ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ

ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Υποστήριξη του ενδιάμεσου λογισμικού POBICOS

σε κινητά τηλέφωνα Android

Διπλωματική Εργασία

Κωνσταντίνος Λέκκας

Επιβλέποντες καθηγητές: Λάλης Σπυρίδων – Γεράσιμος
Αναπληρωτής Καθηγητής Π.Θ.

Θανάσης Λουκόπουλος
Επίκουρος Καθηγητής ΤΕΙ Λαμίας

Βολος 2011

Ευχαριστίες

Η εργασία αυτή δεν θα είχε ολοκληρωθεί χωρίς την ουσιαστική συμβολή του κ. Λάλη, με τον οποίο συνεργαστήκαμε στενά κατά την διάρκεια του τελευταίου εξαμήνου. Θα ήθελα να τον ευχαριστήσω θερμά για την καθοδήγησή του, τις συμβουλές του και τις ευκαιρίες που απλόχερα μου έδωσε.

Επιπλέον, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου, για την υποστήριξή τους όλα αυτά τα χρόνια. Ήταν πάντα δίπλα μου και η βοήθειά τους ήταν απαραίτητη για να πετύχω τους στόχους μου.

Περιεχόμενα

Περιληψη

Σκοπός της εργασίας είναι ο σχεδιασμός και η υλοποίηση ενός ολοκληρωμένου συστήματος που υποστηρίζει την εκτέλεση κατανεμημένων μικρό-εφαρμογών σε κινητά τηλέφωνα τα οποία βρίσκονται σε μια συγκεκριμένη γεωγραφική περιοχή. Οι εν λόγω εφαρμογές εκτελούνται πάνω από το ενδιάμεσο λογισμικό POBICOS, ενώ η πλατφόρμα που επιλέχθηκε για τα κινητά τηλέφωνα είναι το AndroidOS.

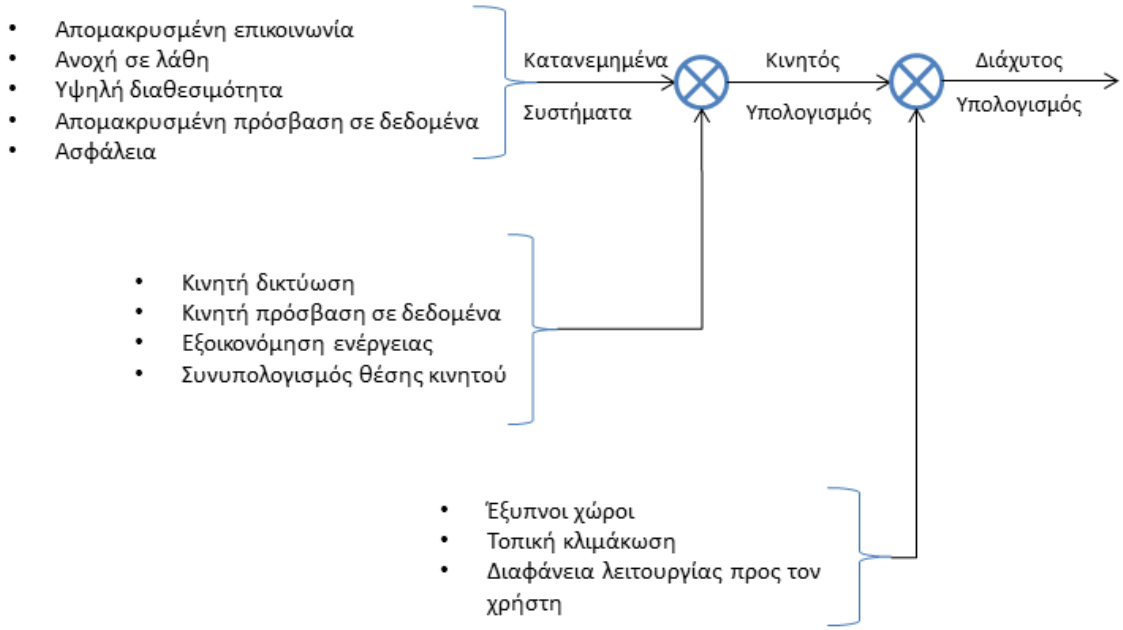
Ένα μεγάλο μέρος της εργασίας έχει αφιερωθεί στην μεταφορά του ενδιάμεσου λογισμικού POBICOS από το TinyOS στο AndroidOS. Τα βήματα του σχεδιασμού και της ανάπτυξης για την μεταφορά αυτή περιγράφονται αναλυτικά στα επόμενα κεφάλαια.

Στα πλαίσια της εργασίας αναπτύχθηκε ένα λειτουργικό πρωτότυπο της πλατφόρμας. Επίσης, παρουσιάζεται μια απλή, ενδεικτική εφαρμογή η οποία χρησιμοποιήθηκε για τις δοκιμές του συστήματος.

Κεφάλαιο 1

Εισαγωγή

Το πεδίο του κινητού και διάχυτου υπολογισμού αποτελεί ένα μεγάλο βήμα εξέλιξης μιας σειράς ερευνητικών δραστηριοτήτων που ξεκινούν από τα μέσα της δεκαετίας του 1970. Σχηματικά, η εξέλιξη αυτή αποτυπώνεται στο Σχ. 1[1]. Επίσης, οι λύσεις πολλών προηγούμενων προβλημάτων γίνονται πιο περίπλοκες - η αύξηση στην πολυπλοκότητα είναι πολλαπλασιαστική και όχι αθροιστική. Για παράδειγμα, ο σχεδιασμός και η υλοποίηση ενός κινητού υπολογιστικού συστήματος είναι πολύ δυσκολότερος από ένα κατανεμημένο, αν θέλουμε να επιτύχουμε την ίδια αξιοπιστία και σταθερότητα.



Σχ. 1

Ο κινητός υπολογισμός γεννήθηκε στις αρχές του 1990 με την έλευση των πρώτων φορητών υπολογιστών – laptop και των ασύρματων τοπικών δικτύων. Μολονότι οι βασικές αρχές σχεδίασης των κατανεμημένων συστημάτων συνέχισαν να έχουν εφαρμογή, τέσσερις νέοι περιορισμοί σχετικοί με την κινητικότητα επέβαλλαν την ανάπτυξη νέων τεχνικών:

1. Οι συνεχείς και απρόβλεπτες διακυμάνσεις στην ποιότητα του δικτύου.
2. Η μειωμένη εμπιστοσύνη/αξιοπιστία των κινητών στοιχείων.
3. Οι περιορισμένοι τοπικοί υπολογιστικοί πόροι.
4. Ο απαραίτητος έλεγχος κατανάλωσης ισχύος.

Ο κινητός υπολογισμός αποτελεί ακόμα και σήμερα ένα ενεργό και συνεχώς αναπτυσσόμενο πεδίο έρευνας.

Λίγο πριν το 2000, η έρευνα στον τομέα του κινητού υπολογισμού άρχισε να ασχολείται με ζητήματα που πλέον τα κατατάσσουμε στον τομέα του διάχυτου υπολογισμού. Το ιδρυτικό μανιφέστο για το πεδίο του διάχυτου υπολογισμού(**pervasive computing**) – εναλλακτικά, απανταχού υπολογισμός (**ubiquitous computing**) – γράφτηκε το 1991 στο άρθρο "**The computer for the 21st Century**" του Mark Weiser[2]. Ο συγγραφέας παρατήρησε ότι «οι πιο σημαντικές τεχνολογίες είναι αυτές που είναι αόρατες. Αυτές που ενσωματώνονται στα οικοδομήματα της καθημερινής ζωής και που τελικά γίνονται δυσδιάκριτες από αυτά». Συνεπώς, η ουσία του διάχυτου υπολογισμού είναι ο εμπλουτισμός του περιβάλλοντος των χρηστών με υπολογιστικές συσκευές που επικοινωνούν, με τρόπο αρμονικό και όχι παρεμβατικό ως προς την καθημερινότητα τους.

Όταν διατυπώθηκε, αυτή η ιδέα ήταν πολύ πιο μπροστά από την εποχή της. Οι τεχνολογίες υλικού και των επικοινωνιών που απαιτούνταν για την υλοποίησή της απλώς δεν υπήρχαν. Σήμερα, για πρώτη φορά, αυτές οι τεχνολογίες είναι πλέον διαθέσιμες, και μάλιστα ιδιαίτερα προσβάσιμες από τον καθένα σε λογικό κόστος. Πλατφόρμες υλικού όπως το Imote2 επιτρέπουν την κατασκευή φθηνών πρωτοτύπων. Λειτουργικά συστήματα όπως το TinyOS, σχεδιασμένα με βάση τις ιδιαίτερες απαιτήσεις του κινητού/διάχυτου υπολογισμού, επιταχύνουν την ανάπτυξη και βελτιώνουν την αξιοπιστία σημαντικά. Συνυπολογίζοντας, τέλος, την συνεχώς αυξανόμενη αποδοχή των smartphone[3] – κινητά τηλέφωνα με αισθητήρες θέσης, κίνησης, μόνιμη σύνδεση στο Internet και ισχυρά λειτουργικά συστήματα όπως το Android OS – το μέλλον που προέβλεψε ο Mark Weiser πριν 2 δεκαετίες είναι πιο κοντά από ποτέ.

Imote2
Σχ.2

Android OS Mobile
Σχ. 3

1.1 Στόχος

Ο βασικός στόχος της εργασίας είναι η υποστήριξη του ενδιάμεσου λογισμικού POBICOS σε κινητά τηλέφωνα Android, έτσι ώστε αυτά να μπορούν να φιλοξενούν εφαρμογές που, ως τώρα, λειτουργούσαν μόνο σε δίκτυα αισθητήρων. Με αυτό τον τρόπο ουσιαστικά θα δημιουργήσει μια κατανεμημένη, κινητή πλατφόρμα προγραμματισμού για κινητά τηλέφωνα. Επίσης, η εργασία έχει ως στόχο την υποστήριξη και του διαμοιρασμού και της εκτέλεσης εφαρμογών σε κινητά που βρίσκονται σε συγκεκριμένες περιοχές ενδιαφέροντος, παρέχοντας έτσι ένα ακόμα σημαντικό εργαλείο για δημιουργία εφαρμογών τύπου crowdsourcing.

Το μεγαλύτερο κομμάτι της εργασίας έχει αφιερωθεί στην μεταφορά του POBICOS από το TinyOS στο Android. Επίσης, αναπτύχθηκε μια υποδομή που διευκολύνει την ανακάλυψη και την επικοινωνία των κινητών συσκευών, χωρίς να προϋποθέτει δυνατότητα απ' ευθείας επικοινωνίας μεταξύ τους. Τέλος, αναπτύχθηκε ένα λειτουργικό πρωτότυπο της πλατφόρμας καθώς και μια απλή, ενδεικτική εφαρμογή για τις δοκιμές του συστήματος.

1.2 Σχετικές εργασίες

Η πλατφόρμα που αναπτύχθηκε στα πλαίσια αυτής της εργασίας βασίστηκε στο ενδιάμεσο λογισμικό POBICOS (***Platform for Opportunistic Behaviour in Incompletely Specified, Heterogeneous Object Communities***)[4]. Περιληπτικά, το POBICOS στοχεύει σε ετερογενή σύνολα αντικειμένων ικανά για επικοινωνία, υπολογισμό, συλλογή πληροφοριών περιβάλλοντος (**sensing**) και έλεγχο διακοπτών, ηλεκτρικών μοτέρ κλπ (**actuating**). Το ενδιάμεσο λογισμικό που τρέχει πάνω σε τέτοια αντικείμενα επιτρέπει την δημιουργία μιας ανοιχτής, κατανεμημένης, υπολογιστικής πλατφόρμας που μπορεί να φιλοξενήσει πολλές διαφορετικές εφαρμογές.

Η μεταφορά του ενδιάμεσου λογισμικού POBICOS στο λειτουργικό σύστημα Android και η υποστήριξη της επιθυμητής λειτουργικότητας, όπως περιγράφηκε παραπάνω, παρουσιάζει κάποιες ενδιαφέρουσες ιδιαιτερότητες και συνδυάζει χαρακτηριστικά από αρκετά πεδία έρευνας. Σχετικές εργασίες μπορούν να αναζητηθούν στους τομείς του κινητού και διάχυτου υπολογισμού, των ασύρματων δικτύων αισθητήρων, ή ακόμα και στο πεδίο του crowdsourcing**.** Παρακάτω αναφέρουμε μερικά ενδεικτικά project.

Για τον τομέα τον ασύρματων δικτύων αισθητήρων:

Αρχικά αναφέρουμε την πλατφόρμα ***Agilla***[5]. Το Agilla είναι ένα middleware για ασύρματα δίκτυα αισθητήρων και το προγραμματιστικό του μοντέλο βασίζεται σε κινητούς πράκτορες (mobile agents). Οι εν λόγω agents μπορούν να μετακινηθούν από κόμβο σε κόμβο, είτε με αντιγραφή είτε με μεταφορά. Είναι υλοποιημένο πάνω από το TinyOS, ενώ ο προγραμματισμός των εφαρμογών του γίνεται σε assembly. Για λόγους εξοικονόμησης ενέργειας, το middleware μπορεί να μεταφέρει τους agents σε κόμβους, με τρόπο που θα φέρει τον υπολογισμό κοντά στα δεδομένα, μειώνοντας έτσι τις –ακριβές σε ενέργεια– ασύρματες μεταδόσεις μηνυμάτων.

Το ***MagnetOS***[6] είναι ένα κατανεμημένο λειτουργικό σύστημα για ad-hoc δίκτυα αισθητήρων. Στόχος του είναι η εύκολη ανάπτυξη δικτυακών εφαρμογών με έλεγχο στην κατανάλωση ισχύος και συμπεριφορά που προσαρμόζεται στις εκάστοτε συνθήκες. Οι ετερογενείς κόμβοι τρέχουν ένα ενοποιημένο *Java virtual machine* και ουσιαστικά φιλοξενούν αντικείμενα Java. Τα αντικείμενα αυτά μπορούν να έχουν threads που μετακινούνται μεταξύ των κόμβων. Οι εφαρμογές του τοποθετούνται αυτόματα και με τρόπο διάφανο προς τον χρήστη στους κόμβους, με τρόπο που μειώνει της κατανάλωση ισχύος και αυξάνει τον χρόνο ζωής του συστήματος.

Το σύστημα ***SmartMessages***[7] έχει ως στόχο την ανάπτυξη ενός προγραμματιστικού μοντέλου και μιας αρχιτεκτονικής για δίκτυα ενσωματωμένων συστημάτων. Οι εφαρμογές που εκτελούνται στο δίκτυο αυτό μπορούν να είναι τετριμμένες, όπως πχ η απλή συλλογή και διάδοση δεδομένων δεδομένων, ή και πολύπλοκες εφαρμογές που απαιτούν συνεργασία πολλών κόμβων. Όπως και στο POBICOS, οι κόμβοι χαρακτηρίζονται από τους πόρους που παρέχουν.

Τέλος, το ***Linksmart Project***[8] (γνωστό και ως "Hydra") ανέπτυξε ένα ενδιάμεσο λογισμικό για ενσωματωμένες συσκευές. Το λογισμικό αυτό θα δίνει στους προγραμματιστές την δυνατότητα να χρησιμοποιούν πολλαπλές, ετερογενής φυσικές συσκευές για τις εφαρμογές τους, προσφέροντας μια εύχρηστη διεπαφή βασισμένη σε web-services για να ελέγχουν οποιαδήποτε συσκευή ανεξάρτητα από την τεχνολογία δικτύωσης που διαθέτει (Bluetooth, ZigBee, WiFi, RFID κλπ). Το Hydra υποστηρίζει μεθόδους ανακάλυψης συσκευών και υπηρεσιών, P2P επικοινωνία και αρχιτεκτονική βασισμένη σε σημασιολογικά μοντέλα.

Για τον τομέα του crowdsourcing σε κινητές συσκευές:

Ο Jeff Howe ήταν ο συγγραφέας που πρώτος εισήγαγε τον όρο crowdsourcing, σε ένα άρθρο του για το περιοδικό Wired, τον Ιούνιο του 2006[9]. Στο άρθρο αυτό μελετούσε τους νέους τρόπους με τους οποίους μπορεί κάποιος να εκμεταλλευτεί την συλλογική νοημοσύνη ενός μεγάλου πλήθος, μέσω του web, για την διεκπεραίωση εργασιών. Η Wikipedia ήταν ίσως το σημαντικότερο παράδειγμα που έδειξε ότι αυτό το μοντέλο μπορεί πράγματι να δουλέψει. Σήμερα στο web υπάρχουν εξειδικευμένες crowdsourcing πλατφόρμες , όπως το Mechanical Turk της Amazon[10] που επιτρέπει σε προγραμματιστές να διατελέσουν εργασίες που, υπολογιστικά, είναι ακόμη δύσκολο να υλοποιηθούν (π.χ. αναγνώριση αντικειμένων σε μία εικόνα). Η τελευταία τάση στο crowdsourcing αφορά την χρήση κινητών τηλεφώνων ως την 'απόλυτη' συσκευή συλλογής πληροφοριών, τόσο από τους αισθητήρες (π.χ. επιταχυνσιόμετρα) όσο και από τους χρήστες τους. Ακολουθεί σύντομη περιγραφή μερικών project που σήμερα βασίζονται στο *mobile crowdsourcing*:

Το ***txteagle***[11] είναι ένα σύστημα που δίνει την δυνατότητα σε ανθρώπους χαμηλού εισοδήματος να κερδίσουν μικρά χρηματικά ποσά ολοκληρώνοντας σύντομες και απλές εργασίες στα κινητά τους τηλέφωνα. Οι χρήστες συχνά πληρώνονται με ηλεκτρονικά αγαθά, όπως πχ με επιπλέον χρόνο ομιλίας. Το σύστημα ήδη λειτουργεί στην Κένυα και στην Ρουάντα σε συνεργασία με εταιρίες κινητής τηλεφωνίας. Οι εργασίες αποστέλλονται στους χρήστες με SMS/MMS και αφορούν κυρίως μεταφράσεις, απομαγνητοφωνήσεις και συμμετοχή σε έρευνες/ψηφοφορίες.

Το ***mCrowd***[12] είναι μια πλατφόρμα για κινητές και διάχυτες εργασίες συλλογής δεδομένων. Υλοποιεί δύο λειτουργίες crowdsourcing: Η πρώτη, συμβατική λειτουργία, χρησιμοποιεί ήδη υπάρχουσες υπηρεσίες όπως το Amazon Mechanical Turk για να κατανείμει εργασίες σε χρήστες που δουλεύουν από το κινητό τους και όχι από τον σταθερό υπολογιστή τους. Η δεύτερη λειτουργία επιτρέπει την συλλογή δεδομένων από τους χρήστες. Για παράδειγμα, μπορεί να ζητηθεί μια φωτογραφία ενός γνωστού αξιοθέατου, η αποστολή μιας συγκεκριμένης ηχογραφημένης λέξης κλπ. Η πλατφόρμα αυτή λειτουργεί ως μια απλή εφαρμογή στο iPhone.

