράρισμα και εφαρμογή της συνάρτησης), τότε αυτά αποστέλλονται στην κλάση ResponseHandler για το χειρισμό της απάντησης.

6.8 OdsRequestor

Η κλάση αυτή χειρίζεται τις περιπτώσεις που ζητάμε δεδομένα από παρελθοντικούς χρόνους. Δέχεται όλες τις απαραίτητες πληροφορίες για την δημιουργία μιας SQL επερώτησης την οποία θα προωθήσει προς το σύστημα ODS (της βάσης δεδομένων) και στη συνέχεια θα πραγματοποιήσει μια αντίστοιχη διαδικασία προώθησης της απάντησης προς τον χρήστη, μέσω του

ResponseHandler. Η λειτουργικότητα της κλάσης αυτής δεν έχει ενσωματωθεί (καθώς ξεφεύγει από τους σκοπούς αυτής της πτυχιακής εργασίας), έχει δημιουργηθεί όμως ο κορμός της προκειμένου να είναι εύκολα υλοποιήσιμη η επιθυμητή λειτουργικότητα.

6.9 EventHandler

Η κλάση αυτή χειρίζεται περιπτώσεις αναμονής κάποιου γεγονότος, που όταν συμβεί θα αποσταλεί ένα κατάλληλο μήνυμα στον χρήστη. Η λειτουργικότητα της κλάσης αυτής δεν έχει ενσωματωθεί (καθώς ξεφεύγει από τους σκοπούς αυτής της πτυχιακής εργασίας), έχει δημιουργηθεί όμως ο κορμός της προκειμένου να είναι εύκολα υλοποιήσιμη η επιθυμητή λειτουργικότητα.

6.10 MonitorHandler

Η κλάση αυτή ασχολείται με την περιοδική προώθηση μιας αίτησης προς το ασύρματο δίκτυο αισθητήρων, ξεκινώντας σε μια δεδομένη χρονική στιγμή και σταματώντας σε μια δεύτερη. Η περίοδος της επανάληψης καθώς και οι χρόνοι έναρξης και λήξης ορίζονται στα στοιχεία της αίτησης. Η διάφορα της με την κλάση QueryInjector είναι πως εδώ δεν παίρνουμε για ένα request μια φορά αποτελέσματα, αλλά πολλές. Παράλληλα η ίδια δουλειά μπορεί να γίνεται και για άλλα requests, όπότε να στέλνονται διαρκώς διαφορετικές επερωτήσεις προς το δίκτυο και στη συνέχεια οι απαντήσεις να πρέπει να ξεχωρίζονται αναλόγως με το ποίο request αφορούν. Για παράδειγμα ας θεωρήσουμε ότι έχουμε τρεις επερωτήσεις των οποίων οι χρόνοι έναρξης και λήξης καθώς και οι περίοδοι, φαίνονται στον πίνακα 6.1

Πίνακας 6.1: Επερωτήσεις που περιμένουν να εξυπηρετηθούν

Επερώτηση	StartDate	StartTime	StopDate	StopTime	Period
A	10-10-2005	11:00:00	10-10-2005	21:00:00	15 ms
B	10-10-2005	11:00:05	10-10-2005	21:00:05	5 ms
Γ	10-10-2005	11:00:10	10-10-2005	21:00:10	3 ms

Αν υποθέσουμε πως ο χρόνος από τη στιγμή που θα προωθηθεί μια επερώτηση, μέχρι τη στιγμή που θα επιστραφούν αποτελέσματα είναι σταθερός, ίσος με 5 ms, τότε η σειρά με την οποία θα προωθούνται οι επερωτήσεις και που θα λαμβάνονται απαντήσεις φαίνεται στον πίνακα 6.2:

Πίνακας 6.2: Χρονικές στιγμές αποστολής και λήψης δεδομένων

Χρονική Στιγμή	Προώθηση Επερώτησης	Χρονική Στιγμή	Λήψη Απάντησης
11:00:00	A	11:00:05	A
11:00:05	B	11:00:10	B
11:00:10	Γ	11:00:15	Γ
11:00:10	B	11:00:15	B
11:00:13	Γ	11:00:18	Γ
11:00:15	B	11:00:20	B
11:00:15	A	11:00:20	A
11:00:16	Γ	11:00:21	Γ
11:00:19	Γ	11:00:24	Γ

Παρατηρώντας τους παραπάνω πίνακες, γίνεται εύκολα αντιληπτό πως πρέπει να κρατάμε μια αντιστοιχία ανάμεσα στις επερωτήσεις και στις απαντήσεις και μάλιστα κάθε επερώτηση πρέπει να διαθέτει μια ουρά στην οποία θα τοποθετούνται όλα τα αποτελέσματα που έρχονται από το δίκτυο. Στη

συνέχεια για κάθε στοιχείο, πραγματοποιείται φιλτράρισμα και υπολογισμός της αντίστοιχης συνάρτησης, ακριβώς όπως και στον QueryInjector. Στην επόμενη σελίδα φαίνεται το διάγραμμα καταστάσεων του MonitorHandler.

6.11 ResponseHandler

Η κλάση αυτή φροντίζει για την λήψη των μηνυμάτων λάθους ή των απαντήσεων και την δημιουργία μιας απάντησης κατά τα πρότυπα του USL Response όπως αυτό περιγράφεται στην αντίστοιχη ενότητα παραπάνω. Αν ο ResponseHandler λάβει ένα μήνυμα λάθους πριν γίνει η προώθηση της αίτησης στο δίκτυο, τότε η απάντηση που θα περιγράφει το λάθος θα φτάσει πολύ γρήγορα στον χρήστη. Αν όμως γίνει η προώθηση της αίτησης στο δίκτυο, είτε μέσω του QueryInjector, είτε μέσω του MonitorHandler, είτε μέσω του EventHandler, τότε ο χρόνος που θα περάσει μέχρι να επιστρέψει η απάντηση εξαρτάται από το χρόνο που θα κάνει να έρθει από το δίκτυο.

7

Παράρτημα Α – Λεπτομέρειες Υλοποίησης

Σε αυτήν την ενότητα γίνεται μια αναλυτική παρουσίαση της κάθε κλάσης της υλοποίησης, προκειμένου να χρησιμοποιηθεί σαν τεκμηρίωση (documentation) για τον αντίστοιχο κώδικά του WSN Driver.

7.1 Requestor

Στην κλάση αυτή πραγματοποιούνται τα εξής:

- Δημιουργείται ένα SocketChannel, για το οποίο εγκαθίσταται ένας selector (listener) που ρυθμίζεται να παρακολουθεί συμβάντα Read, Write και Connect. Η επικοινωνία καθορίζεται ως non-blocking μέσω της εντολής channel.configureBlocking(false) και πραγματοποιείται η σύνδεση.
- Παράλληλα δημιουργούνται ένας *Encoder* και ένας *Decoder* (κωδικοποιητής και αποκωδικοποιητής, αντίστοιχα) για την μετάφραση των δεδομένων που γράφονται στο socket από χαρακτήρες σε bytes, ή αυτών που διαβάζονται από αυτό από bytes σε χαρακτήρες.
- Διαβάζεται από ένα εξωτερικό αρχείο κάποιο request.
- Σε περίπτωση που το κανάλι είναι connectable και η σύνδεση δεν έχει ολοκληρωθεί, ολοκληρώνεται και έτσι έχει πραγματοποιηθεί σύνδεση με τον receiver.
- Σε περίπτωση που υπάρχει δυνατότητα γραψίματος στο κανάλι και ο requestor έχει κάποιο request για να στείλει, τότε το κωδικοποιεί από χαρακτήρες σε bytes και το αποστέλλει, γράφοντάς το στον αντίστοιχο buffer.
- Σε περίπτωση που υπάρχει δυνατότητα ανάγνωσης από το κανάλι, ο requestor το διαβάζει και το αποκωδικοποιεί από bytes σε χαρακτήρες διαβάζοντάς το από τον αντίστοιχο buffer. Μια πλήρης δοσοληψία έχει σε αυτό το σημείο πραγματοποιηθεί.

Επειδή δεν αποτελεί μέρος του WSN Driver δεν θα αναλυθεί περεταίρω.

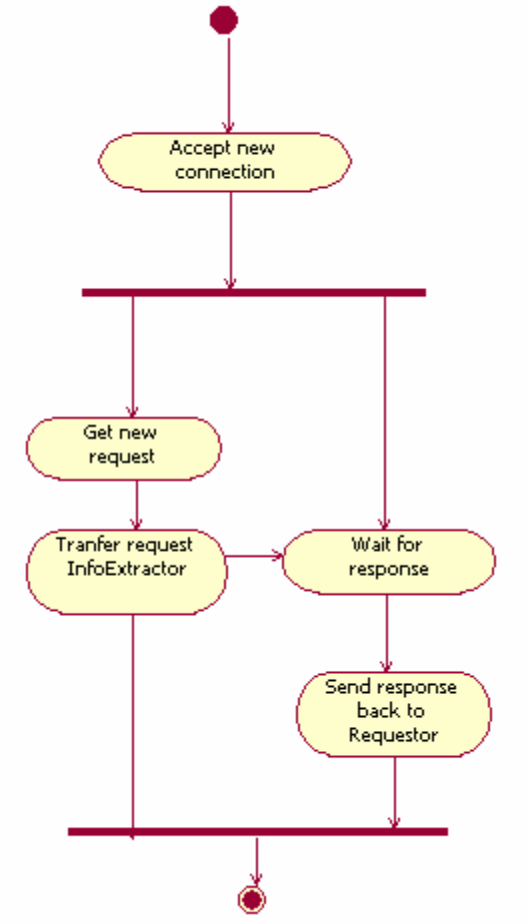
7.2 Receiver

Η εργασία αυτής της κλάσης είναι η δημιουργία ενός *ServerSocketChannel,* η υποδοχή αιτήσεων και η προώθησή τους για επεξεργασία στον InfoExtractor. Αντιστρόφως, δέχεται απαντήσεις για τις διάφορες αιτήσεις από τον ResponseHandler και τις προωθεί στον Requestor. Αναλυτικότερα, ακολουθείται μια σειρά βημάτων προκειμένου να επιτευχθεί η επιθυμητή λειτουργικότητα.

- Αρχικά δηλώνεται το *port* στο οποίο θα αναμένει συνδέσεις ο server και το νήμα (thread) ξεκινά να τρέχει.
- Δηλώνεται ένας *selector* (listener για αιτήσεις από clients)
- Δημιουργείται ένα *ServerSocketChannel*.
- Μέσω της εντολής *channel.configureBlocking(false)* η επικοινωνία ορίζεται ως non-blocking. Δηλώνεται επίσης η Internet διεύθυνση που αντιστοιχεί στο παραπάνω *port* και στη συνέχεια «δένεται» (εντολή *bind*) με το socket το οποίο δημιουργήθηκε.
- Δημιουργείται επίσης ένας *Encoder* και ένας *Decoder* (κωδικοποιητής και αποκωδικοποιητής αντίστοιχα) για την μετάφραση των δεδομένων που γράφονται στο socket από χαρακτήρες σε bytes, ή αυτών που διαβάζονται από αυτό από bytes σε χαρακτήρες.
- Ο *selector* αναμένει αιτήσεις.
- Σε περίπτωση που βρεθεί acceptable (αποδεκτή) σύνδεση, τότε για το συγκεκριμένο κανάλι (*channel*), δημιουργείται ένα απλό socket που αποδέχεται (*accept*) τη σύνδεση, ενώ παράλληλα το κανάλι αυτό δηλώνεται για Read και Write επικοινωνία.
- Σε περίπτωση που ο client (requestor) έχει γράψει κάτι στο socket, τότε αυτό αποκωδικοποιείται, μετατρέπεται σε συμβολοσειρά και στη περίπτωση που πρόκειται για δύο ή περισσότερες αιτήσεις

μαζί, αυτές χωρίζονται με ένα ειδικό σύμβολο (π.χ. το “@”) που σηματοδοτεί το τέλος μιας αίτησης και την αρχή μιας επόμενης. Εννοείται πως αυτό το σύμβολο δεν επιτρέπεται να εμφανίζεται μέσα σε κάποιο request. Αφού μετατραπεί σε συμβολοσειρά προωθείται παρακάτω για επεξεργασία.

- Σε περίπτωση που υπάρχει κάποια απάντηση στην ουρά με τις απαντήσεις που έχουν επιστραφεί από το δίκτυο, τότε αυτή κωδικοποιείται και γράφεται στο socket.
- Η μέθοδος *putrequest* καλείται από τον ResponseHandler και γράφει στην ουρά με τις απαντήσεις μια νέα απάντηση.

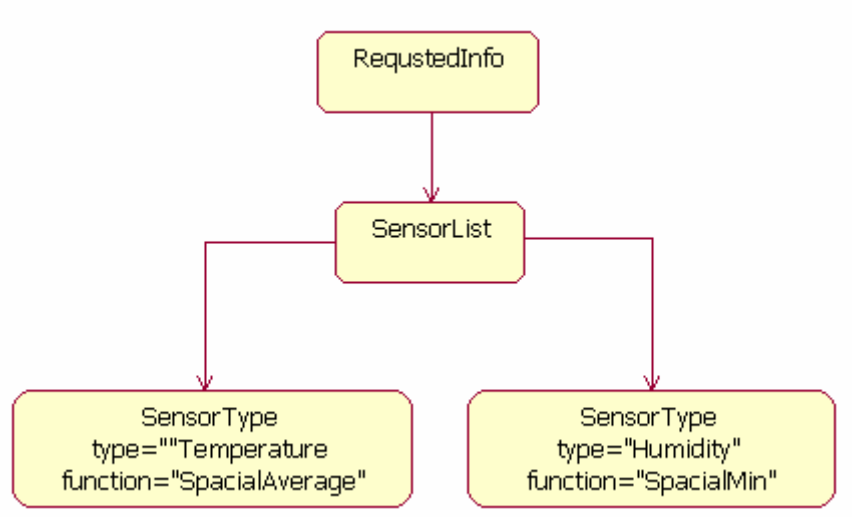


Σχήμα 7.2: Διάγραμμα δραστηριοτήτων του Receiver

Στο παραπάνω διάγραμμα βλέπουμε τις πιθανές δραστηριότητες του receiver.

