Πολυτεχνείο Κρήτης
Τμήμα Ηλεκτρονικών Μηχανικών
και Μηχανικών Ηλεκτρονικών Υπολογιστών

Δημιουργία συστήματος διαχείρισης έξυπνων κτηρίων με τη χρήση του πρωτοκόλλου XMPP

Γιώργος Λογιωτατίδης

Εξεταστική επιτροπή

Γιώργος Σταυρακάκης, καθηγητής (επιβλέπων)
Κωνσταντίνος Καλαϊτζάκης, καθηγητής
Αλέξανδρος Ποταμιάνος, αναπληρωτής καθηγητής

Χανιά, Γενάρης 2008

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth

Περίληψη

Κατασκευάσαμε ένα σύστημα αυτοματισμού κτηρίων συνδέοντας δύο διαφορετικά συστήματα και κόσμους, το διαδίκτυο και τα δίκτυα ανταλλαγής μηνυμάτων με το δίκτυο αυτοματισμού X10. Στην εργασία προσεγγίσαμε των αυτοματισμό κτηρίων με τρόπο διαφορετικό από τον κυρίαρχο, χωρίς τη χρήση δυαδικών πρωτοκόλλων αλλά με τη χρήση XML, ενσωματωμένων λειτουργικών συστημάτων και γλωσσών προγραμματισμού υψηλού επιπέδου. Δώσαμε επίσης βαρύτητα στον άνθρωπο, ο οποίος είναι ο τελικός αποδέκτης οποιουδήποτε συστήματος αυτοματισμού κτηρίων, επιλέγοντας υλικά και μεθόδους οι οποίες είναι οικίες σε αυτόν και προσφέρουν δυνατότητες προσαρμογής στην καθημερινότητα και τρόπους επικοινωνίας του.

Το αποτέλεσμα της εργασίας ήταν η κατασκευή τεσσέρων εφαρμογών, του διαχεριστή μηνυμάτων, της γέφυρας XMPP - X10, των αναγνωριστή παρεβρισκομένων στο χώρο με τη χρήση Bluetooth και της ηλεκτρονικής USB κλειδαριάς, που συνεργάστηκαν άψογα. Αποδείξαμε ότι η τεχνολογία ανταλλαγής μηνυμάτων και ενημέρωσης παρουσίας XMPP είναι κατάλληλη να υποστηρίξει ένα σύστημα αυτοματισμού και επιπλέον προσφέρει τις δυνατότητες για να χτίσουμε τα συστήματα αυτοματισμού για τις ανάγκες του μέλλοντος.

Ευχαριστίες

Αρχικά θέλω να ευχαριστήσω τους γονείς μου, Αθηνά και Γιάννη και την αδερφή μου Μαρία για την πλήρη υποστήριξη τους σε όλες τις επιλογές μου και συγκεκριμένα για τα χρόνια στο πολυτεχνείο.

Ευχαριστώ πολύ τον επιβλέποντα καθηγητή Γ. Σταυρακάκη και τον υποψήφιο διδάκτορα Β. Τριπολιτάκη για την εμπιστοσύνη και τις συμβουλές τους κατά την εκπόνηση της εργασίας αυτής. Επίσης ευχαριστώ τον καθηγητή Κ. Καλαϊτζάκη και τον αν. καθηγητή Α. Ποταμιάνο για την ενασχόλησή τους με τη δουλειά μου. Ευχαριστώ την Ιωάννα που διόρθωσε συντακτικά, ορθογραφικά και νοηματικά το κείμενο.

Ευχαριστώ πολύ τους φίλους μου και ιδιαίτερα την Άντυ, τον Τάσο που ήταν πάντα εκεί στα καλά και άσχημα και τον Αντώνη για τις ατέλειωτες ώρες δημιουργίας, ανταλλαγής ιδεών και απόψεων, αγωνίας και χαράς που περάσαμε παρέα. Τέλος ευχαριστώ τους συντρόφους μου που βοήθησαν πέρα από μηχανικός να γίνω και Άνθρωπος.

Περιεχόμενα

Κατάλογος σχημάτων

Listings

1 Εισαγωγή

1.1 Γενικά

Ένα προγραμματιζόμενο, αυτοματοποιημένο, έξυπνο δίκτυο ηλεκτρονικών συσκευών, το οποίο παρακολουθεί και ελέγχει μηχανικά και ηλεκτρικά συστήματα σε ένα κτήριο ονομάζεται **αυτοματισμός κτηρίου**. Ένα αυτοματοποιημένο κτήριο ονομάζεται ***έξυπνο κτήριο***.

Ένα έξυπνο κτήριο μπορεί να διαχειριστεί τον φωτισμό, τις πόρτες, τα παράθυρα, τα συστήματα ασφαλείας και γενικά το σύνολο τον ηλεκτρικών συσκευών. Τα τελευταία χρόνια στα έξυπνα κτήρια συμπεριλαμβάνονται ολοκληρωμένα συστήματα ψυχαγωγίας πολυμέσων, συστήματα προσαρμογής περιβαλοντικών συνθηκών κ.α.

Στο άμεσο μέλλον τα έξυπνα κτήρια θα παρέχουν διασύνδεση του συνόλου των συσκευών του κτηρίου με το διαδίκτυο και κατά συνέπεια με φορητές συσκευές όπως κινητά τηλέφωνα, θα προσαρμόζουν τις περιβαντολογικές συνθήκες βάσει προσωπικών προτιμήσεων κάθε ατόμου λαμβάνοντας υπόψιν και παραμέτρους εξοικονόμησης ενέργειας. Επίσης θα μπορούν να καταστήσουν λίστες αγορών ακόμη και να κάνουν τις κατάλληλες παραγγελίες, να διαχειρίζονται συνδυαστικά συσκευές όπως να χαμηλώνουν την ένταση της τηλεόρασης όταν χτυπάει το τηλέφωνο.

Η υλοποίηση όλων των παραπάνω αυτοματισμών και της ομαλής συνεργασίας αυτών με τη δυνατότητα κεντρικής διαχείρισης είναι τα απαιτούμενα χαρακτηριστικά ενός σύγχρονου συστήματος αυτοματισμού. Επίσης θα πρέπει να παρουσιάζει προσαρμοστικότητα στις ανάγκες και απαιτήσεις του μέλλοντος.

Ο επίτευξη των στόχων αυτών είναι σίγουρα μια πρόκληση για τους μηχανικούς. Γνωρίζοντας δεν είναι δυνατό να υλοποιηθεί το σύνολο αυτών, με

τις υπάρχουσες τεχνολογίες αυτοματισμών, θα πρέπει να μελετήσουμε και να δημιουργήσουμε τα πρωτόκολλα και τις τεχνολογίες που θα βασίσουμε τους αυτοματισμούς του μέλλοντος. Σε αυτή την εργασία προτείνουμε μία πλατφόρμα ανάπτυξης εφαρμογών έξυπνων κτηρίων και αναπτύσσουμε στη βάση αυτή μερικές εφαρμογές.

1.2 Περιγραφή και προσέγγιση του προβλήματος

Η τεχνολογία κατασκευής έξυπνων κτηρίων σήμερα, βασίζεται κατά κύριο λόγο στη χρήση κλειστών πρωτοκόλλων, εξειδικευμένου και συχνά παρωχημένου υλικού και τεχνολογίες με υψηλό κόστος συντήρησης και περιορισμένων επεκτάσεων. Τα χαρακτηριστικά των συστημάτων αυτών, τα καθιστούν απρόσιτα στο ευρύ κοινό που αδυνατεί αλλά και αρνείται να πληρώσει το υψηλό οικονομικό αντίτιμο για τεχνολογίες που δεν καλύπτουν συνολικά τις ανάγκες του, έχουν κόστος εκμάθησης, δεν συνδέονται εγγενώς με το διαδίκτυο και δεν προσφέρουν δυνατότητα επέκτασης για τα σπίτια του μέλλοντος.

Η ραγδαία αύξηση της υπολογιστικής ισχύος στους προσωπικούς υπολογιστές (PC) συμπαρασύρει και τα ενσωματωμένα συστήματα (Embedded Systems) με αποτέλεσμα να μπορούμε πλέον να προσπεράσουμε τις ούτως ή άλλως παρωχημένες τεχνολογίες των δυαδικών πρωτοκόλλων και να χρησιμοποιήσουμε πρωτόκολλα και πρότυπα από τα πρώτα συστήματα, στα δεύτερα.

Μπορούμε να χρησιμοποιήσουμε ενσωματωμένα συστήματα με πλήρες λειτουργικό σύστημα πολλών παράλληλων διεργασιών (multi tasking) και γλώσσες προγραμματισμού υψηλού επιπέδου. Επίσης μπορούμε να δανειστούμε πρωτόκολλα επικοινωνίας διεθνώς αναγνωρισμένα, χωρίς δικαιώματα και περιορισμούς στη χρήστη τους από το διαδίκτυο.

Στη παρούσα εργασία ασχοληθήκαμε με την υλοποίηση ενός συστήματος διαχείρισης έξυπνων κτηρίων βασισμένο στο πρωτόκολλο XMPP, που χρησιμοποιείται για την άμεση ανταλλαγή μηνυμάτων μεταξύ χρηστών και εφαρμογών στο Διαδίκτυο. Πέρα από την εγγενή σύνδεση με το Διαδίκτυο, το πρωτόκολλο XMPP προσφέρει πολλές επεκτατικές δυνατότητες για να καλύψει τις ανάγκες του μέλλοντος. Επιλέξαμε σε όλα τα επίπεδα λογισμικού, Ελεύθερο Λογισμικό / Λογισμικό Ανοιχτού Κώδικα (ΕΛ/ΛΑΚ), το οποίο μας επιτρέπει να μειώσουμε το κόστος κατασκευής και να αυξήσουμε τις τεχνολογικές δυνατότητες. Βασιστήκαμε σε τεχνολογίες οικίες, που έχουν δοκιμαστεί για χρόνια όπως οι USB μνήμες και το Bluetooth ώστε να μπορεί ο μέσος χρήστης ηλεκτρονικών υπολογιστών να επωφεληθεί άμεσα από το σύστημα μειώνοντας το κόστος εκμάθησης ή τις δυσκολίες προσαρμογής. Η ανάπτυξη του συστήματος και οι επιλογές υλικού και λογισμικού έγιναν με ανθρωποκεντρική κατεύθυνση, ώστε να προσαρμόσουμε στο μέγιστο βαθμό το σύστημα σε ανθρώπινη τροχιά και όχι το αντίστροφο.

1.3 Οργάνωση του κειμένου

Στα επόμενα κεφάλαια της εργασίας θα προσεγγίσουμε με την τεχνική *bottom - up*, δηλαδή από την πιο απλή μονάδα στη πιο σύνθετη, την προτεινόμενη πλατφόρμα λειτουργίας καθώς και τις κοινοποιηθείσες εφαρμογές και τα συμπεράσματα που εξάγαμε από το σύνολο του εγχειρήματος, την ανάπτυξη και τις δοκιμές.

Πιο αναλυτικά παρακάτω στο **Κεφάλαιο 1** σε δύο μεγάλες ενότητες αναλύονται το *υλικό* και το *λογισμικό* που επιλέξαμε. Στη πρώτη ενότητα *"Υλικό"* παρουσιάζονται η πλατφόρμα FOX, τα τεχνικά χαρακτηριστικά της και οι δυνατότητές της και η τεχνολογία αυτοματισμού X10. Στη δεύτερη ενότητα *"Λογισμικό"* παρουσιάζονται το λειτουργικό σύστημα GNU/Linux, η

γλώσσα προγραμματισμού Python και τέλος το λογισμικό HEYU για τη διαχείριση συσκευών X10 από τον υπολογιστή.

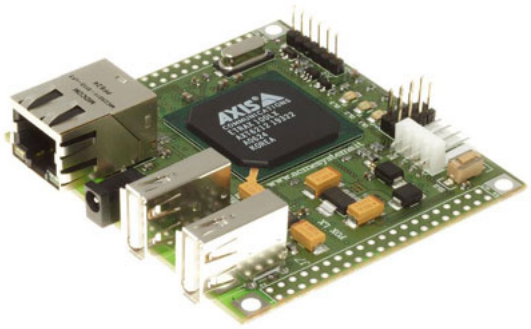
Το **Κεφάλαιο 2** εισάγει τον αναγνώστη στο πρωτόκολλο XMPP που θα χρησιμοποιήσουμε για την δημιουργία του δικτύου μεταξύ των έξυπνων συσκευών, γνωστοποιώντας του ιστορικά στοιχεία, τα τεχνικά χαρακτηριστά, τον τρόπο διευθυνσιοδότησης και τους τύπους των μηνυμάτων. Μία παράγραφος αφιερώνεται στη δυνατότητα διασύνδεσης των δικτύων XMPP με άλλα δίκτυα, που χρησιμοποιήσαμε για να πετύχουμε τη σύνδεση με το δίκτυο X10.

Στο **Κεφάλαιο 3** περιγράφονται αναλυτικά όλες οι εφαρμογές που δημιουργήθηκαν στα πλαίσια της εργασίας, για τη δημιουργία του συστήματος διαχείρισης έξυπνων κτηρίων, όπως ο διαχειριστής μηνυμάτων, η γέφυρα μεταξύ των πρωτοκόλλων XMPP και X10, η ηλεκτρονική USB κλειδαριά και τέλος ο αναγνωριστής παρευρισκομένων σε χώρο με χρήση Bluetooth.

Τα συμπεράσματα από τον συνδυασμό των εφαρμογών σε πραγματικές συνθήκες, σε επίπεδο υλικού, λογισμικού και πρωτοκόλλων καταγράφονται στο **Κεφάλαιο 4**.

Στις τέσσερις παραγράφους του **Κεφαλαίου 5** προτείνονται επεκτάσεις του συστήματος, σε επίπεδο χρηστικότητας, ασφάλειας και επεκτασιμότητας.

Στο παράρτημα του **Κεφαλαίου 6** υπάρχουν αναφορές για το λογισμικό με το οποίο πειραματιστήκαμε κατά τη διάρκεια ανάπτυξης των εφαρμογών XMPP, όπως εξυπηρετητές, πελάτες και βιβλιοθήκες λογισμικού. Παρατίθεται ο κώδικάς των όλων των εφαρμογών που αναπτύξαμε στη παράγραφο 6.2. Τέλος στη παράγραφο 6.3 ερμηνεύουμε ορολογία από όλη την έκταση του κειμένου.



Σχήμα 1: Η πλατφόρμα FOX

1.4 Υλικό

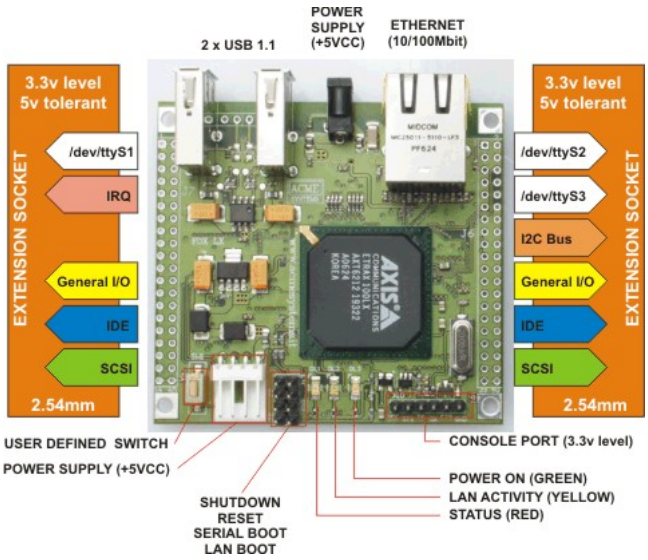
1.4.1 Πλατφόρμα FOX

Η πλατφόρμα FOX είναι ένα ολοκληρωμένο ενσωματωμένο Linux σύστημα με μέγεθος μόλις 66mm x 72mm. Αναπτύσσεται από την εταιρία Acme Systems [8] και τα τεχνικά της χαρακτηριστικά παρουσιάζονται αναλυτικά στον πίνακα 1. Επίσης στον πίνακα 2 παρουσιάζονται τα χαρακτηριστικά του λογισμικού που παρέχει η εταιρία με την πλατφόρμα.

Το FOX αξιοποιώντας τη Μονάδα Διαχείρισης Μνήμης (MMU) του επεξεργαστή ETRAX [9] μπορεί να τρέξει έναν σύγχρονο πλήρη πυρήνα Linux. Σε αντίθεση με όλα τα παλαιότερα ενσωματωμένα συστήματα και με πολλά σύγχρονα, η πλατφόρμα δεν είναι μικροCLinux (uClinux) [1]. Μόνο με μερικά ”μπαλώματα” (patches) στον επίσημο ”vanilla” πυρήνα του linux αρκούν για να μετατρέψουμε το FOX σε ένα Linux σύστημα.

Η πλατφόρμα είναι ιδανική για την ανάπτυξη εφαρμογών αυτοματισμού, αφού με κατανάλωση μόλις 1 Watt μας παρέχει Ethernet δικτύωση

[1] Το uClinux είναι ένα εγχείρημα μεταφοράς του πυρήνα του Linux σε επεξεργαστές χωρίς Μονάδα Διαχείρησης Μνήμης. Αν και το εγχείρημα είναι επιτυχές σε πολλές περιπτώσεις επεξεργαστών, ένα τέτοιο Linux σύστημα είναι αρκετά περιορισμένο σε δυνατότητες



Σχήμα 2: Είσοδοι και έξοδοι της πλατφόρμας FOX

και δύο USB θύρες που μπορούν μεταξύ των άλλων να χρησιμοποιηθούν για τη σύνθεση μέσων αποθήκευσης ή να μετατραπούν σε σειριακές θύρες. Επιπλέον παρέχεται σύνθεση με το I2C δίκτυο και μέχρι 48 γενικής χρήσεως θέσεις εισόδου εξόδου. Τέλος με πρόσθετα επιθέματα παρέχεται σύνδεση με το δίκτυο GSM και το γρήγορα αναπτυσσόμενο ασύρματο δίκτυο δεδομένων ZigBee.

Μέγεθος	66 x 72mm
CPU	Axis ETRAX 100LX, 32bit, RISC, 100Mhz (100MIPS)
Μνήμη	16Mb Flash, 32Mb RAM
Καταλάνωση Ενέργειας	5 Volt 280mA (1 Watt)
Θύρες	1 Ethernet (10/100 Mb/s), 2 USB (1.1), 1 σειριακή
Επεκτάσεις	2 θύρες επέκτασης για IDE, SCSI, σειριακές γραμμές, παράλληλες γραμμές, I/O και I2C
Βάρος	37gr
θερμοκρασία Λειτουργίας	0-70 ^{o}C

Πίνακας 1: Χαρακτηριστικά Υλικού

Πυρήνας	Πλήρης έκδοση του Linux 2.4.31 ή 2.6.15
Εξυπηρετητές	HTTP, FTP, SSH, TELNET
Οδηγοί	USB Flash, FTDI και PROLIFIC USB to Serial Converter
SDK	SDK ανοιχτού κώδικα για συστήματα Linux
Υποστηριζόμενες Γλώσσες	C, C++, PHP, Python etc
Εργαλεία	WEB υπηρεσία μεταγλώτισης με το GNU C

Πίνακας 2: Χαρακτηριστικά Λογισμικού

1.4.2 Τεχνολογία X10

Το X10 είναι ένα διεθνές ανοιχτό βιομηχανικό πρότυπο για την επικοινωνία ηλεκτρονικών συσκευών, με κατεύθυνση τον αυτοματισμό σπιτιών. Χρησιμοποιεί, κατά κύριο λόγο, την καλωδίωση του ηλεκτρικού ρεύματος για να μεταφέρει ψηφιακή πληροφορία, αλλοιώνοντας τη κυματομορφή του ρεύματος. Τα τελευταία χρόνια έχει παρουσιαστεί και ένα πρωτόκολλο επικοινωνίας για το X10 με κύριο μέσω επικοινωνίας τις ραδιοσυχνότητες.

Το X10 εφευρέθηκε το 1975 από την εταιρία Pico Electronics στη Σκωτία και ήταν η πρώτη τεχνολογία για τον αυτοματισμό σπιτιών. Σήμερα παρόλο που υπάρχουν πολλές άλλες λύσεις με περισσότερες δυνατότητες από το X10 όπως το KNX, το INSTEON, το LonWorks, παραμένει πολύ δημοφιλές κυρίως εξαιτίας του χαμηλότερου κόστους απόκτησης σε σχέση με τα υπόλοιπα πρότυπα.

Για τη μεταφορά της πληροφορίας στο πρωτόκολλο X10 ακολουθείται η παρακάτω διαδικασία: Για κάθε μηδενικό που παρουσιάζεται στη κυματομορφή του ηλεκτρικού ρεύματος κωδικοποιείται ένα δυφίο (bit) στη συχνότητα των 120kHz και προστίθεται σε αυτή. Το ρεύμα έχει συχνότητα 50 ή 60 Hz, ανάλογα τη χώρα που βρισκόμαστε, άρα με το πρωτόκολλο X10 μπορούμε να μεταφέρουμε περίπου 120 bit/s, εύρος ζώνης που είναι επαρκές για πολύ βασικές λειτουργίες, όπως ο έλεγχος "ανοιχτό"/"κλειστό"

ηλεκτρικών συσκευών και μεταφορά πληροφορίας από αισθητήρες θερμοκρασίας ή άλλου μεγέθους. Για τη μεταφορά των σημάτων X10 σε μεγάλη απόσταση, ή σε άλλη φάση του ρεύματος μπορούν να χρησιμοποιηθούν αναμεταδότες. Επίσης για τον περιορισμό των σημάτων X10 σε μία συγκεκριμένη περιοχή, όπως το σπίτι μας, θα πρέπει να χρησιμοποιηθούν φίλτρα, ώστε να μην παρεμβάλλουμε εντολές σε άλλα δίκτυα X10.