Τέλος, η εφαρμογή ***Google Maps*** για το Android, αντίθετα από τις προηγούμενες πλατφόρμες, δεν χρησιμοποιεί τους χρήστες αλλά την ίδια την συσκευή για συλλογή χρήσιμων δεδομένων. Η συγκεκριμένη εφαρμογή παρέχει πληροφορίες κυκλοφοριακού φόρτου για την περιοχή που κινείται ο χρήστης. Για να το πετύχει αυτό, συλλέγει, μεταξύ άλλων, τα δεδομένα που παρέχονται από τα επιταχυνσιόμετρα των συσκευών στην γύρω περιοχή. Η λειτουργία αυτή αναφέρεται και ως *crowd-sensing.*

Κεφάλαιο 2

Βασικές Έννοιες

Στο κεφάλαιο αυτό γίνεται μια εισαγωγή στις κύριες τεχνολογίες που χρησιμοποιήθηκαν για την ολοκλήρωση της διπλωματικής εργασίας.

2.1 nesC

Η nesC[13] είναι μια γλώσσα προγραμματισμού για δικτυωμένα ενσωματωμένα συστήματα. Ένα παράδειγμα ενός τέτοιου συστήματος είναι ένα δίκτυο αισθητήρων που (πιθανόν) αποτελείται από χιλιάδες μικροσκοπικές, χαμηλής ισχύος συσκευές, κάθε μια από τις οποίες εκτελεί ταυτόχρονα, οδηγούμενα από γεγονότα προγράμματα (**event-driven**) τα οποία πρέπει να λειτουργήσουν με σημαντικές ελλείψεις σε μνήμη και ενέργεια.

Στόχος της nesC είναι να καλύψει τις ιδιαίτερες ανάγκες αυτού του τομέα, παρέχοντας ένα προγραμματιστικό μοντέλο που συνδυάζει εκτέλεση με βάση γεγονότα (event-driven execution), ευέλικτο μοντέλο ταυτόχρονου προγραμματισμού και «**component-oriented**» σχεδίαση προγραμμάτων. Τα χαρακτηριστικά αυτού του μοντέλου επιτρέπουν στον μεταγλωττιστή της nesC να κάνει πολλές βελτιστοποιήσεις (μείωση μεγέθους του κώδικα) και να αναλύει τον κώδικα εντοπίζοντας πιθανά bug (πχ προβλήματα τύπου *data-race)*.

Η nesC χρησιμοποιήθηκε για την ανάπτυξη του TinyOS, ενός μικρού λειτουργικού συστήματος για δίκτυα αισθητήρων, όπως επίσης και για άλλες σημαντικές εφαρμογές με αισθητήρες. Τόσο η nesC όσο και το TinyOS χρησιμοποιούνται από πολλές ερευνητικές ομάδες σε διεθνές επίπεδο, και μέχρι τώρα δείχνουν να ανταποκρίνονται καλά στην υποστήριξη του πολύπλοκου, ταυτόχρονου προγραμματιστικού μοντέλου που απαιτείται για αυτό το νέο είδος δικτυωμένων συστημάτων.

2.2 TinyOS

Το TinyOS[14] είναι ένα λειτουργικό σύστημα σχεδιασμένο συγκεκριμένα για δικτυωμένα ενσωματωμένα συστήματα. Το TinyOS προσφέρει ένα προγραμματιστικό μοντέλο που είναι κατάλληλο για event-driven προγραμματισμό και επίσης χρειάζεται ελάχιστο χώρο (ο πυρήνας του TinyOS απαιτεί 400 bytes κώδικα και μνήμης, μαζί). Τα βασικά χαρακτηριστικά του TinyOS συνοψίζονται στα εξής:

Αρχιτεκτονική βασισμένη σε Components

Το TinyOS παρέχει ένα σύνολο από επαναχρησιμοποιήσιμα components συστήματος. Μια εφαρμογή μπορεί να συνδέσει διάφορα components χρησιμοποιώντας το λεγόμενο *wiring specification.* Αν και τα περισσότερα components υλοποιούν κάποια λογική σε software, μερικά είναι πολύ χαμηλού επιπέδου και απλώς εξάγουν μια διεπαφή με το hardware.

Tasks και event-based ταυτόχρονος προγραμματισμός

Στο TinyOS υπάρχουν δύο μηχανισμοί ταυτόχρονου προγραμματισμού: Τα *tasks* και τα *events.* Τα **tasks** είναι ένας μηχανισμός υποβολής ασύγχρονων εργασιών προς εκτέλεση. Εκτελούνται μέχρι τέλους και δεν διακόπτουν το ένα το άλλο (**non-preemptive**). Τα components μπορούν να καταθέσουν τέτοιες εργασίες χρησιμοποιώντας την εντολή *post*. Η συγκεκριμένη εντολή τοποθετεί μια εργασία στην ουρά του *TinyOS Task scheduler* και επιστρέφει αμέσως. Η χρήση αυτής της μεθόδου ταυτόχρονου προγραμματισμού είναι κατάλληλη για ενέργειες που δεν έχουν αυστηρές απαιτήσεις ως προς τον χρόνο εκτέλεσης. Τα **events** επίσης εκτελούνται μέχρι τέλους, αλλά μπορούν να διακόψουν ένα άλλο event ή task (**preemptive).** Τα events ειδοποιούν είτε για την ολοκλήρωση μιας ενέργειας 2 φάσεων (περισσότερες πληροφορίες για αυτό στην επόμενη ενότητα) είτε για ένα γεγονός από το εξωτερικό περιβάλλον (πχ λήψη μηνύματος ή λήξη ενός timer). Η εκτέλεση των προγραμμάτων στο TinyOS ουσιαστικά οδηγείται από διαδοχικά εξωτερικά γεγονότα.

Επιπλέον, τα χαμηλού επιπέδου γεγονότα (πχ η λήξη ενός timer) προκαλούν **interrupts.** Τα interrupts αυτά εξυπηρετούνται από τους ***asynchronous event handlers,*** οι οποίοι, αναπόφευκτα, εισάγουν μεγαλύτερη πολυπλοκότητα και μπορούν να δημιουργήσουν κλασσικά προβλήματα ταυτόχρονου προγραμματισμού (πχ data – races). Οι *asynchronous event handlers* είναι ο λόγος που η πλατφόρμα υποστηρίζει preemptive εκτέλεση.

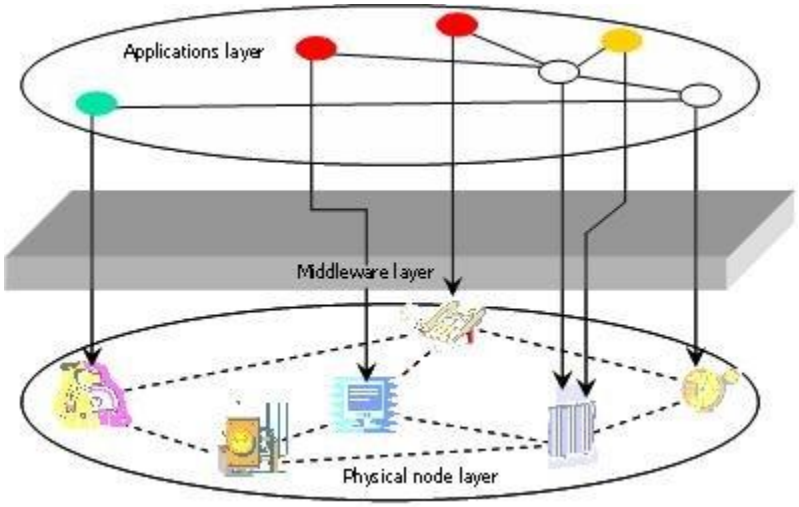
Ενέργειες 2 φάσεων

Επειδή τα tasks εκτελούνται χωρίς να αλληλοδιακόπτονται, το TinyOS δεν έχει λειτουργίες που μπλοκάρουν την εκτέλεση. Όλες οι χρονοβόρες διαδικασίες είναι δύο φάσεων: Η κλήση της διαδικασίας και η ολοκλήρωσή της είναι δυο διαφορετικές διεργασίες. Τα **commands** αποτελούν τον μηχανισμό κλήσης μιας διαδικασίας. Αν η συγκεκριμένη διαδικασία είναι δύο φάσεων, η κλήση της επιστρέφει αμέσως και η ολοκλήρωσή της θα σημανθεί με ένα event. Διαδικασίες που δεν είναι δύο φάσεων (πχ το άνοιγμα ενός LED) δεν παρέχουν events ολοκλήρωσης. Ένα κλασσικό παράδειγμα διαδικασίας δύο φάσεων είναι η αποστολή ενός πακέτου: ένα component καλεί την εντολή *send* για να ξεκινήσει την αποστολή ενός ασύρματου μηνύματος και δέχεται ένα event *sendDone* όταν η αποστολή έχει ολοκληρωθεί. Κάθε component υλοποιεί την μία από τις δύο φάσεις και καλεί την άλλη.

2.3 POBICOS

Το POBICOS είναι ένα ενδιάμεσο λογισμικό που στοχεύει σε ετερογενή σύνολα συνηθισμένων αντικειμένων τα οποία φιλοξενούν κόμβους ικανούς για επικοινωνία, υπολογισμό, συλλογή πληροφοριών περιβάλλοντος (**sensing**) και έλεγχο διακοπτών, ηλεκτρικών μοτέρ κλπ (**actuating**). Λειτουργεί σε τέτοιους κόμβους και επιτρέπει την δημιουργία μιας «κοινότητας αντικειμένων» - μιας ανοιχτής, κατανεμημένης, υπολογιστικής πλατφόρμας που μπορεί να φιλοξενήσει εφαρμογές. Οι εφαρμογές αυτές χρησιμοποιούν sensors και actuators που ανήκουν στα αντικείμενα της εν λόγω κοινότητας με τρόπο μη παρεμβατικό προς τον χρήστη, ούτως ώστε να επιτύχουν τον εκάστοτε στόχο τους.

Το POBICOS υποθέτει ότι οι κοινότητες αντικειμένων σχηματίζονται *ad-hoc*, χωρίς να υπάρχει κάποιο συγκεκριμένο πλάνο. Επομένως, δύο κοινότητες αντικειμένων δεν μπορεί να είναι ίδιες. Επιπλέον, οι κοινότητες αντικειμένων είναι ελλιπώς ορισμένες κατά την φάση του σχεδιασμού της εφαρμογής. Ως αποτέλεσμα, μια εφαρμογή POBICOS θα πρέπει να επιδεικνύει ευκαιριακή συμπεριφορά, δηλαδή θα πρέπει να εκμεταλλεύεται με τον καλύτερο δυνατό τρόπο όσους πόρους, sensors και actuators βρει διαθέσιμους στην συγκεκριμένη κοινότητα αντικειμένων. Ο στόχος του POBICOS είναι η δημιουργία μιας πλατφόρμας (εργαλεία και ενδιάμεσο λογισμικό - **middleware**) που θα επιτρέπει την ανάπτυξη και εκτέλεση εφαρμογών που επιδεικνύουν ευκαιριακή συμπεριφορά σε ετερογενής, ελλιπώς ορισμένες κοινότητες αντικειμένων.



Το POBICOS middleware - Σχ. 4

Το προγραμματιστικό μοντέλο του POBICOS και το API του middleware, με σκοπό να διευκολύνουν τον προγραμματισμό της ευκαιριακής συμπεριφοράς, αποτελούνται από δύο μέρη: Το *domain-neutral* και το *domain-specific.* Κάθε εφαρμογή POBICOS αποτελείται από ένα *δέντρο* συνεργαζόμενων κινητών

οντοτήτων, τους λεγόμενους ***micro-agents.*** Εσωτερικά, κάθε micro-agent είναι μια συλλογή από event-handler. Τα domain-neutral στοιχεία αποτελούνται από λειτουργίες όπως η δημιουργία των micro-agent και η επικοινωνία μεταξύ τους. Αυτά τα στοιχεία είναι διαθέσιμα σε κάθε κόμβο POBICOS και στο εξής θα αναφέρονται ως ***generic.***

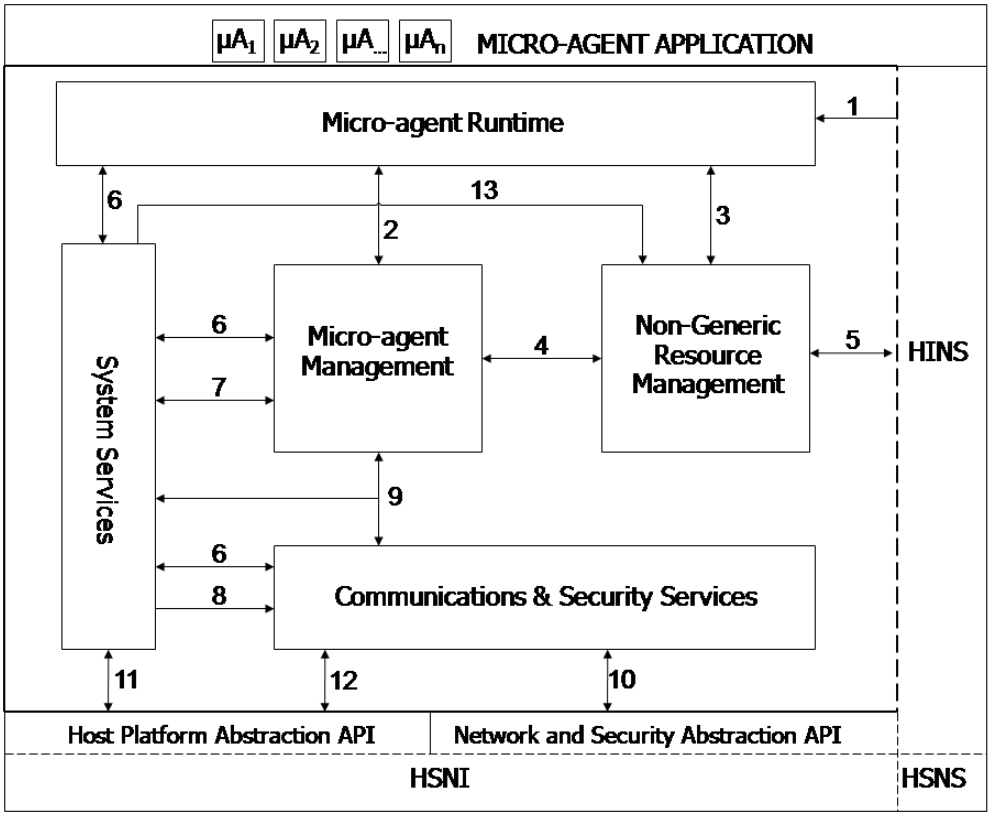
Τα domain-specific στοιχεία επιτρέπουν στον προγραμματιστή να σκέφτεται με βάση ένα συγκεκριμένο χώρο εφαρμογών (για παράδειγμα εφαρμογές για αυτοματισμό σπιτιού). Χρησιμοποιώντας την αντίστοιχη οντολογία που μοντελοποιεί τον εκάστοτε χώρο εφαρμογών, ο προγραμματιστής μπορεί να χρησιμοποιήσει τους ανάλογους sensor και actuators. Τα domain-specific στοιχεία επιτρέπουν στον προγραμματιστή να εκτελέσει ενέργειες που χρησιμοποιούν sensors και actuators εγκαθιστώντας τους ανάλογους event handler / micro agents. Μόνο μερικά από τα υποστηριζόμενα domain-specific χαρακτηριστικά είναι διαθέσιμα σε κάθε κόμβο, γι' αυτό και στο εξής θα αναφέρονται ως ***non-generic.***

Εσωτερικά, το ενδιάμεσο λογισμικό POBICOS περιλαμβάνει, μεταξύ άλλων:

1. Ένα AVR virtual machine για την εκτέλεση των micro-agent.
2. Μονάδα διαχείρισης micro-agent, υπεύθυνη για την επικοινωνία μεταξύ τους καθώς και για την τοποθέτησή τους σε κόμβους.
3. Μονάδα διαχείρισης non-generic πόρων.
4. Υπηρεσίες συστήματος, έλεγχο συστήματος, λειτουργίες ειδικών κόμβων.
5. Υπηρεσίες επικοινωνίας και ασφάλειας.

Τα παραπάνω component χρησιμοποιούν αρκετά πρωτόκολλα, όπως πχ το *Host Candidate Discovery Protocol.*

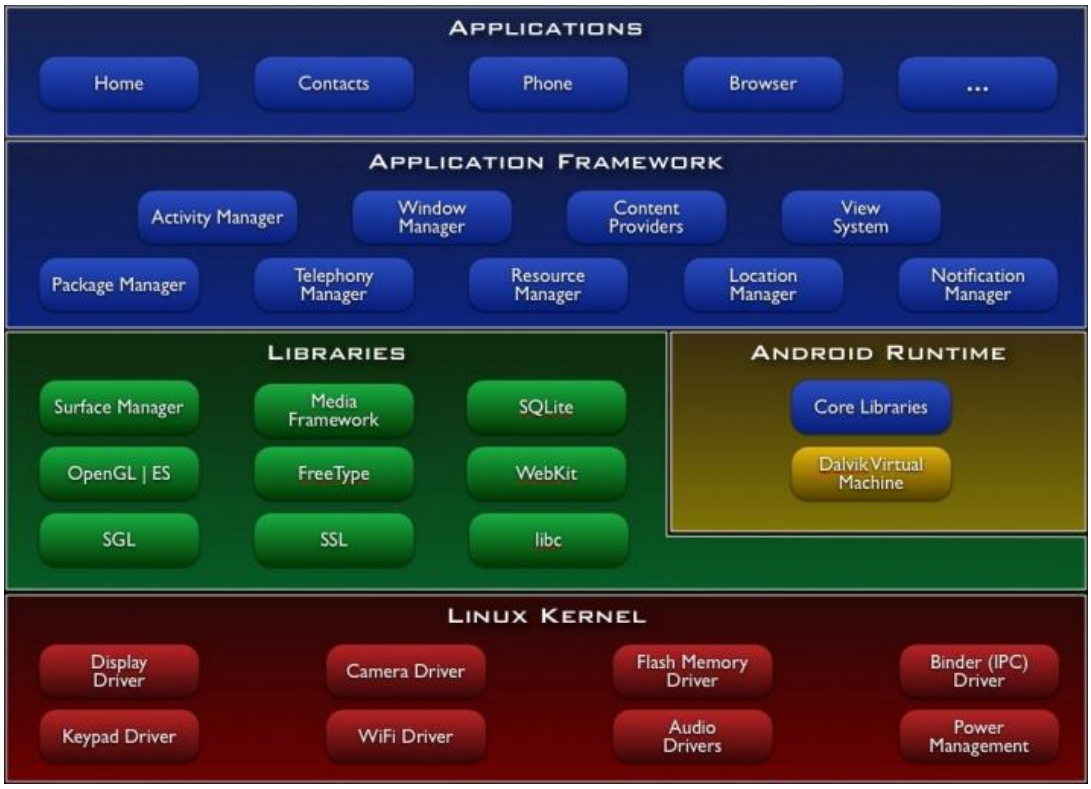
Το middleware έχει σχεδιαστεί με έμφαση στην εύκολη μεταφορά και φορητότητά του. Τα περισσότερα component είναι ανεξάρτητα από την υποκείμενη τεχνολογία δικτύωσης (πχ ZigBee) και από τους sensor και actuator των κόμβων. Ο διαχωρισμός επιτυγχάνεται με την χρήση αφαιρετικών API. Αυτή η σχεδιαστική αρχή έπαιξε καθοριστικό ρόλο στην επιτυχημένη μεταφορά του POBICOS στο Android OS.