7.3 InfoExtractor

Η συγκεκριμένη κλάση δέχεται αιτήσεις (requests) από τον Receiver και φροντίζει για την σωστή εξαγωγή της πληροφορίας που περιέχεται σε αυτές. Κάθε request είναι δομημένο κατά τα πρότυπα της USL. Για παράδειγμα σε περίπτωση που θέλουμε να μάθουμε την μέση θερμοκρασία και την ελάχιστη υγρασία στο δίκτυο, θα περικλείουμε τα συγκεκριμένα χαρακτηριστικά στο tag που αναφέρεται στις πληροφορίες που θέλουμε να πάρουμε από το δίκτυο (πρόκειται για την οντότητα SensorType που βρίσκεται μέσα σε μια λίστα με τέτοιες οντότητες που λέγεται SensorList). Το tag αυτό περιέχει δύο μεταβλητές (attributes), την μεταβλητή type που παίρνει προκαθορισμένες τιμές ανάλογα με τους διαφορετικούς τύπους αισθητήρων που θεωρούμε πως υπάρχουν (προσοχή, όχι



Σχήμα 7.3: Δομή της οντότητας RequestedInfo

αυτούς που κατ'ανάγκη υποστηρίζονται από το συγκεκριμένο δίκτυο) και το function που καθορίζει την συνάρτηση που θα εφαρμοστεί πάνω στα δεδομένα. Για να υλοποιήσουμε το παραπάνω παράδειγμα, θα χρησιμοποιήσουμε τα εξής ζευγάρια (function: Spacial*Average* – type: Temperature) και (function: SpacialMin – type: Humidity). Το tag του RequestedInfo θα είναι κάπως έτσι:

```
<RequestedInfo>
    <SensorList>
      <SensorType function="SpacialAverage" type="Temperature"/>
      <SensorType function="SpacialMin" type="Humidity"/>
    </SensorList>
</RequestedInfo>
```

Βέβαια, υπάρχουν και άλλα attributes τα οποία δεν αναφέρθηκαν στο συγκεκριμένο παράδειγμα και είναι απαραίτητα στο request, όμως το συγκεκριμένο παράδειγμα αποσκοπεί απλώς στο να δώσει μια γενική εικόνα του τρόπου που δομείται η πληροφορία. Επίσης υπάρχουν και άλλες ενδιαφέρουσες οντότητες που προσδιορίζουν ένα request (όπως το QueryFilter , το GroupBy και το Monitor). Η χρησιμότητα των οντοτήτων αυτών περιγράφεται στην ενότητα USL Request.

Για να γίνει η εξαγωγή της πληροφορίας που περιέχεται στο request, πρέπει πρώτα να εξεταστεί αν αυτό συμφωνεί με τους περιορισμούς και τη λογική δόμησης ενός έγκυρου request, κατά τα πρότυπα ενός εξωτερικού .xsd αρχείου (του *WSNRequest.xsd*). Πρόκειται για ένα XML Schema, βάση του οποίου θα γίνει ο έλεγχος εγκυρότητας (validation) του request. Το διάβασμα (parsing θα πραγματοποιηθεί με δύο ανεξάρτητους parsers (Sax και Dom) για. Τα βήματα που ακολουθούνται σε αυτό το στάδιο είναι τα εξής:

Parsing με τον SAX:

- Αρχικά δημιουργείται ένας XML Reader που θα κάνει το parsing
- Γίνονται κάποιες απαραίτητες ρυθμίσεις και ορίζεται εξωτερικό σχήμα, πάνω στο οποίο θα στηριχθεί το Parsing, το *WSNRequest.xsd*
- Εγκαθίσταται ένας ErrorHandler ο οποίος καθορίζει τι θα συμβεί σε περίπτωση λάθους.

- Πραγματοποιείται το parsing. Αν όλα είναι σωστά, τότε η επεξεργασία θα προχωρήσει παρακάτω (στο δεύτερο parsing αλλιώς θα προωθηθεί μήνυμα λάθους στον ResponseHandler, όπου και δομείται η απάντηση προς τον requestor)

Parsing με το Dom4j:

- Η λογική είναι η ίδια με τον SAX. Και πάλι εγκαθίσταται ένας parser και ένας ErrorHandler, ενώ δεν υπάρχουν ρυθμίσεις που πρέπει να αλλάξουν
- Πραγματοποιείται το parsing. Σε περίπτωση που υπάρξει λάθος, όπως και πριν θα προωθηθεί μήνυμα λάθους στον ResponseHandler.
- Το Parsing με το Dom4j δημιουργεί μια δενδροειδή μορφή για το request που μόλις ήλεγξε, ενώ παράλληλα παρέχει ένα πλούσιο σετ εντολών για την επεξεργασία των διαφόρων κόμβων του δένδρου. Αναφερόμαστε σε δένδρο, γιατί η ιεραρχική δομή της USL μας επιτρέπει να αντιμετωπίζουμε ένα request ως μια δενδρική δομή της οποίας οι κόμβοι είναι τα διάφορα elements. Αν το request είναι valid (έγκυρο) και well-formed (καλά δομημένο), τότε δεδομένης της οντότητας που βρίσκεται στην ρίζα του δένδρου θα εκτελεστεί η συνάρτηση walker. Η συνάρτηση αυτή σαρώνει το δένδρο και εξάγει τις πληροφορίες που περιέχονται σε αυτό για να προωθηθούν παρακάτω για επεξεργασία.

Walker:

Αρχικά θεωρούμε τη δημιουργία ενός καθολικού (global) αντικειμένου request, στο οποίο θα δομούνται σε χρήσιμη μορφή οι πληροφορίες που εξάγονται από το request και θα περιέχει τα εξής:

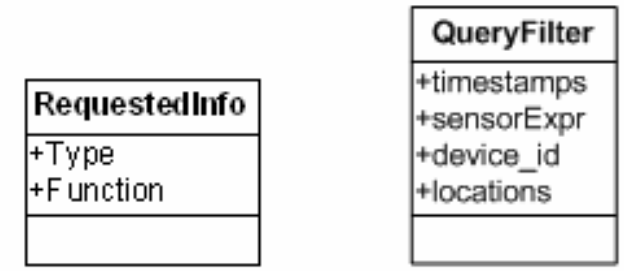
- Μια ουρά με όνομα *requested_info* στην οποία θα καταχωρηθούν οι επιθυμητές από τον requestor πληροφορίες (ζευγάρια *type* και *function*).
- Ένα αντικείμενο τύπου *QueryFilter* (με όνομα *query_fil*), του οποίου η δομή περιγράφεται πιο κάτω και περιέχει τους περιορισμούς που θέλουμε να ικανοποιούνται για τα δεδομένα που ζητάμε.
- Ένα αντικείμενο *Monitor* (με όνομα *monitor*) του οποίου η δομή επίσης περιγράφεται παρακάτω και χρησιμεύει για requests τύπου Monitor
- Ένα αντικείμενο *Event* (με όνομα ev) για την καταχώρηση των πληροφοριών που περιέχονται σε ένα Request που βασίζεται σε Events (π.χ. αν ζητάμε: «σε περίπτωση που η μέση θερμοκρασία γίνει μεγαλύτερη από 30 βαθμούς Κέλσιου κάνε κάτι», τότε πρέπει να εξαχθούν οι πληροφορίες SpacialAvg, Temperature, Greater και 30)
- Μια ουρά με το όνομα *GroupBy* (με όνομα *groupby*) για την αποθήκευση των δεδομένων, βάσει των οποίων θέλουμε να γίνει η ομαδοποίηση
- Και τέλος το *request_id* που περιγράφεται αμέσως μετά.

Σχήμα 7.4: Κλάσση Request

Η συνάρτηση αυτή όπως αναφέρθηκε παραπάνω σαρώνει το request - δένδρο και εξάγει τις πληροφορίες που περιέχονται σε αυτό. Αρχικά διαβάζει το element που βρίσκεται στη ρίζα (το οποίο πρέπει πάντα να είναι το element Request) και

εξάγει τη πληροφορία που περιέχεται στη μεταβλητή ID. Πρόκειται για ένα μοναδικό request id που προσδιορίζει το συγκεκριμένο request και χρησιμεύει για λόγους διάκρισης των διαφορετικών αιτήσεων από τον requestor (δεν πρέπει να ξεχνάμε πως ο requestor μπορεί να πραγματοποιεί requests χωρίς κατ'ανάγκη να έχει παραλάβει απάντηση από τα προηγούμενα). Η τιμή του αποθηκεύεται στη μεταβλητή *request_id* του αντικειμένου *request*. Στη συνέχεια και αφού έχει εξαχθεί το request id, δημιουργείται μια λίστα με τα παιδιά του request και για κάθε παιδί του εκτελείται η συνάρτηση αναδρομικά. Τα παιδιά του Request μπορεί να είναι είτε το Query, είτε το Event (για επεξεργασία πραγματικού χρόνου και για αναμονή κάποιου event αντίστοιχα). Επειδή ένα από τα δύο παιδιά μπορεί να υπάρχει στο Request, ορίζεται μια μεταβλητή (query_or_event) που διαχωρίζει τις δύο περιπτώσεις αναλόγως με τη τιμή στην οποία θα τεθεί. Αναδρομικά, όπως αναφέρθηκε και πιο πριν, θα τρέξουν τα παιδιά του κάθε κόμβου.

Αν επεξεργαζόμαστε το κόμβο παιδί RequestedInfo (στο οποίο περιέχονται οι επιθυμητές πληροφορίες που ζητάει ο requestor από το δίκτυο), τότε για κάθε SensorType που βρίσκεται, δημιουργείται ένα αντικείμενο τύπου *RequestedInfo*, το οποίο όπως φαίνεται και στο παρακάτω διάγραμμα περιέχει μια μεταβλητή type και μια μεταβλητή function.



Σχήμα 7.5: Κλάσεις RequestedInfo και QueryFilter

Το αντικείμενο αυτό τοποθετείται μέσα στην ουρά *requested_info* του αντικειμένου *request*. Η μεταβλητή type αποθηκεύεται επίσης στην ουρά types της κλάσης και η μεταβλητή function στην ουρά functions της κλάσης, για τον

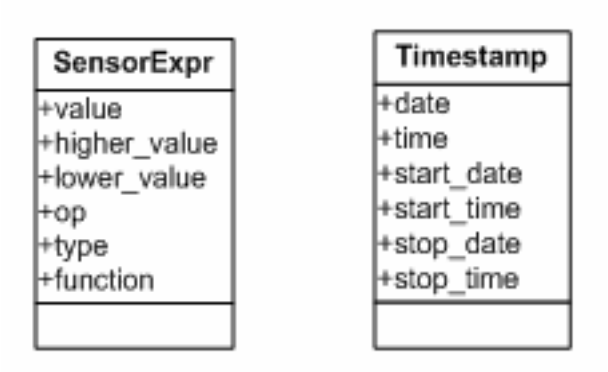
έλεγχο υποστήριξης που θα ακολουθήσει παρακάτω. Η συνάρτηση walker καταλαβαίνει πως βρίσκεται μέσα στο element RequestedInfo ως εξής. Όταν το συναντήσει (διαβάζει το όνομα του element στο οποίο βρίσκεται κάθε φορά) και πρόκειται να τρέξει αναδρομικά για τα παιδιά του, θέτει σε μια global μεταβλητή (με όνομα *metabliti*) μια μοναδική τιμή που προσδιορίζει αυτό το element. Έτσι μέχρι να τρέξει για όλα τα παιδιά του RequestedInfo (που περιέχονται στο SensorList για την ακρίβεια) εκτελεί αποκλειστικά ένα κομμάτι κώδικα που κάνει αυτά που αναφέρθηκαν παραπάνω.

Αν η συνάρτηση συναντήσει το element QueryFilter, το οποίο όπως αναφέρθηκε και προηγουμένως περιέχει τους περιορισμούς της αίτησης, τότε θέτει στην *metabliti* μια μοναδική τιμή που προσδιορίζει το QueryFilter (ακριβώς όπως έκανε παραπάνω) και ετοιμάζεται να γεμίσει το αντικείμενο query_fill του request. Συγκεκριμένα ένα αντικείμενο αυτού του τύπου περιλαμβάνει μια ουρά timestamps με τα στοιχεία των elements που αφορούν σε αιτήσεις με χρονικά στοιχεία (π.χ. «θέλω θερμοκρασίες από τις 20-4-2005 μέχρι τις 20-8-2005»), μια ουρά *sensorExpr* με τους περιορισμούς της αίτησης, το *device_id* που προσδιορίζει μια μοναδική συσκευή που διαθέτει πάνω της αισθητήρες (π.χ. ένα αυτοκίνητο) , καθώς μια ουρά *locations* όπου αποθηκεύονται τα στοιχεία των elements Location. Η δομή του φαίνεται στο παραπάνω διάγραμμα.

Αν κατά την αναδρομική κλήση της συνάρτησης συναντηθεί το element Device, τότε αποθηκεύεται η μοναδική του μεταβλητή *ID*, που αντιστοιχεί σε ένα μοναδικό device_id, το οποίο και αποθηκεύεται στην αντίστοιχη μεταβλητή του *query_fil*.