Η διευθυνσιοδότηση στο σύστημα γίνεται με δύο μεταβλητές, των τεσσάρων δυφίων η καθεμία. Μπορούμε να επιλέξουμε ένα γράμμα από το αγγλικό "A" μέχρι το αγγλικό "P" που ονομάζουμε κωδικό σπιτιού (house code) και ένα αριθμό από το 1 μέχρι το 16 που ονομάζουμε κωδικό μονάδας (unit code). Μπορούμε να ονοματίσουμε δύο ή και παραπάνω διαφορετικές συσκευές με τους ίδιους κωδικούς εάν θέλουμε να τις ελέγχουμε ταυτόχρονα και αδιάσπαστα (εάν για παράδειγμα θέλουμε δύο φώτα του σπιτιού να ανάβουν και να σβήνουν πάντα μαζί). Σύμφωνα με την παραπάνω διευθυνσιοδότηση σε ένα δίκτυο X10 μπορούν να υπάρχουν μόλις 256 συσκευές με διαφορετικές διευθύνσεις.

Οι συσκευές X10 παρεμβάλλονται συνήθως μεταξύ της υπό έλεγχο συσκευής και της πρίζας ηλεκτρικού ρεύματος. Καταλαμβάνουν δηλαδή τη διαθέσιμη πρίζα και προσφέρουν μια νέα X10 πρίζα. Επίσης παρεμβάλλονται ανάμεσα στη λάμπα και στην υποδοχή της λάμπας με την ίδια λογική ή τέλος παίρνουν τη θέση συμβατικών διακοπτών ή μια θέση στον ηλεκτρικό πίνακα του σπιτιού. Οι περισσότερες συσκευές X10 μπορούν απλά να ανοιγοκλείσουν μια συσκευή ή στη περίπτωση λαμπτήρα να αυξομειώσουν και τη φωτεινότητά του, ενώ άλλες πιο εξελιγμένες συσκευές μπορούν να δηλώσουν την κατάστασή τους (επικοινωνία 2-δρόμων), να προγραμματιστούν, να ελεγχθούν τοπικά και όχι μόνο από το ελεγκτή X10 κ.α.

Ο ελεγκτής X10 (X10 controller) έχει δύο μορφές, την αυτόνομη και την

διεπαφή υπολογιστή. Στη πρώτη ο ελεγκτής έχει ενσωματωμένα πλήκτρα ελέγχου για ένα μικρό αριθμό συσκευών (περίπου μέχρι 10) και μπορεί να χρησιμοποιηθεί απευθείας με τη σύνδεσή του σε μια πρίζα. Η διεπαφή υπολογιστή είναι πιο εξελιγμένος ελεγκτής, αφού μας επιτρέπει να διαχειριστούμε τις συσκευές του σπιτιού με τη χρήση ειδικού λογισμικού και να τον προγραμματίσουμε να εκτελεί συγκεκριμένες λειτουργίες χωρίς την παρουσία του υπολογιστή.

Για την εργασία χρησιμοποιήσαμε τρεις μονάδες X10, έναν ελεγκτή υπολογιστή, ένα τερματικό σύνδεσης λαμπτήρα και ένα τερματικό σύνδεσης άλλης ηλεκτρικής συσκευής. Συγκεκριμένα τη μονάδα CM11 για σύνδεση με τον υπολογιστή, τη μονάδα LM12 για τη διαχείριση ενός φωτιστικού και τη μονάδα AM12 για τη διαχείριση μιας καφετιέρας. Η επιλογή του δικτύου X10 και όχι κάποιου άλλου έγινε εξαιτίας του χαμηλού κόστους απόκτησης, της ευκολίας εγκατάστασης, της δημοτικότητας και τέλος της απλότητας του πρωτοκόλλου που μας επέτρεψε να σχεδιάσουμε και να υλοποιήσουμε ένα πρωτότυπο δίκτυο ελέγχου εύκολα κατανοητό και επεκτάσιμο.

Παρά τις ιδιαιτερότητες του δικτύου X10, όπως το χαμηλό εύρος ζώνης, οι έλλειψη αναφοράς κατάστασης σε όλες τις συσκευές και προβλήματα όπως η παύση λειτουργίας εξαιτίας θορύβου στο μέσο μεταφοράς, αποδείχθηκε αξιοπρεπές σύστημα για ένα οικονομικό, μικρού μεγέθους δικτύο αυτοματισμού.

Σχήμα 3: Δύο τερματικές συσκευές και μία συσκευή ελέγχου X10 για υπολογιστή. Ο συνδιασμός αυτός χρησιμοποιήθηκε στη παρούσα εργασία

1.5 Λογισμικό

1.5.1 Λειτουργικό σύστημα GNU/Linux

Το λειτουργικό σύστημα GNU/Linux είναι ένα λειτουργικό τύπου Unix αλλά χωρίς να χρησιμοποιεί κώδικα από το Unix. Αποτελείται από ένα σύνολο προγραμμάτων, όπως μεταγλωττιστές, επεξεργαστές κειμένου, γραφικό περιβάλλον, εξυπηρετητές για διάφορες υπηρεσίες κ.α. και φυσικά από έναν πυρήνα που αναλαμβάνει τον συντονισμό αυτών και την επικοινωνία του υπολογιστή με άλλους υπολογιστές και τον τελικό χρήστη.

Ο πυρήνας του συστήματος, το Linux, είναι ένα από τα πιο χαρακτηριστικά παραδείγματα Ελεύθερου Λογισμικού, αφού ο κώδικάς του μπορεί να αλλαχθεί, χρησιμοποιηθεί και μοιρασθεί στον οποιονδήποτε. Ο πυρήνας του Linux πρωτοεμφανίστηκε ως ερασιτεχνικό έργο το 1991 και ενσωματώθηκε γρήγορα στο έργο GNU, όπου αναζητούσε ένα πυρήνα για να δημιουργήσει ένα σύστημα τύπου Unix, έχοντας αναπτύξει ήδη πολλά από τα άλλα βασικά εργαλεία που χρειάζονταν.

Το Linux χρησιμοποιήθηκε και έγινε γνωστό στην αρχή ως λειτουργικό σύστημα για εξυπηρετητές και πολύ αργότερα, όταν το γραφικό του περιβάλλον και τα προγράμματα γραφείου αναπτύχθηκαν, άρχισε η εμφάνισή του και σε υπολογιστές γραφείου. Ο τεχνικός σχεδιασμός του πυρήνα, αλλά κυρίως το γεγονός ότι το Linux είναι Ελεύθερο Λογισμικό του επέτρεψε να αναπτυχθεί και στον τομέα των υπερυπολογιστών (supercomputer) και στον τομέα των προσωπικών υπολογιστών (PC).

Τα τελευταία χρόνια παρατηρείται μια ραγδαία αύξηση τόσο του ρυθμού ανάπτυξης, όσο και του ρυθμού χρήσης του Linux σε ενσωματωμένα συστήματα. Πολλά καταναλωτικά αγαθά υψηλής τεχνολογίας, όπως κινητά τηλέφωνα, προσωπικά ηλεκτρονικά ημερολόγια, δρομολογητές xDSL είναι στη πραγματικότητα ενσωματωμένα συστήματα που τρέχουν Linux. Το Linux έχει μεταφερθεί σε πολλούς επεξεργαστές για ενσωματωμένα συστήματα όπως η σειρά AVR32 [10] της Atmel, η σειρά XScale PXA [11] της Intel και πολλοί άλλοι, μεταξύ των οποίων και ο ETRAX της AXIS στον οποίο βασίζεται και η πλατφόρμα FOX που χρησιμοποιούμε.

Για την μεταφορά και την υποστήριξη του πυρήνα σε συστήματα με περιορισμένους πόρους έχουν αναπτυχθεί εναλλακτικές βιβλιοθήκες της πρότυπης βιβλιοθήκης της C, glibc όπως η uClibc και η dietlibc, ωστόσο στην εργασία μας χρησιμοποιήσαμε την glibc αφού μας το επέτρεπαν οι δυνατότητες της πλατφόρμας. Επίσης σχεδόν αναγκαίο συστατικό όλων των ενσωματωμένων συστημάτων που βασίζονται στο Linux είναι το λογισμικό BusyBox [17], στο οποίο έχουν ξαναγραφτεί πολλά βασικά προγράμματα με κατεύθυνση τα ενσωματωμένα συστήματα. Μέρος του λογισμικού BusyBox, όπως το κέλυφος εργασίας sh, ο διαχειριστής οθόνης screen, ο επεξεργαστής κειμένου vi, χρησιμοποιήθηκε για την ολοκλήρωση της εργασίας αυτής.

Το Linux ως βασική πλατφόρμα για την ανάπτυξη ενσωματωμένων συστημάτων και συγκεκριμένα στην εργασία μας είναι μια πολύ καλή επιλογή που υπακούει στα κριτήρια της διπλωματικής αυτής. Δηλαδή το Linux είναι Ελεύθερο Λογισμικό με μηδενικό κόστος και χωρίς δικαιώματα χρήσης, που εκμεταλλεύεται τις δυνατότητες των σύγχρονων ενσωματωμένων στο σύνολό τους και μας επιτρέπει τόσο να διευρύνουμε τις δυνατότητές μας και να παρέχουμε περισσότερες και πιο εξελιγμένες λειτουργίες, όσο και να αναπτύξουμε γρήγορα την απαιτούμενη εφαρμογή. Χαρακτηριστικά αναφέρουμε ότι χάρι στην επιλογή του Linux χρησιμοποιήσαμε εύκολα και χωρίς καμία δική μας παραμετροποίηση ή εφαρμογή, δικτύωση μέσω Ethernet, επικοινωνία με Bluetooth συσκευές, χρήση USB μνήμης ως αποθηκευτικό μέσο, χρήση μετατροπέων USB σε σειριακή θύρα, γλώσσες υψηλού επιπέδου για την ανάπτυξη των εφαρμογών μας όπως η Python που παρουσιάζουμε στην επόμενη παράγραφο και εργαλεία για τον έλεγχο των συσκευών X10 που παρουσιάζονται στη παράγραφο 1.6.3.

1.5.2 Γλώσσα προγραμματισμού Python

Η Python είναι μία υψηλού επιπέδου γλώσσα προγραμματισμού που δημιουργήθηκε το 1991 από τον Guido van Rossum. Η Python είναι σχεδιασμένη με έμφαση στην αναγνωσιμότητα και προσπαθεί να μειώσει το λόγο "φόρτος προγραμματιστή" / "φόρτος υπολογιστή". Η σύνταξή της είναι απλή και συνοδεύεται από μία πολύ πλούσια βιβλιοθήκη συναρτήσεων.

Η ανάπτυξη της γλώσσας γίνεται με έναν ανοιχτό, βασισμένο στην κοινότητά της, μοντέλο ανάπτυξης και συντονίζεται από το μη κερδοσκοπικό Ίδρυμα Λογισμικού Python (Python Software Foundation [16]). Αν και η γλώσσα έχει επίσημες τεχνικές προδιαγραφές και πρότυπα για πολλά μέρη της, δεν έχει προτυποποιηθεί στο σύνολό της. Η ανάπτυξη της Python ωστόσο

είναι ζωντανή και συστηματική εξασφαλίζοντας έτσι ένα σίγουρο μέλλον στη γλώσσα.

Η Python είναι Ελεύθερο Λογισμικό που διανέμεται από την άδεια χρήσης του Ιδρύματος Λογισμικού Python και υπάρχουν μεταφραστές τουλάχιστον για τις ακόλουθες πλατφόρμες: Linux, MacOs, Windows, Solaris, VMS.

Στα πλαίσια της διπλωματικής μεταφέραμε (porting) την τελευταία έκδοση του μεταφραστή της γλώσσας προγραμματισμού Python, έκδοση 2.5.1, στη νπλατφόρμα FOX. Η διαδικασία έγινε με τη μέθοδο του *cross compiling*, δηλαδή για να προσπεράσουμε το πρόβλημα της μειωμένης επεξεργαστικής ισχύος και των ελλιπών εργαλείων μεταγλώττισης στην ενσωματωμένη πλατφόρμα, ολοκληρώσαμε όλη τη διαδικασία της μεταγλώττισης σε ένα τρίτο μηχάνημα, άλλης αρχιτεκτονικής από το FOX, δημιουργώντας όμως εκτελέσιμα και βιβλιοθήκες για το FOX. Την ίδια διαδικασία ακολουθήσαμε και για τις παρακάτω βιβλιοθήκες της Python και τις όποιες εξαρτήσεις τους, που ήταν αναγκαίες για την ολοκλήρωση της εργασίας: libexpat, pyxml, twisted python, zlib, pybluez και dateutil.

Στο πακέτο της μεταγλωττισμένης Python για τη πλατφόρμα FOX, αρχιτεκτονική ETRAX, προστέθηκαν οι βιβλιοθήκες και το πρόγραμμα του κατανεμημένου συστήματος ελέγχου εκδόσεων (vcs) bazaar. Με τη χρήση του προγράμματος θα μπορούσαμε να ανανεώνουμε τα υπό ανάπτυξη προγράμματα με έναν εύκολο, πρακτικό, γρήγορο και σχετικά οικονομικό σε μέγεθος δεδομένων τρόπο, δυστυχώς όμως δεν το καταφέραμε εξαιτίας των τεχνικών περιορισμών της πλατφόρμας. Το πακέτο της Python με τις προαναφερθείσες βιβλιοθήκες και προγράμματα διανέμεται μέσω διαδικτύου και ήδη χρησιμοποιείται από μέρος προγραμματιστών που αναπτύσσουν για τη συγκεκριμένη αρχιτεκτονική.

1.5.3 Το πρόγραμμα επικοινωνίας heyu

Το πρόγραμμα HEYU είναι ένα πρόγραμμα τερματικού, χωρίς γραφικό περιβάλλον, για τον εξ' αποστάσεως έλεγχο συσκευών μέσω του δικτύου X10. Το HEYU μπορεί να συνεργαστεί με τις διεπαφές υπολογιστή με το δίκτυο X10 τύπου CM10A, CM12U, RFXCOM, MR26A, CM17 και τέλος με τη διεπαφή CM11, την οποία χρησιμοποιήσαμε και στην εργασία.

Πέρα από τις απλές εντολές ελέγχου όπως "άνοιγμα", "κλείσιμο" το πρόγραμμα υποστηρίζει μία πληθώρα διαφορετικών εντολών για πολλές συσκευές. Επίσης μπορεί να προγραμματίσει τη διεπαφή, για να εκτελεί διεργασίες σε κατάλληλο χρόνο και τέλος μπορεί να διαβάσει μηνύματα όπως την κατάσταση μιας συσκευής ή την τιμή ενός θερμοστάτη.

Τεχνικά το πρόγραμμα είναι χωρισμένο σε δύο μέρη: Τον εξυπηρετητή (deamon) και τον πελάτη. Ο εξυπηρετητής είναι συνδεδεμένος με μία ή περισσότερες διεπαφές X10 μέσω της θύρας USB ή της σειριακής του Η/Υ και ο πελάτης επικοινωνεί με τον εξυπηρετητή για την εκτέλεση κάθε εντολής. Ο σχεδιασμός αυτός αποδείχθηκε ότι παρέχει μια σταθερή ροή εντολών προς το δίκτυο X10 και λειτουργεί άψογα. Πέρα από τη σύνδεση της διεπαφής με τον υπολογιστή και τη ρύθμιση του προγράμματος για σύνδεση σε συγκεκριμένη σειριακή θύρα δεν χρειάστηκε καμία άλλη ιδιαίτερη διαδικασία. Μάλιστα το πρόγραμμα και στον επιτραπέζιο Η/Υ αλλά και στο ενσωματωμένο FOX λειτούργησε επιτυχώς με μετατροπή USB θύρας σε σειριακή.

Το πρόγραμμα διατίθεται υπό μια ιδιαίτερη άδεια χρήσης του δημιουργού του, η οποία δεν είναι στη λίστα της OSI αλλά μας επιτρέπει τη χρήση του προγράμματος για μη εμπορικούς σκοπούς. Παρέχεται μάλιστα για λειτουργικά συστήματα GNU/Linux, MacOS, *BSD, Solaris, ενώ καταφέραμε να το μεταγλωττίσουμε για την πλατφόρμα FOX από την οποία λειτούργησε χωρίς προβλήματα.

Listing 1: Παράδειγματα χρήσης του λογισμικού HEYU

```
1  bash# heyu on A1
   bash# heyu allon A
   bash# heyu off A1
   bash# heyu alloff B
   bash# heyu dim A3 10
```

2 Το πρωτόκολλο XMPP

Το e***X***tensible ***M***essaging and ***P***resence ***P***rotocol *(XMPP)* είναι ένα XML πρωτόκολλο για την μετάδοση μηνυμάτων και πληροφοριών παρουσίας σε πραγματικό χρόνο. Είναι ο πυρήνας του πρωτοκόλλου Jabber. Το πρωτόκολλο είναι δομημένο με τέτοιο τρόπο ώστε να είναι επεκτάσιμο και να υποστηρίζει χαρακτηριστικά όπως η μεταφορά αρχείων και φωνής (VoIP).

Σε αντίθεση με τα περισσότερα πρωτόκολλα ανταλλαγής μηνυμάτων το XMPP είναι βασισμένο σε ανοιχτά πρότυπα. Όπως και το ηλεκτρονικό ταχυδρομείο είναι ένα ανοιχτό σύστημα όπου ο κάθε κάτοχος ενός ονόματος domain και κατάλληλης σύνδεσης στο διαδίκτυο μπορεί να αποκτήσει τον δικό του jabber εξυπηρετητή και να ανταλλάξει μηνύματα με άλλους χρήστες. Υπάρχουν μάλιστα πολλές υλοποιήσεις τόσο εξυπηρετητών όσο και πελατών που διανέμονται κάτω από άδειες ελεύθερου και ανοιχτού κώδικα.

Το *XMPP Standarts Foundation* [42] είναι υπεύθυνο για την ανάπτυξη του πυρήνα και των επεκτάσεων του πρωτοκόλλου XMPP.

2.1 Ιστορία

Το εγχείρημα jabber ξεκίνησε το 1998 από τον Jeremie Miller, ενώ πρώτη δημόσια έκδοση του πρωτοκόλλου και του λογισμικού που το συνόδευε έγινε τον Μάιο του 2000. Το κύριο προϊόν της έκδοσης αυτής ήταν το εξυπηρετητής jabberd.

Αυτή η αρχική έκδοση του Jabber πρωτοκόλλου αποτέλεσε τη βάση για την ανάπτυξη του XMPP που εκδόθηκε στο RFC 3920 [2]. Συχνά το πρω-

[2]Το RFC ακρωνύμιο της φράσης "Request For Comments" είναι σημειώσεις σχετικές με έρευνες και τεχνολογίες του διαδικτύου. Ορισμένα RFC υιοθετούνται από το Internet Engineering Task Force (IETF) ως Διαδικτυακά Πρότυπα

τόκολλο Jabber αναφέρεται ως ανταγωνιστής ενός άλλου ανοιχτού πρωτοκόλλου ανταλλαγής μηνυμάτων, του SIMPLE [3].

Μέσα σε μόλις πέντε χρόνια από τη πρώτη εμφάνιση παρουσιάστηκαν τουλάχιστον έξι λογισμικά εξυπηρετητών για το πρωτόκολλο σε διάφορες γλώσσες προγραμματισμού και με διαφορετική σκοπιά ανάπτυξης. Σήμερα περισσότερα από είκοσι λογισμικά εξυπηρετητή, σαράντα λογισμικά πελατών και είκοσι βιβλιοθήκες προγραμματισμού υπάρχουν για το πρωτόκολλο. Υπό ανάπτυξη είναι και δεκάδες επεκτάσεις, ενώ ήδη έχουν ολοκληρώσει την διαδικασία συγγραφής και έγκρισης, επεκτάσεις όπως το Jabber-RPC [14] που επιτρέπουν τη κλήση συναρτήσεων από απόσταση.

2.2 Χαρακτηριστικά

Το XMPP έχει πέντε βασικά χαρακτηριστικά:

- **Αποκεντροποίηση**: Η αρχιτεκτονική δικτύου είναι ίδια με την αρχιτεκτονική του ηλεκτρονικού ταχυδρομείου. Ο καθένας μπορεί να έχει τον δικό του XMPP εξυπηρετητή και δεν υπάρχει κάποιος κεντρικός εξυπηρετητής.

- **Ανοιχτά Πρότυπα**: Το XMPP βασίζεται σε ανοιχτά πρότυπα που έχουν δημοσιευτεί στα RFC 3920 [12] και RFC 3921 [13]. Δεν χρειάζονται δικαιώματα για την χρήση των παραπάνω προτύπων, ούτε η ανάπτυξή τους επαφίεται σε κάποια εταιρία ή πωλητή.