Η αρχιτεκτονική του ενδιάμεσου λογισμικού POBICOS - Σχ. 5

2.4 Android OS

Το λειτουργικό σύστημα Android[15] είναι ένα νέο ΛΣ για κινητές συσκευές το οποίο απευθύνεται κυρίως στην αγορά των smartphone, αλλά πλέον εισάγεται και στην αγορά των tablet και netbook. Όπως φαίνεται από το παρακάτω σχήμα αποτελείται από ένα σύνολο επιπέδων λογισμικού.



Η software stack του Λειτουργικού Συστήματος Android - Σχ. 6

Στο κατώτερο επίπεδο βρίσκεται ένας τροποποιημένος πυρήνας του Linux 2.6. Στον πυρήνα στηρίζονται βασικές λειτουργίες του συστήματος όπως ασφάλεια, διαχείριση μνήμης, διαχείριση διεργασιών, δικτύωση και διεπαφή με τους οδηγούς των συσκευών. Ο πυρήνας λειτουργεί επίσης ως αφαιρετικό επίπεδο μεταξύ του hardware και των υψηλότερων επιπέδων software.

Πάνω από το επίπεδο του πυρήνα υπάρχουν δύο επίπεδα, οι βιβλιοθήκες και το *Android Runtime.* Οι βιβλιοθήκες αποτελούνται από ένα σύνολο C/C++ βιβλιοθηκών που χρησιμοποιούνται από διάφορα μέρη του Android. Η μεταφορά δεδομένων μεταξύ του Application Framework Layer και του πυρήνα γίνεται μέσω αυτών την βιβλιοθηκών. Το *Android Runtime* αποτελείται από το *Dalvik Virtual Machine*[16] *(Dalvik VM)* και από βιβλιοθήκες οι οποίες δίνουν ένα μεγάλο μέρος της λειτουργικότητας που παρέχεται από τις βασικές βιβλιοθήκες τις Java.

Κάθε εφαρμογή Android τρέχει σε δική της διεργασία, μέσα στο δικό της Dalvik VM. Το Dalvik είναι βελτιστοποιημένο ώστε μια συσκευή να μπορεί να τρέχει ταυτόχρονα πολλά VM με αποδοτικό τρόπο. Το Dalvik VM εκτελεί αρχεία της μορφής *Dalvik Executable (.dex)* η οποία είναι βελτιστοποιημένη για ελάχιστες απαιτήσεις σε μνήμη. Επίσης, το συγκεκριμένο VM στηρίζεται στον πυρήνα Linux για λειτουργίες όπως χαμηλού επιπέδου διαχείριση μνήμης και threading.

Πάνω από τα δύο προηγούμενα επίπεδα, βρίσκεται το *Application Framework*. Το επίπεδο αυτό δίνει στους προγραμματιστές μια μεγάλη γκάμα λειτουργιών, παρουσιάζοντας ακόμα πιο αφαιρετικά τις βιβλιοθήκες των χαμηλότερων επιπέδων.

Τέλος, στο πιο πάνω επίπεδο βρίσκονται οι εφαρμογές, οι οποίες χτίζονται στην Java με την χρήση του Application Framework.

2.4.1 Android NDK - JNI

Το NDK[17] είναι ένα σύνολο εργαλείων που δίνει την δυνατότητα στους προγραμματιστές να ενσωματώσουν στις εφαρμογές τους κομμάτια κώδικα που είναι γραμμένα σε C ή C++.

Ο μηχανισμός που το NDK επιτυγχάνει αυτή την λειτουργικότητα είναι απλός: Ο κώδικας που είναι γραμμένος σε C/C++ πρέπει πρώτα να μετατραπεί σε δυναμικά διαμοιραζόμενη βιβλιοθήκη και έπειτα, με την χρήση του *JNI* (**Java Native Interface**) οι ζητούμενες συναρτήσεις μπορούν να κληθούν κανονικά. Η εν λόγω βιβλιοθήκη εγκαθίσταται στην συσκευή μαζί με το Android app.

Κεφάλαιο 3

Απαιτήσεις και Προδιαγραφές

Σε αυτό το κεφάλαιο γίνεται μια καταγραφή των βασικών προδιαγραφών.

Από την αρχή της διπλωματικής εργασίας τέθηκαν συγκεκριμένες απαιτήσεις, οι οποίες περιγράφονται παρακάτω:

1. Το σύστημα θα πρέπει να λειτουργεί με τρόπο μη παρεμβατικό προς τους χρήστες των κινητών συσκευών, στο πνεύμα του διάχυτου υπολογισμού. Ιδανικά, ο χρήστης απλώς θα εγκαθιστά την εφαρμογή και δεν θα ασχολείται με περαιτέρω τεχνικές λεπτομέρειες. Στο ίδιο πνεύμα, το *Rooting* της συσκευής είναι ανεπιθύμητο.
2. Οι κινητές συσκευές θα πρέπει να χρησιμοποιούν ευκαιριακά οποιοδήποτε τύπο σύνδεσης είναι διαθέσιμος (GPRS ή WiFi) με τρόπο διάφανο προς τον χρήστη.
3. Οι μικρο-εφαρμογές θα πρέπει να υποστηρίζουν συγκεκριμένες απαιτήσεις θέσης και να εκτελούνται μονάχα στις συσκευές που ικανοποιούν τις απαιτήσεις αυτές.
4. Οι micro-agents θα πρέπει να υποστηρίζουν την χρήση non-generic στοιχείων, όπως αυτά περιγράφηκαν στην ενότητα 2.3 .
5. Το POBICOS θα πρέπει να μεταφερθεί με τρόπο που θα επιτρέπει την χρήση του διαθέσιμου toolset για την παραγωγή micro-agent.
6. Απαραίτητη, τέλος, είναι η δημιουργία μιας δοκιμαστικής εφαρμογής που θα χρησιμοποιεί τουλάχιστον ένα non-generic στοιχείο και που θα απαιτεί επικοινωνία μεταξύ των κινητών συσκευών.

Στα επόμενα κεφάλαια ακολουθεί ο σχεδιασμός και η υλοποίηση των βασικών οντοτήτων για την επίτευξη των στόχων που τέθηκαν εδώ.

Κεφάλαιο 4

Μετατροπή POBICOS/TinyOS

4.1 Εισαγωγικές πληροφορίες

Το ενδιάμεσο λογισμικό POBICOS έχει χτιστεί πάνω στο TinyOS και την nesC. Αν και η hardware πλατφόρμα που στοχεύουν είναι συσκευές τύπου Imote, τόσο το POBICOS όσο και το TinyOS παρέχουν simulators για PC,ώστε να διευκολύνουν την διαδικασία της ανάπτυξης. Χρησιμοποιώντας λοιπόν τα αντίστοιχα εργαλεία (TOSSIM και το POBICOS toolset) , ο κώδικας του POBICOS, που κανονικά αποτελείται από εκατοντάδες αρχεία κώδικα nesC, μετατρέπεται σε ένα μοναδικό αρχείο C (**app.c)**. Η μετατροπή αυτή γίνεται ως ένα ενδιάμεσο στάδιο μεταγλώττισης, πριν παραχθεί το τελικό εκτελέσιμο (binary) αρχείο.

Συνεπώς, μπορούμε να χτίσουμε το παραγόμενο app.c ως μια διαμοιραζόμενη βιβλιοθήκη και να το ενσωματώσουμε στο τελικό Android application, χρησιμοποιώντας το *Native Development Kit*. Το NDK, όπως περιγράφεται στην ενότητα 2.4.1, παρέχει το *JNI Framework,* το οποίο μας δίνει την δυνατότητα να δημιουργήσουμε μια C ↔ Java διεπαφή.

Το JNI Framework μας δίνει δύο βασικά εργαλεία:

- Τον μηχανισμό **JNI Call.** Χρησιμοποιώντας JNI Calls, μια εφαρμογή Java/Android μπορεί να καλέσει εγγενής (**native)** συναρτήσεις μιας ήδη φορτωμένης, διαμοιραζόμενης βιβλιοθήκης C.
- Τον μηχανισμό **JNI Callback.** Χρησιμοποιώντας JNI Callbacks, κώδικας από μία βιβλιοθήκη C μπορεί να έχει πρόσβαση σε κλάσεις/αντικείμενα στην Java όπως επίσης και να καλέσει τις αντίστοιχες μεθόδους τους.

Για την υλοποίησή μας σχεδιάσαμε το **Android PoAPI Layer ,** το οποίο αποτελείται από αντικείμενα Java τα οποία λειτουργούν ως «εντολοδόχοι» (**proxies)** για τις κλήσεις JNI Call/Callback. Αναλυτικότερα:

- Χαμηλού επιπέδου TinyOS **commands –**που αντιστοιχούν σε χαμηλού επιπέδου λειτουργίες της πλατφόρμας, πχ αποστολή μηνύματος **–** θα καλούν κάποια μέθοδο του PoAPI επιπέδου χρησιμοποιώντας τον μηχανισμό **JNI Callback**, μέσω του δείκτη περιβάλλοντος του Java VM (***JNIEnv**) [18].

```
async command error_t PoUARTHALI.txByte(uint8_t data) {
/*
 * JNI Callback
 */

 (*cached_JNIEnv)->CallIntMethod(cached_JNIEnv, cached_uart_obj,
         cached_UARTTxByte_mid, data);

 signal PoUARTHALI.txByteReady(SUCCESS);
 return SUCCESS;
}
```

Παράδειγμα χρήσης του μηχανισμού JNI Callback - Σχ. 7

- Χαμηλού επιπέδου TinyOS **events** που αντιστοιχούν σε χαμηλού επιπέδου συμβάντα της πλατφόρμας, πχ άφιξη μηνύματος, θα «περιτυλίγονται» μέσα σε συναρτήσεις C (thin-wrapped) ούτως ώστε το επίπεδο PoAPI να μπορεί να τα ενεργοποιήσει χρησιμοποιώντας τον μηχανισμό **JNI Call.**

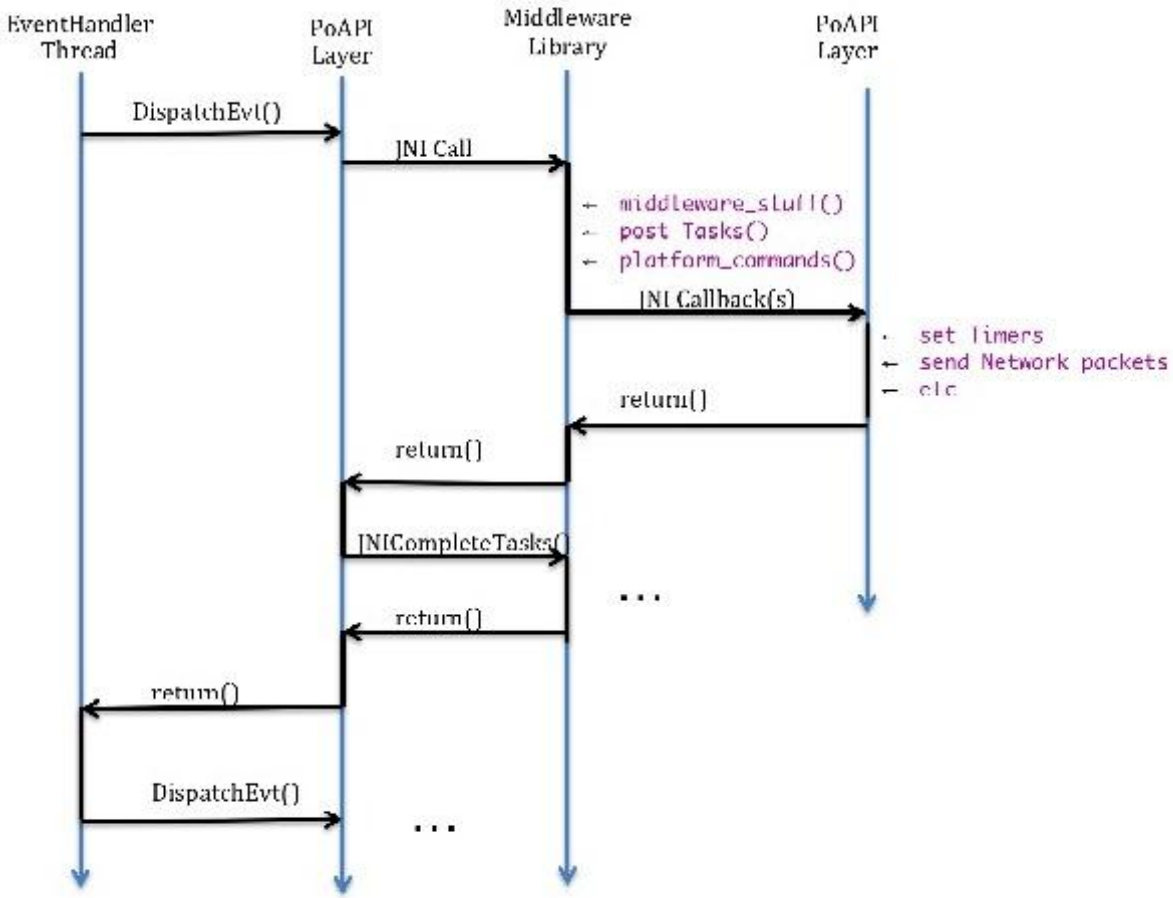
```
void Java_org_lekkas_poclient_PoAPI_UARTService_nativeRxByte(JNIEnv *env,
jobject obj, jbyte b) __attribute__ ((C, spontaneous)) {

signal PoUARTHALI.rxByteReady(b, SUCCESS);
}
```

Παράδειγμα thin-wrapped event για JNI Call - Σχ. 8

```
...
private native void nativeRxByte(byte b);
...
public void JNICall_RxByteReady(byte data) {
nativeRxByte(data);
}
```

Παράδειγμα ορισμού ενός JNI Call στην Java - Σχ. 9

4.2 Διάγραμμα ροής κλήσεων/μηνυμάτων

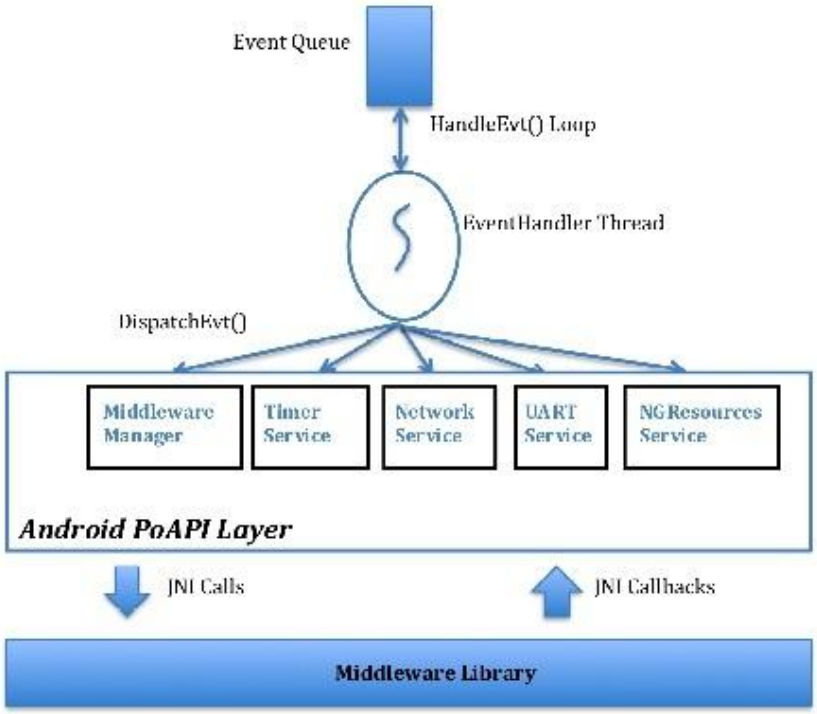


Σχ. 10

Όπως περιγράφηκε στην ενότητα 2.2, η εκτέλεση του middleware συνεχίζει να είναι *event-driven* με την διαφορά ότι, πλέον, τα συμβάντα προέρχονται από την πλατφόρμα Android. Επίσης, το *EventHandler Thread* που για πρώτη φορά παρουσιάζεται εδώ, δημιουργείται και ελέγχεται από την εφαρμογή Android. Οι δύο μοναδικές του λειτουργίες είναι:

1. Απομάκρυνση των συμβάντων από την αντίστοιχη ουρά χρησιμοποιώντας την μέθοδο *HandleEvt().*
2. Εξυπηρέτηση των συμβάντων με την χρήση της μεθόδου *DispatchEvt().*

4.3 Αρχιτεκτονική λειτουργίας του POBICOS middleware στο Android



Σχ. 11

Το PoAPI Layer αποτελείται από κλάσεις Java οι οποίες δημιουργούνται ακριβώς μία φορά (**singleton) .** Είναι το επίπεδο διεπαφής με το υποκείμενο POBICOS middleware. Όπως φαίνεται στο προηγούμενο σχήμα, το PoAPI κάνει δύο πράγματα:

1. Δέχεται JNI Callbacks τα οποία και «μεταφράζει» σε λειτουργίες συμβατές με το περιβάλλον Android. Για παράδειγμα, ένα JNI Callback που θέτει ένα Timer εν τέλει θα προχωρήσει στην χρήση των αντίστοιχων συναρτήσεων του *Android Application Framework* (ενότητα 2.4).
2. Ενημερώνει το middleware για γεγονότα με την χρήση JNI Calls που με την σειρά τους προκαλούν *event signaling*, μιλώντας με όρους του TinyOS.