Αν τώρα τρέχοντας αναδρομικά για τα παιδιά του element QueryFilter συναντήσει κάποιο element SensorExpression, τότε θα δώσει σε μια μεταβλητή με όνομα QF_metabliti την τιμή 3, τιμή που προσδιορίζει ένα SensorExpression, θα δημιουργήσει ένα αντικείμενο τύπου *SensExpr* του οποίου η δομή φαίνεται στο παρακάτω διάγραμμα και θα αποθηκεύσει σε αυτό τα attributes του συγκεκριμένου element (function και type). Η μεταβλητή type αποθηκεύεται

επίσης στην ουρά types της κλάσης και η μεταβλητή function στην ουρά functions της κλάσης, για τον έλεγχο υποστήριξης που θα ακολουθήσει παρακάτω



Σχήμα 7.6: Κλάσεις SensorExpr και Timestamp

Επίσης τρέχοντας αναδρομικά, θα αποθηκεύσει στη μεταβλητή op το τύπο του περιορισμού που θα συναντήσει παρακάτω (Greater, GreateEqual, Within, Less κ.α.) υπό την μορφή παιδιών του SensorExpression, καθώς και τα απαραίτητα attributes που περιλαμβάνει ο περιορισμός (π.χ. *value=”50”* ή higher_value=”20”). Για παράδειγμα αν το αίτημα περιλαμβάνει τον περιορισμό η μέγιστη θερμοκρασία να κυμαίνεται μεταξύ 30 και 50 βαθμών, τότε το κομμάτι που θα σαρώσει η walker είναι κάπως έτσι:

<SensorExpr>

<SensorType function="SpacialAverage" type="Temperature">

<Within lower_value=”30” higher_value="50"/>

</SensorType>

</SensorExpr>

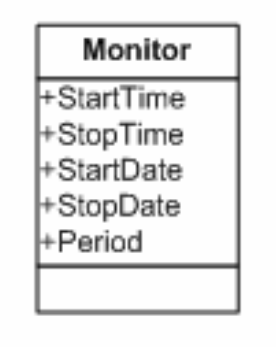
Αφού αποθηκευτούν όλα τα δεδομένα σε ένα αντικείμενο SensorExpr, τότε αυτό τοποθετείται στην ουρά *sensorExpr* του αντικειμένου query_fil. Η διαδικασία επαναλαμβάνεται μέχρι να εξαντληθούν όλα τα SensorExpression elements της αίτησης.

Αν κατά την σάρωση του XML document και ενώ ακόμα δεν έχουμε τελειώσει με τα παιδιά του element QueryFilter (η metabliti εξακολουθεί να έχει την τιμή 3) συναντήσουμε το element Timestamp, τότε ανατίθεται στην QF_metabliti η τιμή 2 και επαναλαμβάνεται παρόμοια διαδικασία, φτιάχνοντας αντικείμενα τύπου *Timestamp* και τοποθετώντας τα στην ουρά *Timestamps* του αντικειμένου query_fil. Η δομή ενός αντικειμένου Timestamp φαίνεται στο παραπάνω διάγραμμα.

Στη περίπτωση που συναντήσουμε element τύπου Location, τότε για κάθε παιδί αυτού του κόμβου (για κάθε element LocationName δηλαδή), αποθηκεύουμε την τιμή του στον πίνακα *locations* του *query_fil* καθώς και στον πίνακα *Locations* της κλάσης InfoExtractor γιατί θα χρησιμεύσει παρακάτω για τον έλεγχο υποστήριξης των συγκεκριμένων περιοχών.

Επίσης σε περίπτωση που συναντήσουμε το element GroupBy, το οποίο περιέχει μια λίστα από SensorTypes , τότε αποθηκεύουμε τις τιμές που περιέχονται στη μεταβλητή *type* για κάθε SensorType στην ουρά *GroupBy* του *request*, τιμές που αποθηκεύουμε και στο πίνακα *types* της κλάσης για τον έλεγχο υποστήριξης που θα γίνει παρακάτω. Οι τιμές που καταχωρούνται στην ουρά *GroupBy*, χρησιμεύουν για λόγους ομαδοποίησης της απάντησης που θα πάρουμε από το ασύρματο δίκτυο αισθητήρων, αφού βέβαια έχουμε στείλει μια SQL-Like επερώτηση, μέσω του TinyOS. Η λογική λοιπόν αυτού του element είναι η ίδια με της ομώνυμης εντολής της SQL.

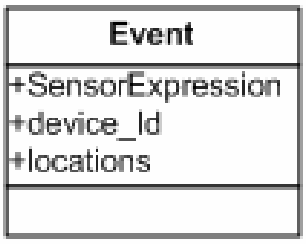
Επίσης, υπάρχει η πιθανότητα να συναντηθεί το element Monitor, που σηματοδοτεί εντολές αυτού του τύπου. Το Monitor αποτελείται υποχρεωτικά από τους κόμβους παιδιά StartTime και StopTime. Κατά την σάρωση αυτών και ενώ έχει δημιουργηθεί ένα global αντικείμενο τύπου Monitor (η δομή του φαίνεται στο παρακάτω διάγραμμα), αποθηκεύονται σε αυτό οι μοναδικές μεταβλητές που βρίσκονται στους κόμβους παιδιά (και αντιστοιχούν στα StartDate, StopDate, StartTime, StopTime και Period).



Σχήμα 7.7: Κλάση Monitor

Τα elements αυτού του είδους αντιστοιχούν σε επερωτήσεις προς το δίκτυο κατά τις οποίες θέλουμε να επαναλαμβάνεται η ίδια επερώτηση συνεχώς με περίοδο Period, ξεκινώντας από την ημερομηνία StartDate την ώρα StartTime και σταματώντας την ημερομηνία StopDate, την ώρα StopTime.

Σε περίπτωση που κατά την αρχική σάρωση, αντί για QueryFilter βρεθεί element Event, τότε θα έχουμε επερώτηση που αντιστοιχεί σε αναμονή για κάποιο event από το δίκτυο. Η λογική επεξεργασίας αυτού του element καθώς και των παιδιών του είναι ακριβώς η ίδια με του QueryFilter.

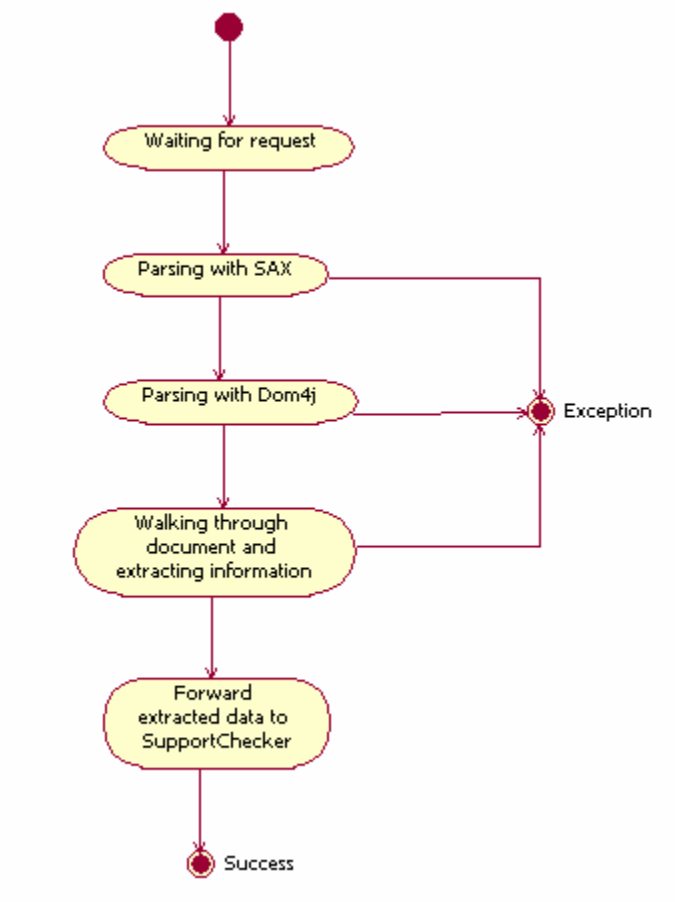


Σχήμα 7.8: Κλάση Event

Συγκεκριμένα έχουμε δημιουργήσει ένα global αντικείμενο τύπου *Event* (η δομή του φαίνεται στο παραπάνω διάγραμμα) με όνομα *ev* και ξεκινάμε να το γεμίζουμε με τις πληροφορίες που βρίσκουμε. Αφού τοποθετήσουμε όλες τις απαραίτητες πληροφορίες σε αυτό το αντικείμενο, το γράφουμε στο αντίστοιχο

αντικείμενο του *request*. Δεν χρειάζεται περαιτέρω ανάλυση του τρόπου εξαγωγής των συγκεκριμένων πληροφοριών, καθώς

Όταν τελειώσει η επεξεργασία και έχουν εξαχθεί όλες οι πληροφορίες που περιέχονταν στο συγκεκριμένο αίτημα, το αντικείμενο request μαζί με τις ουρές για τον έλεγχο υποστήριξης αποστέλλεται στο επόμενο βήμα (SupportChecker) για επεξεργασία.

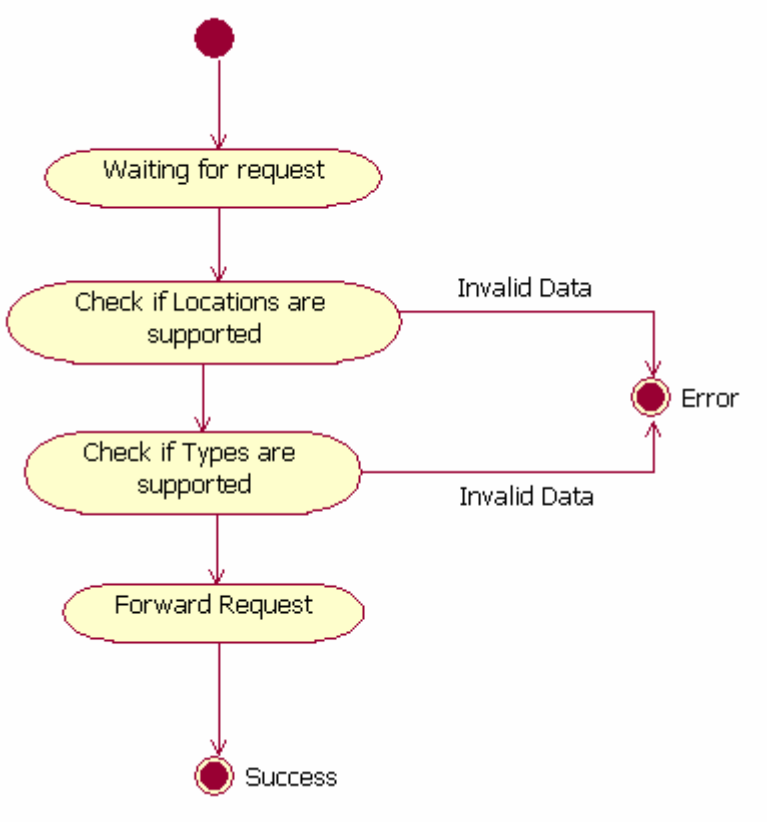


Σχήμα 7.9: Διάγραμμα δραστηριοτήτων της κλάσης InfoExtractor

Στο παραπάνω διάγραμμα δραστηριοτήτων, φαίνονται όλες οι πιθανές καταστάσεις που περιγράφηκαν προηγουμένως. Όπως παρατηρούμε πρόκειται για μια ακολουθιακή εκτέλεση κάποιων βημάτων από την στιγμή που υπάρχει κάποιο νέο request μέχρι τη στιγμή που θα προωθηθεί παρακάτω.

7.4 SupportChecker

Η λειτουργία αυτής της κλάσης είναι αρκετά απλή και ακολουθεί τα βήματα που φαίνονται στο σχήμα 7.10.