- **Ιστορικό**: Το XMPP χρησιμοποιείται συστηματικά από το 1998. Έχουν υλοποιηθεί δεκάδες εξυπηρετητές, πελάτες, βιβλιοθήκες και επεκτάσεις. Τα τελευταία χρόνια υποστηρίζουν την ανάπτυξη και τη χρήση με-

[3] Το Session Initiation Protocol for Instant Messaging and Presence Leveraging Extensions (SIMPLE) είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων και πληροφοριών παρουσίας βασισμένο στο Session Initiation Protocol (SIP)

γάλες εταιρίες του χώρου όπως η Sun Microsystems και η Google. Χαρακτηριστικά αναφέρεται ότι προϊόν της τελευταίας "Google Talk" βασίζεται στην τεχνολογία XMPP.

- **Ασφάλεια**: Οι εξυπηρετητές δεν χρειάζεται να ανήκουν σε κάποιο δημόσιο XMPP δίκτυο, αλλά μπορούν να περιοριστούν για παράδειγμα στο εσωτερικό δίκτυο ενός κτηρίου ή οργανισμού. Επίσης τεχνολογίες κρυπτογράφησης και διαπίστευσης όπως οι SASL και TLS είναι ενσωματωμένα στο πρωτόκολλο. Τέλος μπορούν να χρησιμοποιηθούν επιπλέον επίπεδα (όπως το GPG) για την κρυπτογράφηση των μηνυμάτων μεταξύ των πελατών.

- **Ελαστικότητα**: Η χρήση του πρωτοκόλλου δεν επιβάλλει τη συνολική υλοποίησή του ούτε από τη μεριά του εξυπηρετητή, ούτε από τη μεριά του πελάτη. Χάρη σε αυτό το χαρακτηριστικό μπορούμε να υλοποιήσουμε τα κομμάτια του πρωτοκόλλου που μας εξυπηρετούν, βελτιστοποιώντας με αυτό τον τρόπο οικονομικά μεγέθη (χρόνος, χρήμα) αλλά και υπακούοντας σε τεχνικούς περιορισμούς, όπως η ενσωμάτωση του πρωτοκόλλου σε έναν μικροελεγκτή. Επιπλέον, νέες λειτουργικότητες μπορούν να δομηθούν πάνω από το XMPP. Ήδη έχουν εγκριθεί ως standart από το ίδρυμα του XMPP επεκτάσεις όπως το VoIP, Publish-Subscribe Υπηρεσίες, XMLRPC και SOAP over XMPP αλλά και πολλές άλλες.

2.3 Διευθυνσιοδότηση

Ένα XMPP δίκτυο είναι βασισμένο στη χρήση εξυπηρετητών, δηλαδή οι πελάτες δεν αποστέλλουν μηνύματα απευθείας ο ένας στον άλλο. Επιπλέον είναι αποκεντρωμένο από το σχεδιασμό του, δηλαδή ο καθένας μπορεί να

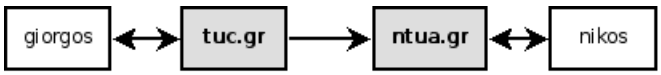
τρέχει το δικό του Jabber εξυπηρετητή, σε αντίθεση με τα περισσότερα δίκτυα ανταλλαγής μηνυμάτων.

Κάθε χρήστης του XMPP έχει ένα μοναδικό όνομα δικτύου που ονομάζεται Jabber ID (JID) και έχει την μορφή μιας διεύθυνσης ηλεκτρονικού ταχυδρομείου, δηλαδή `username@domain.com`. Το *username* είναι φυσικά το όνομα χρήστη και το *domain.com* το domain στο οποίο ανήκει αυτός ο χρήστης.

Επίσης κάθε χρήστης μπορεί να κάνει είσοδο ("logon") στο σύστημα από περισσότερες της μιας διαφορετικές τοποθεσίες, για παράδειγμα μπορεί να έχει συνδεδεμένο και τον προσωπικό του υπολογιστή αλλά και τον φορητό και το κινητό του τηλέφωνο, χρησιμοποιώντας *πόρους* (resources). Ο κάθε πόρος αντιπροσωπεύεται στην διεύθυνση JID ως εξής `username@domain.com/resource`. Έτσι ένας χρήστης μπορεί να έχει κάνει **ταυτόχρονα** είσοδο με τα ακόλουθα στοιχεία για τον προσωπικό του υπολογιστή, τον φορητό και το κινητό του αντίστοιχα: `giorgos@tuc.gr/PC`, `giorgos@tuc.gr/laptop`, `giorgos@tuc.gr/mobile`.

Όταν ο χρήστης `giorgos@tuc.gr` θέλει να στείλει ένα μήνυμα στο κινητό του χρήστη `nikos@ntua.gr` θα στείλει, χρησιμοποιώντας τον Jabber πελάτη, ένα μήνυμα στην διεύθυνση `nikos@ntua.gr/mobile` (με την προϋπόθεση ότι ο Νίκος έχει κάνει είσοδο στο ntua.gr με το κινητό του). Το μήνυμα θα το παραλάβει ο εξυπηρετητής του tuc.gr και θα το παραδώσει στον εξυπηρετητή του ntua.gr, ο οποίος με την σειρά του θα δρομολογήσει κατάλληλα το μήνυμα στο κινητό του Νίκου. Η διαδικασία είναι αντίστοιχη με αυτή του πρωτοκόλλου ηλεκτρονικής αλληλογραφίας (e-mail).

Επίσης μπορεί να συναντήσουμε μηνύματα με παραλήπτες χωρίς το όνομα χρήστη (JID: @tuc.gr) που τα χρησιμοποιούμε για ελέγξουμε χαρακτηριστικά του εξυπηρετητή ή για ανταλλαγή ειδικών μηνυμάτων.



Σχήμα 4: Ο χρήστης `giorgos@tuc.gr` στέλνει ένα μήνυμα στον χρήστη `nikos@ntua.gr`

2.4 Τύποι μηνυμάτων

Στο XMPP ορίζονται τρεις διαφορετικοί τύποι μηνυμάτων, οι οποίοι μπορούν να παραμετροποιηθούν και να επεκταθούν για να καλύψουν όλες τις απαιτήσεις των χρηστών. Τα μηνυμάτων μπορούν να είναι τύπου *presence*, *iq* ή *message*.

Συγκεκριμένα τα μηνυμάτα τύπου *presence* χρησιμοποιούνται για να διαδόσουν τη κατάσταση διαθεσιμότητας μιάς οντότητας Jabber δηλαδή ενός JID. Μια οντότητα μπορεί να είναι *διαθέσιμη (available)*, δηλαδή θα παραλάβει αμέσως όποιο μήνυμα στείλουμε ή μπορεί να είναι *μη διαθέσιμη (unavailable)*, που σημαίνει ότι ο χρήστης δεν είναι συνδεδεμένος με το δίκτυο. Επίσης ορίζονται και άλλοι τύποι διαθεσιμότητας όπως *do not distrub (dnd)*, *away*, *extended away (xa)*, *chat* και *normal*. Τέλος μηνύματα τύπου presence χρησιμοποιούνται για να δηλώσουμε την επιθυμία μας να γραφτούμε στη λίστα ενημέρωσης κατάστασης μιας οντότητας, να αποδεχθούμε τέτοιου τύπου αιτήσεις, ή απλά να ρωτήσουμε τη κατάσταση κάποιου χρήστη.

Τα μηνύματα τύπου *iq* παίρνουν το όνομα τους από τα αρχικά των λέξεων Info / Query, δηλαδή πληροφορία / ερώτημα. Τα μηνύματα αυτά χρησιμοποιούνται για να τη δημιουργία ενός δομημένου τρόπου αίτησης, αποστολής και λήψης πληροφορίας. Με τις παραμέτρους *get*, *set*, *result* και *error* μπορούμε να ανταλλάξουμε οποιαδήποτε πληροφορία μεταξύ οντοτήτων Jabber.

Τέλος τα μηνύματα τύπου *message* χρησιμοποιούνται για τη μεταφορά οποιασδήποτε πληροφορίας που δεν είναι πληροφορία κατάστασης ή δομημένη ανταλλαγή πληροφορίας. Φυσικά τα μηνύματα τύπου *message* είναι δομικός λίθος του πρωτοκόλλου, το οποίο είναι κατά κύριο λόγο ένα σύστημα ανταλλαγής μηνυμάτων μεταξύ των χρηστών. Ένα μήνυμα τύπου *message* μπορεί να πάρει διάφορα χαρακτηριστικά όπως *chat*, *groupchat*, *headline*, *normal* και *error* ανάλογα με τον τύπο επικοινωνίας που θέλουμε.

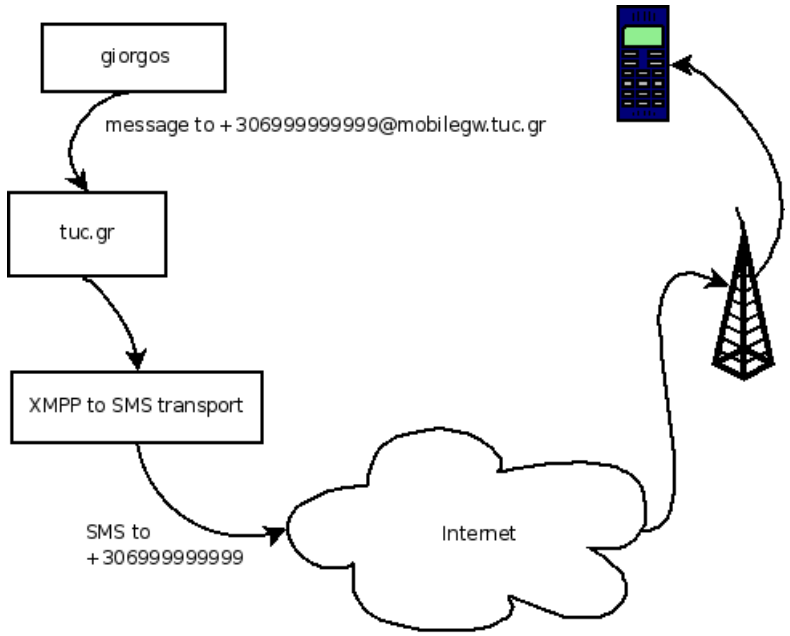
2.5 Διασύνδεση με άλλα πρωτόκολλα

Ένα άλλο ενδιαφέρον και χρήσιμο χαρακτηριστικό του πρωτοκόλλου XMPP είναι οι *μεταφορές (transports)*, επίσης γνωστές και ως *πύλες (gateways)*. Με τη χρήση των μεταφορών το XMPP μας επιτρέπει σε επίπεδο εξυπηρετητή τη διασύνδεση με άλλα δίκτυα είτε είναι δίκτυα ανταλλαγής μηνυμάτων είτε όχι.

Σήμερα υπάρχουν υλοποιημένες αρκετές μεταφορές με άλλα δίκτυα ανταλλαγής μηνυμάτων. Χαρακτηριστικό παράδειγμα είναι η μεταφορά για το ιδιόκτητο δίκτυο MSN. Ένας χρήστης ενός jabber εξυπηρετητή με τη συγκεκριμένη μεταφορά και ένα λογαριασμό στο δίκτυο MSN μπορεί, χωρίς επιπλέον λογισμικό από τη μεριά του να συνδεθεί στο δίκτυο MSN διαμέσω του δικτύου XMPP, να δει την κατάσταση των επαφών του σε αυτό και να ανταλλάξει μηνύματα. Όλη τη διαδικασία της σύνδεσης και της μετάφρασης των μηνυμάτων μεταξύ του δικτύου XMPP και του δικτύου MSN την αναλαμβάνει η μεταφορά. Αντίστοιχες υλοποιήσεις υπάρχουν και για άλλα δίκτυα ανταλλαγής μηνυμάτων.

Μεταφορές μπορούν να χρησιμοποιηθούν και για τη σύνδεση ενός δικτύου XMPP με δίκτυα που δεν είναι άμεσης ανταλλαγής μηνυμάτων, όπως για παράδειγμα το ηλεκτρονικό ταχυδρομείο. Ακολουθεί ένα παράδειγμα

αποστολής μηνήματος SMS από το δίκτυο XMPP με τη χρήση μιας μεταφορας.



Σχήμα 5: Ο χρήστης `giorgos@tuc.gr` στέλνει ένα μήνυμα στο κινητό του Νίκου το οποίο αναλαμβάνει να εξυπηρετήσει η μεταφορά XMPP to SMS

Ο χρήστης `giorgos@tuc.gr` θέλει να στείλει ένα Σύντομο Γραπτό Μήνυμα στο κινητό τηλέφωνο του Νίκου. Ο εξυπηρετητής του παρέχει αυτή τη δυνατότητα, δρομολογόντας κάθε μήνυμα για το domain mobilegw.tuc.gr στον μεταφορέα XMPP to SMS. Ο μεταφορέας θα μεταφράσει το μήνυμα XMPP σε μήνυμα για το κινητό με παραλήπτη, αποστολέα και κείμενο όλα σύμφωνα με το μήνυμα που έλαβε. Στη συνέχεια θα προωθήσει την αίτηση σε κάποια υπηρεσία αποστολής μηνυμάτων στο διαδίκτυο με τη χρήση HTTP (ή κάποιας άλλης μεθόδου που μας παρέχει η υπηρεσία όπως το XML-RPC). Τέλος η υπηρεσία θα το προωθήσει στον παροχέα κινητής τηλεφωνίας του Νίκου ο οποίος θα παραλάβει ένα μήνυμα από εμάς.

3 Εφαρμογές

3.1 Οι βιβλιοθήκες jClient και jComponent

Για την ανάπτυξη των εφαρμογών που αναφέρονται στις επόμενες παραγράφους του κεφαλαίου χρειάστηκε πρώτα να δημιουργήσουμε δύο βιβλιοθήκες, την jClient και τη jComponent. Οι βιβλιοθήκες θα υποστηρίξουν τη δημιουργία πελάτων του δικτύου Jabber και επεκτάσεων του εξυπηρετητή Jabber αντίστοιχα.

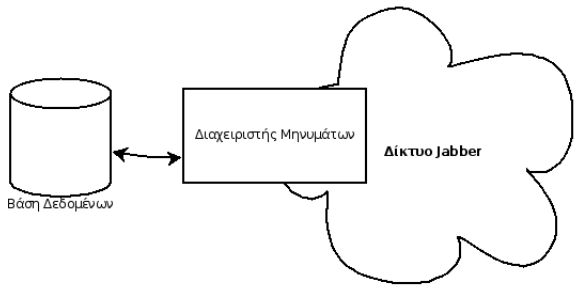
Η επικοινωνία με το πρωτόκολλο του Jabber δεν έγινε σε χαμηλό επίπεδο. Αντίθετα χρησιμοποιήσαμε για αυτό το σκοπό τη βιβλιοθήκη *Twisted Words* από το πακέτο βιβλιοθηκών *Twisted Python*, η οποία ανέλαβε το μέρος της επικοινωνίας με τον εξυπηρετητή. Η βιβλιοθήκες jClient και jComponent απλοποιούν ακόμη περισσότερο τη διαδικασία αρχικοποιώντας ρυθμίσεις και προσφέροντας έτοιμες συναρτήσεις για να απαντήσουμε ή να δημιουργήσουμε νέα μηνύματα, να αποδεχθούμε αυτόματα τις αιτήσεις παρακολούθησης κατάστασης (presence subscribe), καταγράφοντας όλες τις αποτυχημένες ενέργειες με τη χρήση του Logger της Python έτσι ώστε να έχουμε πλήρη έλεγχο του προγράμματος.

Θα πρέπει να σημειώσουμε ότι όπως το σύνολο των εφαρμογών που αναπτύξαμε έτσι και αυτές οι βασικές βιβλιοθήκες που αποτελούν πυρήνα των προγραμμάτων μας, είναι πλήρως και απόλυτα ασύγχρονες βελτιώνοντας έτσι τη χρήση των πόρων και τη συνολική εμπειρία χρήσης σε ένα περιβάλλον πολλών χρηστών. Για να το πετύχουμε αυτό χρησιμοποιούμε *παρατηρητές (observers)*. Οι παρατηρητές είναι αντικείμενα των βιβλιοθηκών Twisted Python και μας επιτρέπουν να παρακολουθούμε μία ροή δεδομένων, στη περίπτωσή μας τη σύνδεση με τον Jabber εξυπηρετητή, και ανάλογα με τα δεδομένα που λαμβάνουμε να τα διοχετεύουμε στις κατάλλη-

λες συναρτήσεις. Οι βιβλιοθήκες που φτιάξαμε διοχετεύουν τους τρεις τύπους μηνυμάτων *presence*, *iq* και *message* στις συναρτήσεις *presenceHandler*, *iqHandler* και *messageHandler* τις οποίες θα υπερφορτώσουμε στα προγράμματά μας για να εκτελούν συγκεκριμένες διεργασίες.

3.2 Πράκτορας διαχείρισης μηνυμάτων

Για την ολοκληρωμένη λειτουργία ένος έξυπνου κτηρίου και τον συντονισμό των συσκευών σύμφωνα με τις απαιτήσεις των χρηστών θα πρέπει να δημιουργήσουμε ένα λογισμικό που θα συνδυάζει τα ληφθέντα μηνύματα και αναλόγως θα τα εκτελεί την κατάλληλη χρονική στιγμή. Το λογισμικό αυτό είναι ο πράκτορας διαχείρισης μηνυμάτων.



Σχήμα 6: Ο διαχειριστής μηνυμάτων συνδέεται με μια βάση δεδομένων και με τις υπόλοιπες συσκευές μέσω το δικτύου Jabber

Ο πράκτορας διαχείρισης μηνυμάτων ή απλά διαχειριστής μηνυμάτων είναι ένας πελάτης στο δίκτυο Jabber και ταυτόχρονα ένα πελάτης σε μία βάση δεδομένων, συνδέσεις που του επιτρέπουν να κάνει ταυτοποίηση χρηστών, έγκριση, καταγραφή και χρονοπρογραμματισμό ενεργειών. Πιο αναλυτικά οι δυνατότητες του διαχεριστή μηνυμάτων είναι οι εξής:

- **Ταυτοποίηση των χρηστών:** Ο διαχειριστής μηνυμάτων μπορεί να

ταυτοποιήσει ένα χρήστη με JID του, με τη MAC διεύθυνση[4] μιας Bluetooth συσκευής ή από τον σειριακό αριθμό ενός USB Flash Drive. Την ταυτοποίηση θα την κάνει σύμφωνα με τα δεδομένα της βάσης στους πίνακες *users*, *usb2jid* (για αντιστοίχηση σειριακού αριθμού USB συσκευής) και *bt2jid* (για αντιστοίχηση MAC διεύθυνσης Bluetooth συσκευής). Η διαδικασία αυτή είναι χρήσιμη σε περιπτώσεις που θέλουμε για λογαριασμό ενός χρήστη να εκτελέσουμε μια εντολή, όταν ο τελευταίος είναι συνδεδεμένος με έμμεσο τρόπο στο δίκτυο και δεν γνωρίζουμε το JID του (π.χ. παρουσιάζεται στον ανιχνευτή Bluetooth).

- **Καταγραφή ενεργειών:** Ο διαχειριστής μηνυμάτων θα καταγράψει αναλυτικά στον πίνακα *log* της βάσης δεδομένων το JID του χρήστη, το JID της συσκευής που έλεγξε, τη δράση, την ώρα και το αποτέλεσμα. Ο διαχειριστής του συστήματος μπορεί με αυτό το αρχείο να βγάλει χρήσιμα συμπεράσματα και να εντοπίσει πιθανές δυσλειτουργίες.

- **Χρονοπρογραμματισμός ενεργειών:** Στο σύστημα που δημιουργήσαμε έχουμε τη δυνατότητα να προγραμματίσουμε την εκτέλεση μια εντολής στο μέλλον, με λεπτομέρεια ημέρας και ώρας. Εάν ο διαχειριστής μηνυμάτων εγκρίνει τη ζητούμενη πράξη θα την αποθηκεύσει σε κατάλληλες δομές και θα την εκτελέσει τη σωστή ώρα.

- **΄Εγκριση ενεργειών:** Κάθε συσκευή, άρα κάθε JID στο δίκτυό μας, έχει τη δυνατότητα να εκτελέσει ένα αριθμό ενεργειών που παραγράφονται στον πίνακα *actions* της βάσης δεδομένων. Ο διαχειριστής μηνυμάτων θα ελέγξει εάν ο αιτών μπορεί να εκτελέσει τη ζητηθείσα ενέργεια από τον πίνακα *priviledges* και αναλόγως θα την εγκρίνει ή θα την απορρίψει.

[4] Η Media Access Control address είναι ένας δεκαεξαδικός σειριακός αριθμός, μοναδικός για κάθε δικτυακή συσκευή.

Η βάση δεδομένων που δημιουργήσαμε είναι γραμμένη σε γλώσσα SQL και χρησιμοποιήσαμε το λογισμικό MySQL για τον εξυπηρετητή. Η σύνδεση στη βάση και όλα τα αιτήματα προς αυτή γίνονται με τρόπο ασύγχρονο. Η μέθοδος αυτή μας επιτρέπει την παράλληλη επεξεργασία πολλών αιτημάτων και την ομαλή συνεργασία του ασύγχρονου πρωτοκόλλου XMPP με τη σύγχρονη βάση δεδομένων. Η επιλογή της MySQL έγινε για να αναδείξουμε την δυνατότητα επέκτασης που μας δίνουν τα ανοιχτά πρότυπα και το Ελεύθερο Λογισμικό. Εάν το σύστημα που σχεδιάζουμε έχει μικρές απαιτήσεις σε παράλληλους χρήστες και διεργασίες μπορούμε να χρησιμοποιήσουμε άλλες μηχανές βάσεων δεδομένων, όπως για παράδειγμα η SQLite, που θα μπορούσαμε να την τοποθετήσουμε πλήρως σε μία ενσωματωμένη πλατφόρμα.