Για λόγους αξιοπιστίας και σταθερότητας, όλα τα αντικείμενα του PoAPI χρησιμοποιούνται από ένα και μόνο Thread, αυτό του *EventHandler.*Νήματα που θέλουν να ενημερώσουν το middleware για ένα συμβάν (πχ την λήξη ενός Timer, τη λήψη ενός μηνύματος κλπ) τοποθετούν το συμβάν στην ουρά EventQueue.Ο EventHandler, με σειριακό τρόπο, θα φροντίσει να ενημερώσει το middleware για τα event αυτά.

Επίσης, το EventHandler Thread εξυπηρετεί κάθε event μέχρι τέλους. Η εκτέλεση των TinyOS task που πιθανόν να έχουν μπει στην ουρά του TinyOS scheduler γίνεται *πρίν* την εξυπηρέτηση του επόμενου event. Όπως θα δείξουμε σε επόμενη ενότητα, ο scheduler έχει υποστεί μερικές μικρές αλλαγές ώστε να υποστηρίζεται η συγκεκριμένη λειτουργία.

4.4 Περιγραφή component/συναρτήσεων για την υποστήριξη του POBICOS στο Android

Η βασική ιδέα είναι να προσαρμοστούν μονάχα τα component που είναι «κοντά» στο hardware, κρατώντας το υπόλοιπο (Hardware Independent – Node Independent) middleware απαράλλαχτο. Τα components που θα χρειαστεί να τροποποιηθούν θα πρέπει να υποστηρίξουν τον μηχανισμό JNI Call/Callback που περιγράφηκε προηγουμένως. Η περιγραφή που ακολουθεί βασίστηκε στα API που ορίζονται στο Deliverable 2.1.3 του POBICOS project .

4.4.1 Host Platform Abstraction API

- **PoMemHALI (PoHWMemM) :** Το component που είναι υπεύθυνο για την χαμηλού επιπέδου διαχείριση μνήμης. Παραμένει ίδιο με αυτό που χρησιμοποιείται για το Imote, χωρίς καμία αλλαγή. Η μνήμη είναι στατική και δεν υπάρχει δυνατότητα δυναμικής δέσμευσης.
- **PoHostHALI (PoHWHostM) :** Δεν χρειάζεται αλλαγές.
- **PoUARTHALI (PoUARTM) :** Υπεύθυνο για την επικοινωνία με την σειριακή θύρα. Στα Imote η σειριακή θύρα χρησιμοποιείται για την φόρτωση/εκκίνηση της εφαρμογής (**application pill sequence**). Για τις ανάγκες της δικής μας πλατφόρμας, θα δημιουργήσουμε μια εικονική θύρα. Η αποστολή δεδομένων από το middleware στην υποκείμενη Android εφαρμογή θα γίνεται με χρήση JNI Callback. Η αντίστροφη διαδικασία θα γίνεται με την χρήση JNI Call. Συγκεκριμένα:

  - *JNI Callbacks*
    - command error_t PoUARTHALI.txByte(uint8_t data)
    - command error_t PoUARTHALI.txStream(uint8_t* data, uint8_t len)
  - *JNI Calls*
    - signal PoUARTHALI.rxByteReady(uint8_t data, error_t error)

  Σημείωση: Η txStream() βασίζεται στην txByte().

— **Timers (PoSWTimerM) :** Υπεύθυνο για τις υπηρεσίες Timer.
Ειδικότερα:

- *JNI Callbacks*
  - command void Timer.startPeriodic[uint8_t num] (uint32_t dt)
  - command void Timer.startOneShot[uint8_t num] (uint32_t dt)
  - command void Timer.stop[uint8_t num] ()
  - command bool Timer.isRunning[uint8_t num] ()
  - command bool Timer.isOneShot[uint8_t num] ()
  - command void Timer.startPeriodicAt[uint8_t num] (uint32_t t0, uint32_t dt)
  - command void Timer.startOneShotAt[uint8_t num] (uint32_t t0, uint32_t dt)
  - command uint32_t Timer.getNow[uint8_t num] ()
  - command uint32_t Timer.gett0[uint8_t num] ()
  - command uint32_t Timer.getdt[uint8_t num] ()
- *JNI Signals*
  - signal Timer.fired[num] ()

Σημειώσεις:

- Οι startPeriodic() και startOneShot() βασίζονται στις startPeriodicAt() και startOneShotAt() αντίστοιχα.
- Οι startPeriodicAt() και startOneShotAt() χρησιμοποιούν το ίδιο JNI Callback.

4.4.2 Network and Security Abstraction API

- **PoHWCommM :** Το συγκεκριμένο component αναλαμβάνει την αποστολή και την λήψη μη-αξιόπιστων πακέτων δεδομένων (**unreliable datagrams)** . Όλες οι ενέργειες επικοινωνίας εν τέλει καταλήγουν στην χρήση αυτού του component και των αντίστοιχων command/event που αυτό παρέχει.

  - *JNI Callbacks*
    - command error_t PoDatagramTransportHALI.sendMsg(addrtype addr, int maxHops, PoMsg_t poMsg);
    - command error_t PoNetworkMngrHALI.joinNetwork(idtype NetworkID, keytype NetworkKey, addrtype *Addr);
    - command error_t PoNetworkMngrHALI.leaveNetwork();
  - *JNI Calls*
    - signal PoDatagramTransportHALI.msgArrived(addrtype src, int Hop, PoMsg_t rxMsg);

4.4.3 Resource Abstraction API

Το συγκεκριμένο API εξαρτάται από τα non generic primitives (events & instructions) που υποστηρίζονται από την συσκευή Android. Για τις ανάγκες της εργασίας υποστηρίχθηκε ένα σχετικά μικρό υποσύνολο των non-generic primitives ώστε να υπάρχει η υποδομή για την δημιουργία εφαρμογών με δυνατότητα διαδραστικότητας με τον χρήστη. Η υποστήριξη επιπλέον εντολών/γεγονότων είναι τετριμμένη και ίσως πραγματοποιηθεί σε επόμενες εκδόσεις της πλατφόρμας.

- **PoResourcesHALI (PoHWResourcesM) :**

  - *JNI Callbacks*
    - command void PoResourcesHALI.pongiDismissDialog();
    - command void PoResourcesHALI.pongiGetDialogInput();
    - command void PoResourcesHALI.pongiCreateDialog(uint8_t *text, uint32_t seconds);
    - command void PoResourcesHALI.pongiNotifyByDisplayingText(uint8_t* msg, uint32_t millis);
    - command void PoResourcesHALI.pongiAlertByDisplayingText();
  - *JNI Calls*
    - signal PoResourcesHALI.dialogInputReceivedEvent();
    - signal PoResourcesHALI.dialogInputTimeoutEvent();

4.4.4 TinyOS2.x Task Scheduler & Boot Sequence.

Όπως αναφέρθηκε στην ενότητα 4.3 , ο task scheduler πρέπει να μπορεί να κληθεί από το επίπεδο PoAPI. Παρακάτω παρουσιάζεται το Boot Sequence του TinyOS 2.x , όπως είναι υλοποιημένο στο RealMainP.nc :

```
int main() __attribute__ ((C, spontaneous)) {

...

__nesc_enable_interrupt();
signal Boot.booted();

/* Spin in the Scheduler */
call Scheduler.taskLoop();

return -1;
}
```

Σχ. 12

Μελετώντας το παραπάνω κομμάτι κώδικα και, με βάση τις λεπτομέρειες υλοποίησης του boot sequence από το TEP 107[19] (**T**inyOS **E**xtension **P**roposals - το σύνολο κειμένων περιγραφής και σχεδιασμού του TinyOS) , παρατηρούμε ότι η παρούσα υλοποίηση είναι ακατάλληλη για τις ανάγκες μας. Η εντολή ***call Scheduler.taskLoop();*** δεν επιστρέφει ποτέ. Σε μία συμβατική TinyOS πλατφόρμα αυτό δεν είναι πρόβλημα, καθώς η ασύγχρονη κλήση των preemptive interrupt handler επιτρέπει στο εκάστοτε application να συνεχίσει την εκτέλεσή του. Για την δική μας πλατφόρμα, όμως, πρέπει να προχωρήσουμε σε μερικές αλλαγές:

1. Η εντολή call Scheduler.taskLoop() αφαιρείται από το boot sequence. Η εκτέλεση των task που βρίσκονται στην ουρά θα γίνεται «χειροκίνητα» μετά από κάθε event, όπως περιγράφεται στο διάγραμμα ροής της ενότητας 4.2.
2. Το component ***RealMainP*** θα παρέχει ειδικά JNI Call ως διεπαφή για την εκπλήρωση των απαιτήσεων που συζητήθηκαν σε αυτή την ενότητα.

```
jint Java_org_lekkas_poclient_PoAPI_MiddlewareManager_nativeMain(JNIEnv *env,
jobject obj) __attribute__ ((C, spontaneous)) {
        return main();
}

int Java_org_lekkas_poclient_PoAPI_MiddlewareManager_nativeCompleteTasks(JNIEnv *env,
jobject obj) __attribute__ ((C, spontaneous)) {
        while (call Scheduler.runNextTask());
        return JNI_OK;
}
```

JNI Calls για την αρχικοποίηση του middleware και την ολοκλήρωση των Task - Σχ. 13

Κεφάλαιο 5

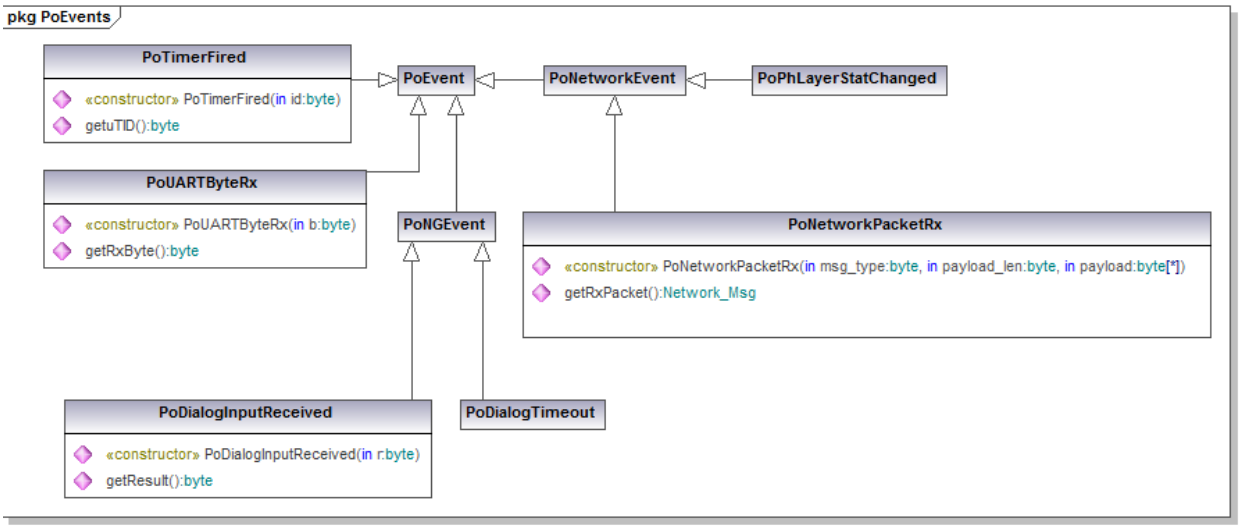
Περιβάλλον Android/Java

5.1 Εισαγωγικές πληροφορίες

Στην προηγούμενη ενότητα εξετάσαμε τις αλλαγές που έγιναν στην υποκείμενη βιβλιοθήκη – middleware. Εδώ θα δούμε πώς αυτές οι αλλαγές συνδέονται με το περιβάλλον Android. Οι αρχές της σχεδίασης θα παρουσιαστούν με την μελέτη των πακέτων, των κλάσεων και των διεπαφών που χρησιμοποιήθηκαν. Παράλληλα, θα δίνουμε επεξηγήσεις πάνω σε θέματα υλοποίησης των σχεδιαστικών αυτών αρχών. Για τις μεθόδους των κλάσεων ακολουθούμε την εξής σύμβαση: Μέθοδοι των οποίων το όνομα ξεκινάει με ***JNICall_*** κάνουν κλήσεις από το Android προς το Middleware (calls), ενώ μέθοδοι με ονομασία τύπου ***JNICallback_*** καλούνται από το Middleware (callbacks) προς το Android. Οι διαφορές και η σημασιολογία των κλήσεων αυτών έχει συζητηθεί με λεπτομέρεια σε αρκετές προηγούμενες ενότητες.

5.2 Πακέτο κλάσεων PoEvents

Στο πακέτο *PoEvents* συγκεντρώνεται το σύνολο των κλάσεων που περιγράφουν τα πιθανά event που μπορεί να εισαχθούν στην Event Queue (βλ. ενότητα 4.3) . Όπως φαίνεται στο ακόλουθο σχήμα, κάποια event classes παρέχουν μεθόδους πρόσβασης στα δεδομένα τους (**accessor methods).** Για παράδειγμα, το event *PoTimerFired* παρέχει την μέθοδο *getuTID()* ώστε να επιστρέφεται το αναγνωριστικό του Timer που έληξε.



Διάγραμμα UML κλάσεων πακέτου PoEvents - Σχ.14

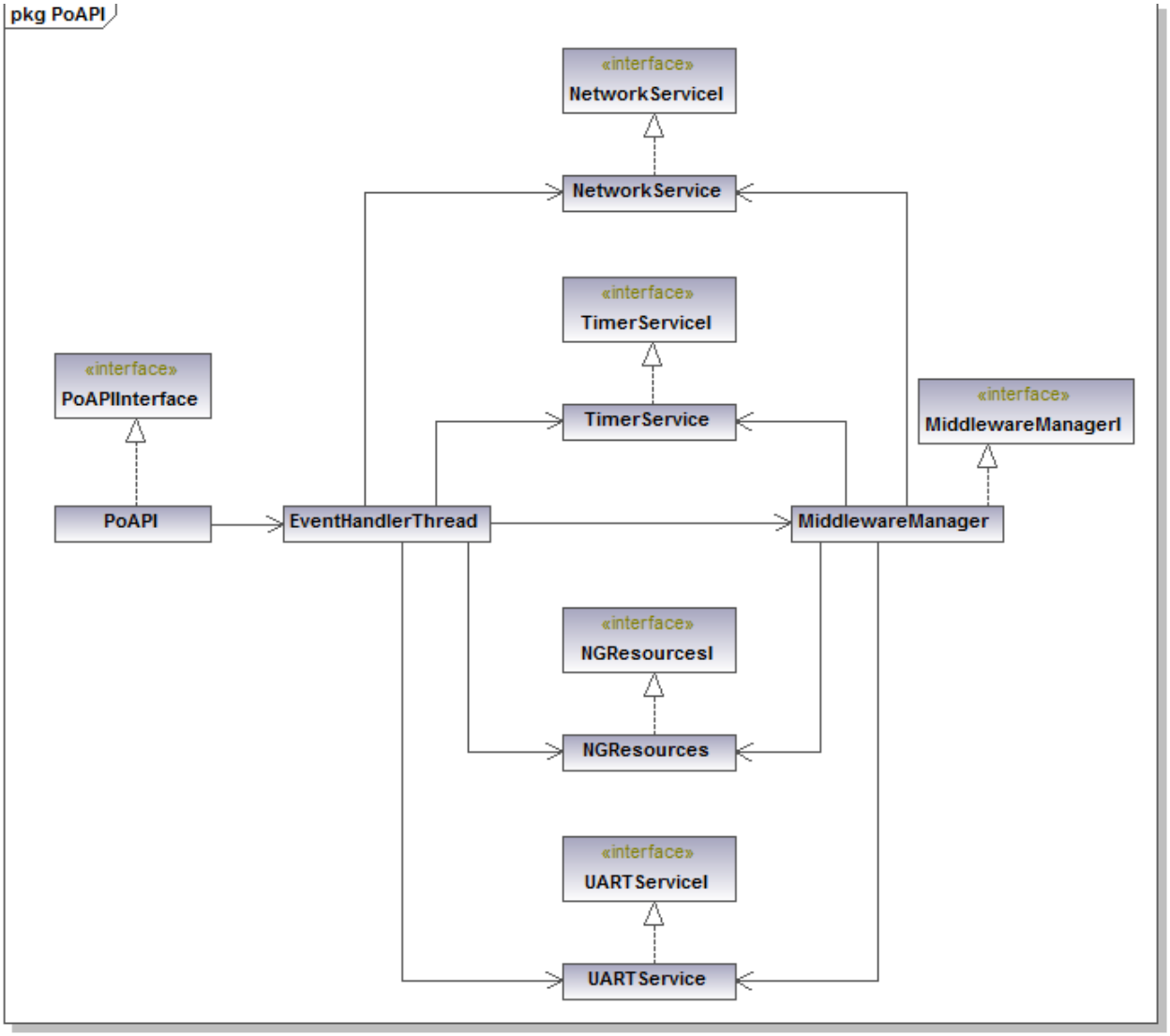
Σύντομη περιγραφή για το πακέτο **org.lekkas.poclient.PoEvents.**

PoEvent : Η υπερκλάση όλων των event.

- **PoTimerFired :** Υποδηλώνει την λήξη ενός Timer.
- **PoUARTByteRx :** Σημάνει την λήψη δεδομένων μέσω του εικονικού UART.
- **PoNetworkEvent**
    - **PoNetworkPacketRx :** Σημάνει την λήψη ενός πακέτου δικτύου.
    - **PoPhLayerStatChanged :** Χρησιμοποιείται μονάχα για debugging σκοπούς, επισημάνει ότι το κινητό αποσυνδέθηκε/επανασυνδέθηκε στο δίκτυο (GPRS ή WiFi)
- **PoNGEvent**
    - **PoDialogInputReceived :** Δημιουργείται όταν ο χρήστης αποκριθεί σε ένα πλαίσιο διαλόγου.
    - **PoDialogTimeout :** Ενημερώνει το middleware application ότι το πλαίσιο διαλόγου «έληξε» πρίν πάρει απάντηση από τον χρήστη.

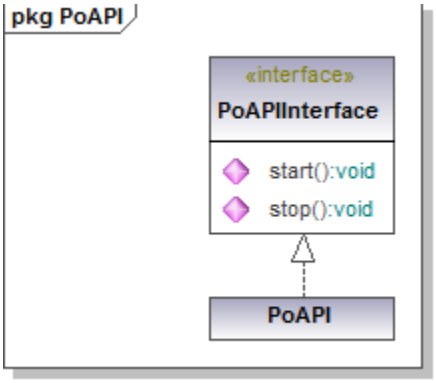
5.3 Πακέτο κλάσεων για το PoAPI Layer

Η βασικές αρχές λειτουργίας του PoAPI Layer περιγράφηκαν στην ενότητα 3.2.3. Εδώ θα δούμε με περισσότερες λεπτομέρειες τον τρόπο που αυτό το επίπεδο σχεδιάστηκε και υλοποιήθηκε. Το διάγραμμα κλάσεων που ακολουθεί δείχνει τις βασικές αρχές σχεδιασμού για το πακέτο **org.lekkas.poclient.PoAPI :**



Διάγραμμα UML κλάσεων πακέτου PoAPI - Σχ. 15

5.3.1 PoAPI



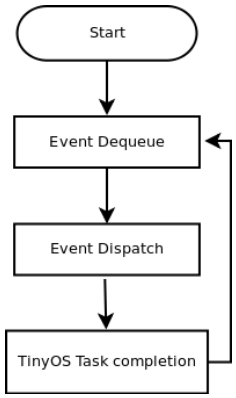
Σχ. 16

Η κλάση αυτή είναι υπεύθυνη για τις διαδικασίες ενεργοποίησης/απενεργοποίησης ολόκληρου του PoAPI Layer.Με την χρήση της μεθόδου *start()* ξεκινά η εκτέλεση του EventHandler Thread, ενώ με την χρήση της μεδόδου *stop()* το EventHandler thread σταματάει την εκτέλεση του και η Event Queue αδειάζει.