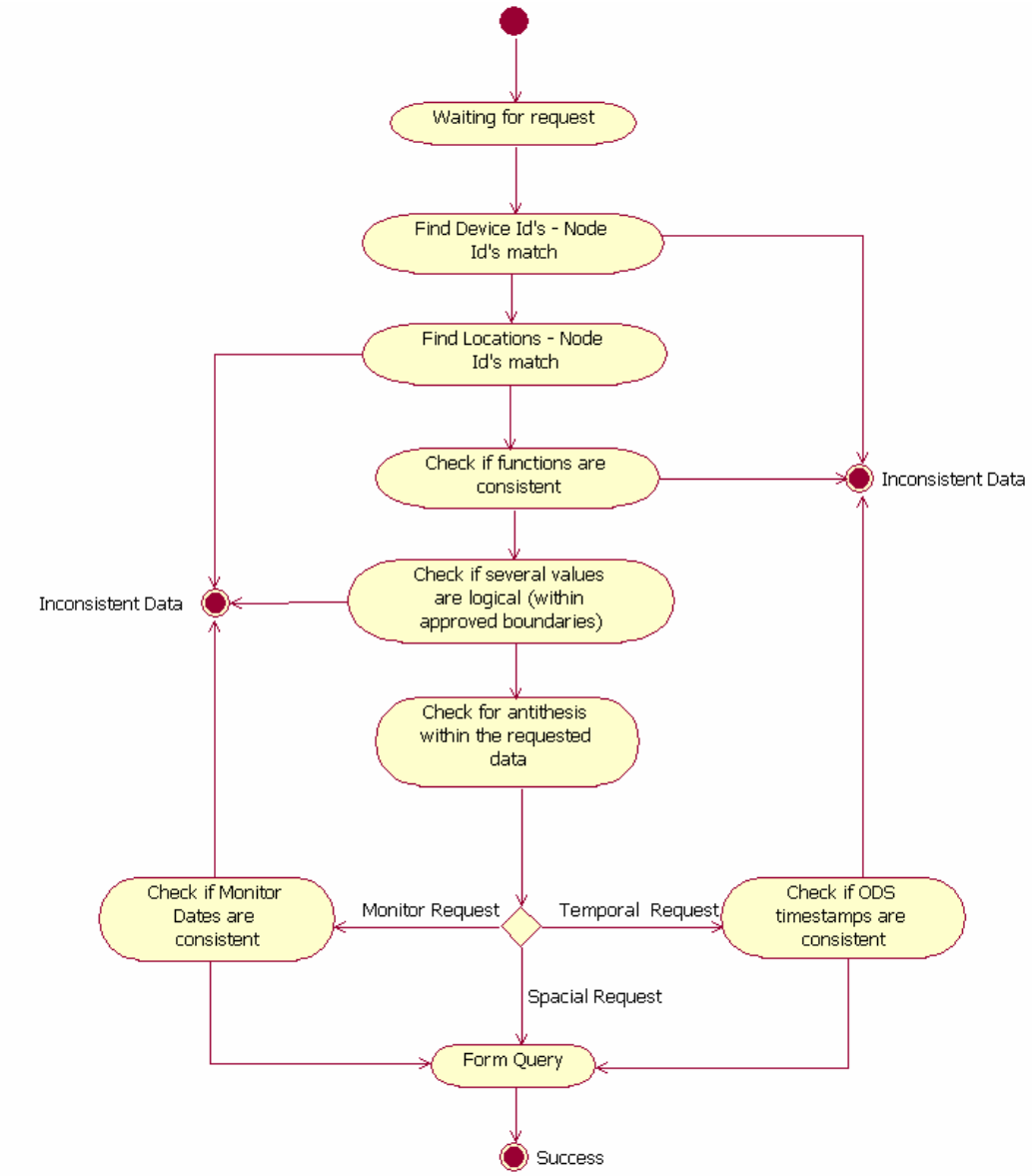


Σχήμα 7.10: Διάγραμμα δραστηριοτήτων της κλάσης SupportChecker

7.5 SemanticChecker

Αρχικά υπάρχει δηλωμένο ένα flag που προσδιορίζει την παρουσία ή μη κάποιου λάθους και συνεπώς καθορίζει αν η επεξεργασία μπορεί να συνεχίσει ή όχι. Το thread του SemanticChecker βρίσκεται σε κατάσταση αναμονής (wait), μέχρι να γραφτεί κάτι στην ουρά με τα requests, οπότε και θα ξυπνήσει για να αρχίσει την επεξεργασία. Το πρώτο πράγμα που κάνει από τη στιγμή που θα πάρει ένα request (υπό τη μορφή που το χτίσαμε στην κλάση InfoExtractor), είναι να εξάγει τις πληροφορίες που περιέχονται σε αυτό ώστε να μπορεί να τις

χειρίζεται εύκολα και γρήγορα. Στη συνέχεια ακολουθεί μια σειρά από κλήσεις σε συναρτήσεις που επιτελούν κάποιο σημασιολογικό έλεγχο.



Σχήμα 7.11: Διάγραμμα δραστηριοτήτων της κλάσης SemanticChecker

getMessage( )

Όση ώρα δεν υπάρχει κάτι στην ουρά requests για επεξεργασία, ο SemanticChecker βρίσκεται σε κατάσταση αναμονής. Σε περίπτωση που γραφτεί

κάτι στην ουρά, θα ξυπνήσει και θα ξεκινήσει η επεξεργασία. Το πρώτο από τα αντικείμενα που γράφτηκαν στην ουρά ανατίθεται στο global αντικείμενο req και στη συνέχεια ξεκινάει η επεξεργασία.

putRequest(*Request req1,Vector functions***)**

Παίρνει σαν ορίσματα ένα αντικείμενο τύπου request (που περιέχει δομημένες και ευδιάκριτες όλες τις απαραίτητες πληροφορίες ενός request) και μια ουρά functions, που περιέχει όλες τις συναρτήσεις που εμφανίζονται στο request. Το αντικείμενο req1 θα χρησιμοποιηθεί στην μέθοδο processRequest για την κατάτμηση της πληροφορίας και την κλήση των κατάλληλων μεθόδων για το χειρισμό κάθε πληροφορίας αντίστοιχα. Η ουρά functions χρησιμοποιείται από την μέθοδο checkFunctions.

processRequest(*int request_id***)**

Η επεξεργασία ξεκινά. Αρχικά γίνεται κατάτμηση των πληροφοριών που περιέχονται στο αντικείμενο req, σε ουρές και επιμέρους αντικείμενα, για την ευκολότερη επεξεργασία τους. Στη συνέχεια καλούνται μια προς μια οι μέθοδοι σημασιολογικού ελέγχου, με τη παρακάτω σειρά:

findDIds( ) και **findLIds( )**

Το TinyDB έχει κάποιους περιορισμούς όπως αναφέρθηκε και παραπάνω. Ένας από αυτούς είναι το γεγονός ότι δεν υποστηρίζει το λογικό or στις επερωτήσεις. Έτσι δεν θα μπορούσαμε να ζητήσουμε :(*location="room1" or location="room2" or location ="room3"*) . Μάλιστα στο συγκεκριμένο παράδειγμα και πάλι θα περιοριζόμασταν γιατί δεν υποστηρίζονται Locations στις επερωτήσεις του TinyDB. Προκειμένου να παρακάμψουμε αυτά τα εμπόδια, ζητάμε όλα τα δεδομένα που ικανοποιούν τους υπόλοιπους περιορισμούς του QueryFilter και στη συνέχεια αφού συγκεντρώσουμε αυτά που επιθυμούμε, κάνουμε την επεξεργασία μόνοι μας (π.χ. εύρεση μέσης, ελάχιστης και μέγιστης

τιμής ή εύρεση αριθμού αισθητήρων που ικανοποιούν το αίτημα). Ο τρόπος με τον οποίο γίνεται αυτό το φιλτράρισμα των απαιτούμενων απαντήσεων, στηρίζεται στην μορφή της απάντησης που επιστρέφεται από το TinyDB. Συγκεκριμένα μαζί με τις απαντήσεις που θα έρθουν από το δίκτυο, θέλουμε να επιστρέφεται και το node_id (κωδικός κάθε αισθητήρα που στέλνει απάντηση) Οι μέδοδοι *findDIds()* και *findLIds()* βρίσκουν για κάθε location ή κάθε device τα αντίστοιχα node_id's αν υπάρχουν και τα τοποθετούν σε μια ουρά, ώστε όταν έρθουν οι απαντήσεις από το δίκτυο, να συγκριθούν τα επιστρεφόμενα node_id's με τα επιθυμητά και να πραγματοποιηθεί το φιλτράρισμα. Σε περίπτωση που για κάποιο location ή device δεν υπάρχει αντίστοιχο node_id, τότε το flag παίρνει την τιμή 1 και σταματάει η επεξεργασία. Ο τρόπος λειτουργίας των παραπάνω μεθόδων είναι αρκετά απλός. Σαρώνουν ένα προς ένα όλα τα locations καθώς και το device αν υπάρχει και ελέγχουν μια ουρά με προκαθορισμένες αντιστοιχίες τιμών. Αν το πρώτο στάδιο τελειώσει επιτυχώς προχωράμε στο επόμενο μέρος της σημασιολογικής επεξεργασίας που αφορά στο αν οι συναρτήσεις που δηλώθηκαν είναι Temporal, Spatial ή NoFunction.

checkFunctions( )

Ένας δεύτερος περιορισμός του TinyDB είναι το γεγονός ότι στις επερωτήσεις προς το δίκτυο μέσω αυτού, δεν επιτρέπονται συναρτήσεις διαφορετικού τύπου. Συγκεκριμένα οι συναρτήσεις χωρίζονται σε Spatial, Temporal και NoFunction. Αν σε μια επερώτηση υπάρξουν συναρτήσεις από δύο ή τρεις διαφορετικούς τύπους (function="TempAverage" type="Temperature", function="SpatialMin" type="Humidity"), τότε θα υπάρξει λάθος στο TinyDB. Είναι λοιπόν απαραίτητος αυτός ο έλεγχος, που σε περίπτωση που αποκαλύψει παραβίαση αυτού του περιορισμού θα θέσει στο flag την τιμή 1 και θα σταματήσει η επεξεργασία. Εννοείται πως κάθε φορά που διακόπτεται η επεξεργασία αποστέλλεται στον ResponseHandler κατάλληλος κωδικός λάθους που ακολουθείται από ανάλογη μορφοποίηση απάντησης προς τον χρήστη (requestor). Ο τρόπος λειτουργίας της μεθόδου είναι ο εξής: Έχουμε 2 ουρές με

προαποθηκευμένες τις αντίστοιχες συναρτήσεις κάθε τύπου (Temporal και Spatial αντίστοιχα σε κάθε ουρά). Σε περίπτωση που βρούμε Temporal συνάρτηση, τροποποιούμε μια αρχικοποιημένη μεταβλητή, σε περίπτωση που βρούμε Spatial μια άλλη και σε περίπτωση που βρούμε NoFunction, μια τρίτη. Αν στο τέλος όλων των συναρτήσεων παραπάνω των μια μεταβλητών έχουν τροποποιηθεί, οι συναρτήσεις δεν είναι έγκυρες και η μεταβλητή flag παίρνει τη τιμή 1 ώστε να σταματήσει η επεξεργασία.

checkBoundaries(*String type,int max,int min***)**

Στη συνέχεια και εφόσον η επεξεργασία συνεχίζεται κανονικά, ελέγχουμε κάποια όρια τιμών που ζητούνται. Για παράδειγμα, υπάρχει περίπτωση στο *Request* να υπάρχει ο περιορισμός η θερμοκρασία να είναι μεγαλύτερη από 20.000 βαθμούς Κελσίου, γεγονός εξωπραγματικό για έναν αισθητήρα. Δεν υπάρχει νόημα λοιπόν στο να συνεχίσουμε την επεξεργασία, καταναλώνοντας πόρους άσκοπα. Για κάθε τύπο υπάρχει μια μέγιστη και μια ελάχιστη τιμή, που θεωρούνται γνωστές.

check4Antithesis( )

Αφού πραγματοποιηθεί με επιτυχία ο έλεγχος των ορίων του κάθε μεγέθους που ζητάται στους περιορισμούς του QueryFilter, γίνεται έλεγχος αμοιβαία αποκλειόμενων περιορισμών. Ελέγχεται δηλαδή αν στους περιορισμούς του QueryFilter συγκαταλέγονται και περιορισμοί που μπορεί μεν αυτόνομοι να στέκουν, δεν μπορούν όμως να ισχύουν ταυτόχρονα για ένα αισθητήρα. Για παράδειγμα έστω ότι το request απαιτούσε τη μέση θερμοκρασία των αισθητήρων για τους οποίους η επιτάχυνση είναι ταυτόχρονα μεγαλύτερη από 30 και μικρότερη από 10 μονάδες. Όπως είναι λογικό μια τέτοια κατάσταση δεν θα μπορούσε να υπάρξει ποτέ και θα ήταν άσκοπη η προώθηση του συγκεκριμένου request στο δίκτυο. Συνολικά μπορούν να υπάρξουν οι εξής εσφαλμένες καταστάσεις:

- Να απαιτηθεί για κάποιο μέγεθος και για δεδομένη συνάρτηση για αυτό το μέγεθος, τιμή μεγαλύτερη (Greater) από μια σταθερά α και στη συνέχεια του ίδιου request να απαιτηθεί για το ίδιο μέγεθος και την ίδια συνάρτηση τιμή μικρότερη (Less) από σταθερά β όπου β<=α ή τιμή ίση (Equals) με σταθερά β όπου β<=α ή τιμή μεταξύ (Within) σταθεράς β και γ όπου γ<=α Π.χ:

  SpatialAverage(Temperature)>30 και *SpatialAverage(Temperature)<20*

  SpatialAverage(Temperature)>30 και *SpatialAverage(Temperature)=20*

  SpatialAverage(Temperature)>30 και *10<SpatialAverage(Temperature)=20*

- Να απαιτηθεί για κάποιο μέγεθος και για δεδομένη συνάρτηση για αυτό το μέγεθος, τιμή μικρότερη (Less) από μια σταθερά α και στη συνέχεια του ίδιου request να απαιτηθεί για το ίδιο μέγεθος και την ίδια συνάρτηση τιμή μεγαλύτερη (Greater) από σταθερά β όπου β>=α ή τιμή ίση (Equals) με σταθερά β όπου β>=α ή τιμή μεταξύ (Within) σταθεράς β και γ όπου β>=α Π.χ:

  SpatialAverage(Temperature)<30 και *SpatialAverage(Temperature)>40*

  SpatialAverage(Temperature)<30 και *SpatialAverage(Temperature)=42*

  SpatialAverage(Temperature)<30 και *35<SpatialAverage(Temperature)=45*

- Να απαιτηθεί για κάποιο μέγεθος και για δεδομένη συνάρτηση για αυτό το μέγεθος, τιμή ίση (Equals) από μια σταθερά α και στη συνέχεια του ίδιου request να απαιτηθεί για το ίδιο μέγεθος και την ίδια συνάρτηση τιμή μεγαλύτερη (Greater) από σταθερά β όπου β>=α, ή τιμή ίση (Equals) με σταθερά β όπου β≠α, ή τιμή μεταξύ (Within) σταθεράς β και γ όπου β>=α ή γ<=α Π.χ:

  SpatialAverage(Temperature)=30 και *SpatialAverage(Temperature)>40*

  SpatialAverage(Temperature)=30 και *SpatialAverage(Temperature)<23*

  SpatialAverage(Temperature)=30 και *35<SpatialAverage(Temperature)=45*

  SpatialAverage(Temperature)=30 και *12<SpatialAverage(Temperature)=26*

- Να απαιτηθεί για κάποιο μέγεθος και για δεδομένη συνάρτηση για αυτό το μέγεθος, τιμή μεταξύ (Within) μιας σταθεράς α και μιας άλλης β και στη συνέχεια του ίδιου request να απαιτηθεί για το ίδιο μέγεθος και την ίδια συνάρτηση τιμή μεγαλύτερη (Greater) από σταθερά γ όπου γ>=β, ή τιμή ίση (Equals) με σταθερά γ όπου γ εκτός του διαστήματος αυτού, ή τιμή μεταξύ (Within) σταθεράς γ και δ όπου γ>=β ή δ<=α Π.χ:

  20<SpatialAverage(Temperature)<30 και *SpatialAverage(Temperature)>40*

  20<SpatialAverage(Temperature)<30 και *SpatialAverage(Temperature)<10*

  20<SpatialAverage(Temperature)<30 και *SpatialAverage(Temperature)=45*

  20<SpatialAverage(Temperature)<30 και *12<SpatialAverage(Temperature)=15*

- Να απαιτηθεί το μέγιστο ενός μεγέθους να είναι μικρότερο από μια σταθερά α και το ελάχιστο του ίδιου μεγέθους να είναι μεγαλύτερο από μια σταθερά β για την οποία ισχύει β>α Π.χ:

  SpatialMaximum(Temperature)<30 και *SpatialMinimum(Temperature)>40*

- Να απαιτηθεί το μέγιστο ενός μεγέθους να είναι μικρότερο από μια σταθερά α και το ελάχιστο του ίδιου μεγέθους να είναι μεταξύ μιας σταθεράς β και μιας άλλης γ για τις οποίες ισχύει β>α Π.χ:

  SpatialMaximum(Temperature)<30 και *35< SpatialMinimum(Temperature)<40*

- Να απαιτηθεί το μέγιστο ενός μεγέθους να είναι μεταξύ των σταθερών α και β και το ελάχιστο του ίδιου μεγέθους να είναι μεγαλύτερο από μια σταθερά β για την οποία ισχύει β>γ Π.χ:

  20<SpatialMaximum(Temperature)<30 και *SpatialMinimum(Temperature)>40*

- Να απαιτηθεί το μέγιστο ενός μεγέθους να είναι μεταξύ των σταθερών α και β και το ελάχιστο του ίδιου μεγέθους να είναι μεταξύ των σταθερών γ και δ για τις οποίες ισχύει γ>β Π.χ:

$20<SpatialMaximum(Temperature)<30$ και $34<SpatialMinimum(Temperature)<40$

Σε περίπτωση λάθους, η μεταβλητή flag παίρνει την τιμή 1 και στη συνέχεια προωθείται στον ResponseHandler το κατάλληλο μήνυμα λάθους (SYNTAX-ERROR). Αν δεν συμβεί κάποιο λάθος, τότε πραγματοποιείται έλεγχος των ημερομηνιών του στοιχείου Monitor (αν υπάρχει φυσικά).

checkMonitorDates( )

Η συνάρτηση αυτή φροντίζει για τον έλεγχο εγκυρότητας των στοιχείων ημερομηνίας και ώρας που δόθηκαν στην οντότητα Monitor. Τα στοιχεία αυτά αφορούν στην περιοδική επανάληψη μιας αίτησης (η περίοδος δίνεται στο στοιχείο Period) στο δίκτυο, ξεκινώντας από την ημερομηνία StartDate, την ώρα StartTime και σταματώντας την ημερομηνία StopDate και την ώρα StopTime. Πιο συγκεκριμένα αρχικά η κάθε πληροφορία σπάει σε κομμάτια (π.χ. το StartDate στα κομμάτια year, month και day και το στοιχείο StartTime στα κομμάτια hour και minute) και στη συνέχεια πραγματοποιούνται λογικοί έλεγχοι σε αυτά. Συγκεκριμένα εξετάζονται τα παρακάτω:

- Αν η ημερομηνία έναρξης να είναι μεγαλύτερη από την ημερομηνία λήξης. Για παράδειγμα αν ζητούσαμε StartDate: 3-5-2005 και StopDate: 3-5-2004, τότε προφανώς δεν πρέπει να προωθηθεί η αίτηση στο δίκτυο.
- Αν οι ημερομηνία έναρξης και λήξης είναι η ίδια, τότε εξετάζονται οι ώρες έναρξης και λήξης. Π.χ αν ζητάμε StartTime:11:20 και StopTime: 10:20 τότε προφανώς δεν πρέπει να προωθηθεί η αίτηση στο δίκτυο
- Αν το StartDate και το StopDate είναι έγκυρα. Συγκεκριμένα δεν επιτρέπεται στην ημερομηνία το έτος να ξεπερνά το τρέχον έτος, ο

μήνας να είναι μεγαλύτερος του 12ου ή μικρότερος του 1ου και η μέρα να είναι μεγαλύτερη της 31ης ή μικρότερη της 1ης.

- Αν το StartTime και το StopTime είναι έγκυρα. Συγκεκριμένα δεν επιτρέπεται ώρα να είναι μεγαλύτερη του 24 και μικρότερη του 0, όπως και τα λεπτά δεν επιτρέπεται να είναι μεγαλύτερα από 59 και μικρότερα από 0.
- Αν η μεταβλητή Period (που αναφέρεται στο κάθε πόση ώρα θέλουμε να επαναλαμβάνεται η επερώτηση προς το δίκτυο) έχει λογική τιμή. Δεν είναι λογικό δηλαδή να επιθυμούμε την επανάληψη της επερώτησης κάθε 3000 ώρες (τουλάχιστον όχι στις περισσότερες εφαρμογές). Βέβαια αυτός ο έλεγχος αναλόγως την εφαρμογή μπορεί να παραμεριστεί αν το επιθυμούμε.

Σε περίπτωση που σε κάποιον από τους παραπάνω ελέγχους διαπιστωθεί λάθος, τότε η καθολική μεταβλητή flag τίθεται την τιμή 1, που σηματοδοτεί το λάθος και πυροδοτεί την την προώθηση της ανάλογης απάντησης στον requestor, μέσω του ResponseHandler. Αν όχι, τότε η επερώτηση που θα σχηματιστεί παρακάτω, θα προωθηθεί στην κλάση MonitorHandler, η οποία μεριμνά για την έναρξη και των λήξη της αποστολής της επερώτησης στο δίκτυο (και της λήψης των αντίστοιχων απαντήσεων) με περίοδο που δίνεται στην μεταβλητή Period.

checkTimestamps( )

Η μέθοδος αυτή έχει παρόμοια λειτουργικότητα με την μέθοδο check_monitor_dates(), αφού και εδώ πραγματοποιούνται οι ίδιοι χρονικοί έλεγχοι με πριν, μόνο που τώρα γίνονται στα αντικείμενα τύπου Timestamp, που περιέχονται στην ουρά TimeExpressionConditions. Και πάλι πραγματοποιούνται έλεγχοι για τα στοιχεία StartDate, StopDate, StartTime και StopTime κάθε αντικειμένου. Η ύπαρξη χρονικών περιορισμών σηματοδοτεί την προώθηση της αίτησης προς τη βάση και όχι προς το δίκτυο, αφού δεν επιθυμούμε real-time

αλλά αποθηκευμένα δεδομένα. Σε περίπτωση παραβίασης κάποιου περιορισμού θα τεθεί η τιμή 1 στη μεταβλητή flag, γεγονός που σηματοδοτεί λάθος και ενεργοποιεί την προώθηση ανάλογης απάντησης προς τον client μέσω του ResponseHandler.

findFunction(*String func***)** και **findCatalogWord(***String word***)**

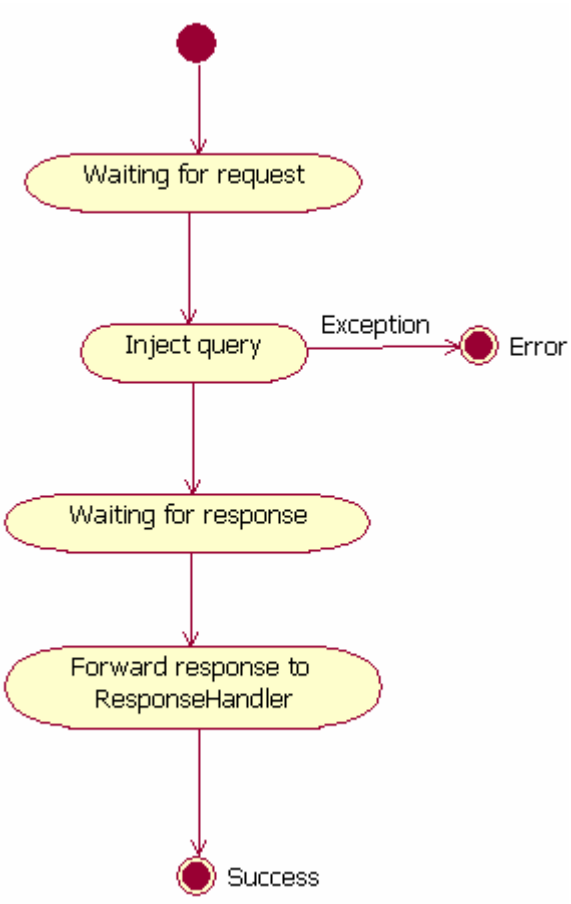
Οι μέθοδοι αυτές χρησιμοποιούνται από την μέθοδο formQuery που περιγράφεται πιο κάτω και αντιστοιχίζουν το όνομα μιας συνάρτησης ή κάποιου άλλου δεδομένου αντίστοιχα που περιέχεται στο Request, με μια λέξη κατά τα πρότυπα ονοματολογίας του TinyDB.

formQuery( )

Η μέθοδος αυτή δημιουργεί μια επερώτηση προς το δίκτυο, κατά τα πρότυπα που επιβάλλει το TinyDB.

Στη συνέχεια, αφού δεν έχει υπάρξει κάποιο λάθος το οποίο θα σταματούσε την επεξεργασία, η αίτηση θα προχωρήσει παρακάτω για περαιτέρω επεξεργασία. Αν η αίτηση αυτή περιλαμβάνει monitor τιμές, αφορά δηλαδή στην περιοδική επανάληψη μιας επερώτησης στο δίκτυο, τότε θα προωθηθεί στη κλάση MonitorHandler. Αν περιέχει χρονικές τιμές (Timestamps δηλαδή), τότε αφορά τη βάση και θα προωθηθεί προς την κλάση OdsRequestor. Αν αφορά το δίκτυο, θα προωθηθεί στην κλάση QueryInjector και τέλος αν αφορά στην παρακολούθηση κάποιου event στη κλάση EventHandler.

7.6 QueryInjector



Σχήμα 7.12: Διάγραμμα δραστηριοτήτων της κλάσης QueryInjector

getMessage( )

Όση ώρα δεν υπάρχει κάποια επερώτηση για προώθηση προς το δίκτυο, η κλάση αυτή βρίσκεται σε αναμονή.

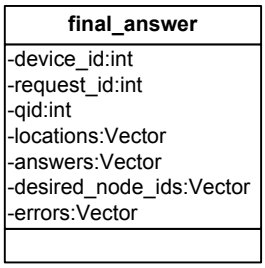
putInfo(*String select, Vector loc, Vector requested_information, int dev_id, Vector Required_Node_Ids, int req_id***)**

Η μέθοδος αυτή, καλείται από τον SemanticChecker και φροντίζει για την αποστολή των απαραίτητων πληροφοριών προς τη κλάση QueryInjector. Συγκεκριμένα προωθούνται η επερώτηση, οι επιθυμητές περιοχές της αίτησης, οι

απαραίτητες πληροφορίες, ο κωδικός της συσκευής, οι αντίστοιχοι κωδικοί των επιθυμητών κόμβων και ο κωδικός της αίτησης.

processRequest(*String query***)**

Η κλάση αυτή φροντίζει για την προώθηση μιας επερώτησης στο tinyDB και συνεπώς στο ασύρματο δίκτυο αισθητήρων. Επειδή μπορούν να προωθηθούν πολλές διαφορετικές επερωτήσεις προτού έρθουν οι πρώτες απαντήσεις, πρέπει να φροντίζουμε να κρατάμε μια αντιστοιχία μεταξύ επερωτήσεων που έγιναν προς το δίκτυο και των πληροφοριών που επέστρεψαν από αυτό για να μην υπάρχει σύγχυση των αποτελεσμάτων. Έτσι η μέθοδος αυτή δημιουργεί ένα αντικείμενο τύπου final_answer, του οποίου κ κλάση φαίνεται στο παρακάτω σχήμα, η οποία περιέχει όλες τις απαραίτητες πληροφορίες που χρειάζονται για μια απάντηση.