Η αποθήκευση και διαχείριση των μελλοντικών γεγονότων γίνεται, αντίθετα με τα υπόλοιπα δεδομένα, χρησιμοποιώντας τις δομές των λεξικών στην γλώσσα Python, με την ιδιαιτερότητα ότι τα αποθηκεύουμε στο δίσκο και όχι στη μνήμη[5]. Με τον τρόπο αυτόν καταναλώνουμε λιγότερους πόρους σε ένα ενσωματωμένο σύστημα, δεν χάνουμε τα μελλοντικά δεδομένα σε περίπτωση επανεκκίνησης και έχουμε πάντα τα δεδομένα μας σε μορφή εύκολη προς διαχείριση από τη γλώσσα προγραμματισμού.

Ο χρονοπρογραμματισμός των μηνυμάτων γίνεται με τη μέθοδο του *polling*, εξετάζοντας κάθε δευτερόλεπτο εάν υπάρχει κάποια εντολή προς εκτέλεση. Όλα τα γεγονότα που δεν έχουν ώρα εκτέλεσης, θα εκτελεστούν με την αλλαγή της μέρας, δηλαδή στις 12 το βράδυ, ενώ τα γεγονότα τα οποία δεν έχουν ημέρα εκτέλεσης θεωρούμε ότι αφορούν το σήμερα. Εάν ζητηθεί να εκτελεστεί ένα γεγονός σε παρωχημένη ώρα, τότε ο διαχειριστής θα το εκτελέσει άμεσα.

[5]Τη δυνατότητα αυτή μας την παρέχει η υπό-βιβλιοθήκη dirdbm του συνόλου βιβλιοθηκών Twisted Python

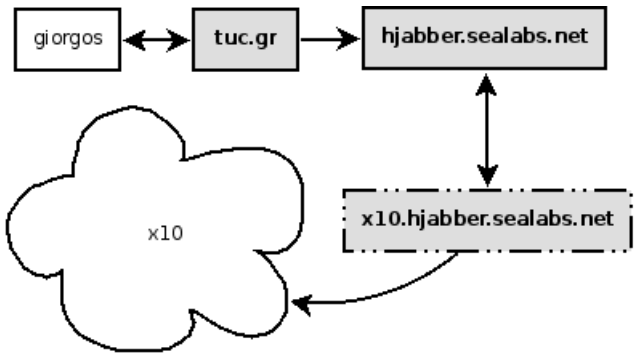
Η λειτουργία του διαχειριστή μηνυμάτων εξαρτάται από τα μηνύματα που δέχεται από τις υπόλοιπες συσκευές, οπότε αναλυτικότερα θα παρουσιαστεί η δράση του, καθώς και οι απαντήσεις στα μηνύματα που δέχεται στα κεφάλαια που ακολουθούν.

3.3 Γέφυρα επικοινωνίας X10 και XMPP

Η *γέφυρα*, ή αλλιώς *μεταφορά* (*transport*) επικοινωνίας μεταξύ του δικτύου X10 και του δικτύου XMPP είναι η υποδομή σε λογισμικό που μας επιτρέπει την παρακολούθηση της κατάστασης και τον έλεγχο συσκευών που υποστηρίζουν το πρωτόκολλο X10. Για τη δημιουργία της γέφυρας δεν κατασκευάσαμε έναν απλό Jabber πελάτη, σε αντίθεση με τις υπόλοιπες εφαρμογές, αλλά κατασκευάσαμε μία επέκταση για το λογισμικό του εξυπηρετητή Jabber.

Οι επεκτάσεις των εξυπηρετητών Jabber μπορούν να είναι δύο διαφορετικών ειδών. Ο πρώτος τύπος επέκτασης είναι άμεσα συνδεδεμένος με το λογισμικό του εξυπηρετητή για τον οποίο γράφουμε την επέκταση, καθώς χρησιμοποιεί ειδικές διεπαφές (API) του τελευταίου. Ο δεύτερος τρόπος είναι προτυποποιημένος από το ίδρυμα XMPP στο XEP-0114 [15] και ορίζει μία μέθοδο επικοινωνίας με τη χρήση sockets. Η χρήση αυτής της μεθόδου μας δίνει την ελευθερία να επιλέξουμε οποιοδήποτε λογισμικό εξυπηρετητή μας διευκολύνει χωρίς να μας περιορίζει σε κάποιο συγκεκριμένα, την ελευθερία επιλογής γλώσσας προγραμματισμού της επέκτασης, αφού το μόνο προαπαιτούμενο για τη γλώσσα είναι η υποστήριξη sockets και τέλος τη δυνατότητα η επέκταση να εκτελείται σε κάποιο απομακρυσμένο μηχάνημα, διαφορετικό από αυτό του εξυπηρετητή.

Η ανάπτυξη της επέκτασης στη παρούσα εργασία έγινε με την δεύτερη μεθοδολογία για τους λόγους που αναπτύχθηκαν νωρίτερα, στη γλώσσα



Σχήμα 7: Ο χρήστης `giorgos@tuc.gr` στέλνει ένα μήνυμα στον χρήστη `lamp@x10.hjabber.sealabs.net`, το οποίο επεξεργάζεται η επέκταση x10.hjabber.sealabs.net και εκτελεί τη κατάλληλη εντολή στο δίκτυο X10

προγραμματισμού Python, όπως και οι υπόλοιπες εφαρμογές. Λειτούργησε μάλιστα σε διαφορετική πλατφόρμα και υπολογιστή από τον κεντρικό Jabber εξυπηρετητή που χρησιμοποιήσαμε για τις δοκιμές μας. Συγκεκριμένα η επέκταση εκτελείται στην πλατφόρμα FOX και ο Jabber εξυπηρετητής σε τρίτο προσωπικό υπολογιστή.

Η αρχή λειτουργίας της επέκτασης είναι απλή και μοιάζει με την αρχή λειτουργίας άλλων επεκτάσεων των εξυπηρετητών Jabber, οι οποίες συνδέουν το XMPP με άλλα δίκτυα ανταλλαγής μηνυμάτων. Η επέκταση μετά τη σύνδεσή της με τον εξυπηρετητή δηλώνει ότι είναι υπεύθυνη για το υπό-domain **x10**.hjabber.sealabs.net και ο εξυπηρετητής αναλαμβάνει να δρομολογεί οποιοδήποτε μήνυμα γι'αυτό το υπό-domain στην επέκταση. Δηλαδή έστω ότι αποστέλεται το μήνυμα με παραλήπτη `lamp@x10.hjabber.sealabs.net` από τον χρήστη `giorgos@tuc.gr`, τότε ο Jabber εξυπηρετητής `tuc.gr` θα το παραδώσει στον Jabber εξυπηρετητή `hjabber.sealabs.net` και αυτός στη συνέχεια στην επέκταση `x10.hjabber.sealabs.net`.

Η επέκταση έχει τη δυνατότητα να αποστέλλει μηνύματα ως οποιοσδήποτε χρήστης του υπό-domain της. Εκμεταλλευόμενοι αυτή τη δυνατότητα,

κατασκευάσαμε τη επέκταση με τέτοιο τρόπο ώστε να επεξεργάζεται, να εκτελεί και να δηλώνει κατάσταση για πολλά διαφορετικά JID. Τα JID αυτά είναι εικονικά και αντιπροσωπεύουν τις διάφορες συσκευές X10 που έχουμε συνδεδεμένες στο δίκτυο. Δηλαδή στο παραπάνω παράδειγμα, και όπως φαίνεται από το Σχήμα 7, ο χρήστης *lamp* δεν υπάρχει και αντί γι'αυτόν το μήνυμα θα το επεξεργαστεί και θα το απαντήσει η επέκταση.

Για την εκτέλεση των X10 εντολών ο υπολογιστής που εκτελεί την επέκταση πρέπει να είναι συνδεδεμένος δια μέσου της σειριακής θύρας με τη συσκευή ελέγχου X10, στη περίπτωσή μας τη συσκευή CM11. Η επέκταση καλεί το πρόγραμμα HEYU με τις κατάλληλες παραμέτρους, για κάθε εντολή που λαμβάνει.

Επειδή οι X10 συσκευές που χρησιμοποιήσαμε δεν είχαν τη δυνατότητα αμφίδρομης επικοινωνίας, δεν μπορούμε να πληροφορηθούμε την κατάσταση αυτών με κάποια εντολή. Για να αντιμετωπίσουμε το πρόβλημα αυτό, αποθηκεύουμε στη μνήμη της επέκτασης την κατάσταση, αφού πρώτα τις κλείσουμε όλες ώστε να τις αρχικοποιήσουμε.

Listing 2: Αρχείο Καταγραφής X10 συσκευών

```
<x10>
        <device>
                <jid>lamp@x10.hjabber.sealabs.net</jid>
                <H>A</H>
                <A>1</A>
                <status>Off</status>
                <description></description>
                <twoway>False</twoway>
                <type>Lamp</type>
        </device>
        <device>
                <jid>coffemachine@x10.hjabber.sealabs.net</jid>
                <H>A</H>
                <A>2</A>
```

```
15                  <status>Off</status>
                    <description></description>
                    <twoway>False</twoway>
                    <type>Appliance</type>
            </device>
20  </x10>
```

Οι εικονικοί χρήστες τους οποίους αντιπροσωπεύει η επέκταση ορίζονται στο αρχείο *x10.xml*. Στο Listing 2 παρουσιάζεται το αρχείο που χρησιμοποιήσαμε στην εργασία μας. Το αρχείο είναι δομημένο με XML και για κάθε συσκευή πρέπει να οριστούν όλα τα πεδία εκτός από το πεδίο *description* το οποίο είναι προαιρετικό. Αναλυτικά το πεδίο *jid* δηλώνει το JID της εικονικής συσκευής, που πρέπει να είναι μοναδικό για κάθε συσκευή. Τα πεδία *H* και *A* δηλώνουν το *house code* και *unit code* αντίστοιχα για τη συσκευή X10 που θέλουμε να ελέγξουμε με αυτό το JID, το πεδίο *status* την κατάσταση της συσκευής, το πεδίο *twoway* παίρνει τη τιμή *True* εάν η συσκευή είναι αμφίδρομη και τέλος ο πεδίο *type* δηλώνει τον τύπου της συσκευής και άρα τις εντολές που μπορούμε να εκτελέσουμε σε αυτή[6]. Το αρχείο x10.xml διαβάζεται μία φορά κατά την εκτέλεση της επέκτασης και με βάση αυτό γίνονται οι κατάλληλες αρχικοποιήσεις στο λογισμικό.

Listing 3: Μήνυμα κειμένου από τον χρήστη `client@hjabber.sealabs.net` στον εικονικό χρήστη `lamp@x10.hjabber.sealabs.net`

```
<message type='chat' id='purple3da96ef2' from='client@hjabber.sealabs.net'
  to='lamp@x10.hjabber.sealabs.net'>
        <body>on</body>
</message>
```

[6]Ο πρωτόκολλο X10 ορίζει μερικές επιπλέον εντολές πέρα των βασικών *On* και *Off* για συγκεκριμένους τύπους συσκευών. Για παράδειγμα μια X10 συσκευή λαμπτήρα έχει τη δυνατότητα ελέγχου του ποσοστού φωτισμού.

Listing 4: Μήνυμα παρουσίας από τον εικονικό χρήστη `lamp@x10.hjabber.sealabs.net` στον χρήστη `client@hjabber.sealabs.net`

```
1 <presence to='client@hjabber.sealabs.net/Home'
    from='lamp@x10.hjabber.sealabs.net'>
      <show/>
      <status>On</status>
  </presence>
```

Μετά τις αρχικοποιήσεις η επέκταση είναι έτοιμη να λάβει μηνύματα από χρήστες. Για κάθε μήνυμα τύπου *message* που λαμβάνει εκτελεί με τη σειρά τις συναρτήσεις *translateCommand* και *requestAuth* και σε περίπτωση λάθους τη συνάρτηση *notAvailableCommand*.

Στη συνάρτηση *translateCommand* χρησιμοποιούμε ένα δυεπίπεδο αναλυτή κειμένου για να μετατρέψουμε την ανθρώπινη εντολή σε εντολή κατανοητή από τη μηχανή. Πρώτα εντοπίζουμε και υπολογίζουμε το χρόνο που μας ζητάτε να εκτελέσουμε την εντολή. Η χρήση της συνάρτησης *parser* από τη βιβλιοθήκη *dateutil* είναι υπεύθυνη για το αποτέλεσμα. Στη συνέχεια εντοπίζοντας λέξεις στην αγγλική και ανάλογα με την τωρινή κατάσταση της συσκευής που θέλουμε να επηρεάσουμε επιλέγεται μια εντολή. Για παράδειγμα εάν μια συσκευή είναι στη κατάσταση "Off" και στείλουμε μια εντολή που περιέχει τη λέξη "on" τότε η εντολή θα μεταφραστή σε "On" για X10. Εάν όμως η συσκευή είναι στη κατάσταση "Dimmed" τότε εάν έρθει μια εντολή "On" θα επιλέξουμε να αυξήσουμε στο μέγιστο τη φωτεινότητά της με την εντολή "brightb 1", καθώς η εκτέλεση της εντολής "on" δεν θα έφερνε το επιθυμητό αποτέλεσμα.

Η μεταφρασμένη εντολή δίνεται ως παράμετρος στη μορφή του λεξικού στη συνάρτηση *requestAuth*, η οποία θα τη στείλει στον διαχειριστή μηνυμάτων με ένα μήνυμα τύπου IQ, ο οποίος με τη σειρά θα ελέγξει εάν ο συγκεκριμένος χρήστης μπορεί να εκτελέσει την εντολή και εάν μπορεί, θα

την αποθηκεύσει στη μνήμη του μέχρι να έρθει η ώρα εκτέλεσης.

Τέλος η επέκταση επεξεργάζεται μηνύματα τύπου IQ από τον διαπιστευμένο διαχειριστή μηνυμάτων. Όταν ένα μήνυμα φτάσει από τον διαχειριστή μηνυμάτων, τότε εάν έχει τις κατάλληλες παραμέτρους θα εκτελεστούν οι συναρτήσεις *changeDeviceStatus* και *sendSignal*. Η πρώτη θα αποθηκεύσει τη νέα κατάσταση της συσκευής και θα ενημερώσει όσους χρήστες το έχουν ζητήσει. Η δεύτερη θα εκτελέσει το λογισμικό *HEYU* με τις κατάλληλες παραμέτρους.

Listing 5: Αίτηση καταχώρησης δράσης από την επέκταση x10 `x10.hjabber.sealabs.net` στον διαχειριστή μηνυμάτων `em@hjabber.sealabs.net`

```
<iq xmlns='jabber:client' to='em@hjabber.sealabs.net/em'
    from='x10.hjabber.sealabs.net' id='jchJNsO' type='get'>
        <query xmlns='ha:device'>
                <userId type='jid'>client@hjabber.sealabs.net</userId>
                <deviceId action='on'>lamp@x10.hjabber.sealabs.net
                        </deviceId>
                <time xmlns='ha:time'>
                        <year>2008</year>
                        <month>1</month>
                        <day>1</day>
                        <hour>0</hour>
                        <minute>0</minute>
                </time>
        </query>
</iq>
```

Listing 6: Μήνυμα εκτέλεσης εντολής από τον διαχεριιστή μηνυμάτων `em@hjabber.sealabs.net` προς την επέκταση `x10.hjabber.sealabs.net`

```
<iq xmlns='jabber:client' to='x10.hjabber.sealabs.net'
  from='em@hjabber.sealabs.net/em' id='jchJNsO' type='result'>
        <query xmlns='ha:device'>
```

```
                <userId type='jid'>client@hjabber.sealabs.net</userId>
 5              <deviceId action='on'>lamp@x10.hjabber.sealabs.net
                        </deviceId>
                <time xmlns='ha:time'>
                        <year>2008</year>
                        <month>1</month>
10                      <day>1</day>
                        <hour>0</hour>
                        <minute>0</minute>
                </time>
        </query>
15  </iq>
```

Όπως παρατηρούμε από τα Listing 5 και 6 το IQ μήνυμα δομείται με βάση ενός δικού μας namespace που ονομάζουμε *ha:device*. Το *ha:device* έχει πεδία *userId* με πιθανούς τύπους *jid* ή *bt* ή *usb*, *deviceId* με εντολή και χρόνο εκτέλεσης.

Η λειτουργία του προγράμματος είναι σχεδιασμένη με τέτοιο τρόπο που η παρουσία του *διαχειριστή μηνυμάτων* είναι αναγκαία. Σημειώνουμε ότι την τελική εντολή που θα εκτελεστεί τη στέλνει ο διαχειριστής στην επέκταση και εάν δεν υπάρξει αυτή η διαδικασία η επέκταση δεν θα αλλάξει τη κατάσταση καμίας συσκευής όσα γραπτά μηνύματα και να λάβει. Η επιλογή εμπλοκής του διαχειριστή μηνυμάτων έγινε για να υπάρχει επίπεδο ελέγχου για το ποιός χρήστης μπορεί να εκτελέσει ποιά εντολή με κεντρικό χρονοπρογραμματισμό και καταγραφή όλων των δράσεων. Φυσικά σε μία απλούστερη εφαρμογή που δεν υπάρχουν επίπεδα ασφαλείας, η λειτουργίες του διαχειριστή μηνυμάτων θα μπορούσαν πολύ εύκολα να ενσωματωθούν στην επέκταση.

3.4 Ηλεκτρονική κλειδαριά με χρήση USB

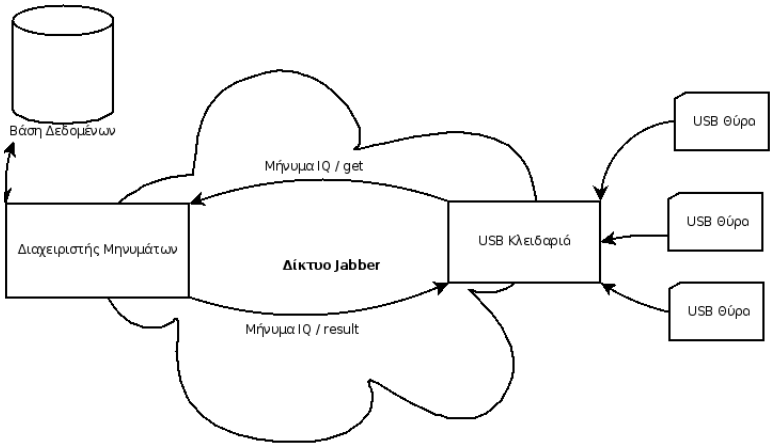
Ένα ολοκληρωμένο σύστημα αυτοματισμού θα πρέπει να έχει οπωσδήποτε ένα σύστημα ηλεκτρονικού έλεγχου πρόσβασης σε χώρους ή και υπολογιστές, το οποίο θα πρέπει να μπορεί να συνδυαστεί με τα υπόλοιπα αυτοματοποιημένα συστήματα του κτηρίου. Κατασκευάσαμε λοιπόν μια ηλεκτρονική κλειδαριά, ακολουθώντας τις ίδιες πρακτικές και λογική όπως και στις υπόλοιπες εφαρμογές.

Η επιλογή του ηλεκτρονικού κλειδιού έγινε με βάση το κόστος απόκτησης, την ευκολία μεταφοράς και διαχείρισης των κλειδιών. Με βάση αυτά τα κριτήρια επιλέχθηκε για ηλεκτρονικό κλειδί η χρήση ενός USB μέσου αποθήκευσης τύπου flash (USB flash drive).

Οι μνήμες τύπου flash έχουν αναπτυχθεί σε τέτοιο βαθμό ώστε να μας επιτρέπουν να συνδυάζουμε πολύ μεγάλους αποθηκευτικούς χώρους, της τάξης των Gb, στο μισό μέγεθος ενός σπιρτόκουτου και με κόστος μόλις λίγων ευρώ. Επίσης κάθε μνήμη έχει ένα δικό της σειριακό αριθμό που την κάνει ξεχωριστή και άρα μπορεί να χρησιμοποιηθεί ως κλειδί. Τέλος ο καθένας έχει τουλάχιστον μία μνήμη τύπου flash συνήθως μαζί με τα κλειδιά του, άρα το κόστος περιορίζεται στη κατασκευή της υποδομής του κτηρίου, ενώ δεν υπάρχει κόστος εκπαίδευσης ή αλλαγής συνηθειών των χρηστών.

Σύμφωνα με τα παραπάνω κατασκευάσαμε έναν Jabber πελάτη, ο οποίος συνδέεται στο δίκτυο και στη συνέχεια παρακολουθεί κάποια ορισμένη, ή το σύνολο των θυρών USB για νέες συσκευές. Η παρακολούθηση και η επικοινωνία με τις USB συσκευές γίνεται με τη χρήση του λογισμικού HAL και τις κατάλληλες βιβλιοθήκες της Python με την τεχνική των callbacks. Με αυτή τη μέθοδο το πρόγραμμά μας δεν μπλοκάρει τη λειτουργία του περιμένοντας την είσοδο κάποιας συσκευής και συνεχίζει να επικοινωνεί κανο-

νικά και με το δίκτυο Jabber, αναφέροντας κατάσταση ή οποιαδήποτε άλλη πληροφορία μπορεί να του ζητηθεί.