5.3.2 EventHandler Thread

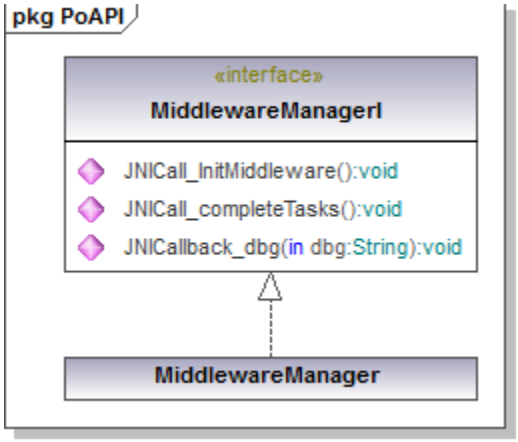
Το EventHandler Thread δημιουργεί, κατά την εκκίνησή του, ένα *instance* της κλάσης *MiddlewareManager.* Στην συνέχεια, χρησιμοποιώντας μεθόδους που αυτή η κλάση παρέχει, αρχικοποιεί το middleware.Τέλος, το νήμα μπαίνει σε έναν ατέρμονο βρόχο στον οποίο:

1. Αφαιρεί γεγονότα από την Event Queue.
2. Τα εξυπηρετεί χρησιμοποιώντας την *DispatchEvt().*
3. Εκτελεί τα tasks του middleware, χρησιμοποιώντας την μέθοδο *JNICall_completeTasks()* της κλάσης MiddlewareManager.
4. Επιστρέφει στο βήμα 1.



Event Handler Flow diagram – Σχ 17

5.3.2 MiddlewareManager

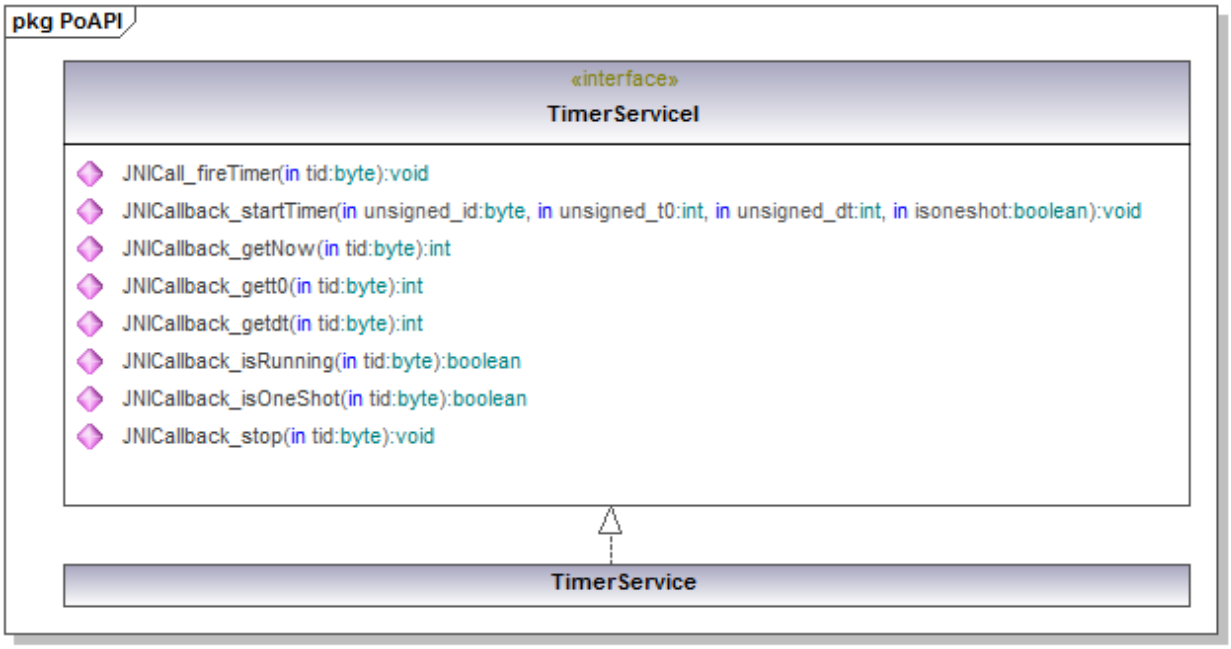


Σχ. 18

Για την κλάση αυτή:

- Η μέθοδος *JNICall_InitMiddleware()* ολοκληρώνει τις απαραίτητες αρχικοποιήσεις κυρίως όσον αφορά το περιβάλλον JNI, κάνοντας, για παράδειγμα, caching στις αναφορές των αντικειμένων/μεθόδων του Dalvik VM. Η διαδικασία αυτή εξοικονομεί χρόνο κατά την εκτέλεση μελλοντικών JNI Callback. Στην συνέχεια εκτελεί την *main()* όπως περιγράφεται στην ενότητα 3.2.4.4 .
- Η μέθοδος *JNICall_completeTasks()* εκτελεί όσα tasks βρίσκονται στην ουρά του TinyOS Task Scheduler (βλ. ενότητα 3.2.4.4) .
- Η μέθοδος *JNICallback_dbg()* χρησιμοποιείται για debugging. Όλες οι κλήσεις συναρτήσεων για εμφάνιση debug μηνυμάτων στο POBICOS καταλήγουν σε αυτό το callback. Για την ώρα, τα μηνύματα αυτά εμφανίζονται στο debugging console (**Logcat**) του περιβάλλοντος ανάπτυξης Eclipse για Android.

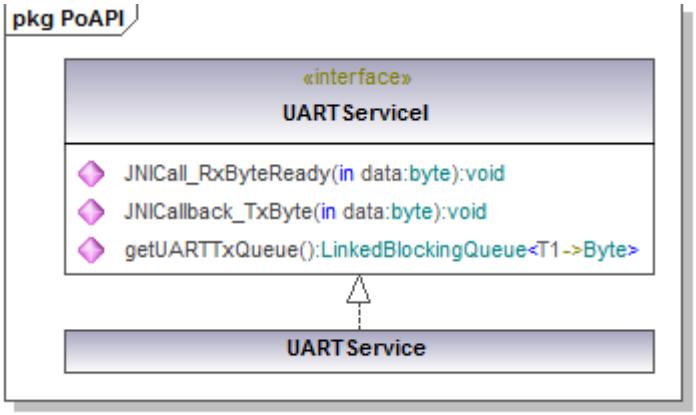
5.3.4 TimerService



Σχ. 19

Η κλάση αυτή υποστηρίζει πλήρως το Timer Interface του TinyOS. Έτσι, το POBICOS middleware μπορεί να λειτουργεί σε πραγματικό χρόνο. Για την υλοποίηση των Timer, έχει χρησιμοποιηθεί η κλάση *ScheduledThreadPoolExecutor* του *Concurrent* Framework της Java, η οποία δίνει την δυνατότητα για υποβολή (και ακύρωση) χρονοπρογραμματισμένων task. Το pool αποτελείται από μόνο ένα thread και κάθε Timer Id αντιστοιχεί σε ένα μοναδικό χρονοπρογραμματισμένο task. Αυτά τα task, μόλις εκτελεστούν, απλώς εισάγουν ένα *PoTimerFired* event στην Event Queue.

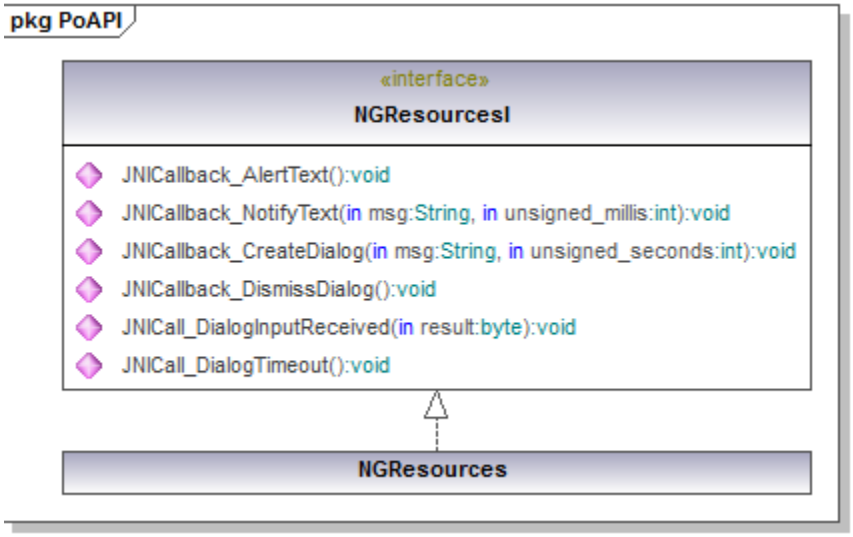
5.3.5 UARTService



Σχ. 20

Η υπηρεσία της εικονικής UART θύρας ουσιαστικά χρησιμοποιείται για έναν και μόνο σκοπό: την φόρτωση στην μνήμη και την μετέπειτα την εκτέλεση του POBICOS application. Για την ώρα ο μόνος χρήστης της εικονικής UART είναι το *Application Pill Loader*, το οποίο θα μελετηθεί σε επόμενη ενότητα. Η ουρά *UARTTxQueue* χρησιμοποιείται ως buffer για τα εξερχόμενα από το middleware δεδομένα. Από την άλλη, τα εισερχόμενα δεδομένα που έχουν προορισμό το middleware λαμβάνονται από την Event Queue.

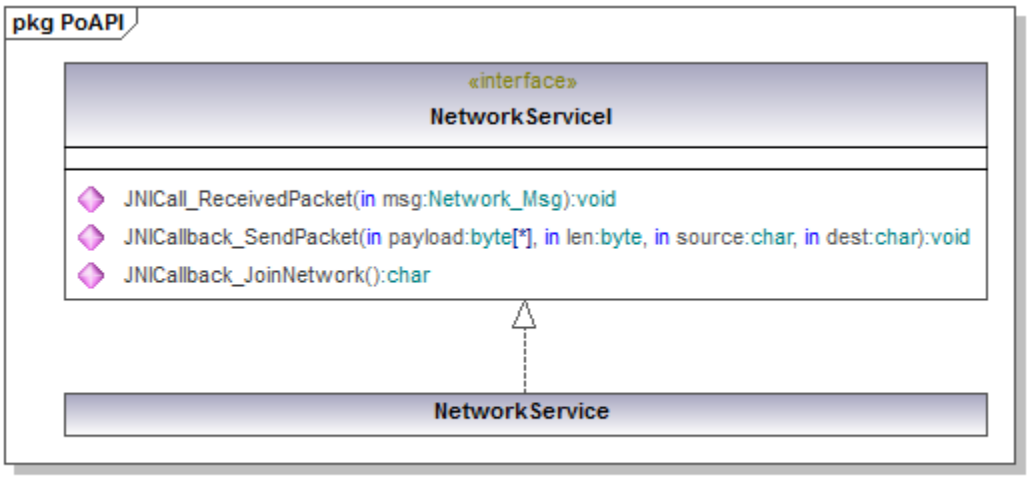
5.3.6 NGResources



Σχ. 21

Η κλάση αυτή υποστηρίζει ένα μέρος των non-generic primitive του POBICOS για διαδραστική επικοινωνία με τον χρήστη. Για την υλοποίηση των λειτουργιών αυτών χρησιμοποιήθηκαν στοιχεία UI του Android Application Framework, όπως *Activities, Text Views, Toast Notifications*, *Alert Dialogs και άλλα.*

5.3.7 NetworkService

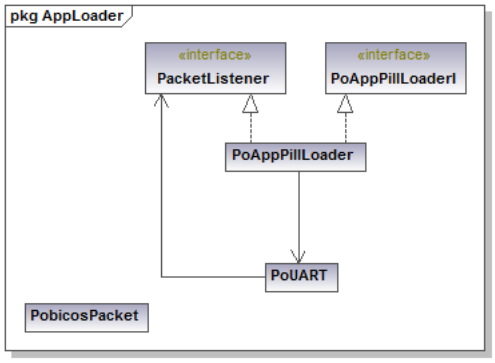


Σχ. 22

Η συγκεκριμένη κλάση αποτελεί την βάση της δικτυακής επικοινωνίας. Αναλαμβάνει την μεταφορά δικτυακών πακέτων από και προς το middleware. Η δικτυακή λειτουργία του συστήματος περιγράφεται αναλυτικά σε ξεχωριστή ενότητα παρακάτω.

5.4 Πακέτο για το AppLoader

Η φόρτωση ενός POBICOS application σε έναν κόμβο είναι μια διαδικασία μη τετριμμένη που απαιτεί συγκεκριμένα πρωτόκολλα επικοινωνίας. Το ίδιο ισχύει και για την έναρξη/παύση της εφαρμογής. Βασισμένοι στο αντίστοιχο εργαλείο του POBICOS toolchain (υλοποιημένο από το Warsaw University of Technology) υλοποιήσαμε μια έκδοση προσαρμοσμένη στις απαιτήσεις μας. Όπως αναφέρθηκε στην ενότητα 5.3.5, το AppLoader χρησιμοποιεί την εικονική υπηρεσία UART .

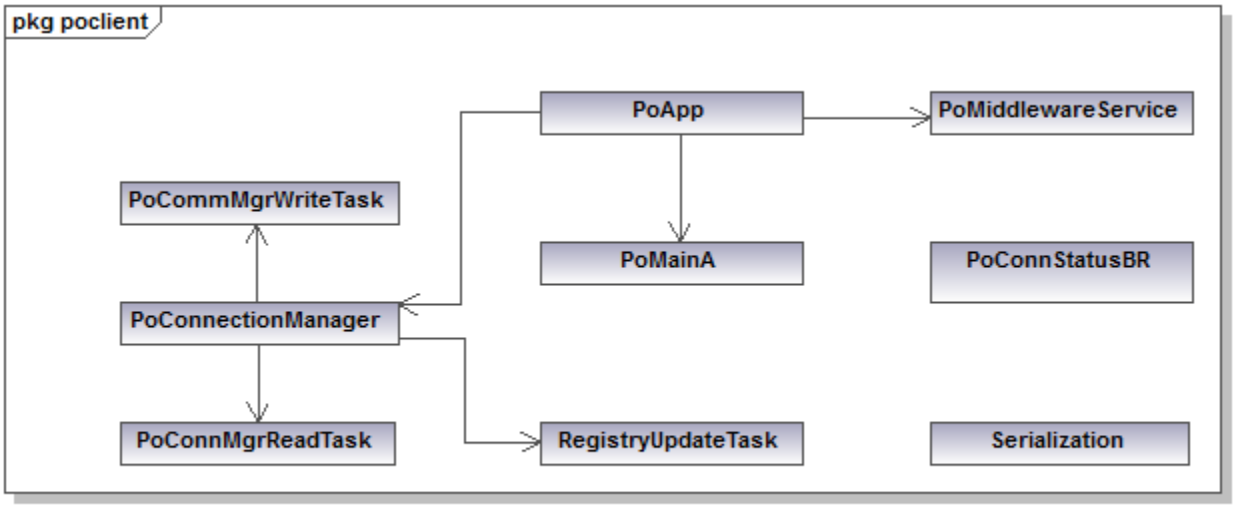


Σχ. 23

Για την συγκεκριμένη εργασία οι λεπτομέρειες του σχεδιασμού και της υλοποίησης του AppLoader δεν χρειάζεται να μελετηθούν σε βάθος . Οι ενδιαφερόμενοι μπορούν να αναζητήσουν περισσότερες πληροφορίες από τις παραπομπές και τον κώδικα της εργασίας[20].

5.5 Πακέτο Εφαρμογής Android

Στις ενότητες 5.1-5.4 περιγράφηκε με λεπτομέρεια η διεπαφή του Android με την βιβλιοθήκη του ενδιάμεσου λογισμικού. Το τελευταίο πακέτο περιέχει κλάσεις που λειτουργούν σε υψηλότερο επίπεδο και χρησιμοποιούν αποκλειστικά το Android Application Framework.



Σχ. 24

Επιγραμματικά:

- **PoMainA**: Το βασικό Android Activity.
- **PoMiddlewareService** : Το Android Service που είναι υπεύθυνο για την εκκίνηση/τερματισμό του middleware.
- **PoConnStatusBR** : Το Android Broadcast Receiver που ενημερώνει για τις συνδέσεις/αποσυνδέσεις .
- **PoConnectionManager**: Το βασικό component που ασχολείται με θέματα συνδεσιμότητας. Δημιουργεί τα Read/Write thread.
- **PoCommMgrWriteTask**: Νήμα με μοναδικό σκοπό την αποστολή πακέτων στο Forwarder/Registry.
- **PoCommMgrReadTask**: Νήμα επιφορτισμένο με την λήψη πακέτων. Τα ληφθέντα πακέτα θα τοποθετηθούν στην Event Queue, όπως περιγράφεται στην ενότητα 3.2.3 .
- **Serialization:** Συλλογή συναρτήσεων για λειτουργίες σειριοποίησης δεδομένων.