Σχήμα 7.13: Κλάση FinalAnswer

Αρχικά τοποθετούνται μέσα στην ουρά locations του αντικειμένου αυτού οι περιοχές της αίτησης (στην επερώτηση δεν επιτρέπονται περιοχές παρά μονάχα οι αντίστοιχοι κωδικοί, όμως στην απάντηση πρέπει να αναφέρονται οι περιοχές που ζητήθηκαν αρχικώς). Επίσης τοποθετούνται στο αντικείμενο αυτό ο κωδικός της συσκευής αν υπάρχει, ο κωδικός της αίτησης, τα επιθυμητά node_id's, οι ζητούμενες πληροφορίες (ζευγάρια συνάρτησης και μεγέθους π.χ. function:avg και type temperature) και το qid ή query id που αντιστοιχεί σε έναν μοναδικό κωδικό που αντιστοιχίζει το tinyDB στην τρέχουσα επερώτηση. Στη συνέχεια το αντικείμενο αυτό που ακόμα δεν έχει γεμίσει πλήρως, τοποθετείται σε μια ουρά

(την pending_answers), με τέτοια αντικείμενα. Όταν μια απάντηση έρθει, περιέχει έναν μοναδικό κωδικό, που είναι ο ίδιος με της επερώτησης που πραγματοποιήθηκε για να πάρουμε αυτήν την απάντηση από το δίκτυο. Έτσι κάνοντας μια απλή αναζήτηση μέσα σε αυτήν την ουρά βρίσκουμε σε ποιό αντικείμενο πρέπει να αντιστοιχίσουμε τα αποτελέσματα που επεστράφησαν. Προτού όμως τοποθετήσουμε κάτι μέσα σε αυτό το αντικείμενο κάνουμε τα εξής. Κάνουμε φιλτράρισμα των απαντήσεων για να πάρουμε τα δεδομένα των κόμβων που μας ενδιαφέρουν και εφαρμόζουμε τις συναρτήσεις που προδιαγράφονται στο request για τα δεδομένα που πήραμε. Για παράδειγμα αν ζητάμε avg(Temperature) και min(Humidity) για τα δωμάτια room1 και room7 που έχουν κωδικούς (node_id's) 1,2,3 το πρώτο δωμάτιο και 7,8 το δεύτερο. Τότε θα ζητήσουμε αρχικά τις τιμές θερμοκρασίας και υγρασίας από όλους τους κόμβους, στο δίκτυο, και θα πάρουμε δύο στήλες σαν και αυτές

Πίνακας 7.1 : Τιμές πριν από το φιλτράρισμα

Node Id’s	Temperature	Humidity
1	23	11
2	25	11
3	21	14
4	33	12
5	32	13
6	12	16
7	26	17
8	23	12
9	11	9

Η απάντηση έχει ένα μοναδικό id που αντιστοιχεί στο qid ενός εκ των αντικειμένων την ουρά pending_anwers. Έτσι, παίρνοντας το σωστό αντικείμενο, ας το ονομάσουμε Α βλέπουμε ποιά node_id's θέλουμε. Στη συνέχεια θα φιλτράρουμε τα αποτελέσματα ώστε να πάρουμε τα δεδομένα των επιθυμητών κόμβων, άρα καταλήγουμε στα εξής:

Πίνακας 7.2: Τιμές μετά από το φιλτράρισμα

Node Id's	Temperature	Humidity
1	23	11
2	25	11
3	21	14
7	26	17
8	23	12

Αφού φιλτράρουμε τα αποτελέσματα, πραγματοποιούμε τους κατάλληλους υπολογισμούς πάνω στα δεδομένα αυτά, με βάση τις πληροφορίες που περιέχονται στα ζευγάρια συνάρτηση-τύπος στο αντικείμενο Α. Πρώτα δηλαδή βρίσκουμε τη μέση θερμοκρασία που είναι 23.6 και στη συνέχεια τη μέση τιμή της υγρασίας που είναι 13. Εφόσον έχουμε και τις απαντήσεις λοιπόν, τις τοποθετούμε στο αντικείμενο Α, στην ουρά answers για την ακρίβεια και αποστέλλουμε το αντικείμενο στον ResponseHandler.

makeSelection( )

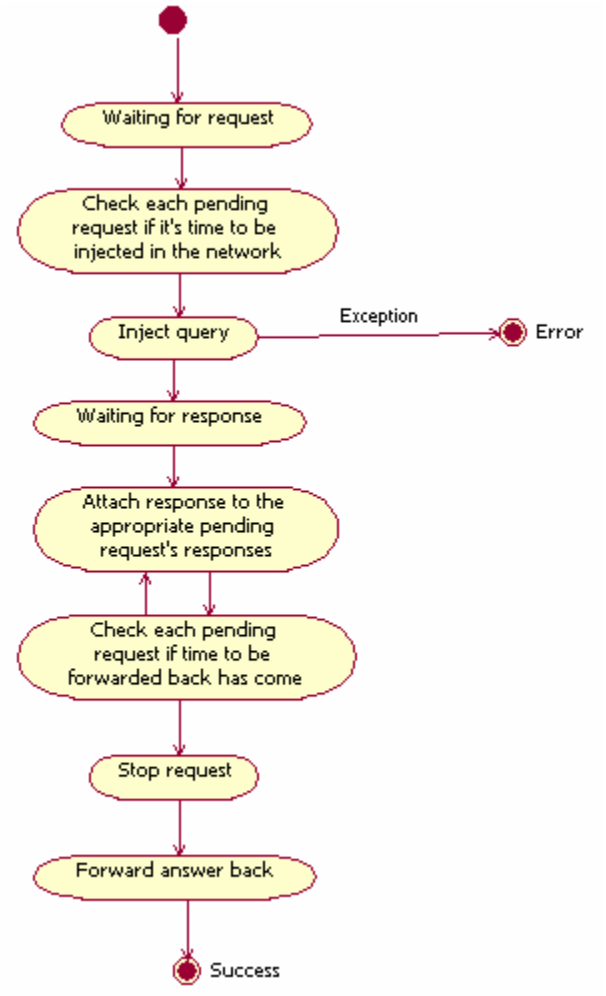
Αυτή είναι η μέθοδος που πραγματοποιεί το φιλτράρισμα των απαντήσεων που ήρθαν από το δίκτυο. Ελέγχει την ουρά desired_nodes καθώς και τις απαντήσεις που ήρθαν από το δίκτυο και αν βρει κάποια αντιστοιχία,

μεταξύ των node_id's σημαίνει πως το συγκεκριμένο αποτέλεσμα θα συγκαταλέγεται στα επιθυμητά αποτελέσματα.

makeCalculation(*Vector types_values,String function***)**

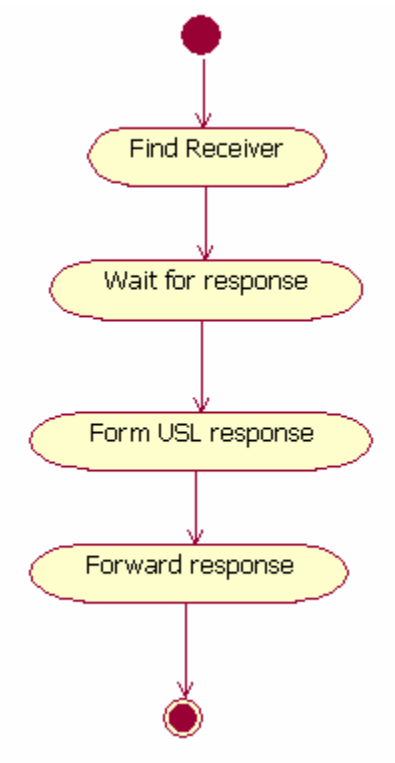
Η μέθοδος αυτή πραγματοποιεί την συνάρτηση function Πάνω στα δεδομένα που περιέχονται στην ουρά types.

7.7 MonitorHandler



Σχήμα 7.14: Διάγραμμα δραστηριοτήτων της κλάσης MonitorHandler

7.8 ResponseHandler



Σχήμα 7.15: Διάγραμμα δραστηριοτήτων της κλάσης ResponseHandler

getMessage( )

Όση ώρα δεν υπάρχει κάποια απάντηση ώστε να φτιαχτεί το ανάλογο USL Response, η κλάση παραμένει σε κατάσταση αναμονής. Αν υπάρξει απάντηση από κάποια άλλη κλάση του Driver, τότε τα επιμέρους τμήματά της θα εξαχθούν σε ουρές και μεταβλητές της ResponseHandler, ώστε η επεξεργασία τους να είναι πιο εύκολη.

putRequest(*FinalAnswer req1***)**

Η κλήση αυτής της μεθόδου γίνεται από άλλες κλάσεις του Driver. Η λειτουργία της αφορά στην τοποθέτηση μιας απάντησης τύπου FinalAnswer στην ουρά με τις απαντήσεις του ResponseHandler και η αφύπνιση του τελευταίου

ώστε να ξεκινήσει τη λειτουργία του. Η απάντηση μπορεί να περιέχει δεδομένα, ή να περιέχει το κωδικό κάποιου λάθους.

findReceiver(*Receiver r***)**

Η μέθοδος αυτή καλείται από τον ίδιο τον receiver, με σκοπό να ειδοποιήσει τον ResponseHandler για την ύπαρξή του κατά την έναρξη λειτουργίας του συστήματος.

createResponse ( )

Με βάση τα δεδομένα που έχει πάρει, ο ResponseHandler ξεκινάει να δομεί το USL Response, δηλαδή μια USL απάντηση. Η δομή της περιγράφεται στην ενότητα USL Response. Συνοπτικά τοποθετούνται όλες οι πληροφορίες που αφορούσαν την αίτηση (locations, device_id, request_id, τιμές απαντήσεων π.χ. function=”SpacialAverage” Type=”Acceleration”), μαζί με τις αντίστοιχες τιμές τους.

8

Παράρτημα Β - TinyDB

Στο παράρτημα αυτό γίνεται παρουσιάζεται TinyDB με τη χρήση κατάλληλων παραδειγμάτων που υποδεικνύουν τη λειτουργία του. Αναφέρονται τόσο γενικές πληροφορίες, όσο και πληροφορίες εγκατάστασης και εκτέλεσης του.

8.1 TinyDB

Το TinyDB είναι ένα σύστημα επεξεργασίας επερωτήσεων που χρησιμοποιείται για την εξαγωγή πληροφοριών από ένα δίκτυο TinyOS αισθητήρων. Αντίθετα με τις υπάρχουσες λύσεις για την επεξεργασία δεδομένων σε TinyOS, το TinyDB δεν προϋποθέτει το γράψιμο ενσωματωμένου C κώδικα για τους αισθητήρες. Αντιθέτως, παρέχει μια απλή, SQL-like διεπαφή, μέσω της οποίας προσδιορίζονται τα δεδομένα που επιζητούμε, μαζί με επιπρόσθετες παραμέτρους, όπως η συχνότητα με την οποία θέλουμε να μας επιστρέφονται τα δεδομένα κ.α. Δεδομένης μιας επερώτησης που προσδιορίζει τα δεδομένα που μας ενδιαφέρουν, το TinyDB συλλέγει τα δεδομένα αυτά, τα φιλτράρει, τα επεξεργάζεται και τα προωθεί πίσω σε κάποιο PC. Είναι πολύ σημαντικό πως όλα αυτά τα κάνει μέσω αποδοτικών (από πλευράς καταναλωτικής ισχύος) αλγορίθμων.

Ο βασικός στόχος του TinyDB είναι να κάνει τη ζωή του προγραμματιστή αρκετά πιο εύκολη με το να επιτρέπει την ανάπτυξη εφαρμογών προσανατολισμένων στα δεδομένα, αρκετά πιο εύκολα από ότι ίσχυε μέχρι σήμερα. Το βασικό του πλεονέκτημα είναι πως απελευθερώνει τον προγραμματιστή από το φορτίο της γραφής χαμηλού επιπέδου κώδικα για συσκευές αισθητήρων, συμπεριλαμβάνοντας διεπαφές για δίκτυα αισθητήρων. Μερικά από τα χαρακτηριστικά του TinyDB είναι τα εξής:

Διαχείριση Μεταδεδομένων

Αυτό επιτυγχάνεται με την παροχή ενός καταλόγου (catalog.xml) για τη περιγραφή των μεταβλητών και των εντολών, που είναι απαραίτητες για τις επερωτήσεις στο δίκτυο. Οι μεταβλητές μπορεί να είναι μετρήσεις από κάποιον αισθητήρα, ή εσωτερικές software/hardware παράμετροι. Οι εντολές και οι παράμετροι μπορούν δημιουργούνται μέσω των TinySchema κομματιών του TinyOS.

Επερωτήσεις υψηλού επιπέδου

Το TinyDB χρησιμοποιεί μια δηλωτική γλώσσα επερωτήσεων που επιτρέπει την περιγραφή των επιθυμητών δεδομένων, χωρίς να είναι απαραίτητος ο προσδιορισμός του πως θα τα πάρουμε. Αυτό διευκολύνει αρκετά την δημιουργία εφαρμογών και εγγυάται πως οι εφαρμογές θα συνεχίσουν να δουλεύουν αποδοτικά καθώς το δίκτυο αλλάζει.

Τοπολογία Δικτύου

Το TinyDB χειρίζεται το υποκείμενο δίκτυο που βρίσκεται από κάτω με το να εντοπίζει γείτονες, να κρατάει πίνακες δρομολόγησης και να φροντίζει πως κάθε κόμβος του δικτύου μπορεί αποδοτικά και έγκυρα να μεταδίδει τα δεδομένα του προς τον χρήστη.

Πολλαπλές Επερωτήσεις

Το TinyDB επιτρέπει την αποστολή πολλών διαφορετικών επερωτήσεων προς τους ίδιους κόμβους, την ίδια στιγμή. Οι επερωτήσεις μπορούν να έχουν διαφορετικές συχνότητες και να έχουν πρόσβαση σε διαφορετικούς τύπους δεδομένων.

Επέκταση μέσω της κοινής χρήσης επερωτήσεων

Προκειμένου να επεκταθεί το TinyDB δίκτυο αισθητήρων, απλώς εγκαθιστούμε τον βασικό κώδικα του TinyDB σε καινούργιους κόμβους και το TinyDB κάνει τα υπόλοιπα. Οι κόμβοι μοιράζονται επερωτήσεις. Όταν ένας κόμβος μάθει για κάποια επερώτηση που δεν τρέχει ακόμα σε αυτόν, τότε αυτόματα ζητά την αποστολή των δεδομένων της επερώτησης και ξεκινά να την τρέχει. Δεν χρειάζεται προγραμματισμός των νέων κόμβων, πέρα από την εγκατάσταση του TinyDB.