Σχήμα 8: Η USB κλειδαριά επικοινωνεί με τις θύρες USB και στέλνει αιτήσεις διαπίστευσης στον Διαχειριστή Μηνυμάτων, ο οποίος με τη σειρά του επικοινωνεί με τη Βάση Δεδομένων για να ανακτήσει τις κατάλληλες πληροφορίες και σύμφωνα με αυτές απαντά στη USB κλειδαριά.

Κατά την είσοδο μας συσκευής USB καλείται η συνάρτηση *deviceAdded* η οποία εφόσον η εισαχθείσα συσκευή είναι συσκευή αποθήκευσης και έχει σειριακό αριθμό καλεί τη συνάρτηση *requestAuth*. Η *requestAuth* θα στείλει ένα κατάλληλο μήνυμα τύπου IQ / get (Listing 7) προς τον διαχειριστή μηνυμάτων ο οποίος θα απαντήσει με ένα μήνυμα IQ / result (Listing 8 και 9) το οποίο θα επιτρέψει ή όχι την πρόσβαση στο χώρο. Το πρόγραμμά είναι ρυθμισμένο για να επιτρέπει πρόσβαση σε κλειδωμένους υπολογιστές και ανάλογα με το μήνυμα που θα λάβει από τον διαχειριστή μηνυμάτων θα ξεκλειδώσει ή όχι το κλειδωμένο μηχάνημα στο οποίο εκτελείται.

Από τη πλευρά του διαχειριστή μηνυμάτων ακολουθείται η παρακάτω διαδικασία. Πρώτα μεταφράζεται ο σειριακός αριθμός του USB κλειδιού σε JID με τη συνάρτηση *usb2jid*. Στη συνέχεια θα ακολουθήσει τη γενική διαδικασία ελέγχου δικαιωμάτων και θα απαντήσει άμεσα στο μήνυμα που έλαβε

με ένα μήνυμα έγγρισης ή απόρριψης του αιτήματος. Τέλος θα καταγράψει την έκβαση του αποτελέσματος στη βάση δεδομένων.

Listing 7: IQ μήνυμα από τη κλειδαριά στον διαχειριστή μηνυμάτων

```
<iq xmlns='jabber:client' from='door@hjabber.sealabs.net/usb' type='get'
 id='jcFmsiV' to='em@hjabber.sealabs.net/em'>
<query xmlns='ha:device'>
   <userId type='usb'>Kingston_DataTravelerMini_5B7303770601 0:0</userId>
   <deviceId action='open'>door@hjabber.sealabs.net</deviceId>
</query>
</iq>
```

Listing 8: IQ μήνυμα θετικής απάντησης από τον διαχειριστή μηνυμάτων στη κλειδαριά

```
<iq xmlns='jabber:client' to='door@hjabber.sealabs.net/usb'
 from='em@hjabber.sealabs.net/em' id='jcFmsiV' type='result'>
<query xmlns='ha:device'>
   <userId type='usb'>Kingston_DataTravelerMini_5B7303770601 0:0</userId>
   <deviceId action='open'>door@hjabber.sealabs.net</deviceId>
</query>
</iq>
```

Listing 9: IQ μήνυμα αρνητικής απάντησης από τον διαχειριστή μηνυμάτων στη κλειδαριά

```
<iq xmlns='jabber:client' to='door@hjabber.sealabs.net/usb'
 from='em@hjabber.sealabs.net/em' id='jcFmsiV' type='result'>
<query xmlns='ha:device'>
   <userId type='usb'>Kingston_DataTravelerMini_5B7303770601 0:0</userId>
   <deviceId action='open'>door@hjabber.sealabs.net</deviceId>
</query>
<error>Access Denied</error>
</iq>
```

Η χρήση του σειριακού αριθμού μιας συσκευής ως μοναδικό κριτήριο διαπίστευσης ενός χρήστη, καθιστά την διαδικασία ευάλωτη σε οποιονδήποτε με υψηλό βαθμό γνώσεων ηλεκτρονικών. Για το λόγο αυτό το σύστημα μπορεί εύκολα να επεκταθεί και πέρα από τον σειριακό αριθμό της συσκευής να διαβάζει και ειδικά δεδομένα από τη συσκευή. Η χρήση τεχνικών όπως η OTP (One Time Password) μπορεί να αυξήσει τα επίπεδα ασφαλείας, ακόμη και εάν η μνήμη USB κλαπεί και τα δεδομένα ασφαλείας αντιγραφούν. Τέλος το λογισμικό θα μπορούσε να επεκταθεί κατάλληλα ώστε να απαιτεί δύο επίπεδα διαπίστευσης, κάτι που ο χρήστης έχει (μία USB μνήμη) και κάτι που ο χρήσης γνωρίζει (ένα προσωπικό κωδικό).

3.5 Αναγνώριση παρεβρισκομένων σε χώρο με χρήση Bluetooth

Η αναγνώριση παρευρισκομένων σε χώρο γίνεται συνήθως με τη χρήση φωτοαισθητήρων, οι οποίοι με χαμηλό κόστος μπορούν να μας δηλώσουν την κίνηση ατόμων σε ένα χώρο και να χρησιμοποιηθούν για ασφάλεια ή για απλούς αυτοματισμούς, όπως ο έλεγχος των λαμπτήρων. Στην εργασία αυτή κατασκευάσαμε έναν αναγνωριστή παρευρισκομένων με τη χρήση του πρωτοκόλλου Bluetooth αυξάνοντας κατά πολύ τις δυνατότητες αυτοματισμού.

Ο πρωτόκολλο Bluetooth είναι ένα σύγχρονο πρωτόκολλο ασύρματης επικοινωνίας συσκευών, που χρησιμοποιείται κατά κόρον στα κινητά τηλέφωνα και άλλες μικροσυσκευές για την ανταλλαγή αρχείων και τη μεταφορά φωνής. Κάθε συσκευή Bluetooth έχει μία μοναδική διεύθυνση MAC, κατά αντιστοιχία της μοναδικής διεύθυνσης MAC που έχει μια κάρτα δικτύου σε ένα δίκτυο τύπου Ethernet. Όπως και στη περίπτωση της USB μνήμης έτσι και σε αυτή τη περίπτωση η διασύνδεση με το δίκτυο Bluetooth είναι ένα χαρακτηριστικό όλων των σύγχρονων τηλεφώνων και δεν χρειάζεται

κανένα ιδιαίτερο χειρισμό από τον χρήστη, απλά να έχει μαζί του το κινητό του τηλέφωνο.

Η εφαρμογή που αναπτύξαμε έχει δύο σκέλη: Πρώτον το κομμάτι της ανίχνευσης Bluetooth συσκευών στο χώρο και δεύτερον την αναφορά στο δίκτυο Jabber. Για το πρώτο κομμάτι χρησιμοποιήσαμε μία Bluetooth συσκευή πρώτης τάξης που μας δίνει μία εμβέλεια 100 μέτρων, η οποία καλύπτει ένα πολύ μεγάλο χώρο με έναν μόλις δέκτη. Η χρήση άλλης τάξης δεκτών (για παράδειγμα η δεύτερη τάξη μας δίνει εμβέλεια 10 μέτρα) μπορεί να επιλεγεί για τον περιορισμό του δέκτη σε συσκευές εντός ενός μικρότερου χώρου. Η σύνδεση της Bluetooth συσκευής έγινε με θύρα USB και στη συνέχεια χρησιμοποιήθηκε το stack του πυρήνα του Linux και η βιβλιοθήκη PyBluez της Python για την επικοινωνία με αυτή.

Η διαδικασία ανίχνευσης συσκευών είναι μία χρονοβόρα διαδικασία η οποία μπλοκάρει η διεργασία που την εκτελεί. Γι' αυτό το λόγο χρησιμοποιήσαμε τη τεχνική των Deferreds που μας παρέχει το σύνολο βιβλιοθηκών Twisted Python για να την τρέξουμε σε νήμα και να συλλέξουμε την παρεχόμενη πληροφορία ασύγχρονα, χωρίς να μπλοκάρουμε την αρχική διεργασία. Ένα τυπικό "πέρασμα" του ανιχνευτή διαρκεί περίπου 10 δευτερόλεπτα, χωρίς την ανάγνωση των ονομάτων των Bluetooth συσκευών που θα το έκανε περισσότερο χρονοβόρο, γεγονός που θα καθιστούσε τη λειτουργία του προγράμματος με το δίκτυο Jabber αδύνατη, εάν δεν ακολουθούσαμε την προαναφερθείσα μεθοδολογία.

Το δεύτερο κομμάτι της εφαρμογής αφορά την επικοινωνία με το δίκτυο Jabber. Για τη δήλωση των ευρεθέντων συσκευών Bluetooth θα επεκτείνουμε τα μηνύματα παρουσίας *presence* του ίδιου του πρωτοκόλλου XMPP. Υπενθυμίζουμε ότι τα μηνύματα παρουσίας χρησιμοποιούνται για τη δήλωση της παρουσίας ενός full JID και παίρνει τιμές όπως "away", "dnd" κ.α.

Επίσης τις καταστάσεις αυτές μπορούμε να τις συνοδεύσουμε με προσωποποιημένα μηνύματα.

Σύμφωνα με τα παραπάνω, το πρόγραμμά μας συνδέεται στο Jabber δίκτυο που το ορίζουμε και στη συνέχεια αρχίζει ένα αέναο κύκλο ανίχνευσης Bluetooth συσκευών. Μετά το τέλος κάθε ανίχνευσης εάν δεν έχει βρει συσκευές θα στείλει ένα μήνυμα τύπου *presence* με κατάσταση *unavailable* ώστε να δηλώσει τη απουσία ατόμων. (Listing 10)

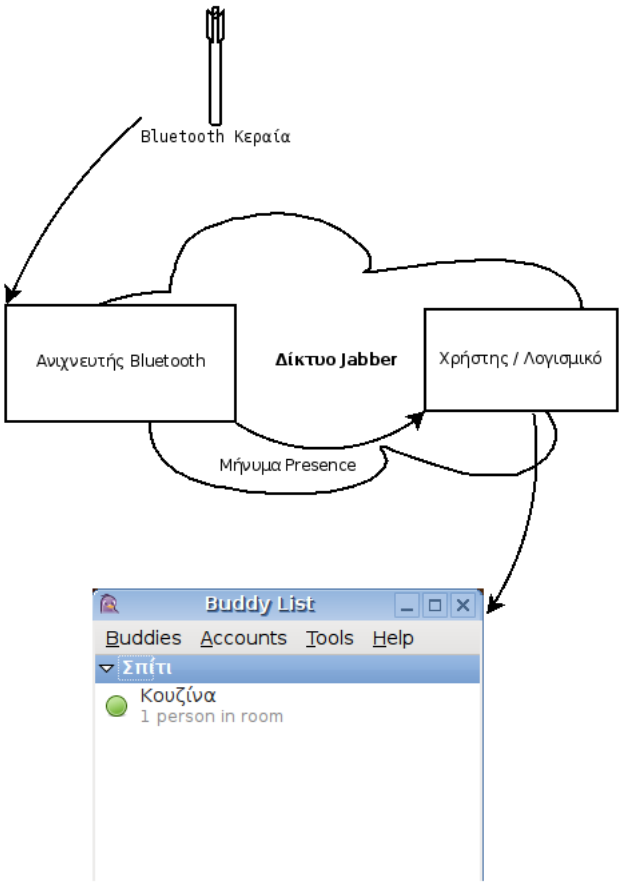
Listing 10: Presence μήνυμα απουσίας ατόμων από τον ανηχνευτή με JID `kitchen@hjabber.sealabs.net` στον χρήστη με JID `client@hjabber.sealabs.net`

```
<presence to='client@hjabber.sealabs.net/Home'
        from='kitchen@hjabber.sealabs.net/bt' type='unavailable'/>
```

Εάν αντιθέτως βρεθούν συσκευές θα στείλει ένα μήνυμα τύπου *presence* με κατάσταση *available* και συνοδευτικό κείμενο που δηλώνει των αριθμών των ευρεθέντων συσκευών, άρα των ατόμων σε ένα δωμάτιο. Σε αυτή τη περίπτωση επεκτείνουμε τον τύπου *presence* και προσθέτουμε ένα στοιχείο *devices* μέσα στο οποίο καταγράφονται όλες οι MAC διευθύνσεις των ευρεθέντων συσκευών. Επίσης προσθέτουμε και ένα στοιχείο *persons* στο οποίο καταγράφουμε των αριθμό των ατόμων σε μορφή κατανοητή από λογισμικό.

Listing 11: Presence μήνυμα παρουσίας ενός ατόμου με MAC διεύθυνση 00:00:11:11:22:22 από τον ανηχνευτή με JID `kitchen@hjabber.sealabs.net` στον χρήστη με JID `client@hjabber.sealabs.net`

```
<presence to='client@hjabber.sealabs.net/Home'
        from='kitchen@hjabber.sealabs.net/usb'>
        <show/>
```



Σχήμα 9: Ο ανιχνευτής Bluetooth επικοινωνεί με τη κεραία και βρίσκει τις συσκευές Bluetooth στο χώρο. Στη συνέχεια με τη χρήση του δικτύου Jabber της ανακοινώνει στον χρήστη που έχει εγγραφεί στη λίστα ανακοινώσεων κατάστασης του ανιχνευτή. Ο χρήστης μπορεί άμεσα να ενημερωθεί για τη κατάσταση ενός χώρου από οποιονδήποτε λογισμικό Jabber πελάτη.

```
        <status>1 person in room</status>
        <persons xmlns='ha:presence'>1</persons>
        <devices xmlns='ha:presence'>
7               <device>00:00:11:11:22:22</device>
        </devices>
    </presence>
```

Χρησιμοποιώντας τη πληροφορία που μας παρέχει αυτός ο ανιχνευτής μπορούμε να ελέγξουμε τις περιβαλοντικές συνθήκες και τις συσκευές ενός χώρου με ιδιαίτερα εξελιγμένο τρόπο. Για παράδειγμα ο διαχειριστής μηνυμάτων θα μπορούσε να διαβάζει από τη βάση δεδομένων προτιμήσεις για κάθε άτομο που βρίσκεται σε ένα δωμάτιο. Με αυτό τον τρόπο θα μπορούσαμε να ελέγχουμε τη θερμοκρασία ενός δωματίου όχι μόνο με βάση των αριθμό των ατόμων αλλά και σύμφωνα με τις προτιμήσεις τους. Επίσης θα μπορούσαμε να παραμετροποιήσουμε τη λειτουργία των συσκευών, όπως για παράδειγμα να ανοίγουμε τα κατάλληλα φωτιστικά και τον υπολογιστή, κάνοντας ταυτόχρονα είσοδο στο δίκτυο με το κατάλληλο όνομα χρήστη που αντιστοιχεί στη συσκευή Bluetooth που ανακαλύψαμε στο χώρο και τέλος να αναδρομολογήσουμε όλες τις κλήσεις που αφορούν το συγκεκριμένο άτομο στο δωμάτιο που βρίσκεται κάθε στιγμή.

4 Εφαρμογή σε πραγματικές συνθήκες και συμπεράσματα

4.1 Ο συνδιασμός των εφαρμογών σε κτήριο

Το σύστημα αυτοματισμού βασισμένο στο XMPP και το σύνολο των εφαρμογών που αναπτύξαμε, λειτούργησαν για ένα μήνα, ελέγχοντας δύο ηλεκτρικές συσκευές και έναν ηλεκτρονικό υπολογιστή χωρίς να παρουσιαστεί κανένα πρόβλημα.

Συγκεκριμένα η γέφυρα επικοινωνίας X10 και XMPP όπως και ο αναγνωριστής παρεβρισκομένων με τη χρήση Bluetooth, εκτελέστηκαν στη πλατφόρμα FOX ταυτόχρονα. Η ηλεκτρονική κλειδαριά έτρεχε σε έναν ηλεκτρονικό υπολογιστή και έλεγχε τη χρήση του. Στον ίδιο υπολογιστή έτρεχαν επίσης ο διαχειριστής μηνυμάτων και η βάση δεδομένων που αποθήκευε την απαιτούμενη πληροφορία.

Όλες οι εφαρμογές και επιπλέον δύο φυσικοί χρήστες που συνδέονταν στο δίκτυο μέσω πελατών Jabber, συγκεκριμένα των προγραμμάτων Pidgin και Psi, συνδέονταν στον εξυπηρετητή Jabberd 2.x που εγκαταστήσαμε γι' αυτό το σκοπό, σε εσωτερικό δίκτυο TCP/IP.

4.2 Η χρήση Ελεύθερου Λογισμικού / Λογισμικού Ανοιχτού Κώδικα

Σε όλες τις πτυχές της εργασίας αυτής χρησιμοποιήθηκε με απόλυτη επιτυχία Ελεύθερο Λογισμικό / Λογισμικό Ανοιχτού Κώδικα, εκτός από τη περίπτωση του προγράμματος HEYU, το οποίο διαθέτει τον κώδικά του υπό ένα ιδιαίτερο καθεστώς.

Η γλώσσα προγραμματισμού Python μας έδωσε τη δυνατότητα να αναπτύξουμε πρωτότυπες εφαρμογές συνδυάζοντας πολλές διαφορετικές

τεχνολογίες, σε μία ενσωματωμένη πλατφόρμα. Η ελεύθερη διανομή του κώδικά της επέτρεψε τη μεταφορά της γλώσσας και βιβλιοθηκών της σε μία πλατφόρμα για την οποία δεν υπήρχε υποστήριξη νωρίτερα.

Το λειτουργικό σύστημα GNU/Linux συνδύασε με χαρακτηριστική σταθερότητα και ευκολία όλα τα υποσυστήματα της πλατφόρμας, όπως δίκτυο Ethernet, δίκτυο Bluetooth, USB αποθηκευτικός χώρος, σειριακή θύρα και επιπλέον μας δίνει τη δυνατότητα να εκτελούμε πολλές διεργασίες παράλληλα. Εκτελέσαμε σε μία μόνο συσκευή την μεταφορά μεταξύ του δικτύου XMPP και X10 και τον ανιχνευτή συσκευών Bluetooth, λειτουργώντας και τα δύο παράλληλα χωρίς να παρουσιάζεται το παραμικρό πρόβλημα.

Τέλος το πρόγραμμα HEYU παρόλο που δεν ακολουθεί κάποια από τις αναγνωρισμένες άδειες χρήσεις του OSI, επιτρέπει τη διανομή και βελτίωση του κώδικα, οπότε μας δίνει τη δυνατότητα να βελτιώσουμε την ήδη πολύ καλή λειτουργία του.

Συμπερασματικά το ΕΛ/ΛΑΚ κάλυψε συνολικά τις ανάγκες της εργασίας και παράλληλα μας δίνει τη δυνατότητα για περαιτέρω ανάπτυξη αυτής χωρίς κόστος, δικαιώματα και περιορισμούς από πατέντες.

4.3 Η χρήση των πρωτοκόλλων XMPP και X10

Το πρωτόκολλο X10 λειτούργησε σε γενικές γραμμές ικανοποιητικά. Παρουσίασε ωστόσο προβλήματα μη ανταπόκρισης των συσκευών σε εντολές, πιθανώς εξαιτίας θορύβου στη γραμμή ηλεκτρισμού, τα οποία διορθώθηκαν αργότερα χωρίς δικιά μας παρέμβαση. Σημαντικό μειονέκτημα των συσκευών ήταν η αδυναμία αποστολής αναφοράς κατάστασης, που περιορίζει σημαντικά τις δυνατότητες διαχείρισής τους ή μπορεί να αποσυντονίσει το σύστημα εάν συνδυαστεί με το προηγούμενο πρόβλημα του θορύβου.

Το πρωτόκολλο XMPP ήταν στο σύνολό του άριστο, δίνοντας απεριόριστη ελευθερία στον προγραμματιστή. Το ίδρυμα XMPP παρέχοντας πλήρη τεκμηρίωση για τον πυρήνα και για τις επεκτάσεις του μας επέτρεψε να κατασκευάσουμε το σύστημα αυτό και να αποδείξουμε ότι το πρωτόκολλο XMPP μπορεί να χρησιμοποιηθεί για τη κατασκευή συστημάτων αυτοματισμού και για τον απομακρυσμένο έλεγχο συσκευών. Από την εμπειρία που έχουμε από τη χρήση του πρωτοκόλλου σε δίκτυα ανταλλαγής μηνυμάτων μεταξύ ανθρώπων γνωρίζουμε ότι ένα δίκτυο XMPP μπορεί να εξυπηρετήσει πολλές χιλιάδες χρήστες, γεγονός που καθιστά την υλοποίηση κατάλληλη ακόμη και για μεγάλες εγκαταστάσεις.

Στο Διαδίκτυο μπορεί να βρει κανείς μερικές υλοποιήσεις γεφυρών μεταξύ του δικτύου XMPP και δικτύων αυτοματισμών, συνήθως του X10 αλλά υπάρχουν και αναφορές για το δίκτυο xAP [41]. Από αυτές ξεχωρίζει η υλοποίηση του Simon Aurell [44] ο οποίος πραγματοποίησε τη διασύνδεση των πρωτοκόλλων χρησιμοποιώντας ένα πελάτη για κάθε συσκευή και όχι μία μεταφορά όπως στη περίπτωσή μας. Δεν μπορούσαμε όμως να συγκρίνουμε τα αποτελέσματα της υλοποίησης αυτής με την εργασία γιατί δεν υπάρχει διαθέσιμη στο διαδίκτυο.