Κεφάλαιο 6

Αρχιτεκτονική Δικτύωσης

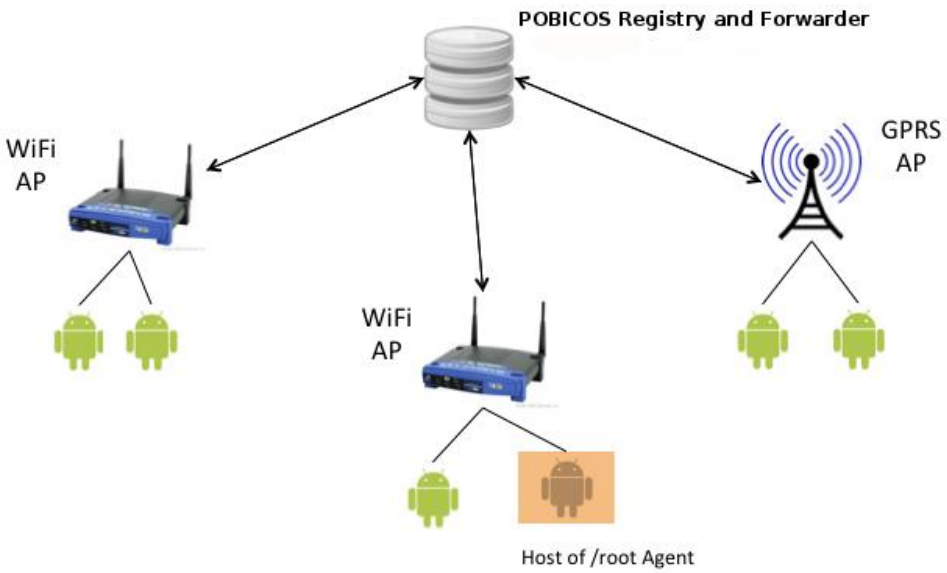
6.1 Εισαγωγικές πληροφορίες

Το ενδιάμεσο λογισμικό POBICOS σχεδιάστηκε ώστε να λειτουργεί σε ασύρματους, κατανεμημένους κόμβους που υποστηρίζουν την δημιουργία ad-hoc τοπικών δικτύων (IEEE 802.11/ZigBee). Με την μεταφορά του POBICOS σε κινητές συσκευές Android, οι κόμβοι μπορούν πλέον να βρίσκονται σε αποστάσεις που ξεπερνούν κατά πολύ την εμβέλεια του IEEE 802.11 και παρόλα αυτά να εκτελούν την ίδια , *location-aware* εφαρμογή. Επομένως, για την προσέγγισή χρησιμοποιούμε την στατική υποδομή που είναι διαθέσιμη στα κινητά τηλέφωνα (WiFi και GPRS Access Points) για να δημιουργήσουμε ένα δίκτυο κόμβων POBICOS. Τα μηνύματα πλέον δεν αποστέλλονται απευθείας μεταξύ των κόμβων αλλά προωθούνται σε ένα γνωστό "***POBICOS Registry and Forwarder"*** το οποίο υποστηρίζει:

1. Την δρομολόγηση μηνυμάτων μεταξύ των κόμβων.
2. Την εγγραφή (***registration)*** συσκευών Android.

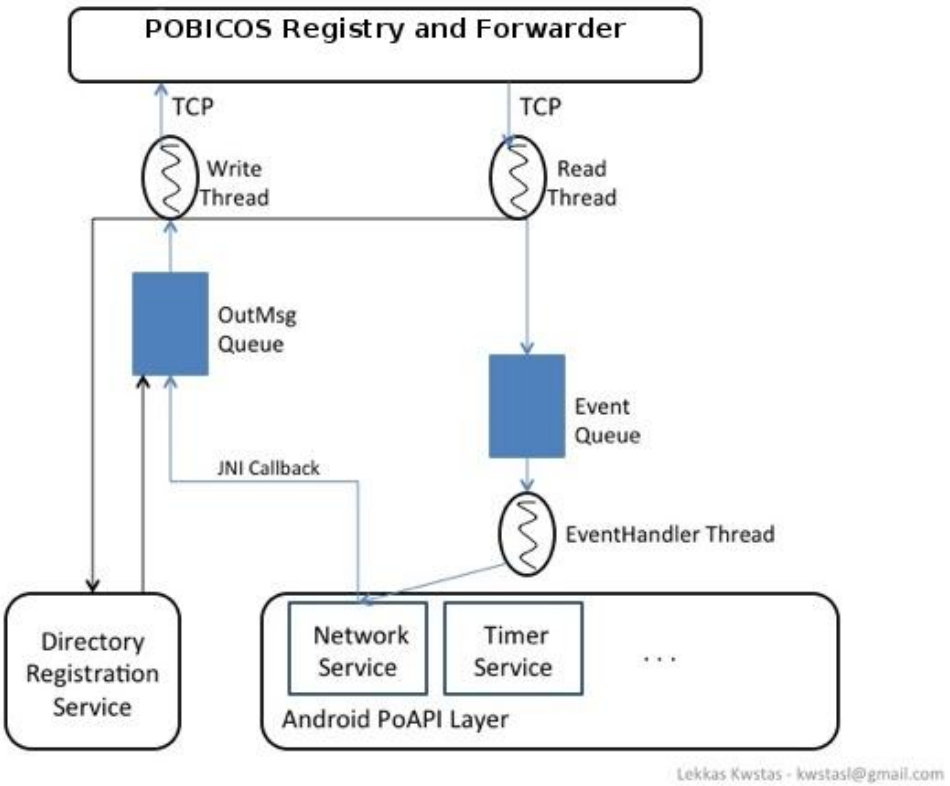
Από άποψη σχεδιασμού είναι πιο σωστό να θεωρήσουμε ότι το Registry και το Forwarder είναι δύο διαφορετικές οντότητες. Ιδανικά, το Forwarder πρέπει να αποτελείται από ένα σύνολο κόμβων ώστε να παραμένει λειτουργικό ανεξάρτητα από τον αριθμό των συσκευών που επικοινωνούν ταυτόχρονα (απαιτήσεις κλιμάκωσης - scaling). Το ίδιο ισχύει βέβαια και για το Registry, το οποίο θα μπορούσε να αποτελείται από έναν αριθμό μηχανημάτων, οργανωμένων σε ιεραρχική δομή. Για τις ανάγκες του project θα αντιμετωπίζουμε τις δύο αυτές οντότητες ως ένα μοναδικό, ενοποιημένο σύστημα.

Τέλος, οι κινητές συσκευές επικοινωνούν με το Forwarder δημιουργώντας μια σταθερή σύνδεση TCP με αυτό. Η χρήση του UDP, αν και σχεδιαστικά πιο σωστή, παρουσίαζε ένα σημαντικό τεχνικό πρόβλημα: Τα περισσότερα σταθερά σημεία πρόσβασης που θα χρησιμοποιηθούν για πρόσβαση στο Internet (WiFi / GPRS) λειτουργούν με NAT (Network Address Translation) και επιπλέον βρίσκονται πίσω από Firewall. Παρόλο που υπάρχουν τεχνικές για αντιμετώπιση αυτών των περιορισμών (πχ η τεχνική UDP Hole Punching που χρησιμοποιείται και από το Skype[21]) αποφασίστηκε, για λόγους απλότητας, να δουλέψουμε με το TCP. Η επιλογή αυτή δεν είναι δεσμευτική και σε επόμενες εκδόσεις της πλατφόρμας μπορεί να υποστηριχθεί το UDP με κάποιες μικρές αλλαγές.



Σχ. 25

6.2 Η δομή της εφαρμογής



Σχ. 26

6.3 Τα μηνύματα του POBICOS Registry & Forwarder

Υπάρχουν δύο τύποι μηνυμάτων που μεταδίδονται από/προς το Registry & Forwarder:

1. Τα ***POBICOS messages:*** ενθυλακωμένα μηνύματα POBICOS που αποστέλλονται μεταξύ κόμβων.
2. Τα ***Registry request/reply messages:*** Μηνύματα σχετικά με το μητρώο εγγραφών που χειρίζεται εγγραφές/διαγραφές κόμβων. Τα μηνύματα αυτά μεταφέρουν επίσης πληροφορίες σχετικά με την τρέχουσα θέση των κόμβων.

6.4 Μορφή EBNF

Όλα τα μηνύματα έχουν την γενική μορφή:

```
NetworkMsg = MsgType PayLen Payload
MsgType    = uint8_t
Paylen     = uint8_t
Payload    = {byte}*Paylen
```

6.5 Μηνύματα POBICOS Forwarder

```
PobicosMsg      = MsgType PayLen Payload
MsgType         = POBICOS_MSG
POBICOS_MSG     = 0x01
PayLen          = uint8_t
Payload         = SrcNodeAddr DstNodeAddr PoMsgLen PoMsg PoLocationInfo
DstNodeAddr     = uint16_t
SrcNodeAddr     = uint16_t
PoMsgLen        = uint8_t
PoMsg           = {byte}*PoMsgLen
PoLocationInfo = RadiusKM GPSLat GPSLong
RadiusKM        = uint16_t
GPSLat          = uint64_t
GPSLong         = uint64_t
```

6.6 Μηνύματα Registry Request

```
DirectoryReq     = MsgType PayLen Payload
MsgType          = REGISTRY_REQ
REGISTRY_REQ     = 0x02
PayLen           = uint8_t
Payload          = ClassOfDevice GPSLat GPSLong Addr Seed
ClassOfDevice    = MOBILE | LAPTOP | PC
MOBILE           = 0x01
LAPTOP           = 0x02
PC               = 0x03
GPSLat           = uint64_t
GPSLong          = uint64_t
Addr             = uint16_t
Seed             = uint32_t
```

6.7 Μηνύματα Registry Reply

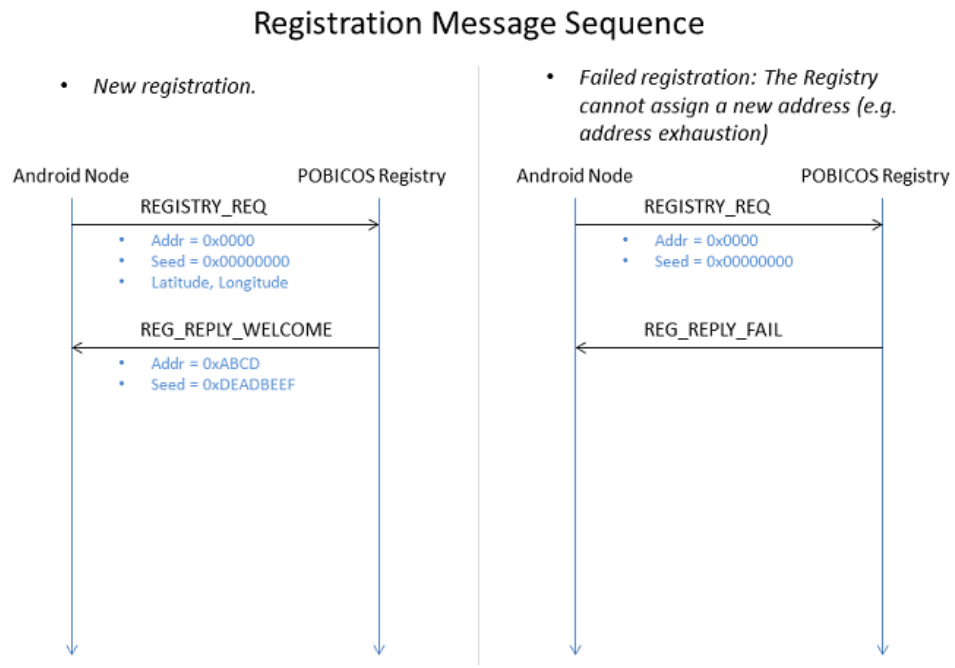
```
DirectoryReply      = MsgType PayLen Payload
MsgType             = REG_REPLY_FAIL | REG_REPLY_WELCOME | REG_REPLY_PONG
REG_REPLY_FAIL      = 0x03
REG_REPLY_WELCOME   = 0x04
REG_REPLY_PONG      = 0x05
PayLen              = uint8_t
Payload             = Addr Seed
Addr                = uint16_t
Seed                = uint32_t
```

Σχετικά με την διαδικασία εγγραφής:

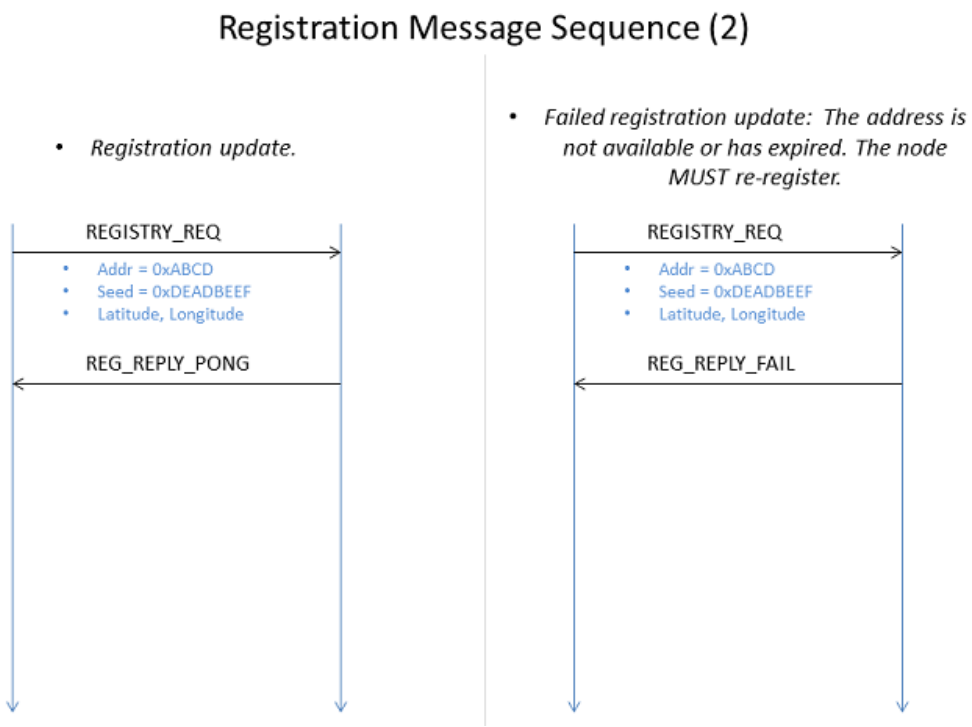
1. Οι εγγραφές ανανεώνονται με την αποστολή περιοδικών μηνυμάτων τύπου “Registry Request”. Οι κόμβοι μπορεί μεν να αποσυνδέονται από το Registry & Forwarder αλλά έχουν περιορισμένο χρόνο για να επανασυνδεθούν και να ανανεώσουν/επανακτήσουν την παλιά του διεύθυνση.
2. Οι κόμβοι που αποτυγχάνουν να ανανεώσουν την εγγραφή τους αυτόματα διαγράφονται από το μητρώο. Οι διαγραμμένοι κόμβοι πρέπει να επαναλάβουν την διαδικασία “Address Registration” και να λάβουν μια νέα διεύθυνση.
3. Μηνύματα τύπου POBICOS Forwarder προς διευθύνσεις που δεν ισχύουν, πετιούνται.
4. Μια διεύθυνση κόμβου είναι επαναχρησιμοποιήσιμη μόνο αν λήξει (δηλαδή ο προηγούμενος κάτοχός της αποτύχει να την ανανεώσει μέσα στο επιτρεπόμενο χρονικό όριο) .
5. Ο αριθμός “seed” λαμβάνεται από το Registry κατά την διάρκεια της διαδικασίας εγγραφής. Από εκείνη την στιγμή και μετά, ο αριθμός αυτός θα πρέπει να αποστέλλεται μαζί με την διεύθυνση κόμβου για όλα τα μηνύματα τύπου Registry Request.

6.8 Διαγράμματα ροής μηνυμάτων

Τα παρακάτω διαγράμματα περιγράφουν το πρωτόκολλο επικοινωνίας μεταξύ του Registry και των κινητών συσκευών.



Σχ. 27



Σχ. 28

6.9 Registry & Forwarder

6.9.1 Εισαγωγικές πληροφορίες

Όπως αναφέρθηκε και στην ενότητα 6.1 , το Registry και το Forwarder μπορούν, σχεδιαστικά, να είναι δύο διαφορετικές (και μάλιστα κατανεμημένες) υπηρεσίες. Για πραγματικές εφαρμογές με χιλιάδες κινητά ταυτόχρονα συνδεδεμένα αυτό είναι απαραίτητη προϋπόθεση, καθώς η χρήση ενός μόνο κόμβου και για τις δύο αυτές λειτουργίες δεν κλιμακώνει. Στην περίπτωσή μας, όπου στόχος δεν είναι να επιτευχθεί καλή κλιμάκωση για πολύ μεγάλους αριθμούς κινητών αλλά να υλοποιηθεί κατ' αρχάς η επιθυμητή λειτουργικότητα, μπορούμε να συμβιβαστούμε με ένα ενιαίο Registry & Forwarder (**RF**).

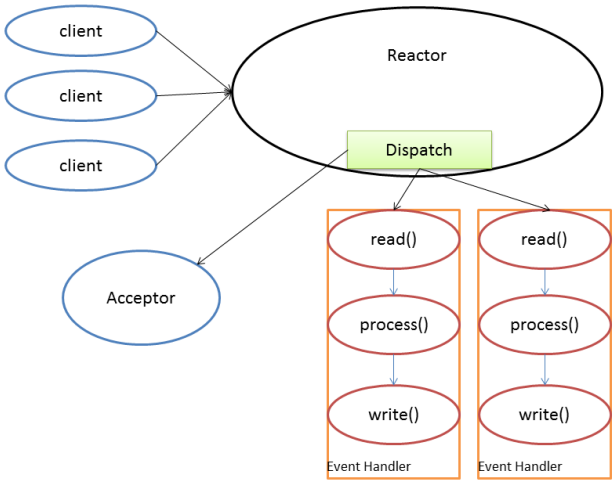
6.9.2 Αρχές σχεδιασμού

Η σχεδίαση/υλοποίηση του RF έγινε με τρόπο όσο το δυνατόν πιο αποδοτικό. Συγκεκριμένα, το RF λειτουργεί με το λεγόμενο "Reactor Pattern" [22]. Ο σχεδιασμός αυτός ακολουθεί τις αρχές του Event-Driven προγραμματισμού και λειτουργεί με ένα μόνο thread, στα πρότυπα high-scalability συστημάτων (όπως για παράδειγμα το nginx[23]).

6.9.3 Υλοποίηση

Το RF υλοποιήθηκε σε Java, με την χρήση των *java.nio non-blocking APIs.*Κύριες λειτουργικές μονάδες – κλάσεις του είναι:

- Ο **Reactor:** Εξυπηρετεί γεγονότα IO δημιουργώντας έναν RequestHandler.
- Ο **Request Handler**: Εκτελεί non-blocking λειτουργίες στα socket καλώντας τις *read() , process(), write() .*
- Ο **Accept Handler**: Δέχεται νέες συνδέσεις.
- Το **Registry:** Το μητρώο τον εγγραφών με τις αντίστοιχες μεθόδους αναζήτησης/εγγραφής/διαγραφής



Σχ. 29

Όπως επίσης αναφέρθηκε στην ενότητα 3.3, ο RF προωθεί πακέτα POBICOS μόνο σε κόμβους που ικανοποιούν τις απαιτήσεις τοποθεσίας του εκάστοτε application.

Κεφάλαιο 7

Επίγνωση Θέσης

Οι μικροεφαρμογές έχουν την δυνατότητα να περιορίζουν την εκτέλεσή τους σε συγκεκριμένες συσκευές που βρίσκονται εντός μιας προκαθορισμένης περιοχής. Ήδη από το προηγούμενο κεφάλαιο έγινε αναφορά στα πρωτόκολλα που αναλαμβάνουν να ενημερώσουν το Registry για τις θέσεις των κινητών.

