ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Triennale in Informatica

UN FRAMEWORK PER APPLICAZIONI DI MONITORAGGIO E DOMOTICA BASATO SU TECNOLOGIE ANDROID E ARDUINO

Tesi di Laurea in Laboratorio di Applicazioni Mobili

Relatore:
Chiar.mo Prof.
LUCIANO BONONI

Presentata da:
DANIELE BASCHIERI
Matr. 0000405155

Correlatore:
Dott.
LUCA BEDOGNI

Correlatore:
Dott.
MARCO DI FELICE

Sessione 2
Anno Accademico 2011/2012

Sommario

Questa tesi tratta dello sviluppo di un progetto chiamato Faxa e di una sua concreta applicazione nell'ambito della domotica (CasaDomotica).

Faxa è un framework per la comunicazione via wireless tra dispositivi che supportano il sistema operativo Android e dispositivi Arduino Ethernet, comunicazione che avviene localmente attraverso il wi-fi.

Il progetto si inserisce nel panorama più ampio chiamato "Internet of Things", ovvero internet delle cose, dove ogni oggetto di uso domestico è collegato ad Internet e può essere quindi manipolato attraverso la rete in modo da realizzare una vera e propria "smart house"; perchè ciò si attui occorre sviluppare applicazioni semplici e alla portata di tutti.

Il mio contributo comincia con la realizzazione del framework Faxa, così da fornire un supporto semplice e veloce per comporre programmi per Arduino e Android, sfruttando metodi ad alto livello.

Il framework è sviluppato su due fronti: sul lato Android è composto sia da funzioni di alto livello, necessarie ad inviare ordini e messaggi all'Arduino, sia da un demone per Android; sul lato Arduino è composto dalla libreria, per inviare e ricevere messaggi.

Per Arduino: sfruttando le librerie Faxa ho redatto un programma chiamato "BroadcastPin". Questo programma invia costantemente sulla rete i dati dei sensori e controlla se ci sono ordini in ricezione.

Il demone chiamato "GetItNow" è una applicazione che lavora costantemente in background. Il suo compito è memorizzare tutti i dati contenuti nei file xml inviati da Arduino. Tali dati corrispondono ai valori dei sensori connessi al dispositivo. I dati sono salvati in un database pubblico, potenzialmente accessibili a tutte le applicazioni presenti sul dispositivo mobile.

Sul framework Faxa e grazie al demone "GetItNow" ho implementato "CasaDomotica", un programma dimostrativo pensato per Android in grado di interoperare con apparecchi elettrici collegati ad un Arduino Ethernet, impiegando un'interfaccia video semplice e veloce.

L'utente gestisce l'interfaccia per mezzo di parole chiave, a scelta comandi vocali o digitali, e con essa può accendere e spegnere luci, regolare ventilatori, attuare la rilevazione di temperatura e luminosità degli ambienti o quanto altro sia necessario. Il tutto semplicemente connettendo gli apparecchi all'Arduino e adattando il dispositivo mobile con pochi passi a comunicare con gli elettrodomestici.

"E se anche il quadro della tua vita risultasse un Picasso,
non preoccuparti varrebbe ugualmente miliardi."

Anonimo

Indice

Elenco delle figure

Capitolo 1

Introduzione

1.1 Internet Of Things

Il mondo sta mutando ad una velocità sorprendente, la stessa idea di Internet è cambiata radicalmente nell'arco di pochissimi anni. Quando si parla di Internet of Things non si parla di un'ipotesi futura, ma di un presente possibile che sta sempre più prendendo piede.

Internet of Things è la possibilità di fornire un indirizzo IP ad ogni oggetto, in modo tale che questo abbia una interfaccia sulla rete internet. Non si tratta quindi solo di comandare e monitorare a distanza i dispositivi domestici, ma di dare a questi ultimi un ruolo attivo di protagonisti, instaurando una rete di comunicazione fra essi. Si avranno infatti confezioni di farmaci che notificano all'utente se non sono stati assunti, e sveglie che si riprogrammano in base alle condizioni di traffico.[1]

Il passaggio da IPv4 a IPv6 avvenuto recentemente permette un'incredibile svolta, in quanto ogni dispositivo domestico potrebbe potenzialmente essere dotato di un indirizzo IP univoco che lo contraddistingue e gli fornisce un accesso per comunicare sulla rete (lo stesso A.S.Tanenbaum afferma: "IPv6 permetterebbe di utilizzare $7 * 10^{23}$ indirizzi IP per metro quadro"). [2]

Un altro fattore fondamentale per la diffusione di Internet of Things sono i costi delle componenti elettroniche in continuo calo: con pochissimi euro è possibile acquistare dispositivi elettronici sempre più potenti e veloci, Arduino e Raspberry Pi solo per fare qualche esempio.[3] Questi dispositivi, configurati in modo opportuno, permettono di integrare oggetti domestici nati prima di questa rivoluzione fornendo un indirizzo IP a quasi tutti gli oggetti comuni, dalle semplici lampade ad oggetti più complessi come le lavatrici.[4]

Ultime e non meno importanti sono alcune caratteristiche degli attuali dispositivi mobili come la diffusione, le ridotte dimensioni e la potenza di calcolo: 20 anni fa poche persone in Italia possedevano un telefono cellulare, mentre oggi si stima che in media ci siano in circolazione almeno 1.2 dispositivi per abitante, senza tener conto di quanti troppo anziani o troppo giovani non lo possiedono affatto. La potenza di calcolo di questi dispositivi è diventata paragonabile a quella dei computer e addirittura superiore nel caso di alcuni modelli in circolazione, e questo fornisce una spinta enorme al progetto di Internet of Things in quanto fa degli Smartphone uno strumento di controllo completo e versatile ma soprattutto sempre disponibile. Le ridotte dimensioni infatti ne fanno i compagni di viaggio ideali, e il costo ridotto ne favorisce la diffusione sul mercato. Gli Smartphone sono la chiave di volta di questo sistema in quanto non essendo oggetti dedicati ad un unica funzione possono offrire le interfacce giuste per controllare ogni apparecchio domestico.[5] [6]

Con la nascita dei Tablet una nuova porta si è aperta nell'universo dell'Internet of Things: infatti il tablet ha una mobilità maggiore di un laptop ma è altrettanto efficiente, perfetto come strumento per la gestione di un ambiente domestico.