8.2 TinySQL

Η γενική μορφή της TinySQL, της γλώσσας δηλαδή που χρησιμοποιείται από το χρήστη για την δημιουργία επερωτήσεων μέσω του TinyDB, είναι η παρακάτω:

SELECT <aggregates>, <attributes>

[FROM {sensors | <buffer>}]

[WHERE <predicates>]

[GROUP BY <exprs>

[HAVING having-list]]

[EPOCH DURATION integer]

[INTO <buffer>]

[TRIGGER ACTION <command>]

Η δομή της θυμίζει την SQL. Το στοιχείο «EPOCH DURATION» δηλώνει την περίοδο επανάληψης της επερώτησης προς το δίκτυο. Ένα ενδεικτικό παράδειγμα των παραπάνω είναι το εξής:

SELECT nodeid, nestNo, light

FROM sensors

WHERE light > 400

EPOCH DURATION 1s

Στη συγκεκριμένη επερώτηση ζητάμε να μας επιστραφούν τα πεδία *nodeid, nestNo, light* από τους αισθητήρες εκείνους για τους οποίους η φωτεινότητα είναι μεγαλύτερη από 400 μονάδες, κάθε ένα δευτερόλεπτο.

Οι απαντήσεις που θα επιστρέψουν από το δίκτυο θα έχουν την εξής μορφή:

Πίνακας 8.1: Απαντήσεις από το δίκτυο

Epoch	Nodeid	nestNo	Light
0	1	17	455
0	2	25	389
1	1	17	422
1	2	25	405

Όπου epoch είναι η χρονική περίοδο που έγινε η επερώτηση και σε κάθε στήλη περιέχεται το αντίστοιχο πεδίο που ζητήσαμε.

Επερωτήσεις τύπου Event:

Ζητούν από το TinyDB να πραγματοποιήσει την επερώτηση μόνο στη περίπτωση που συμβεί κάποιο ενδιαφέρον συμβάν. Για παράδειγμα σε περίπτωση που πατηθεί κάποιο πλήκτρο, ή έρθει κάποιο μήνυμα.

8.3 Εγκατάσταση του TinyDB

Για το συγκεκριμένο παράδειγμα θα χρειαστούμε τρεις κόμβους. Δίνουμε σε καθέναν ένα id (0,1,2) και στη συνέχεια συνδέουμε το κόμβο 0 με τη σειριακή θύρα του PC.

Θα χρειαστούμε την συνεργασία με την κλάση TinyDBMain που βρίσκεται στο tools/java/net/tinyos/tinydb. Αρχικά πρέπει να κάνουμε build τις κλάσεις της java. Για να το πετύχουμε αυτό πρέπει κάποια πακέτα να ενσωματωθούν στο CLASSPATH. Τα πακέτα που χρειάζονται είναι τα JLex.jar, cup.jar, και plot.jar. Όλα είναι διαθέσιμα στο tools/java/jars. Αφού ενσωματωθούν, κάνουμε build τις κλάσεις γράφοντας σε μια κονσόλα:

cd path/to/tinyos/tools/java/net/tinyos/tinydb

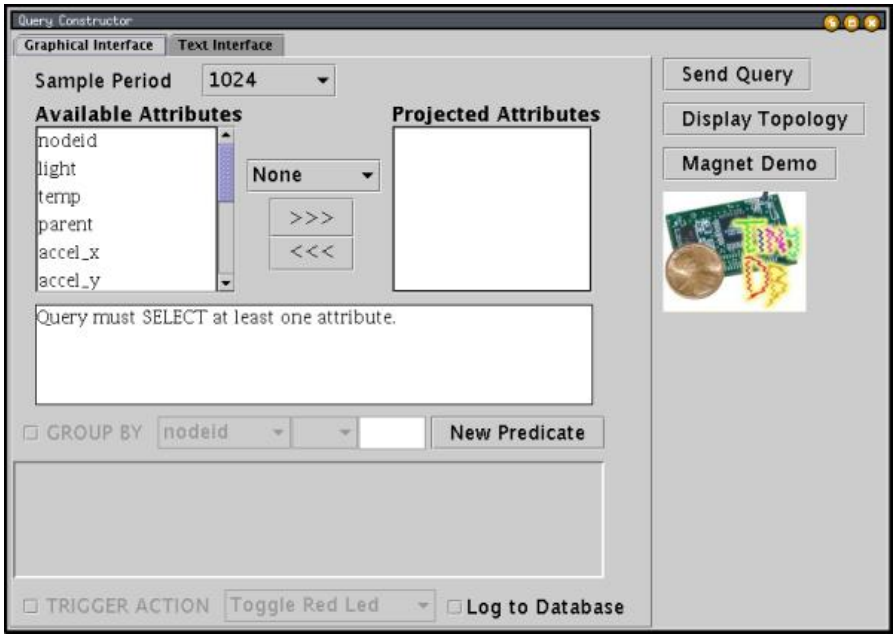
make

Αυτό μπορεί να πάρει μερικά λεπτά. Στη συνέχεια είμαστε έτοιμοι να τρέξουμε το γραφικό περιβάλλον, πληκτρολογώντας:

cd ../../..

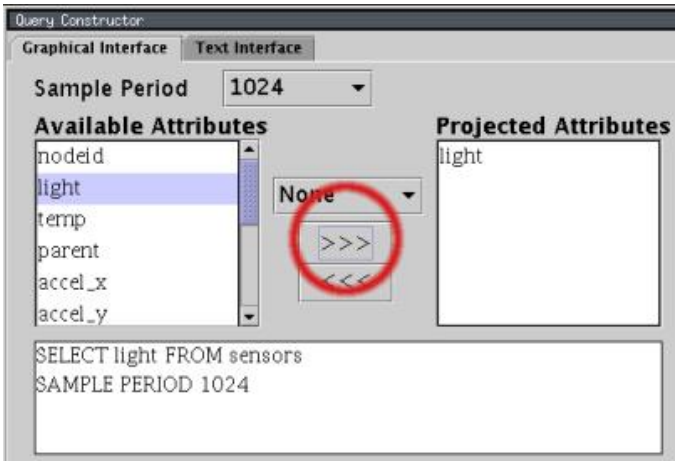
java net.tinyos.tinydb.TinyDBMain

Στη συνέχεια θα εμφανιστεί το γραφικό περιβάλλον που φαίνεται στο σχήμα 8.1



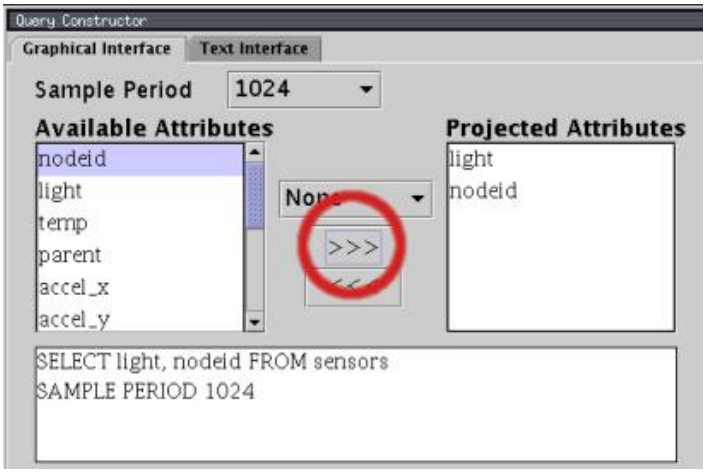
Σχήμα 8.1: Γραφικό Περιβάλλον του TinyDB

Για να φτιάξουμε μια επερώτηση αρχικά επιλέγουμε τα πεδία που ζητήσουμε να πάρουμε από το δίκτυο, από την λίστα που βρίσκεται στα αριστερά. Ας επιλέξουμε το πεδίο light πατώντας στο κουμπί " >>> " .



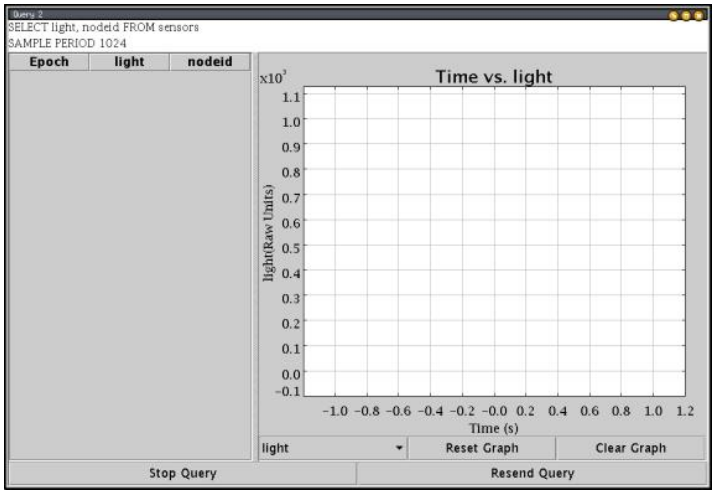
Σχήμα 8.2: Εισαγωγή πεδίου light

Αν προσέξουμε θα δούμε πως στο κάτω μέρος της οθόνης εμφανίζεται η επερώτηση. Στο παράδειγμα αυτό θα παίρνουμε αποτελέσματα κάθε 1024 ms. Επιλέγουμε και το nodeid:



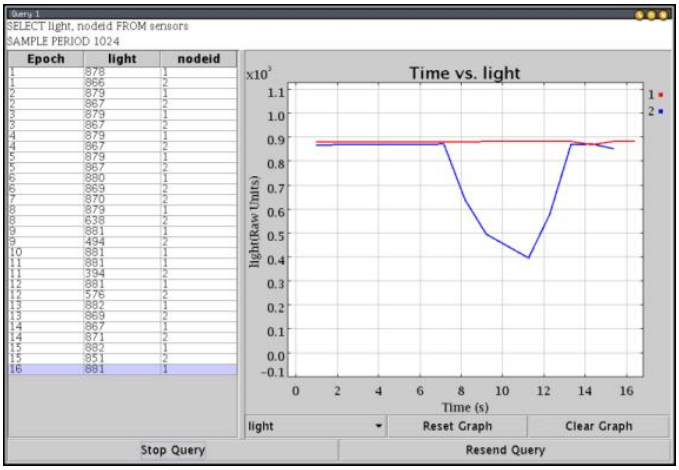
Σχήμα 8.3: Εισαγωγή πεδίου nodeid

Αν τώρα πατήσουμε το κουμπί Send Query θα πάρουμε την οθόνη του σχήματος 8.4 η οποία σταδιακά θα γεμίζει, όπως στο σχήμα 8.5



Σχήμα 8.4: Κενό γράφημα

Με το που θα εμφανιστεί η οθόνη αυτή θα αναβοσβήσει ένα Led στο κόμβο) και σύντομα θα ανοίξουν και τα led's των άλλων δύο κόμβων. Τα κίτρινα Led's θα ανάβουν κάθε ένα δευτερόλεπτο δείχνοντας ότι το σύστημα δουλεύει σωστά. Αν δεν ανάψουν τα κίτρινα Led's, πατάμε το κουμπί Resend Query. Σύντομα θα εμφανιστεί ένα γράφημα σαν το παρακάτω:



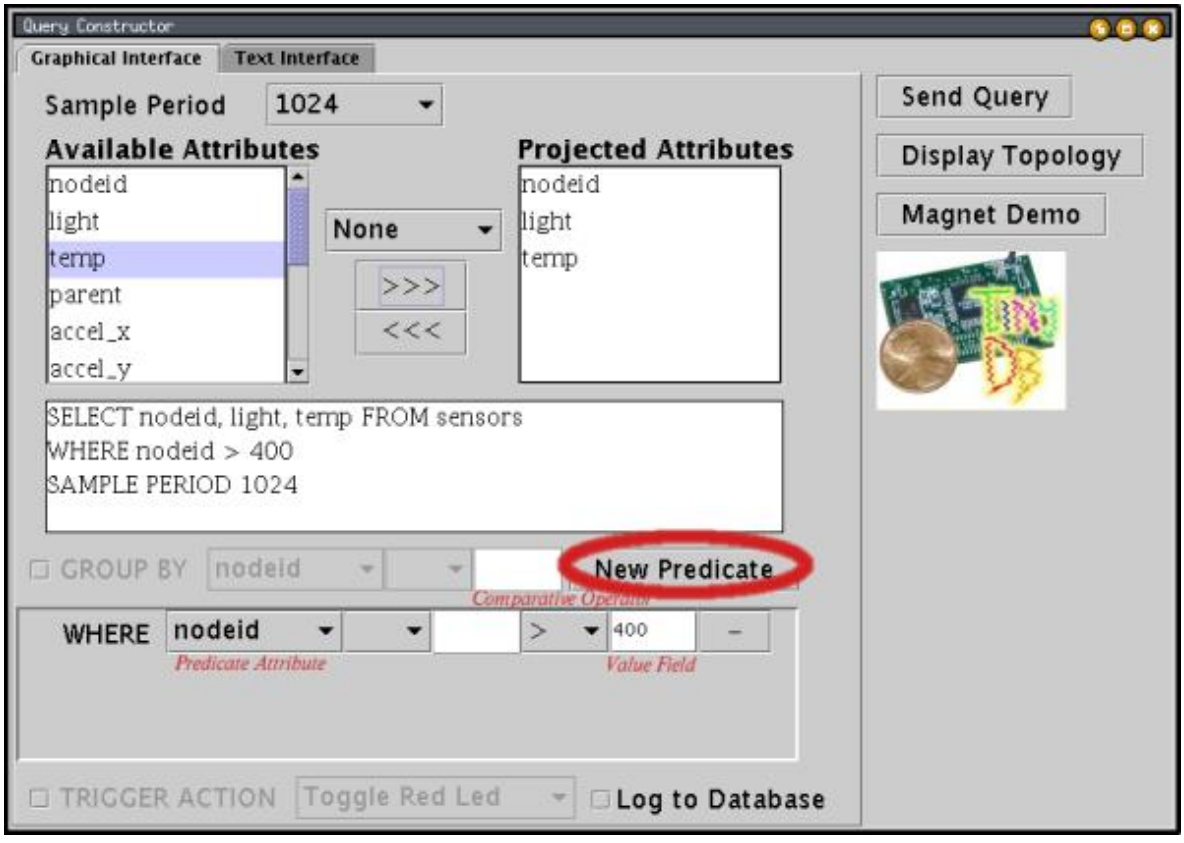
Σχήμα 8.5: Γράφημα που σιγά σιγά γεμίζει

Μερικά παραδείγματα:

♦Έστω ότι θέλουμε ένα query της μορφής

SELECT nodeid, light, temp
FROM sensors
WHERE light > 400
SAMPLE PERIOD 1024

Για να φτιάξουμε μια τέτοια επερώτηση χρησιμοποιούμε το κουμπί “New Predicate” . Επιλέγουμε Light από το predicate μενού, > από το μενού με τα σύμβολα ανισότητας και γράφουμε 400 στο πεδίο τιμών.