7.1 Υποστήριξη γεωγραφικών απαιτήσεων

Όπως φαίνεται από την EBNF μορφή των μηνυμάτων της ενότητας 6.5, το πεδίο PoLocationInfo χρησιμοποιείται για τις ανάγκες των *location-aware* εφαρμογών. Οι απαιτήσεις θέσης εξάγονται από την προς εκτέλεση εφαρμογή και ενσωματώνονται σε κάθε μήνυμα που στέλνεται από το Root agent προς το Forwarder. Στην συνέχεια, το Forwarder, με τρόπο διάφανο προς την εφαρμογή, προωθεί τα μηνύματα **αποκλειστικά** στους διαθέσιμους κόμβους που ικανοποιούν τις συγκεκριμένες απαιτήσεις. Αν δεν υπάρχουν κόμβοι που πληρούν τις προϋποθέσεις το μήνυμα απλώς γίνεται drop (το POBICOS middleware είναι σχεδιασμένο να ανταπεξέρχεται σε τέτοια πιθανά χαμένα μηνύματα). Οι απαιτήσεις θέσης περιγράφονται από

1) Την ακτίνα του νοητού, 'γεωγραφικού' κύκλου, μέσα στον οποίο επιτρέπεται η εκτέλεση της εφαρμογής, σε μονάδες km.
2) Το κέντρο της ζητούμενης περιοχής, με συντεταγμένες GPS.

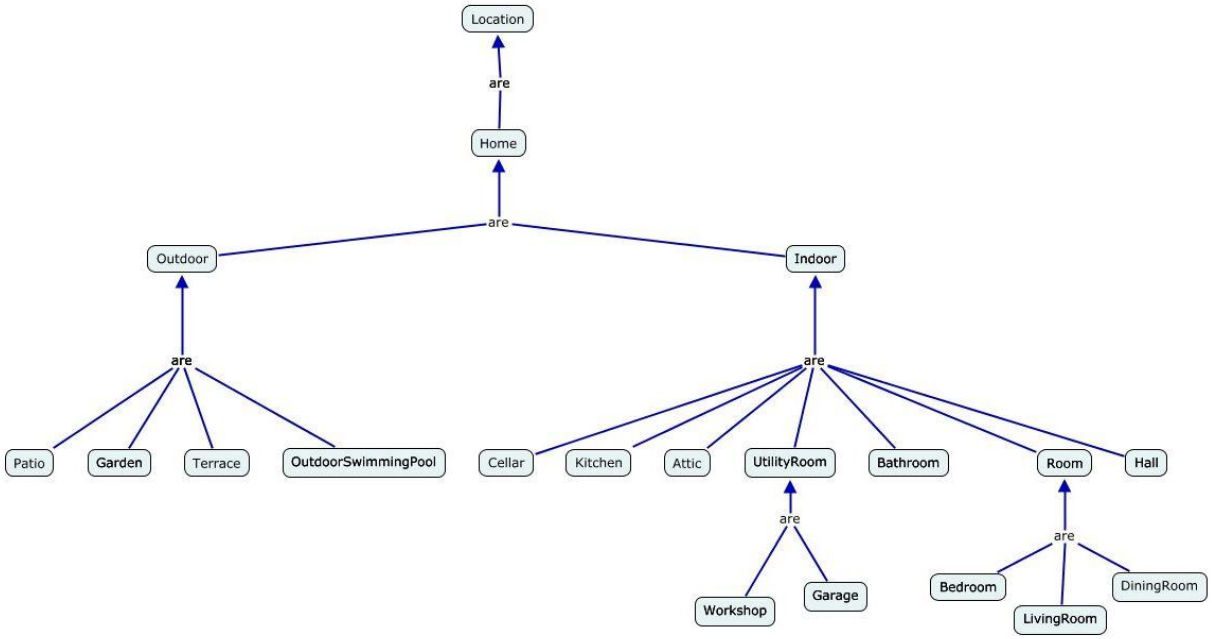
Οι υπόλοιπες συσκευές, με την χρήση των Registry update μηνυμάτων (ενότητα 6.6) ενημερώνουν περιοδικά το Registry για την τρέχουσα θέση τους. Με αυτό τον τρόπο, για τα μηνύματα που αποστέλλονται από την συσκευή του Root agent, ισχύουν τα εξής:

- Αν το μήνυμα είναι τύπου broadcast, προωθείται σε όλους τους κόμβους που βρίσκονται εντός της ζητούμενης περιοχής.
- Αν το μήνυμα είναι unicast **και** ο κόμβος προορισμού είναι α) ακόμα εγγεγραμμένος και β) ικανοποιεί τις απαιτήσεις θέσης, τότε το μήνυμα παραδίδεται κανονικά.
- Σε κάθε άλλη περίπτωση, το μήνυμα απορρίπτεται. Το POBICOS αναλαμβάνει τυχόν επαναληπτικές αποστολές και χειρίζεται την περίπτωση όπως θα έκανε για κάθε χαμένο μήνυμα.

Το Registry & Forward βρίσκει τις συσκευές που ικανοποιούν τις απαιτήσεις τοποθεσίας χρησιμοποιώντας τον ***τύπο Haversine***[24]**,** ο οποίος υπολογίζει την απόσταση δύο σημείων πάνω σε μία σφαίρα. Αν λοιπόν, το κινητό βρίσκεται εντός του κύκλου που ορίζεται από τις απαιτήσεις της εφαρμογής (GPS συντεταγμένες κέντρου και ακτίνα) τότε θεωρούμε ότι **ικανοποιεί** τις απαιτήσεις θέσης.

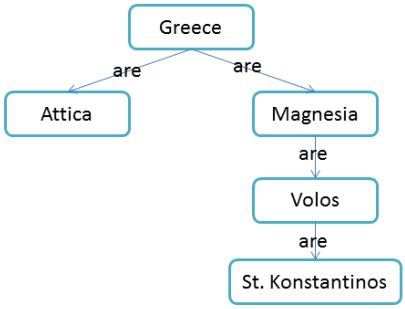
7.2 Εναλλακτικές προσεγγίσεις

Το POBICOS παρέχει έναν δικό του μηχανισμό για τον καθορισμό/εξακρίβωση απαιτήσεων μιας εφαρμογής. Κάθε κόμβος περιγράφεται από μια δομή, που συγκεντρώνει τα χαρακτηριστικά του αντικειμένου ως ένα σύνολο εννοιών που προέρχονται από διάφορες ταξονομίες. Παρακάτω φαίνεται μια ταξονομία σχετική με την θέση των αντικειμένων σε ένα σπίτι.



Location taxonomy - Σχ 30

Κάθε εφαρμογή καθορίζει, μέσω ενός '*διάστηματος εννοιών*'(concept-interval), τις απαιτήσεις εκτέλεσης. Για το παράδειγμα της προηγούμενης ταξονομίας, μια εφαρμογή μπορεί να θέσει την απαίτηση **[LivingRoom, Indoor].** Έτσι, αν ένας κόμβος περιγράφεται από μια έννοια που είναι υποκλάση της γενικότερης έννοιας του concept-interval (στην περίπτωσή μας το Indoor), τότε θεωρούμε ότι ικανοποιεί τις απαιτήσεις της εφαρμογής. Επομένως, θεωρητικά, θα μπορούσαμε να δημιουργήσουμε μια ταξονομία θέσης που δεν θα περιγράφει μόνο ένα σπίτι, αλλά θα καλύπτει μια μεγαλύτερη γεωγραφική περιοχή.



Παράδειγμα ταξονομίας θέσης για γεωγραφικές περιοχές – Σχ 31

Η προσέγγιση αυτή έχει ιδιαίτερο ενδιαφέρον, καθώς εκμεταλλεύεται την ήδη υπάρχουσα υποδομή του POBICOS και είναι πλήρως συμβατή με το μοντέλο εκτέλεσης εφαρμογών του. Παρουσιάζει, όμως, κάποια σοβαρά τεχνικά προβλήματα και ελλείψεις: Η ταξονομία θέσης, όσο λεπτομερής και αν είναι, δεν μπορεί να φτάσει την ακρίβεια των δεκτών GPS, που έχουν σφάλμα λίγων μόνο μέτρων. Επίσης, οι δομές που χαρακτηρίζουν τους κόμβους ορίζονται κατά την μεταγλώττιση και, στην τωρινή υλοποίηση του POBICOS, παραμένουν ως έχουν, πράγμα ακατάλληλο για κινητές συσκευές. Για παράδειγμα, κόμβος που χαρακτηρίστηκε με την έννοια “St. Konstantinos” δεν μπορεί να χαρακτηριστεί εκ νέου με βάση την νέα του θέση. Για τους δύο παραπάνω λόγους, η ιδέα της χρήσης των ταξονομιών εγκατελήφθηκε.

Κεφάλαιο 8

Δοκιμαστική Εφαρμογή

Σε αυτό το κεφάλαιο περιγράφεται μια δοκιμαστική εφαρμογή που χρησιμοποιεί την υποδομή που μελετήθηκε στα κεφάλαια 1-3.

8.1 Λογική Επιλογής

Η πλατφόρμα που αναπτύχθηκε μπορεί να φιλοξενήσει μια ευρεία γκάμα μικρο-εφαρμογών. Μία πιθανή εφαρμογή είναι συλλογή δεδομένων από τους αισθητήρες των κινητών, χτίζοντας έτσι ένα μεγάλο data set από το οποίο μπορούν να εξαχθούν χρήσιμες πληροφορίες (πχ traffic monitoring με βάση της τιμές των επιταχυνσιόμετρων). Μια άλλη εφαρμογή θα μπορούσε να είναι η χρήση της πλατφόρμας ως μια κατανεμημένη, κινητή και αμιγώς προγραμματιστική πλατφόρμα. Μια τέτοια εφαρμογή θα είχε πρωτίστως ερευνητικό ενδιαφέρον, καθώς οι νέες cloud-based πλατφόρμες (Amazon WS, Google AppEngine κλπ) πλέον παρέχουν «άπειρη» υπολογιστική ισχύ για εφαρμογές κατανεμημένου υπολογισμού, σε χαμηλό κόστος.

Η χρήση μιας τέτοιας πλατφόρμας παρουσιάζει επίσης ιδιαίτερο ενδιαφέρον για εφαρμογές στον τομέα του crowdsourcing. Η ιεραρχική δομή των μικροεφαρμογών POBICOS, οι ελάχιστες απαιτήσεις τους σε μνήμη και υπολογιστική ισχύ, καθώς επίσης και η ελευθερία που παρέχεται από το υποκείμενο προγραμματιστικό περιβάλλον καθιστούν την πλατφόρμα μας ως μια καλή επιλογή για ανάπτυξη/εκτέλεση εφαρμογών για crowdsourcing.

8.2 Υλοποίηση

Το POBICOS παρέχει τα κατάλληλα εργαλεία για την παραγωγή μικροεφαρμογών. Με δεδομένο ότι ο πυρήνας του middleware έχει μείνει απαράλλαχτος κατά την μεταφορά του στο Android, τα ίδια αυτά εργαλεία μπορούν να χρησιμοποιηθούν για την παραγωγή εφαρμογών που θα εκτελούνται στην νέα πλατφόρμα που αναπτύξαμε.

8.3 Εισαγωγή στις εφαρμογές POBICOS

Μία εφαρμογή POBICOS αποτελείται από ένα σύνολο συνεργαζόμενων, ταυτόχρονα εκτελούμενων, υπολογιστικά ελαφρών διεργασιών, τους ονομαζόμενους **agents.** Οι agents ενός application οργανώνονται σε δομή δέντρου και έχουν σχέσεις γονέα-παιδιού.

Ο **Root** agent δημιουργείται αυτόματα από το POBICOS middleware και αποτελεί το σημείο εκκίνησης της εφαρμογής. Ο Root μπορεί έπειτα να δημιουργήσει έναν ή περισσότερους agent οι οποίοι, με την

σειρά τους, μπορούν να δημιουργήσουν άλλους agent κ.ο.κ. Οι κόμβοι – φύλλα στο δέντρο συνήθως αλληλεπιδρούν με το περιβάλλον μέσω των sensor και actuator που διαθέτουν.

Σχετικά με την επικοινωνία, ένας agent μπορεί να ανταλλάσει μηνύματα μονάχα με τον γονέα και τα παιδιά του. Τα μηνύματα από τον γονέα προς τα παιδιά ονομάζονται **commands,** ενώ τα μηνύματα από ένα παιδί προς τον γονέα του ονομάζονται **reports.** Τα μηνύματα αυτά στέλνονται είτε με **best-effort** είτε με **reliable** τρόπο.

Οι εφαρμογές του POBICOS αποτελούνται από micro-agents δύο κατηγοριών: τους generic και τους non-generic.

Οι generic agents χειρίζονται μόνο generic συμβάντα στο POBICOS και καλούν (μέσα από τους event handler τους) μόνο generic εντολές. Ένας non-generic agent μπορεί να τρέξει σε οποιονδήποτε κόμβο υποστηρίζει το ενδιάμεσο λογισμικό POBICOS.

Οι non-generic agents χειρίζονται ένα υποσύνολο non-generic συμβάντων και/ή καλούν ένα υποσύνολο non-generic εντολών. Ένας non-generic agent μπορεί να τρέξει μονάχα σε κόμβους που υποστηρίζουν τις αντίστοιχες εντολές και συμβάντα (δηλαδή έχουν τα κατάλληλα sensor/actuator)

8.4 Η δοκιμαστική polling εφαρμογή

Η εφαρμογή αναπτύχθηκε με την λογική των εφαρμογών τύπου crowdsourcing. Η λειτουργία του app είναι η ακόλουθη: Ένας root agent δημιουργείται αρχικά στο mobile που εκκινεί την εφαρμογή (e.g. Developer phone) . Έπειτα και για ένα περιορισμένο χρονικό διάστημα, ο root agent δημιουργεί συνεχώς non-generic agents σε συσκευές που μπορούν να τους υποστηρίξουν. Ο κάθε non-generic agent, μόλις δημιουργηθεί, εμφανίζει μια ερώτηση στον χρήστη. Μόλις ο χρήστης απαντήσει, τα αποτελέσματα αποστέλλονται με ένα report στον Root agent. Μετά το τέλος του προκαθορισμένου χρόνου, τα αποτελέσματα παρουσιάζονται στο developer phone που φιλοξενεί το root application.

8.4.1 Πηγαίος Κώδικας AndroidDemoRootAgent

```
#include <pobicos.h>
#include <string.h>

#define DIALOG_AGENT                    0x8001FFFF
#define POLL_EXPIRATION_TIMEOUT                 60000
#define POLL_TIMER_ID                   0
#define CREATION_TIMEOUT                30

int     agentCnt = 0;
int     yesReplyCnt = 0;
int     noReplyCnt = 0;

EVENT_HANDLER(PoInitEvent){
        PongObjectQualifier oq;

        PoEnableEvent(PoChildCreatedEvent);
```

```
        PoEnableEvent(PoReportArrivedEvent);
        PoEnableEvent(PoTimeoutEvent);

        PoBuildUpObjectQualifier(&oq, PO_INIT,
PONGO_MONITOR__CONSUMER_ELECTRONICS_OBJECT);
        PoCreateNonGenericAgents(DIALOG_AGENT, &oq, PO_CREATE_MULTIPLE,
CREATION_TIMEOUT, 0);
        PoSetTimer(POLL_TIMER_ID, POLL_EXPIRATION_TIMEOUT);
}

EVENT_HANDLER(PoChildCreatedEvent) {
        PoMsg msg;
        PoAgentID cid;
        PoAgentType ctype;
        PoReqHandle h;

        PoGetChildInfo(&cid,&ctype,&h);
        agentCnt++;

        sprintf((char *)msg.data, "%s", "Are you happy? :-) :-(");
        msg.len=strlen((char *)msg.data)+1;
        PoSendCommand(cid,&msg,PO_MSG_BESTEFFORT);
}

EVENT_HANDLER(PoReportArrivedEvent) {
        PoMsg msg;
        PoAgentID cid;
        PoAgentType ctype;
        PoReqHandle h;

        PoGetChildInfo(&cid,&ctype,&h);
        PoGetReport(&msg);
        if(msg.data[0] == '0')          // this is a NO
                noReplyCnt++;
        else                                    // this is a YES
                yesReplyCnt++;
}

EVENT_HANDLER(PoTimeoutEvent) {
        uint8_t buf[50];

        /*Total, YES, NO */
        sprintf((char *)buf, "__ROOT__%d-%d-%d",
                agentCnt, yesReplyCnt, noReplyCnt);

        PoDbgString((char *)buf);
        PoRelease(PoGetMyID());
```

```
}
```

Σημείωση: Όπως έχουμε ήδη αναφέρει, οι root agents δεν υποστηρίζουν non-generic λειτουργίες όπως η εμφάνιση μηνυμάτων στον χρήστη. Για να ξεπεράσουμε τον περιορισμό αυτό, τα developer phones (συσκευές που εκκινούν εφαρμογές POBICOS) μπορούν να δημιουργήσουν απλά message boxes για την εμφάνιση αναφορών. Ο τρόπος που αυτό επιτυγχάνεται είναι με την χρήση της generic εντολής *PoDbgString()* με ένα "magic" string ("__ROOT__").

8.4.2 Πηγαίος Κώδικας AndroidDemoDialogAgent

```
#include <pobicos.h>
#include <string.h>

EVENT_HANDLER(PoInitEvent){
        PoEnableEvent(PONGE_DIALOG_INPUT_RECEIVED);
        PoEnableEvent(PoCommandArrivedEvent);
}

EVENT_HANDLER(PONGE_DIALOG_INPUT_RECEIVED) {
        PoMsg msg;
        uint32_t reply = pongiGetDialogInput();
        sprintf((char*)msg.data, "%d", (int)reply);
        msg.len = strlen((char*)msg.data) + 1;
        PoSendReport(&msg,PO_MSG_BESTEFFORT);
}

EVENT_HANDLER(PoCommandArrivedEvent){
        PoMsg msg;
        PoGetCommand(&msg);
        pongiCreateDialog((uint8_t *)msg.data, (uint32_t)0 );
}
```

8.5 Σχόλια / Αποτελέσματα

Όπως φαίνεται και από τους πηγαίους κώδικες των AndroidDemoRootAgent και AndroidDemoDialogAgent, η δημιουργία εφαρμογών με την χρήση του POBICOS API είναι μια εύκολη διαδικασία. Μέσα σε ελάχιστες γραμμές κώδικα δημιουργήσαμε ένα application το οποίο μπορεί να τρέξει σε όλα τα διαθέσιμα κινητά μιας συγκεκριμένης περιοχής. Η εφαρμογή, σύμφωνα με τις αρχικές απαιτήσεις, λειτουργεί με τρόπο μη παρεμβατικό προς τον χρήστη.