Σε άλλες υλοποιήσεις [45] απλά δημιουργείται μια σύνδεση με το δίκτυο XMPP για όλες τις υπό έλεγχο συσκευές, η οποία λειτουργεί ο ”δρόμος διαφυγής” (gateway). Η υλοποίηση αυτή δεν εκμεταλλεύεται τις δυνατότητες του δικτύου για ενημερώσεις παρουσίας (presence notifications), δεν οπτικοποιεί την κατάσταση των συσκευών και για τη χρήση της θα πρέπει να θυμόμαστε τα κωδικά ονόματα συσκευών για να δώσουμε τις εντολές.

5 Μελλοντικές Επεκτάσεις

5.1 Δημιουργία συσκευών που να υποστηρίζουν εγγενώς το πρωτόκολλο

Στη σημερινή αγορά αυτοματισμού υπάρχουν είτε συσκευές που υποστηρίζουν ένα από τα συστήματα ελέγχου, είτε πρίζες που προσαρμόζονται πριν από την υπό έλεγχο συσκευή, είτε συσκευές υπερύθρων που μπορούν να ελέγξουν μια συσκευή με αντίστοιχο δέκτη. Μονοπωλιακά συμφέροντα και πατέντες αυξάνουν δραματικά το κόστος μιας τέτοιας συσκευής αλλά και ταυτόχρονα την περιορίζουν αισθητά σε επίπεδο δυνατοτήτων. Έτσι μια συσκευή ελέγχου φωτιστικού με αναφορά κατάστασης κοστίζει αρκετές δεκάδες ευρώ, χωρίς να υπάρχει ούτε υλικό ούτε τεχνολογικό αντίκρισμα.

Επιπλέον ο έλεγχος αυτών των συσκευών από ένα τεχνολογικά διαφορετικό σύστημα, όπως αυτό της παρούσας εργασίας, αυξάνει την πολυπλοκότητα του τελικού συστήματος και ίσως να μην μας επιτρέπει να αξιοποιήσουμε στο σύνολό τους της δυνατότητες μια "έξυπνης" συσκευής. Συγκεκριμένα στο παράδειγμά μας, για την διασύνδεση του πρωτοκόλλου X10 με το XMPP, χρειαστήκαμε να υλοποιήσουμε έναν διαχειριστή γεγονότων, μια μεταφορά X10 και να χρησιμοποιήσουμε ένα εξωτερικό πρόγραμμα επικοινωνίας του υπολογιστή με το X10, ενώ θα μπορούσαμε να έχουμε μόνο τον διαχειριστή γεγονότων και να μπορούμε να "μιλάμε" απευθείας στις συσκευές.

Η δημιουργία συσκευών που να υποστηρίζουν εγγενώς τη σύνδεση στο δίκτυο XMPP, σημαίνει αυτομάτως φθηνό και αξιόπιστο υλικό, πρωτόκολλα βασισμένα στην XML με πολύ περισσότερες δυνατότητες και ευκολίες στην ανάπτυξη και συντήρηση από τα σημερινά δυαδικά, απευθείας διασύνδεση

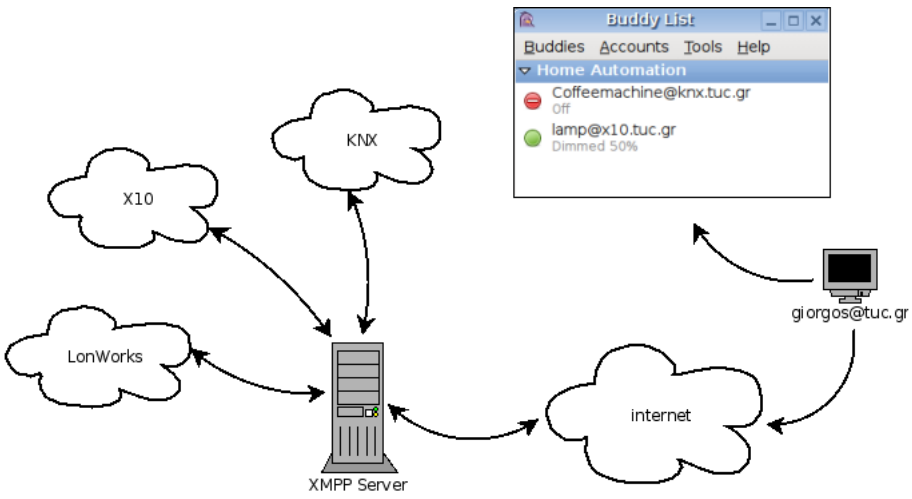
με το διαδίκτυο για εφαρμογές εξ'αποστάσεως, δυνατότητα αναβάθμισης χαρακτηριστικών κ.α.

Μια τέτοια συσκευή θα πρέπει να μπορεί να συνδεθεί σε ένα δίκτυο, όπως είναι το Ethernet, το WiFi, το ZigBee να αποκτήσει μια διεύθυνση IP (σημειώνουμε ότι το IPv6 μας επιτρέπει να έχουμε μια μοναδική διεύθυνση IP για κάθε συσκευή του σπιτιού μας, αφού διευθυνσιοδοτεί 2^{128} διαφορετικές συσκευές) και στη συνέχεια να κάνει μια σύνδεση TCP σε προκαθορισμένο εξυπηρετητή σύμφωνα με το XMPP. Τέλος θα πρέπει να στέλνει αναφορές παρουσίας (presence notifications) και να δρα κατάλληλα σε κάθε ληφθείσα εντολή.

5.2 Διασύνδεση με άλλα συστήματα αυτοματισμών

Σήμερα υπάρχουν δεκάδες διαφορετικά συστήματα αυτοματισμού κτηρίων, τα οποία διαφέρουν τόσο στον φυσικό τρόπο επικοινωνίας των συσκευών όσο και στο πρωτόκολλο που ακολουθούν οι συσκευές για την ανταλλαγή της πληροφορίας. Πέρα από το σύστημα X10 το οποίο μας απασχόλησε σε αυτή την εργασία και υλοποιήσαμε μια σύνδεση μεταξύ αυτού και του συστήματος XMPP, υπάρχουν και άλλα συστήματα όπως το INSTEON [38], το KNX [36], το LonWorks [40], το C-Bus [39] κ.τ.λ. Μερικά από αυτά τα συστήματα χρησιμοποιούν καλώδια μεταφοράς των σημάτων, άλλα είναι ασύρματα και άλλα χρησιμοποιούν την υπάρχουσα καλωδίωση ενός κτηρίου, συνήθως αυτή του ηλεκτρικού ρεύματος.

Σε όλες τις περιπτώσεις όμως υπάρχουν τερματικά που συνδέονται με ηλεκτρονικό υπολογιστή για τον έλεγχο των συσκευών και τη συλλογή δεδομένων. Ακολουθώντας την ίδια τακτική που ακολουθήσαμε και στο κεφάλαιο 3.2 "Γέφυρα επικοινωνίας X10 και XMPP" θα μπορούσαμε να φτιάξουμε γέφυρες επικοινωνίας μεταξύ καθενός διαφορετικού συστήματος και



Σχήμα 10: Ο χρήστης μπορεί από το ένα απλό πρόγραμμα ανταλλαγής μηνυμάτων (στο σχήμα το πρόγραμμα Pidgin) να ενημερώνεται σε πραγματικό χρόνο για την κατάσταση των συσκευών που παρακολουθεί και να δίνει άμεσα εντολές σε αυτές, σε μία κοινή, κατανοητή στον άνθρωπο γλώσσα.

του πρωτοκόλλου XMPP. Με αυτό τον τρόπο θα υλοποιούσαμε ένα εικονικό δίκτυο, πάνω από τα υπάρχοντα δίκτυα αυτοματισμού, από το οποίο θα μπορούσαμε να αντλήσουμε πληροφορία και να ελέγξουμε το σύνολο των συσκευών, χωρίς επιπλέον λογισμικό από τη πλευρά του πελάτη και χωρίς να απασχολεί τον τελευταίο το δίκτυο στο οποίο ανήκουν οι συσκευές που θέλει να ελέγξει.

Με αυτό τον τρόπο θα μπορούσαμε να έχουμε τον πλήρη έλεγχο από ένα πρόγραμμα και ένα πίνακα ελέγχου του γραφείου μας αλλά και του σπιτιού μας, παρόλο που οι κατασκευαστές του κτηρίου επέλεξαν διαφορετικά συστήματα. Σε ένα άλλο παράδειγμα θα μπορούσαμε να συλλέγουμε πληροφορίες από άλλου τύπου αισθητήρες σε διαφορετικά ερευνητικά κέντρα και να συλλέγουμε το σύνολο των πληροφοριών εύκολα σε μία XML βάση δεδομένων για περαιτέρω επεξεργασία και δράση.

Συνοψίζοντας η διασύνδεση του συνόλου των διαφορετικών συστημάτων

αυτοματισμού δημιουργεί ένα επίπεδο αφαίρεσης, μη αντιληπτό από τον τελικό χρήστη, ο οποίος μπορεί να ελέγξει τα πάντα με τις ίδιες εντολές, από το ίδιο τελικό και απλό τεχνολογικά πρόγραμμα. Με αυτό τον τρόπο ξεπερνάμε στην πράξη την ασυμβατότητα των πρωτοκόλλων και οι όποιοι τεχνολογικοί και όποιους οικονομικούς περιορισμούς υπάρχουν εξαιτίας του μονοπωλίου και των κλειστών προτύπων, αφού ο χρήστης μπορεί να επιλέξει την συσκευή που του χρειάζεται με μόνο κριτήριο τις ανάγκες του.

5.3 Ασφάλεια μεταφοράς δεδομένων και διαπίστευσης χρηστών

Το XMPP είναι από τον σχεδιασμό του ένα δίκτυο βασισμένο σε εξυπηρετητές, αλλά αποκεντρωποιημένο. Δηλαδή κάθε ενδιαφερόμενος χρήστης μπορεί να δημιουργήσει ένα δικό XMPP εξυπηρετητή και να ανταλλάσσει μηνύματα με άλλους χρήστες. Ακριβώς επειδή δεν υπάρχει κάποια κεντρική διαχείριση των χρηστών, δεν μπορεί να υπάρξει και κάποια κεντρική διαπίστευση ότι ο χρήστης έχει όντως την ταυτότητα που υποστηρίζει το XMPP μήνυμα που λάβαμε.

Το πρόβλημα είναι σημαντικό αλλά εάν το σκεφτούμε, παρουσιάζεται καθημερινά, αφού ακριβώς με τον ίδιο τρόπο λειτουργεί και η παραδοσιακή αλληλογραφία, όπου ο καθένας μπορεί να συμπληρώσει στο πεδίο του αποστολέα το όνομα που επιθυμεί χωρίς να του ζητήσει κάποιος ταυτότητα, αλλά και το ηλεκτρονικό ταχυδρομείο, το οποίο έχει ακριβώς τη δομή και τα χαρακτηριστικά ενός δικτύου του XMPP. Με λίγα λόγια και στα τρία αυτά μέσα επικοινωνίας, μπορεί ο καθένας να υποστηρίξει ότι είναι κάποιος άλλος χωρίς να το ελέγξει κανείς.

Φυσικά στην περίπτωση της παραδοσιακής αλληλογραφίας, ένα τηλέφωνο επιβεβαίωσης μπορεί να μας δώσει την λύση, αλλά σίγουρα μια συσκευή αυτοματισμού απλά θα υπακούσει τυφλά στον ιδιοκτήτη της, όπως

αυτή το αναγνωρίζει από την ηλεκτρονική διεύθυνσή του (Jabber ID - JID / e-mail). Το πρόβλημα παίρνει τρομακτικές διαστάσεις εάν αναλογιστούμε ότι στο μέλλον που θα είναι οι περισσότερες, αν όχι όλες, οι συσκευές αυτοματοποιημένες και συνδεδεμένες στο δίκτυο ένας χρήστης που προσποιείται ότι είναι ο ιδιοκτήτης θα μπορεί να ανοίξει την πόρτα του σπιτιού, να απενεργοποιήσει τον συναγερμό κ.ο.κ.! Προφανώς η λύση δεν είναι να μην υιοθετήσουμε τεχνολογίες αυτοματισμού που θα κάνουν τη ζωή μας καλύτερη, αλλά να φροντίσουμε να αναπτύξουμε παράλληλα με αυτές τις λύσεις και τις κατάλληλες διαδικασίες διαπίστευσης των χρηστών και κρυπτογράφησης του κειμένου.

Σήμερα υπάρχουν ήδη τεχνολογίες που μπορούμε να κληροδοτήσουμε στο XMPP τόσο για την διαπίστευση, όσο και για την κρυπτογράφηση. Σε αυτό το σημείο θα πρέπει να παρατηρήσουμε ότι στα πρωτόκολλα του XMPP υπάρχουν σαφώς ορισμένες διαδικασίες για την ασφαλή, κρυπτογραφημένη διαπίστευση των χρηστών σε ένα εξυπηρετητή και για την επικοινωνία των εξυπηρετητών μεταξύ τους. Στη περίπτωση που εξετάζουμε όμως θα χρειαστούμε κρυπτογράφηση και διαπίστευση από άκρη σε άκρη (end to end), δηλαδή από τον ένα χρήστη στον άλλο, γιατί δεν μπορούμε να εμπιστευτούμε όλους τους εξυπηρετητές που θα μεσολαβήσουν για την παράδοση του μηνύματός μας.

Για αυτούς τους σκοπούς μπορούμε να χρησιμοποιήσουμε τεχνολογίες όπως το OpenPGP για το οποίο υπάρχει και κατάλληλο πρότυπο από το Ίδρυμα XMPP [19]. Επίσης θα ήταν σημαντικό να υπάρξουν υλοποιήσεις για τα υπό δοκιμή πρότυπα του XMPP "Encrypted Session Negotiation" XEP-0116 [20] και "Cryptographic Design of Encrypted Sessions" XEP-0188 [21] τα οποία υλοποιούν διαπίστευση των χρηστών και κρυπτογράφηση των δεδομένων από άκρη σε άκρη.

Αναπτύσσοντας τέτοια συστήματα κάθε συσκευή θα χρησιμοποιεί επιπλέον μηχανισμούς για την αναγνώριση του ιδιοκτήτη πέρα από τη διεύθυνση που αναγράφεται στο μήνυμα που παρέλαβε καθιστώντας πολύ δύσκολη την αλλοίωση, επανάληψη [7] ή ακόμη και την απλή ανάγνωση του μηνύματος από κάποιον τρίτο.

5.4 Υψηλού επιπέδου διεπαφές χρηστών

Στη παρούσα διπλωματική δημιουργήσαμε μια γέφυρα μεταξύ του πρωτοκόλλου XMPP και του X10. Για την επικοινωνία των συσκευών μεταξύ τους στο επίπεδο XMPP, για παράδειγμα του υπολογιστή με την λάμπα X10, δημιουργήσαμε ένα απλό DTD που παρουσιάσαμε νωρίτερα για να ορίσουμε τα βασικά πεδία μιας εντολής, όπως είναι το πεδίο της εντολής αυτής καθαυτής, της ώρας εκτέλεσης κ.α. Όμως η γλώσσα XML δεν δημιουργήθηκε για να την γράφουν άνθρωποι και για αυτό τον λόγο ενσωματώσαμε έναν απλό αναλυτή φυσικής γλώσσας στη γέφυρά μας. Με αυτόν τον τρόπο όταν ένας άνθρωπος στέλνει ένα μήνυμα στην X10 λάμπα του, με τη χρήση του προγράμματος ανταλλαγής μηνυμάτων της επιλογής του, η γέφυρα αναλαμβάνει να το μετατρέψει σε XML και να το προωθήσει κατάλληλα ώστε να εκτελεστεί η εντολή. Η αναφερθείσα υλοποίηση είναι εύκολη στη χρήση και ο καθένας μπορεί να ελέγξει τις συσκευές του από έναν απλό XMPP πελάτη, δεν έχει όμως χαρακτηριστικά υψηλού επιπέδου.

Αναφερόμενοι σε χαρακτηριστικά υψηλού επιπέδου για διεπαφές χρηστών, αναφερόμαστε σε φωνητικές εντολές τις οποίες θα πρέπει να κατανοεί το σύστημα. Επίσης αναφερόμαστε σε γραφικά περιβάλλοντα εικονικής ή και πραγματικής αναπαράστασης του χώρου, που δηλαδή θα

[7] Η "τυφλή επανάληψη" μηνυμάτων θα χρειαστεί και επιπλέον ανάπτυξη πέρα από την υλοποίηση κρυπτογράφησης και διαπίστευσης. Μπορεί να είναι εξίσου επικίνδυνη αφού εάν δεν υπάρχει σχετική πρόβλεψη ένας κακόβουλος χρήστης μπορεί να επαναλάβει μηνύματα σχετικά με το άνοιγμα της πόρτας ή το κλείσιμο του υπολογιστή μας!

μπορούμε να ενημερωθούμε για την κατάσταση του φωτισμού κοιτώντας απευθείας το δωμάτιο που μας ενδιαφέρει είτε μέσα από κάποιο εικονικό φωτορεαλιστικό μοντέλο, είτε και από φωτογραφία του χώρου σε πραγματικό χρόνο.

Ακόμη αναφερόμαστε σε λογισμικά για την διαχείριση της αλληλεπίδρασης των συσκευών με τις άλλες συσκευές και το περιβάλλον, με έμφαση στην ευκολία της χρήσης, χωρίς όμως να στερούμαστε δυνατοτήτων. Έτσι θα πρέπει να μπορούμε μέσα από ένα ανθρωποκεντρικό περιβάλλον να ορίσουμε μια συσχέτιση μεταξύ της παρουσίας μας σε ένα δωμάτιο, με την εκκίνηση του υπολογιστή και το άνοιγμα μια λάμπας, με όχι περισσότερες από 3 κινήσεις drag 'n' drop (σείρε και άφησε) στα κατάλληλα αντικείμενα.

6 Παράρτημα

6.1 Λογισμικό σχετικό με το XMPP

Το XMPP αναπτύχθηκε εκθετικά σε μία μόλις πενταετία. Από την πρώτη επίσημη εμφάνισή του το 2000 μέχρι σήμερα έχει αναπτυχθεί μια πλήρης σειρά εξυπηρετητών, πελατών και βιβλιοθηκών για την παιρεταίρω ανάπτυξη του πρωτοκόλλου. Θα πρέπει να σημειωθεί ότι κανένα λογισμικό ούτε εξυπηρετητής, ούτε πελάτης δεν υλοποιεί πλήρως το σύνολο του πρωτοκόλλου όπως αυτό ορίζεται από τα RFCs, όμως η ελαστικότητα που παρέχει το πρωτόκολλο δεν το καθιστά αυτό ως πρόβλημα και έτσι μπορούν να συνυπάρξουν πελάτες με εξυπηρετητές και οι τελευταίοι με τη σειρά τους με άλλους εξυπηρετητές με διαφορετικό ποσοστό κάλυψης του πρωτοκόλλου ο καθένας. Άλλωστε η ίδια η φύση του πρωτοκόλλου που είναι φιλική προς στις επεκτάσεις και ο ρυθμός παρουσίασης νέων επεκτάσεων καθιστά την πλήρη κάλυψη του συνόλου πολύ δύσκολη. Ο κάθε χρήστης ή διαχειριστής θα πρέπει να επιλέξει το προϊόν σύμφωνα με τις ανάγκες του. Ακολουθεί μια μικρή ενδεικτική λίστα με Ελεύθερο Λογισμικό / Λογισμικό Ανοιχτού Κώδικα που ο καθένας μπορεί να χρησιμοποιήσει για να χρησιμοποιήσει το XMPP.