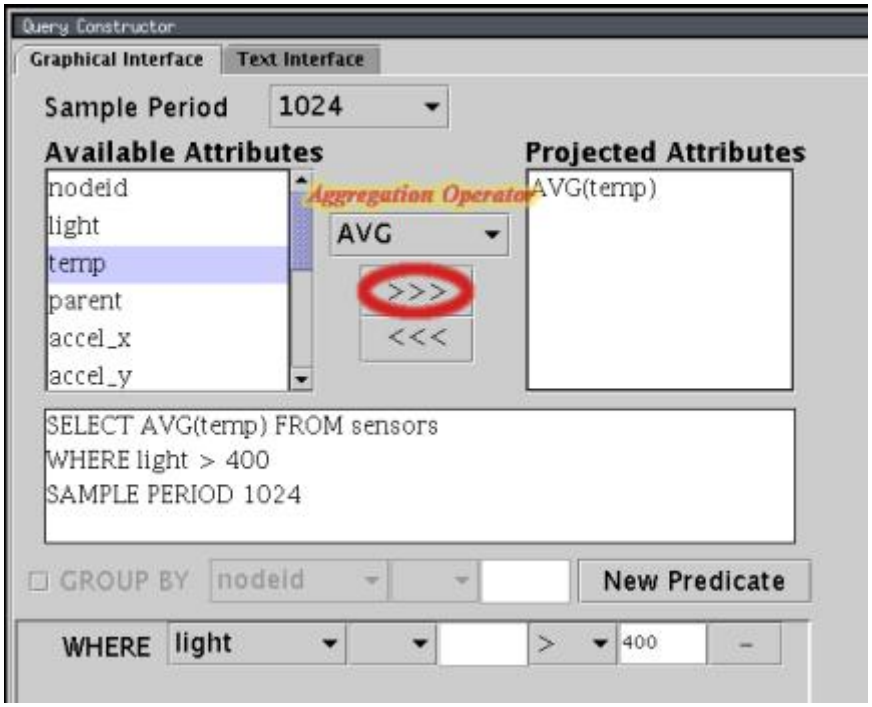


Σχήμα 8.6: Εικόνα του προηγούμενου παραδείγματος

♦Έστω ότι θέλουμε μια επερώτηση της μορφής:

SELECT AVG(temp)
FROM sensors
WHERE light > 400
SAMPLE PERIOD 1024

Επιλέγουμε το AVG από το μενού των συναρτήσεων πριν βάλουμε το πεδίο temp στα επιθυμητά αποτελέσματα.



Σχήμα 8.7: Εικόνα του προηγούμενου παραδείγματος

8.4 Μια απλή εφαρμογή Java για συνεργασία με το TinyDB

```
package net.tinyos.tinydb;

import net.tinyos.tinydb.parser.*;
import java.util.Vector;
import java.io.*;
public class DemoApp implements ResultListener{
    public DemoApp() {
        try {
            TinyDBMain.initMain(); //parse the query
            q = SensorQueryer.translateQuery("SELECT light", (byte)1);
            //inject the query, registering ourselves as a listener for result
            System.out.println("Sending query.");
            TinyDBMain.injectQuery( q, this);
        } catch (IOException e) {
            System.out.println("Network error.");
        } catch (ParseException e) {
            System.out.println("Invalid Query.");
        }
    }

    /* ResultListener method called whenever a result arrives */
    public void addResult(QueryResult qr) {
        Vector v = qr.resultVector(); //print the result
        for (int i = 0;  i <  v.size(); i++) {
            System.out.print("\t" + v.elementAt(i) + "\t|");
        }
        System.out.println();
    }

    public static void main(String argv[]) {
        new DemoApp();
    }

    TinyDBQuery q;
}
```

Για να τρέξουμε αυτό το πρόγραμμα, στήνουμε τους κόμβους όπως και πριν και στη συνέχεια μπαίνουμε στο φάκελο tools/java/ και πληκτρολογούμε java.net.tinyos.tinydb.DemoApp. Το αποτέλεσμα θα είναι το εξής:

Πίνακας 8.2: Απαντήσεις από το δίκτυο

```
Listening for client connections on port 9000
SerialPortIO: initializing
Successfully opened COM1
client connected from localhost.localdomain (127.0.0.1)
Sending query.
      1      |      835    |
      2      |      833    |
      3      |      833    |
      4      |      833    |
      5      |      833    |
      6      |      833    |
      7      |      833    |
...
```

Οι πρώτες γραμμές του DemoApp αρχικοποιούν το TinyDB, κάνουν parse την επερώτηση και την στέλνουν στο δίκτυο:

```
TinyDBMain.initMain();

//parse the query
q = SensorQueryer.translateQuery("SELECT light", (byte)1);

//inject the query, registering ourselves as a listener for result
System.out.println("Sending query.");
TinyDBMain.injectQuery( q, this);
```

Η κλήση TinyDB.initMain() διαβάζει το TinyDB configuration file και στήνει την επικοινωνία με το δίκτυο. Η default επιλογή είναι να ανοίγει από μόνη της σύνδεση με τη σειριακή θύρα, αλλά μπορεί να ρυθμιστεί ώστε να μοιράζεται τη σύνδεση αυτή μέσω του SerialForwarder.

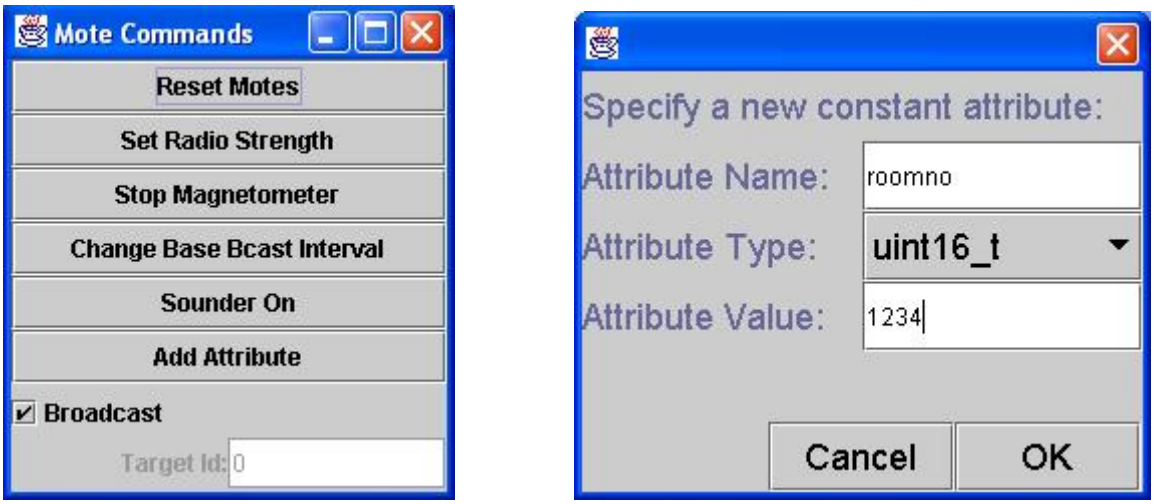
Η κλήση SensorQueryer.translateQuery(..) μεταφράζει αυτήν την SQL Like επερώτηση σε ένα αντικείμενο τύπου TinyDBQuery. Η δεύτερη παράμετρος καθορίζει ότι το query id θα είναι 1. Αυτό το id μπορεί να χρησιμοποιηθεί για να ακυρώσουμε ή να αλλάξουμε το query αφού έχει εισαχθεί στο δίκτυο.

Τέλος η κλήση TinyDBMain.injectQuery(..) εισάγει την επερώτηση στο δίκτυο και ξεκινά να την εκτελεί. Η δεύτερη παράμετρος προσδιορίζει πως το

αντικείμενο DemoApp θα οριστεί ως listener για τα αποτελέσματα αυτού του query.

Προσθέτοντας Μεταβλητές

Για να προσθέσουμε μια μεταβλητή, απλώς πατάμε το κουμπί “Add Attribute” στο παράθυρο “Mote Commands”. Απλώς γράφουμε το όνομα της μεταβλητής και πατάμε “OK”.



Σχήμα 8.8: Διαδικασία πρόσθεσης μεταβλητών

Για να θέσουμε αυτή τη μεταβλητή για ένα μόνο κόμβο απενεργοποιούμε την επιλογή Broadcast και γράφουμε το Target Id δηλαδή το id του κόμβου που μας ενδιαφέρει.

9

Βιβλιογραφία

[1] Tilemahos Hasiotis, George Alyfantis, Vassileios Tsetsos, Odysseas Sekkas, Stathes Hadjiefthymiades. "*Sensation: A Middleware Integration Platform for Pervasive Applications in Wireless Sensor Networks*"

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "*A Survey on Sensor Networks*", IEEE Communications Magazine, August 2002

[3] K. Martinez, J. K. Hart, and R. Ong, "*Sensor Network Applications*", IEEE Computer, August 2004

[4] M. Satyanarayanan, "*Pervasive Computing: Vision and Challenges*" IEEE Personal Communications, August 2001

[5] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, *"The Emergence of Networking Abstractions and Techniques in TinyOS "*, NSDI, 2004

[6] D. Estrin, D. Culler and K. Pister, and G. Sukhatme, "*Connecting the Physical World with Pervasive Networks*", IEEE Pervasive Computing, January-March 2002

[7] K. Romer, O. Kasten, and F. Mattern, "*Middleware Challenges for Wireless Sensor Networks*", Mobile Computing and Communications Review, Volume 6, Number 2, October 2002

[8] K. Romer, "*Programming Paradigms and Middleware for Sensor Networks*", GI/ITG Fachgespräch Sensornetze, Karlsruhe, February 2004

[9] S. R. Madden, J. Hellerstein, and W. Hong, "*TinyDB: In-Network Query Processing in TinyOS*", Version 0.4, September, 2003

[10] S. R. Madden, "*The Design and Evaluation of a Query Processing Architecture for Sensor Networks*", Ph.D. Thesis. UC Berkeley. Fall, 2003 (TinyDB)

[11] P. Bonnet, J. E. Gehrke, and P. Seshadri, "*Querying the Physical World*", IEEE Personal Communications, vol. 7, no. 5, pp 10-15, Oct. 2000. (Cougar)

[12] Y. Yao, and J. E. Gehrke, "*The Cougar Approach to In-Network Query Processing in Sensor Networks*", Sigmod Record, Volume 31, Number 3, September 2002.

[13] C.-C. Shen, Ch. Srisathapornphat, and Ch. Jaikaeo, "*Sensor Information Networking Architecture and Applications*", IEEE Personal Communications, August 2001 (SINA)

[14] A. Boulis and M. B. Srivastava, "*A Framework for Efficient and Programmable Sensor Networks*", In proceedings of OPENARCH 2002, New York, June, 2002.

[15] T. L. and M. Martonosi. "*Impala: A Middleware System for*

Managing Autonomic, Parallel Sensor Systems". ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, June 2003

[16] W. Heinzelman, A. Murphy, H. Carvalho and M. Perillo,"*Middleware to Support Sensor Network Applications*", IEEE Network Magazine Special Issue. Jan. 2004 (MiLAN)

[17] Sh. Li A1, S. H. Son A1, J. A. Stankovic, "*Event Detection Services Using Data Service Middleware in Distributed Sensor Networks*", Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Volume 2634 / 2003

[18] W3C Architecture Domain, Extensible Markup Language (XML), www.w3.org/XML/

[19] W3C Architecture Domain, XML Schema, www.w3.org/XML/Schema

[20] D.H.Crocker, "*Standard for the format of ARPA Internet text messages*", STD11, RFC 822, UDEL, August 1982

[21] F. Z., and L. Guibas, "*Wireless Sensor Networks: An Information Processing Approach (Morgan Kaufmann Series in Networking)*", Morgan Kaufmann, July 2004

[22] J.Hill et al., "*System Architecture Directions for Networked Sensors*," Proc. Int'l Conf. Arcitectural Support for Programming Languages and Operating Systems (ASPLOS), ACM Press, 2000, pp. 93-104

[23] J.Gehrke, and A. Madden. "Query Processing in Sensor Networks," IEEE Pervasive Computing, January-March 2004