8.6 Σενάριο χρήσης

Παρακάτω περιγράφουμε ένα απλό σενάριο χρήσης της ενδεικτικής crowdsourcing εφαρμογής. Χρησιμοποιήθηκαν δύο κινητά Android, το Samsung Galaxy 550 (Android 2.2) και το Google IDEOS

(Android 2.2). Θεωρούμε ότι η εφαρμογή που υποστηρίζει το POBICOS είναι ήδη εγκατεστημένη και στις δύο συσκευές.

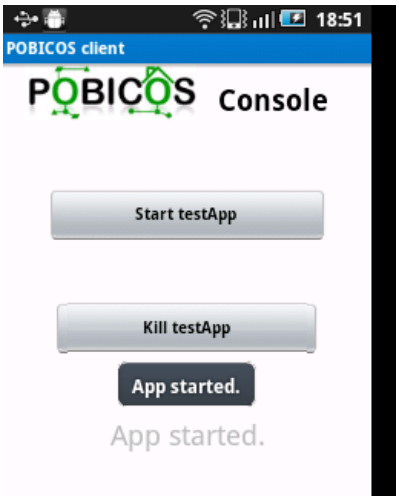
Φάση 1: Εκκίνηση της εφαρμογής.

Ο χρήστης του κινητού Α έχει τον ρόλο του developer που θέλει να εκκινήσει την προεγκατεστημένη, crowdsourcing POBICOS εφαρμογή μέσω του *application pill*. Στην τωρινή υλοποίησή μας, η συγκεκριμένη ενέργεια απαιτεί έναν μυστικό κωδικό.

Κινητό Α: Αρχική οθόνη εφαρμογής - Σχ. 32

Κινητό Α: Επιλογές εφαρμογής - Σχ. 33

Κινητό Α: Εισαγωγή κωδικού developer μετά την επιλογή του Application Pill Mode.
Σχ. 34



Κινητό Α: Διεπαφή λειτουργίας σε developer mode.
Σχ. 35

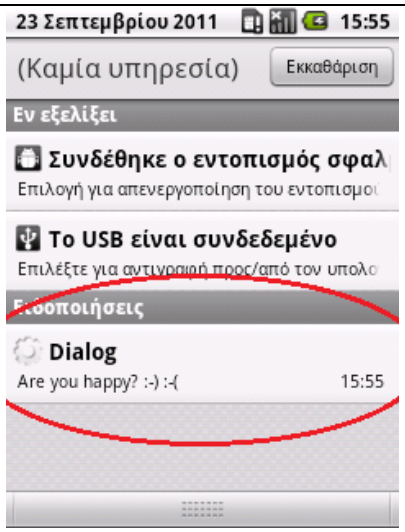
Το μήνυμα "*Connected*" υποδηλώνει πως το κινητό έχει συνδεθεί επιτυχώς στο POBICOS Registry & Forwarder. Μετά την επιτυχημένη εισαγωγή του κωδικού, ο developer αποκτά πρόσβαση στην κονσόλα που επιτρέπει την έναρξη και τον τερματισμό της crowdsourcing εφαρμογής. Στο στιγμιότυπο του Σχ. 32 , το κουμπί "Start testApp" έχει μόλις πατηθεί.

Η διαδικασία του *application pill loading/starting* έχει πλέον ξεκινήσει.

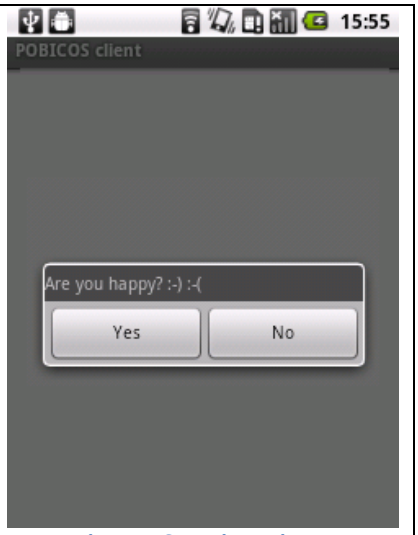
Φάση 2: Δημιουργία agent στο κινητό ενός απλού χρήστη.

Το application έχει πλέον ξεκινήσει, με τον root agent να φιλοξενείται στο κινητό A. Το κινητό B, για το οποίο ακολουθούν screenshots, βρέθηκε να ικανοποιεί τις απαιτήσεις του application ως προς την τοποθεσία και τις συνθήκες δημιουργίας POBICOS εφαρμογών (που στην περίπτωσή μας είναι η υποστήριξη των ζητούμενων non-generic primitives).

Κινητό B: Εμφάνιση Notification - Σχ. 36



Κινητό B: Drop-Down Notification Menu στο κινητό του χρήστη - Σχ. 37



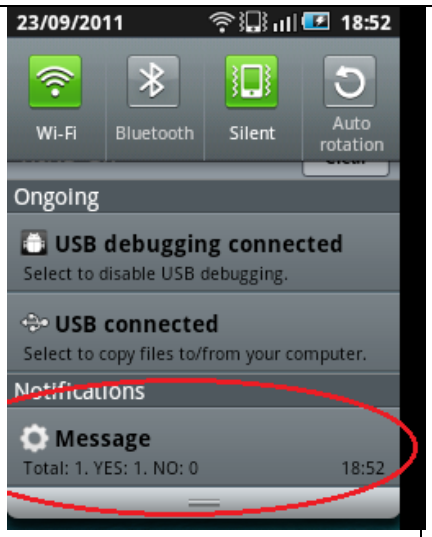
Κινητό B: Διαδικασία απάντησης – Σχ. 38

Ο *AndroidDemoDialogAgent*, που έχει δημιουργηθεί στο κινητό B, έχει μόλις δεχθεί το c*ommand* από τον root agent. Μία ειδοποίηση εμφανίζεται διακριτικά, ζητώντας από τον χρήστη να απαντήσει σε μία ερώτηση.

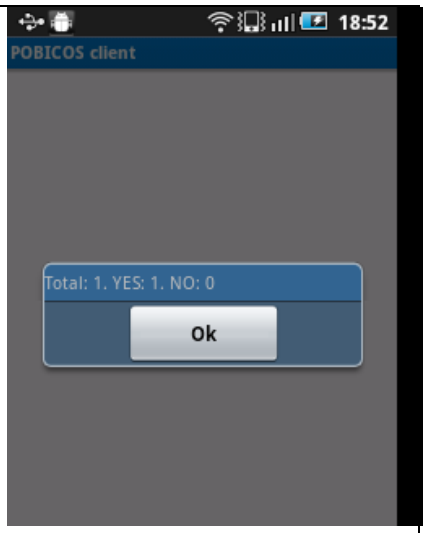
Φάση 3: Συλλογή αποτελεσμάτων.

Το χρονικό περιθώριο μέσα στο οποίο ο root agent περίμενε απαντήσεις έχει πλέον περάσει. Οι απαντήσεις των χρηστών (στην περίπτωσή μας, του χρήστη του κινητού Β) παρουσιάζονται στον developer – κάτοχο του κινητού Α.

Κινητό Α: Notification Αποτελεσμάτων
Σχ. 39



Κινητό Α: Drop-down notification menu
Σχ. 40



Κινητό Α: Εμφάνιση αποτελεσμάτων
Σχ. 41

Κεφάλαιο 9

Μετρήσεις

Παρακάτω ακολουθούν κάποιες μετρήσεις, ως μια εικόνα της απόκρισης του συστήματος. Σημειώνουμε ότι, λόγω των πρωτοκόλλων “liveness” του POBICOS, οι μετρήσεις ενδέχεται να εμπεριέχουν ‘θόρυβο’, τόσο στις χρονικές μετρήσεις όσο και στις αντίστοιχες για το μέγεθος των Tx/Rx δεδομένων. Έγιναν 10 πειράματα για κάθε μέτρηση, με το Registry & Forwarder.Μερικά χρήσιμα στοιχεία:

- **RTT** : **90 ms** (με βάση το διαγνωστικό εργαλείο “ping”) από τα κινητά προς το Registry.
- **NG Agent binary image length: 43 bytes**

1) Εγγραφή συσκευής στο μητρώο, μόλις συνδεθεί στο Internet. Περιλαμβάνεται το κόστος της δημιουργίας TCP σύνδεσης.

Μέσος χρόνος **365.2 ms** Standard Deviation: **211.6 ms**	Bytes Tx/Rx *-πάντα σταθερό (εν 6.4)-* **25/8**

2) Ανανέωση εγγραφής, όσο η συσκευή είναι συνδεδεμένη στο Internet.

Μέσος χρόνος **172.7 ms** Standard Deviation: **80.6 ms**	Bytes Tx/Rx *-πάντα σταθερό* (εν 6.4)- **25/8**

3) Δημιουργία τοπικού NG agent. Δεν περιλαμβάνεται η διαδικασία application pill loading.

Μέσος χρόνος **682.7 ms.** Standard deviation: **170.5 ms**

4) Δημιουργία απομακρυσμένου NG agent. Συνυπολογίζεται η μεταφορά του Agent image στον απομακρυσμένο κόμβο. Η διαδικασία application pill loading δεν περιλαμβάνεται στις μετρήσεις.

Μέσος χρόνος **1,591 ms** Standard deviation: **219.7 ms**	Bytes Tx/Rx *-κατά προσέγγιση, καθώς εμπεριέχονται bytes από τα Liveness πρωτόκολλα-* **162b / 132b** **8 msgs / 9 msgs**

5) RTT για ένα κενό ping-pong μήνυμα μεταξύ δύο agent που βρίσκονται στην ίδια συσκευή.

Μέσος χρόνος **50.7 ms** Standard deviation: **7 ms**

6) RTT για ένα κενό μήνυμα μεταξύ δύο απομακρυσμένων agent. Παρατηρούμε ότι ο μέσος χρόνος είναι περίπου ίσος με τον διπλάσιο μέσο χρόνο ανανέωσης εγγραφής. Το αποτέλεσμα είναι λογικό, αφού τα μικρά σε μέγεθος registration update μηνύματα μπορούν να θεωρηθούν ως RTT στο επίπεδο εφαρμογής μεταξύ agent <-> registry.

Μέσος χρόνος **368.85 ms** Standard deviation: **170.6 ms**	Bytes Tx/Rx *-σταθερό για κενά μηνύματα-* **46/46**

Κεφάλαιο 10

Συμπεράσματα και Βελτιώσεις

Το project που υλοποιήθηκε είχε κάποιες βασικές απαιτήσεις. Κάποιες σχεδιαστικές αποφάσεις δεν έγιναν με γνώμονα την βέλτιστη λύση αλλά την γρήγορη υλοποίηση ενός λειτουργικού πρωτότυπου. Για παράδειγμα, όπως έχει συζητηθεί στην ενότητα 3.4, μια ολοκληρωμένη λύση για το Registry & Forwarder θα είχε ως εξής:

1. Ξεχωριστά συστήματα για τις ενέργειες του Registry και του Forwarder.
2. Το Forwarder θα αποτελείται από πολλούς server. Ιδανικά, θα υπάρχουν server μοιρασμένοι σε διάφορες τοποθεσίες για forwarding με low latency.
3. Το Registry θα αποτελείται και αυτό από πολλούς server, δομημένους σε μια ιεραρχική δομή, όπως ισχύει στο **DNS** και το **H**ome **L**ocation **R**egister (GSM).

Όσο αφορά την υποστήριξη non-generic primitive, για την ώρα υποστηρίζονται μόνο βασικές UI λειτουργίες. Μελλοντικές εκδόσεις της πλατφόρμας θα υποστηρίζουν μεγαλύτερη γκάμα non-generic event/instruction και θα έχουν αυξημένες δυνατότητες sensing/actuating. Το μοντέλο – οντολογία που περιγράφει αυτά τα primitives στο POBICOS PROJECT μπορεί επίσης να εμπλουτιστεί σύμφωνα με τις νέες δυνατότητες του mobile domain. Για παράδειγμα, θα μπορούσε να προστεθεί ένα sensing instruction που θα συλλέγει δεδομένα από το επιταχυνσιόμετρο.

Στο middleware υπάρχει χώρος για πολλές βελτιστοποιήσεις. Τα μηνύματα δικτύου έχουν πολύ μικρό επιτρεπόμενο μέγεθος, με αποτέλεσμα να δημιουργείται υπολογίσιμο overhead κατά την μεταφορά δεδομένων, λόγω των σχετικά μεγάλων header που προστίθενται στα πακέτα λόγω του PoAPI Layer. Επίσης, αρκετά πρωτόκολλα (κυρίως τα σχετιζόμενα με 'liveness' διαδικασίες) θα μπορούσαν να προσαρμοστούν στις ανάγκες του mobile domain με κατάλληλη αλλαγή των αντίστοιχων timer.

Σχετικά την υλοποίηση του PoAPI και του Android Application, είναι σίγουρο ότι θα έχουν ξεφύγει κάποια bug. Υπολογίζοντας μόνο τον κώδικα που γράφτηκε σε Android/Java το μέγεθος φτάνει τις 2.600 γραμμές. Γενικά οι γραμμές κώδικα δεν είναι το κατάλληλο μέτρο για εξαγωγή συμπερασμάτων όσο αφορά την πολυπλοκότητα των εφαρμογών, αλλά αν μη τι άλλο δείχνει μια γενική εικόνα της απαιτούμενης δουλειάς.

Τέλος, οι απαιτήσεις τοποθεσίας είναι "hard-coded" στο εκάστοτε application. Για την εισαγωγή νέων απαιτήσεων θέσης απαιτείται ένα εξωτερικό εργαλείο που θέτει τις πληροφορίες αυτές εκ νέου στο πακέτου της εφαρμογής(application bundle). Αντί για αυτό, θα μπορούσαμε να δημιουργήσουμε ένα επιπλέον User Interface, στο οποίο ο χρήστης επιλέγει την επιθυμητή τοποθεσία μέσω χαρτών (πχ google maps).

Κεφάλαιο 11

Επίλογος

Η μεταφορά του POBICOS από τον χώρο των δικτύων αισθητήρων σε συσκευές κινητής τηλεφωνίας ήταν μια απαιτητική εργασία που είχε να διδάξει πολλά. Ο σωστός σχεδιασμός κατανεμημένων, κινητών και διάχυτων συστημάτων απαιτεί μεγάλη έμφαση στην λεπτομέρεια και ουσιαστική κατανόηση των μέσων επικοινωνίας και των υποκείμενων πρωτοκόλλων. Όπως χιουμοριστικά είχε πει ο Leslie Lamport, “κατανεμημένο σύστημα είναι το σύστημα στο οποίο η βλάβη ενός υπολογιστή, ο οποίος δεν γνώριζες ότι υπάρχει, μπορεί να καταστήσει άχρηστο τον δικό σου υπολογιστή». Το πρόβλημα μπορεί να γίνει πολλαπλασιαστικά μεγαλύτερο στον τομέα του διάχυτου/απανταχού υπολογισμού.

Το τελικό Project έχει επιτύχει ένα ιδιαίτερα ενδιαφέρον αποτέλεσμα: Συσκευές αισθητήρων και κινητά τηλέφωνα μπορούν να συμμετέχουν στην ίδια κοινότητα αντικειμένων, εκτελώντας την ίδια εφαρμογή. Κάτι τέτοιο δεν θα ήταν φυσικά εφικτό αν το POBICOS δεν είχε εξ’ αρχής σχεδιαστεί με αυτό το όραμα. Η πειραματική του λειτουργία όμως, επιβεβαιώνει πως συνηθισμένες, ετερογενής συσκευές μπορούν να συνεργαστούν για έναν κοινό σκοπό. Η απαραίτητη τεχνολογία, το hardware και το software, είναι πλέον μέρος της καθημερινότητάς μας. Είναι πλέον ζήτημα καθορισμού καθολικών προτύπων/middleware για την διαλειτουργικότητα των συσκευών αυτών. Τόσο στην έρευνα όσο και στην αγορά έχουν γίνει τα πρώτα βήματα προς αυτή την κατεύθυνση και είναι θέμα χρόνου να καθιερωθεί κάποιο κοινά αποδεκτό πρότυπο επικοινωνίας/μοντέλο προγραμματισμού για πλατφόρμες διάχυτου υπολογισμού.

Βιβλιογραφία

[1] Android SDK Guide - *Google*

[2] Android NDK Guide - *Google*

[3] Java Native Interface: Programmer's Guide and Specification – *Orcacle corporation*

[4] nesC 1.2 Language Reference Manual – *D.Gay, P. Levis, D.Culler, E.Brewer*

[5] Effective Java, 2nd Edition – *Joshua Bloch*

[6] TinyOS Programming – *Philip Levis*

[7] Java Concurrency in Practice – *T.Peierls, J.Bloch, J.Bowbeer, D.Holmes, D.Lea*

[8] *POBICOS Project Papers*

Αναφορές

[1] http://www.csd.cs.cmu.edu/research/areas/mopercomp/
[2] http://sandbox.xerox.com/want/papers/ubi-sciam-sep91.pdf
[3] http://www.mobilemarketingwatch.com/idc-estimates-50-growth-in-worldwide-smartphone-market-in-2011-14227/
[4] POBICOS Project - http://www.ict-pobicos.eu/summary.htm
[5] Agilla Middleware - http://mobilab.cse.wustl.edu/projects/agilla/
[6] MagnetOS - http://www.cs.cornell.edu/people/egs/magnetos/
[7] SmartMessages - http://discolab.rutgers.edu/sm/
[8] Hydra Middleware – http://www.hydramiddleware.eu/
[9] Crowdsourcing - http://www.wired.com/wired/archive/14.06/crowds.html
[10] Amazon Mechanical Turk - https://www.mturk.com/mturk/welcome
[11] txteagle - http://reality.media.mit.edu/pdfs/hcii_txteagle.pdf
[12] mCrowd - http://lass.cs.umass.edu/~yan/pubs/mCrowd_Demo_SenSys09.pdf
[13] http://nescc.sourceforge.net/
[14] http://www.tinyos.net/
[15] http://developer.android.com/guide/basics/what-is-android.html
[16] http://www.dalvikvm.com/
[17] http://developer.android.com/sdk/ndk/overview.html
[18] Java Native Interface: Programmer's Guide and Specification- http://java.sun.com/docs/books/jni/
[19] http://www.tinyos.net/tinyos-2.x/doc/html/tep107.html
[20] http://sourceforge.net/projects/pobicos/
[21] UDP Hole Punching - http://en.wikipedia.org/wiki/UDP_hole_punching
[22] Reactor Programming Pattern using Java nio - http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf
[23] Nginx High-Performance HTTP Server - http://wiki.nginx.org/

[24] http://en.wikipedia.org/wiki/Haversine_formula