6.1.1 Πελάτες

- **PSI:** Το PSI από το ελληνικό γράμμα Ψ, το αρχικό της φράσης "Ψυχική Ενέργεια" είναι ένας πελάτης του δικτύου XMPP που χαρακτηρίζεται από σταθερότητα και συνέπεια στο πρωτόκολλο. Η κονσόλα του πρωτοκόλλου σε περιβάλλον αποσφαλμάτωσης είναι ιδιαίτερα χρήσιμη κατά την ανάπτυξη εφαρμογών. Διαθέσιμο για πλατφόρμες GNU/Linux, Mac OS X και Windows ως Ελεύθερο Λογισμικό. [22]

- **Pidgin:** Το Pidgin, παλαιότερα γνωστό ως Gaim, είναι ένας πελάτης δικτύων ανταλλαγής μηνυμάτων. Πέρα από το XMPP υποστηρίζει και άλλα 14 δίκτυα. Είναι Ελεύθερο Λογισμικό και παρέχεται για τις πλατφόρμες GNU/Linux, MacOS X και Windows. [23]

6.1.2 Εξυπηρετητές

- **Jabberd 1.x:** Το Jabberd14 όπως ονομάζεται η σειρά Jabberd 1.x είναι ένας Jabber εξυπηρετητής με πλήρη υποστήριξη των RFCs 3920 και 3921, του πυρήνα δηλαδή του πρωτοκόλλου XMPP. Το Jabberd14 είναι ο ιστορικότερος XMPP εξυπηρετητής και η λειτουργία του έχει δοκιμαστεί εκτενώς με πολλούς παράλληλους χρήστες. Είναι διαθέσιμος ως Ελεύθερο Λογισμικό για πολλές πλατφόρμες. [24]

- **Jabberd 2.x:** Το Jabber2 ξεκίνησε την πορεία του ως αντικαταστάτης της σειράς Jabber 1.x όμως στη πορεία οι ομάδες ανάπτυξης των δύο λογισμικών πήραν διαφορετικό δρόμο. Ο σχεδιασμός του επιτρέπει αποκέντρωση των στοιχείων του ώστε να επιτυγχάνεται καλύτερος έλεγχος, μεγαλύτερη ασφάλεια και υψηλότερο αριθμό ταυτόχρονα συνδεδεμένων χρηστών. Διανέμεται για λειτουργικά GNU/Linux, BSD, Solaris και Windows ως Ελεύθερο Λογισμικό. [25]

- **Openfire:** Ο εξυπηρετητής Openfire, παλαιότερα γνωστός ως Wilfire Server, είναι γραμμένος στη γλώσσα Java και διανέμεται και ως Ελεύθερο Λογισμικό αλλά και με εμπορική άδεια στις πλατφόρμες που υποστηρίζει η γλώσσα. Ιδιαίτερο χαρακτηριστικό του είναι ότι καλύπτει σε μεγαλύτερο βαθμό το πρωτόκολλο από οποιοδήποτε άλλο λογισμικό, σύμφωνα με τα μετρούμενα χαρακτηριστικά του jabber.org. [26]

- **ejabberd:** Ο ejabberd γραμμένο στη γλώσσα Erlang, επίσης με υψη-

λό δείκτη καλυπτόμενων χαρακτηριστικών στο jabber.org, θεωρείται ένας από τους πιο αξιοπόπιστούς και γρήγορους XMPP εξυπηρετητές. Έχει επιλεγεί από το jabber.org για να εξυπηρετεί περισσότερους από 290 χιλιάδες χρήστες καθημερινά. Το ejabberd είναι και αυτό Ελεύθερο Λογισμικο. [27]

6.1.3 Βιβλιοθήκες

- **Twisted Words:** Βιβλιοθήκη της γλώσσας προγραμματισμού Python. Είναι υποσύνολο της συλλογής βιβλιοθηκων Twisted Python. Αρκετά καλή υποστήριξη του πρωτοκόλλου τόσο για την κατασκευή πελατών, όσο και επεκτάσεων εξυπηρετητών. Διανέμεται σύμφωνα με την άδεια του MIT. [28]

- **Loudmouth:** Η βιβλιοθήκη LoudMouth υλοποιεί μια ασύγχρονη πλήρη βιβλιοθήκη σε C για τη συγγραφή πελατών XMPP δικτύων. Διανέμεται ως Ελεύθερο Λογισμικό. [29]

- **JSO:** Η βιβλιοθήκη JSO προσθέτει στη γλώσσα Java υποστήριξη του πρωτοκόλλου XMPP σε "χαμηλό" επίπεδο, και προσφέρεται για τη συγγραφή πελατών, επεκτάσεων αλλά και εξυπηρετητών. Διανέμεται ως Ελεύθερο Λογισμικό. [30]

6.2 Ο πηγαίος κώδικας των εφαρμογών

Listing 12: jClient.py

```
1  #!/usr/bin/env python
   # * coding: utf8 *

   from twisted.words.protocols.jabber import jid,xmlstream,client
   from twisted.words.xish import domish
6  from twisted.internet import reactor
   from twisted.python import log, logfile
   from string import join, letters
```

```
    from random import sample
    from sys import stderr, exit
11  import os.path

    class Jclient:

        def __init__(self, server, username, password, resource="Jclient",
            port=5222, observers=None, enableLog=False, lf=None):
16          self.username = username
            self.password = password
            self.resource = resource
            self.server = server
            self.port = port
21          self.observers = observers
            self.enableLog = enableLog
            self.myJid = jid.JID('%s/%s' % (self.username, self.resource))

            if self.enableLog:
26              if not lf:
                    stderr.write("Logging requested but no logfile given")
                    exit(1)

                self.logfile = logfile.DailyLogFile(os.path.basename(lf), os.
                    path.dirname(lf))
31              self.logger = log.startLogging(self.logfile)
                print("Logging started")

        def connect(self):
            factory = client.basicClientFactory(self.myJid, self.password)
36          factory.addBootstrap('//event/stream/authd', self.authd)
            reactor.connectTCP(self.server, self.port, factory)

        def disconnect(self):
            presence = domish.Element(('jabber:client', 'presence'))
41          presence['from'] = "%s/%s" % (self.username, self.resource)
            presence['type'] = 'unavailable'
            self.xmlstream.send(presence)

        def authd(self, xmlstream):
46          #save xmlstream
            self.xmlstream = xmlstream
            # send presence
            presence = domish.Element(('jabber:client', 'presence'))
            xmlstream.send(presence)
51          self.watchdog()

        def watchdog(self):
            if not self.observers:
                self.observers = {}
56             # self.observers['/*'] = self.logger
                self.observers['/iq'] = self.iqHandlerCaller
                self.observers['/message'] = self.messageHandlerCaller
                self.observers['/presence'] = self.presenceHandlerCaller

61          for trigger, function in self.observers.items():
                self.xmlstream.addObserver(trigger, function)

        def iqHandlerCaller(self, xmlstream):
            """ Calling iqHandler in a try/except section with logging
```

```
66          abilities"""
            log.callWithLogger(log.Logger(), self.iqHandler, xmlstream)

        def messageHandlerCaller(self, xmlstream):
            """ Calling messageHandler in a try/except section with logging
71          abilities"""
            log.callWithLogger(log.Logger(), self.messageHandler, xmlstream)

        def presenceHandlerCaller(self, xmlstream):
            """ Calling presenceHandler in a try/except section with logging
76          abilities"""
            log.callWithLogger(log.Logger(), self.presenceHandler, xmlstream)

        def logger(self, xmlstream):
            print xmlstream.toXml()
81
        def iqHandler(self, xmlstream):
            return

        def messageHandler(self, xmlstream):
86          """
            Returns sender's jid, receiver's jid and message
            """
            sender = xmlstream['from']
            receiver = xmlstream['to']
91          try:
                message = xmlstream.body.children[0]
            except:
                message = None

96          if 'id' in xmlstream.attributes:
                id = xmlstream['id']
            else:
                id = self.generateId()

101         return sender, receiver, id, message

        def send(self, msg):
            self.xmlstream.send(msg)

106     def generateId(self):
            return 'jc' + join(sample(letters, 5), '')

        def newMsg(self, to, message, id=None, type="chat"):
            if not id:
111             # generate id
                id = self.generateId()

            msg = domish.Element(('jabber:client', 'message'), attribs={'to':
                to, 'from':self.myJid.full(), 'type': type, 'id': id})
            msg.addElement('body', content=message)
116         return msg

        def replyMsg(self, xmlstream, message, type=None):
            """
            Creates reply AND sends it
121         """
            msg = self.newMsg(xmlstream['from'], message, None, type or
                xmlstream['type'])
```

```
            self.send(msg)


    def presenceHandler(self, xmlstream):
        """
        Should implement auto add and auto delete from
        presence lists
        """

        if 'type' in xmlstream.attributes:
            if xmlstream['type'] == 'error':
                pass
            elif xmlstream['type'] == 'probe':
                msg = domish.Element(('jabber:client', 'presence'),
                    attribs = {'to': xmlstream['from'], 'from':xmlstream['
                    to']})
                # always free to chat
                msg.addElement('show', content="chat")
            else:
                msg = domish.Element(('jabber:client', 'presence'),
                    attribs = {'type':xmlstream['type'], 'to':xmlstream['
                    from'], 'from':xmlstream['to']})

            try:
                self.xmlstream.send(msg)
            except:
                pass
```

Listing 13: jComponent.py

```
#!/usr/bin/env python
# * coding: utf8 *

from twisted.words.protocols.jabber import jid,xmlstream
from twisted.words.protocols.jabber import component
from twisted.words.xish import domish
from twisted.internet import reactor
from twisted.python import log, logfile
from string import join, letters
from random import sample
import sys
import os.path

class Jcomponent:

    def __init__(self, server, username, password, port=5347, observers=
        None, enableLog=False, lf=None):
        self.username = username
        self.password = password
        self.server = server
        self.port = port
        self.observers = observers
        self.enableLog = enableLog
        self.myJid = jid.JID('%s' % (self.username))

        if self.enableLog:
```

```
            if not lf:
                stderr.write("Logging requested but no logfile given")
                exit(1)

30
            self.logfile = logfile.DailyLogFile(os.path.basename(lf), os.
                path.dirname(lf))
            self.logger = log.startLogging(self.logfile)

    def connect(self):
35      factory = component.componentFactory(self.username, self.password)
        factory.addBootstrap('//event/stream/authd', self.
            componentConnected)
        reactor.connectTCP(self.server, self.port, factory)

    def componentConnected(self, xmlstream):
40      self.xmlstream = xmlstream
        self.watchdog()

    def callLater(self, *args, **kwargs):
        reactor.callLater(*args, **kwargs)
45
    def watchdog(self):
        disco = "/iq[@type='get']/query"
        disco += "[@xmlns='http://jabber.org/protocol/disco#info']"

50      if not self.observers:
            self.observers = {}
            #self.observers['/*'] = self.logger
            self.observers['/iq'] = self.iqHandlerCaller
            self.observers['/message'] = self.messageHandlerCaller
55          self.observers['/presence'] = self.presenceHandlerCaller
            self.observers[disco] = self.discoHandlerCaller

        for trigger, function in self.observers.items():
            self.xmlstream.addObserver(trigger, function)
60
    def logger(self, xmlstream):
        print "LOG: %s" % xmlstream.toXml()

    def iqHandlerCaller(self, xmlstream):
65      """ Calling iqHandler in a try/except section with
        logging abilities"""
        log.callWithLogger(log.Logger(), self.iqHandler, xmlstream)

    def discoHandlerCaller(self, xmlstream):
70      log.callWithLogger(log.Logger(), self.discoHandler, xmlstream)

    def messageHandlerCaller(self, xmlstream):
        """ Calling messageHandler in a try/except section with
        logging abilities"""
75      log.callWithLogger(log.Logger(), self.messageHandler, xmlstream)

    def presenceHandlerCaller(self, xmlstream):
        """ Calling presenceHandler in a try/except section with
        logging abilities"""
80      log.callWithLogger(log.Logger(), self.presenceHandler, xmlstream)

    def iqHandler(self, xmlstream):
```

```
            return

    def discoHandler(self, xmlstream):
        xmlstream.swapAttributeValues('to', 'from')
        xmlstream['type'] = "result"
        xmlstream.query.children = []
        i = xmlstream.query.addElement('identity')
        i['category'] = 'gateway'
        i['type'] = 'X10'
        i['name'] = 'X10 Bridge'
        xmlstream.query.addElement("feature")["var"] = "http://jabber.org/
            protocol/disco#info"
        self.xmlstream.send(xmlstream)

    def messageHandler(self, xmlstream):
        """
        Returns sender's jid, receiver's jid and message
        """
        sender = xmlstream['from']
        receiver = xmlstream['to']
        try:
            message = xmlstream.body.children[0]
        except:
            message = None

        if 'id' in xmlstream.attributes:
            id = xmlstream['id']
        else:
            id = self.generateId()

        return sender, receiver, id, message

    def send(self, msg):
        self.xmlstream.send(msg)

    def generateId(self):
        return 'jc' + join(sample(letters, 5), '')

    def newMsg(self, frm, to, message, id=None, type="chat"):
        if not id:
            # generate id
            id = self.generateId()

        msg = domish.Element(('jabber:client', 'message'), attribs={'to':
            to, 'from':frm, 'type': type, 'id': id})
        msg.addElement('body', content=message)

        return msg

    def replyMsg(self, xmlstream, message, type=None):
        """
        Creates reply AND sends it
        """
        msg = self.newMsg(xmlstream['from'], message, None, type or
            xmlstream['type'])
        self.send(msg)

    def presenceHandler(self, xmlstream):
        """
```

```
        Should implement auto add and auto delete from
        presence lists
        """
        if 'type' in xmlstream.attributes:
            if xmlstream['type'] == 'error':
                pass
            elif xmlstream['type'] == 'probe':
                msg = domish.Element(('jabber:client', 'presence'),
                    attribs = {'to': xmlstream['from'], 'from':xmlstream['
                    to']})
                # always free to chat
                msg.addElement('show', content="chat")
            else:
                msg = domish.Element(('jabber:client', 'presence'),
                    attribs = {'type':xmlstream['type'], 'to':xmlstream['
                    from'], 'from':xmlstream['to']})
            try:
                self.xmlstream.send(msg)
            except:
                pass
```

Listing 14: Διαχειριστής Μηνυμάτων

```
#!/usr/bin/env python

from twisted.enterprise import adbapi
from twisted.internet import reactor
from twisted.words.xish import domish
from twisted.persisted import dirdbm
from jclient import Jclient
from time import strftime
import sys

class Jeventmanager(Jclient):
    def __init__(self, server, username, password, resource="em", port
        =5222):
        Jclient.__init__(self,server, username, password, resource, port)
        self.queue = dirdbm.Shelf("./events")
        self.futureEvent()
        self.connect()

    def iqHandler(self, xmlstream):
        print "Received: %s" % xmlstream.toXml()
        # auth request
        # uri is xmlns
```

```
        if xmlstream.query and xmlstream.attributes['type'] == 'get':
            # ok we received a query
            if xmlstream.query.uri == 'ha:device':
                # we received a auth query
                try:
                    userJid = xmlstream.query.userId.children[0]
                    userJidType = xmlstream.query.userId.getAttribute('
                        type', 'jid')
                except KeyError:
                    sys.exit(1)


                try:
                    deviceJid = xmlstream.query.deviceId.children[0]
                    action = xmlstream.query.deviceId.getAttribute('action
                        ')
                except KeyError:
                    sys.exit(1)


                if userJidType == 'usb':
                    self.usb2jid(userJid).addCallback(self.callAction,
                        deviceJid, action, xmlstream, self.queueResult)
                elif userJidType == 'jid':
                    self.callAction(userJid, deviceJid, action, xmlstream,
                         self.queueResult)
                elif userJidType == 'bt':
                    self.bt2jid(userJid).addCallback(self.callAction,
                        deviceJid, action, xmlstream, self.queueResult)

    def bt2jid(self, device):
        query = 'SELECT `users`.`jid` FROM `users`, `bt2jid` WHERE `users
            `.`id` = `bt2jid`.`id` AND `bt2jid`.`mac` LIKE "%s"' % device
        return dbpool.runQuery(query)

    def usb2jid(self, device):
        query = 'SELECT `users`.`jid` FROM `users`, `usb2jid` WHERE `users
            `.`id` = `usb2jid`.`id` AND `usb2jid`.`serial` LIKE "%s"' %
            device
```

```
53          return dbpool.runQuery(query)

        def queueResult(self, result, xmlstream):
            message = xmlstream
            message['to'] = xmlstream.getAttribute('from')
58          message['id'] = xmlstream.getAttribute('id', self.generateId())
            message['from'] = self.myJid.full()



            if len(result) == 1:
63              message['type'] = 'result'
                try:
                    userJid, deviceJid, action = result[0]
                except:
                    print "Error fetching userJid, deviceJid and action"
68                  return

                now = strftime("%Y%m%d%H%M")
                try:
                    time = message.query.time
73                  key = "%04d%02d%02d%02d%02d" % (int(time.year.children[0])
                        ,int(time.month.children[0]), int(time.day.children
                        [0]),int(time.hour.children[0]), int(time.minute.
                        children[0]))
                except AttributeError:
                    key = now

                # The given hour has passed. Execute now
78              if now >= key:
                    key = strftime("%Y%m%d") + "0000"


                try:
                    self.queue[key].append(message)
83              except KeyError:
                    self.queue[key] = [message]

            else:
                # we got access denied. Errorize message and send now.
88              # Do not queue
```

```
                message['type'] = 'error'
                message.addElement('error', content='Access Denied')
                self.send(message)

93          return result

        def logActions(self, result, xmlstream):
            if len(result) == 1:
                userJid, deviceJid, action = result[0]
98              query = 'INSERT INTO log (`userJid`, `deviceJid`, `action`, `
                    time`, `result`) VALUES ("%s", "%s", "%s", NOW(), "%s")' %
                     (userJid, deviceJid, action, "successful")
                return dbpool.runQuery(query)
            else:
                # failed record as failed
                pass
103
        def checkPriviledge(self, userjid, devicejid, action):
            query = 'SELECT `users`.`jid`, `actions`.`jid`, `actions`.`action`
                 FROM `users`, `actions`, `priviledges` WHERE `actions`.`
                action` = "%s" AND `actions`.`jid` = "%s" AND `priviledges`.`
                id` = `actions`.`id` AND (`priviledges`.`jid` = 0 OR `
                priviledges`.`jid` = "%s")' % (action, devicejid, userjid)
            return dbpool.runQuery(query)

108     def callAction(self, userJid, deviceJid, action, xmlstream, callback):
            if type(userJid) == tuple:
                # we have used bt2jid or usb2jid functions
                if len(userJid) == 1:
                    userJid = userJid[0][0]
113             else:
                    # we didn't find id
                    # FIX THIS MUST RETURN IQ ERROR 'ID NOT FOUND'
                    return

118         d = self.checkPriviledge(userJid, deviceJid, action)
            d.addCallback(callback, xmlstream)
            d.addCallback(self.logActions, xmlstream)
```

```
        def futureEvent(self):
123         """
            Should record events refering to future time
            """
            reactor.callLater(1, self.futureEvent)
            print "Queue is ",
128         print self.queue.keys()
            now = strftime("%Y%m%d")
            now += "0000"
            list = []

133         # get day events
            try:
                list += self.queue[now]
                del(self.queue[now])
            except KeyError:
138             # nothing to do!
                pass

            # get day time events
            now = strftime("%Y%m%d%H%M")
143         try:
                list += self.queue[now]
                del(self.queue[now])
            except KeyError:
                pass
148
            for item in list:
                self.send(item)

    # initialize connection to DB
153 dbpool = adbapi.ConnectionPool("MySQLdb", "localhost", db="ha", user="root
        ", passwd="")

    if __name__ == "__main__":
        if len(sys.argv) == 4:
            Jeventmanager = Jeventmanager(sys.argv[1], sys.argv[2], sys.argv
                [3])
158     else:
```

```
        Jeventmanager = Jeventmanager("localhost", "em@hjabber.sealabs.net
            ", "1")



reactor.run()
```

Listing 15: Γέφυρα X10 με XMPP

```
#!/usr/bin/env python

from twisted.internet import reactor, defer, threads, utils
from twisted.words.xish import domish
from jcomponent import Jcomponent
import sys
from commands import getoutput
from xml2obj import Xml2Obj
from dateutil.parser import parse

PRG = 'heyu'
CONFIG = 'x10.xml'

class notAvailableCommand(Exception):pass

class jX10gateway(Jcomponent):
    def __init__(self,  server, username, password, port=5347):
        Jcomponent.__init__(self, server, username, password,port)
        self.eventManager = 'em@hjabber.sealabs.net/em'
        self.devices = {}
        self.loadConfig()
        self.connect()

    def loadConfig(self):
        parser = Xml2Obj()
        elements = parser.Parse(CONFIG)

        if elements.name != 'x10':
            # this is not a x10 config file
            print 'This is not a x10 config file!'
            sys.exit(1)

        for element in elements.children:
            broken = 0
            jid = None
            if element.name == 'device':
                # ok we have a devicce
                    d = {}
                    for delement in element.children:
                        if delement.name == 'jid':
                            jid = delement.getData()
                            continue

                        d[delement.name] = delement.getData()
```

```
                    for key in ['H', 'A', 'type', 'twoway']:
                        if d.has_key(key) is False:
48                          print 'This is not a valid x10 device entry!
                                Key %s not found!' % key
                            broken = 1
                    if jid == None:
                        print 'This is not a valid x10 device entry! Key %
                            s not found!' % key
                        broken = 1
53
                    # check if jid already exists
                    if self.devices.has_key(jid):
                        print 'This jid %s already exists!' % (jid)
                        broken = 1
58
                    if broken == 0:
                        # add to devices list
                        try:
                            d['roster'] = []
63                          self.devices[jid] = d.copy()
                        except KeyError:
                            print 'Error adding device %s' % jid
                            continue

68
        print self.devices

    def componentConnected(self, xmlstream):
        Jcomponent.componentConnected(self, xmlstream)
73      self.initDevices()

    def initDevices(self):
        for key, device in self.devices.items():
            if device['twoway'] == 'True':
78              #get status from device
                pass
            else:
                device['status'] = 'Off'
                threads.deferToThread(self.sendSignal, 'Off', device['H'],
                     device['A'])
83
            self.sendPresence(key)

    def sendPresence(self, djid, jid=None):
        try: device = self.devices[djid]
88      except KeyError:
            return

        presence = domish.Element(('jabber:client', 'presence'))
        presence['from'] = djid
93      if device['status'] == 'Off':
            presence.addElement('show', content='dnd')
        else:
            presence.addElement('show')
        presence.addElement('status', content=device['status'])
98
        if jid == None:
            #now send to all
```

```
                for jid in device['roster']:
                    presence['to'] = jid
                    self.send(presence)
            else:
                presence['to'] = jid
                self.send(presence)

    def sendSignal(self, command, H, A):
        if command == None:
            return
        command = command.lower()

        try:
            signal, value = command.split(' ', 1)
        except ValueError:
            signal = command
            value = ""

        command = "%s %s %s%s %s" % (PRG, signal, H, A, value)
        getoutput(command)
        return command

    def changeDeviceStatus(self, command, receiver):
        if command == None:
            return command
        c = command.capitalize().split()[0]

        print c
        if c == 'Off':
            status = 'Off'
        elif c == 'Dim' or c == 'Bright':
            status = 'Dimmed'
        elif c == 'On' or c == 'Brightb':
            status = 'On'
        else:
            status = command

        try: self.devices[receiver]['status'] = status
        except KeyError: pass

        self.sendPresence(receiver)

        return command

    def translateCommand(self, command, receiver):
        cmd = {}
        cmd['time'] = parse(command, fuzzy=True)

        c = command.lower().split()
        type = self.devices[receiver]['type']
        status = self.devices[receiver]['status']
        if 'on' in c:
            if self.devices[receiver]['status'] == 'Dimmed':
                command = 'brightb 1'
            else:
                command = 'on'
        elif 'off' in c:
            command = 'off'
```

```
            elif 'up' in c and type == 'Lamp' and status != 'On':
                command = 'brightb 5'
            elif 'down' in c and type == 'Lamp' and status != 'Off':
                command = 'dim 5'
        else:
            print "raise!"
            raise notAvailableCommand

        cmd['command'] = command


        return cmd

    def messageHandler(self, xmlstream):
        sender, receiver, id, message = Jcomponent.messageHandler(self,
            xmlstream)
        if not message:
            return

        try:
            H = self.devices[receiver]['H']
            A = self.devices[receiver]['A']
        except KeyError:
                return

        d = threads.deferToThread(self.translateCommand, message, receiver
            )
        d.addErrback(self.notAvailableCommand, sender, receiver)
        d.addCallback(self.requestAuth, sender, receiver)

    def notAvailableCommand(self, failure):
        print "Not available command!"
        return failure

    def requestAuth(self, command, sender, receiver):
        """Builds an iq message to 'eventManager'
             requesting auth and sends it"""
        print command
        (year, month, day, hour, minute) = command['time'].timetuple()
            [0:5]
        message = domish.Element(('jabber:client', 'iq'))
        message['from'] = self.myJid.full()
        message['to'] = self.eventManager
        message['type'] = 'get'
        message['id'] = self.generateId()
        message.addElement('query')
        message.query['xmlns'] = 'ha:device'
        message.query.addElement('userId', content=sender.split('/')[0])
        message.query.userId['type'] = 'jid'
        message.query.addElement('deviceId', content=receiver)
        message.query.deviceId['action'] = command['command']
        message.query.addElement('time')
        message.query.time['xmlns'] = 'ha:time'
        message.query.time.addElement('year', content=str(year))
        message.query.time.addElement('month', content=str(month))
        message.query.time.addElement('day', content=str(day))
        message.query.time.addElement('hour', content=str(hour))
```

```
            message.query.time.addElement('minute', content=str(minute))

            print message.toXml()
            self.xmlstream.send(message)

    def iqHandler(self, xmlstream):
        if xmlstream.getAttribute('from') == self.eventManager and
            xmlstream.getAttribute('type') == 'result':
            # ok we got something we trust
            try:
                device = xmlstream.query.deviceId.children[0]
                command = xmlstream.query.deviceId.getAttribute('action',
                    None)
                H = self.devices[device]['H']
                A = self.devices[device]['A']
            except KeyError:
                print "Wrong xmlstream received"

            d = threads.deferToThread(self.changeDeviceStatus, command,
                device)
            d.addCallback(self.sendSignal, H, A)


    def presenceHandler(self, xmlstream):
        if xmlstream.getAttribute('type') == 'probe':
            self.sendPresence(xmlstream.getAttribute('to', None),
                xmlstream.getAttribute('from'))
            # add him to devices local roster
            try:
                device = xmlstream.getAttribute('to')
                jid = xmlstream.getAttribute('from')
                self.devices[device]['roster'].append(jid)
            except KeyError:
                return
        elif xmlstream.getAttribute('type') == 'subscribe':
            msg = domish.Element(('jabber:client', 'presence'), attribs =
                {'type':'subscribed', 'to':xmlstream['from'], 'from':
                xmlstream['to']})
            self.send(msg)
            self.devices[xmlstream.getAttribute('to')]['roster'].append(
                xmlstream.getAttribute('from'))
            self.sendPresence(xmlstream.getAttribute('to', None),
                xmlstream.getAttribute('from'))

if __name__ == "__main__":
    if len(sys.argv) == 4:
        jX10gateway = jX10gateway(sys.argv[1], sys.argv[2], sys.argv[3])
    else:
        JX10gateway = jX10gateway("localhost", "x10.hjabber.sealabs.net",
            "secret")


reactor.run()
```

Listing 16: Ηλεκτρονική USB Κλειδαριά

```
#!/usr/bin/env python

# call before importing jcline to load our own reactor
from twisted.internet import glib2reactor
glib2reactor.install()

from twisted.internet import threads, defer
import twisted.internet
from twisted.words.xish import domish
from jclient import Jclient
import dbus, subprocess, sys
from dbus.mainloop.glib import DBusGMainLoop
from commands import getoutput

class Jusb(Jclient):
    def __init__(self, server, username, password, resource="usb", port
        =5222, eventManager="em@hjabber.sealabs.net/em", devices=None):
        """
        deviceConnected is False if no device is connected and
        contains full device udi if device is connected
        """
        Jclient.__init__(self,server, username, password, resource, port)

        # save authorised event manager
        self.eventManager = eventManager
        self.deviceConnected = False

        # listen on specific devices eg. /dev/sdb (not in /dev/sdb1,2,3
        # use None to listen on all devices
        self.devices = devices

        self.connect()
        self.dbusInit()
        # used for debug
        #twisted.internet.reactor.callLater(2, self.simulateUsb)
        twisted.internet.reactor.run()

    def simulateUsb(self):
        """ Recursive function calls every 5 seconds requestAuth
```

```
            simulating
        a usb device insertion. Usefull for debugging"""
        self.deviceConnected = True
        self.requestAuth("lalala")
        #twisted.internet.reactor.callLater(5, self.simulateUsb)
43
    def dbusInit(self):
        """ dBus initilazation process. We connect to DeviceAdded and
         DeviceRemoved signals"""
        DBusGMainLoop(set_as_default=True)
48      self.bus = dbus.SystemBus()
        self.halManagerObject = self.bus.get_object("org.freedesktop.Hal",
             "/org/freedesktop/Hal/Manager")
        self.halManager = dbus.Interface(self.halManagerObject, "org.
            freedesktop.Hal.Manager")
        self.halManager.connect_to_signal("DeviceAdded", self.deviceAdded)
        self.halManager.connect_to_signal("DeviceRemoved", self.
            deviceRemoved)
53
    def deviceRemoved(self, device_udi, *args):
        """ If we have recognized a device we set
        deviceConnected to False"""
        if device_udi == self.deviceConnected:
58          self.deviceConnected = False

    def deviceAdded(self, device_udi, *args):
        """ We get devices serial number and call requestAuth"""
        device_udi_obj = self.bus.get_object("org.freedesktop.Hal",
            device_udi)
63      properties = device_udi_obj.GetAllProperties(dbus_interface="org.
            freedesktop.Hal.Device")

        # filter out not wanted devices
        device = properties.get("block.device")
        if self.devices is not None:
68          if device not in self.devices: return

        serial = properties.get("storage.serial")
```

```
            if not serial == None:
73              self.deviceConnected = device_udi
                self.requestAuth(serial)

        def requestAuth(self, serial):
            """Builds an iq message to 'eventManager'
78           requesting auth and sends it"""
            message = domish.Element(('jabber:client', 'iq'))
            message['from'] = self.myJid.full()
            message['to'] = self.eventManager
            message['type'] = 'get'
83          message['id'] = self.generateId()
            message.addElement('query')
            message.query['xmlns'] = 'ha:device'
            message.query.addElement('userId', content=serial)
            message.query.userId['type'] = 'usb'
88          message.query.addElement('deviceId', content=self.myJid.userhost()
                )
            message.query.deviceId['action'] = 'open'
            self.xmlstream.send(message)

        def iqHandler(self, xmlstream):
93          """Handling events from eventManager"""
            if xmlstream.attributes['from'] ==  self.eventManager and
                xmlstream.attributes['type'] == 'result' and self.
                deviceConnected is not False:
                print "Access granted!"
                self.unlock()
            else:
98              print "Access denied!"

        def unlock(self):
            command = "gnome screensaver command  d"
            getoutput(command)
103
    if __name__ == "__main__":
        if len(sys.argv) == 4:
            jusb = Jusb(sys.argv[1], sys.argv[2], sys.argv[3])
        else:
```

```
108         jusb = Jusb("hjabber.sealabs.net", "door@hjabber.sealabs.net", "1"
                )


    r
```

Listing 17: Ανιχνευτής με χρήση Bluetooth

```
    #!/usr/bin/env python

    import bluetooth
    from twisted.words.xish import domish
 5  from twisted.internet import reactor, threads, defer
    from jclient import Jclient
    import sys


10  class JbtScanner(Jclient):
        def __init__(self, server, username, password, resource="bt", port
            =5222, eventManager="em@hjabber.sealabs.net/em"):
            Jclient.__init__(self,server, username, password, resource, port)

            # save authorised event manager
15          self.eventManager = eventManager
            self.connect()
            self.devices = {}
            self.call(None)

20      def call(self, devices):
            # Create thread so we don't block procedure
            scanner = threads.deferToThread(self.scanBt)
            scanner.addCallback(self.sendPresence)
            scanner.addCallback(self.updateGlobalList)
25          # recursive call
            scanner.addCallback(self.call)

        def scanBt(self):
            """
30          Scans for active bluetooth devices in area.
```

```
        Returns list with MAC addresses
        """
        devices = bluetooth.discover_devices(lookup_names = False)
        print devices
        return devices

    def updateGlobalList(self, devices):
        """
        Keeps an updated local list of connected bt devices.
        Can be used in the future to recognise clients and customize
        enviromental parameters
        """
        self.devices = {}
        for device in devices:
            try: self.devices[device] = True
            except: pass

        return devices

    def sendPresence(self, devices):
        """ Sends presence changes with found device info """
        # don't send presence if nothing changed
        if len(devices) == len(self.devices):
            return devices

        presence = domish.Element(('jabber:client', 'presence'))
        l = len(devices)

        if l == 0:
            presence['type'] = 'unavailable'
            content = 'Room empty'
        elif l == 1:
            content = "1 person in room"
        else:
            content = "%d persons in room" % l

        presence.addElement('show')
        presence.addElement('status', content=content)
        presence.addElement('persons', content=str(l))
```

```
70          presence.addElement('devices')
            for device in devices:
                presence.devices.addElement('device', content=device)
            presence.persons['xmlns'] = 'ha:presence'
            presence.devices['xmlns'] = 'ha:presence'
75          self.send(presence)

            return devices


    if __name__ == "__main__":
80      if len(sys.argv) == 4:
            jbtscanner = JbtScanner(sys.argv[1], sys.argv[2], sys.argv[3])
        else:
            jbtscanner= JbtScanner("localhost", "kitchen@hjabber.sealabs.net",
                 "1")


85  reactor.run()
```

6.3 Ορολογία

Ελεύθερο Λογισμικό: Το Ελεύθερο Λογισμικό είναι λογισμικό που μπορεί να χρησιμοποιηθεί, να μελετηθεί και να τροποποιηθεί χωρίς περιορισμούς. Επίσης μπορεί να αντιγραφεί και μοιραστεί τροποποιημένο ή όχι με μοναδικό περιορισμό οι παραλήπτες να απολαμβάνουν τις ίδιες ελευθερίες με τον δότη. Πρέπει να σημειωθεί ότι ο όρος 'Ελεύθερο' αναφέρεται στις ελευθερίες που το συνοδεύουν και δεν αφορά το κόστος. Το Ίδρυμα Ελεύθερο Λογισμικού είναι υπεύθυνο για την ιδεολογική βάση του, από το 1980 και σήμερα συνεχίζει τη διάδοση, την εξέλιξη και προστασία του Ελεύθερου Λογισμικού. (Ορισμός του Ε.Λ. από το Ίδρυμα Ε.Λ.[33])

Λογισμικό Ανοιχτού Κώδικα: Το Λογισμικό Ανοιχτού κώδικα είναι λογισμικό που διατίθεται με μία άδεια χρήσης σύμφωνη με τον ορισμό της Πρωτοβουλίας Ανοιχτού Κώδικα. Αν και το Ελεύθερο Λογισμικό θεωρείται

Λογισμικό Ανοιχτού Κώδικα, δεν ισχύει και το αντίστροφο, αφού τη Πρωτοβουλία Ανοιχτού Κώδικα έχει διαφορετικά κριτήρια από το Ίδρυμα Ελεύθερου Λογισμικού (Ορισμός του Λ.Α.Κ. από τη Π.Α.Κ. [34])

ΕΛ/ΛΑΚ: Συντομογραφία που χρησιμοποιούμε για να αναφερθούμε συνολικά σε **Ε**λεύθερο **Λ**ογισμικό και **Λ**ογισμικό **Α**νοιχτού **Κ**ώδικα.

XMPP: Το e**X**tensible **M**essaging and **P**resence **P**rotocol (XMPP) είναι ένα ανοιχτό πρωτόκολλο βασισμένο στην XML για ανταλλαγή μηνυμάτων και πληροφοριών παρουσίας σε πραγματικό χρόνο. [42]

Jabber: Ονομάζεται η τεχνολογία ανταλλαγής μηνυμάτων και πληροφοριών παρουσίας που βασίζεται στο XMPP. [43]

X10: Το X10 είναι ένα διεθνές ανοιχτού πρότυπο επικοινωνίας μεταξύ ηλεκτρικών συσκευών. Για την επικοινωνία χρησιμοποιεί είτε τις υπάρχουσες ηλεκτρικές καλωδιώσεις, είτε ραδιοσυχνότητες. Χρησιμοποιείται για αυτοματισμούς σε σπίτια. [37]

KNX: Το KoNneX είναι ο απόγονος των συστημάτων αυτοματισμού EIB, EHS και BatiBUS. Με πολύ περισσότερες δυνατότητες από το X10 είναι το επικρατέστερο ανοιχτό πρότυπο για έξυπνα κτήρια στην Ευρώπη. [36]

LonWorks: Το LonWorks είναι ένα σύστημα αυτοματισμού από την εταιρία Echelon, αντίστοιχων δυνατοτήτων με το KNX αλλά δημοφιλές στην Αμερική. [40]

XML: Η eXtensible Markup Language είναι μια γενικής χρήσης γλώσσα περιγραφής. Ο πρωταρχικός της σκοπός είναι να αποτελεί έναν δομημένο τρόπο ανταλλαγής πληροφοριών μεταξύ διαφορετικών συστημάτων, τεχνολογιών και λογισμικού. [35]

OSI: Δείτε το λήμμα "Λογισμικό Ανοιχτού Κώδικα"

GNU: Αρχικά της φράσης "GNU is Not Unix". Το GNU είναι αναδρομικό ακρονύμιο, ένα σύνηθες αστείο κατά την ονομασία έργων Ελεύθερου

Λογισμικού. Αναφέρεται στο έργο που ξεκίνησε το 1983 από τον Richard Stallman και στόχο έχει τη δημιουργία ενός λειτουργικού τύπου Unix αλλά μόνο με Ελεύθερο Λογισμικό.

HAL: Αρχικά του Hardware Abstration Layer, ενός επιπέδου αφαίρεσης που επιτρέπει στους προγραμματιστές και τους χρήστες την επικοινωνία με υλικό με εύκολο, ανεξάρτητο της συσκευής και γενικό τρόπο. Στο GNU/Linux υλοποιείται από το ομώνυμο λογισμικό που είναι Ελεύθερο Λογισμικό.

OTP: Αρχικά του One Time Password. Μεθοδολογία κατά την οποία σε ένα σύστημα που χρειάζεται κωδικό διαπίστευσης κάθε κωδικός είναι έγκυρος για μία και μοναδική φορά. Έτσι ακόμη και εάν ο κωδικός μαθευτεί μετά τη πρώτη χρήση του θα λήξει.

MySQL: Ισχυρή βάση δεδομένων. Διανέμεται ως Ελεύθερο Λογισμικό για πολλές πλατφόρμες. [31]

6.4 Άδεια Χρήσης

Το λογισμικό που αναπτύχθηκε σε αυτή την εργασία είναι Ελεύθερο Λογισμικό, μπορεί να διανεμηθεί και/ή να τροποποιηθεί σύμφωνα με τους όρους της GNU Γενικής Άδειας Χρήσης (GNU GPL `http://www.fsf.org/licensing/licenses/gpl.html`), όπως δημοσιεύεται από το Ίδρυμα Ελεύθερου Λογισμικού, της έκδοσης 3.

Το παρόν κείμενο και το σύνολο των πινάκων και των εικόνων, πέρα των εικόνων των σχημάτων 1, 2 και 3, διατήθονται σύμφωνα με τους όρους της άδειας GNU Ελεύθερης Τεκμητίωσης (GNU FDL `http://www.fsf.org/licensing/licenses/fdl.html`).

Το λογισμικό και το κείμενο διατήθεται με την ελπίδα ότι θα είναι χρήσιμο, αλλά χωρίς ΚΑΜΜΙΑ ΕΓΓΥΗΣΗ.

Αναφορές

[1] DJ Adams, *Programming Jabber*, OReilly, 2002.

[2] William Wright and Dana Moore, *Jabber Developers Handbook*, Sams, 2003.

[3] Magnus L. Hetland, *Practical Python*, Apress, 2002.

[4] Wikipedia, *Home Automation*,

`http://en.wikipedia.org/wiki/Home_automation`

[5] Wikipedia, *Extensible Messaging and Presence Protocol*,

`http://en.wikipedia.org/wiki/Xmpp`

[6] Wikipedia, *X10*,

`http://en.wikipedia.org/wiki/X10`

[7] Wikipedia, *Home Automation*,

`http://en.wikipedia.org/wiki/Home_automation`

[8] Acme Systems, *FOX Board*,

`http://www.acmesystems.it`

[9] AXIS, *Extrax 100LX CPU*,

`http://www.axis.com/products/dev_etrax_100lx/`

[10] Atmel, *AVR32 CPU*,

`http://www.atmel.com/products/AVR32/`

`http://www.avr32linux.org`

[11] Intel, *Xscale CPU*,

`http://www.intel.com/design/intelxscale/`

[12] XMPP Standards Foundation, *XMPP, Extensible Messaging and Presence Protocol: Core*

http://www.xmpp.org/rfcs/rfc3920.html

[13] XMPP Standards Foundation, *XMPP, Extensible Messaging and Presence Protocol: Instant Messaging and Presence*,

http://www.xmpp.org/rfcs/rfc3921.html

[14] XMPP Standards Foundation, *Jabber-RPC*,

http://www.xmpp.org/extensions/xep-0009.html

[15] XMPP Standards Foundation, *Jabber Component Protocol*,

http://www.xmpp.org/extensions/xep-0114.html

[16] Python Software Foundation,

http://www.python.org/psf

[17] Busybox, *The Swiss Army Knife of Embedded Linux*,

http://www.busybox.net

[18] HEYU, *X10 Controlling Software*,

http://heyu.tanj.com/

[19] XMPP Standards Foundation, *Current Jabber OpenPGP Usage*,

http://www.xmpp.org/extensions/xep-0027.html

[20] XMPP Standards Foundation, *Encrypted Session Negotiation*,

http://www.xmpp.org/extensions/xep-0116.html

[21] XMPP Standards Foundation, *Cryptographic Design of Encrypted Sessions*,

http://www.xmpp.org/extensions/xep-0188.html

[22] PSI, *Instant Messaging Application*,

http://psi-im.org

[23] Pidgin, *Instant Messaging Application*,

`http://www.pidgin.im`

[24] Jabberd, *Jabber Server*,

`http://www.jabber.org`

[25] Jabberd 2, *Jabber Server*,

`http://jabberd2.xiaoka.com`

[26] Openfire, *Jabber Server*,

`http://www.igniteraltime.org/projects/openfire/`

[27] ejabberd, *Jabber Server*,

`http://www.process-one.net/en/ejabberd/`

[28] Twisted Words, *Jabber Library*,

`http://www.twsitedmatrix.com`

[29] Loudmouth, *Jabber Library*,

`http://www.loudmouth-project.org`

[30] JSO, *Jabber Library*,

`http://jso.jabberstudio.org`

[31] Mysql, *Relational Database*,

`http://www.mysql.org`

[32] XMPP Standards Foundation,

`http://www.xmpp.org/`

[33] Free Software Foundation, *The Free Software Definition*,

`http://www.fsf.org/licensing/essays/free-sw.html`

[34] Open Source Initiative, *The Open Source Definition*,

http://opensource.org/docs/osd

[35] World Wide Web Consortium, *Extensible Markup Language (XML)*,

http://www.w3.org/XML/

[36] KNX, *Automation System*,

http://www.knx.org

[37] X10, *Automation System*,

http://www.x10.com

[38] INSTEON, *Automation System*,

http://www.insteon.net/

[39] C-bus, *Automation System*,

http://www2.clipsal.com/cis/technical/

[40] LonWorks, *Automation System*,

http://www.echelon.com

[41] xAP, *Automation System*, 2005,

http://www.xapautomation.org/

[42] XMPP, *Instant Messasing and Presence Protocol*,

http://www.xmpp.org

[43] Jabber, *Technology based on XMPP*,

http://www.jabber.org

[44] Simon Aurell, *Remote Controlling Devices Using Instant Messaging*, 2005,

http://portal.acm.org/ft_gateway.cfm?id=1088371&type=pdf

[45] mi4, *xAP Jabber Gateway*,

http://www.mi4.net/modules.php?name=Content&pa=

showpage&pid=37