Internet of Things è molto di più che scambiare messaggi tra dispositivi, è la possibilità di mettere in comunicazione e far sì che si comprendano differenti interfacce e differenti dispositivi, la possibilità che svariati oggetti comunichino tra di loro per un obiettivo comune, acquisendo in questo modo un comportamento "intelligente", in modo di migliorare la nostra esperienza di vita. [8]

In una casa domotica è fondamentale la complicità tra tutti gli apparecchi: non si parla più di un singolo dispositivo ma ogni dispositivo acquisisce consapevolezza dell'ambiente che lo circonda, ciascuno secondo le proprie caratteristiche; la possibilità quindi di comunicare tra i vari dispositivi fa la grande forza di questo sistema, un sistema olistico che si rafforza più che linearmente all'aumentare dei dispositivi connessi e che è in grado di offrire sempre maggiori servizi.

1.2 Domotica

Il termine "domotica" deriva dalla parola latina "domus", che significa casa. La Domotica infatti si occupa di migliorare la qualità della vita all'interno delle abitazioni e, più in generale, in tutti gli ambienti in cui vivono gli uomini.

L'ambito di lavoro è interdisciplinare, vede coinvolti professionisti di vari ambiti: ingegneria edile, automazione, elettronica, telecomunicazioni e informatica e questa necessaria integrazione di varie discipline è dovuta alle molteplici applicazioni della domotica in ogni ambiente domestico.

I principali obiettivi della domotica possono essere riassunti in 5 punti: [?]

- migliorare la qualità della vita.
- migliorare la sicurezza.
- semplificare la progettazione, l'installazione e la manutenzione della tecnologia.
- ridurre i costi di gestione.
- convertire i vecchi ambienti e i vecchi impianti.

I sistemi domotici, per poter essere presentati al grande pubblico, devono essere affidabili, semplici da usare e non devono mettere in pericolo l'utilizzatore anche in caso di guasto o danneggiamento accidentale. Il target di destinazione di queste tecnologie non è un professionista ma bensi l'utente medio, pertanto l'interfaccia deve necessariamente essere *user friendly*. Un'altra caratteristica fondamentale è il basso costo dei componenti: soltanto così si può avere una reale diffusione di queste tecnologie.

Fissati quindi gli obiettivi ci concentriamo ora sulle aree in cui la domotica viene applicata.

Le aree di gestione di una casa possono essere suddivise in quattro settori:

- gestione dell'ambiente: è a sua volta suddivisa in climatizzazione, riscaldamento, illuminazione, distribuzione dell'energia, azionamento di sistemi d'apertura e d'ingresso ecc ecc. Il controllo dell'ambiente viene automatizzato grazie alla presenza di sensori e attuatori e ciò permette la termoregolazione dei locali e un consumo più oculato delle risorse.
- gestione degli apparecchi: sempre più spesso i più comuni elettrodomestici, dalla lavatrice al frigorifero o al forno, montano dispositivi elettronici che ne consentono la telegestione sia a scopo diagnostico sia per regolare e attivare gli apparecchi domestici.
- comunicazione e informazione: questo settore spazia dai telefoni analogici o voip al citofono o all'accesso a internet. Ma può anche comprendere la semplice trasmissione di dati per il controllo remoto.
- sicurezza: questa si occupa della gestione degli accessi, della protezione antifurto e antintrusione ed del videocontrollo. Tuttavia la sicurezza non si occupa solo degli agenti esterni ma anche di guasti domestici come fughe di gas, incendi o allagamenti, eventi dannosi che se non controllati in tempo potrebbero compromettere o distruggere la casa.

1.3 Android

Android è un sistema operativo *open source*, progettato per dispositivi mobili basato sul kernel linux. Include un sistema operativo di base, i middleware per le comunicazione e le applicazioni di base. [9]

Android fornisce un database *SQLite*, librerie *SGL* per la grafica 2D, supporta la grafica 3D grazie allo standard *OpenGL ES 2.0*, fornisce anche funzioni native per la sintesi vocale e per il riconoscimento vocale. Le applicazioni vengono eseguite dalla *Dalvik Virtual Machine*, una macchina virtuale molto simile alla *Java Virtual Machine* della Sun, ma con caratteristiche più performanti e adatta agli smartphone.

1.3.1 Storia

Android Inc. è stata fondata a Palo Alto in California, nell'Ottobre del 2003. I fondatori sono Andy Rubin, Rich Miner, Nick Sears e Chirs White. Inizialmente la società tenne segreti i reali obiettivi del proprio lavoro, fingendo di lavorare su software per dispositivi mobili.

In realtà il team di Rubin stava sviluppando un sistema operativo per dispositivi mobili basato sul kernel Linux.

Il 17 agosto del 2005 Google ha acquistato l'azienda, per riuscire ad entrare nel crescente mercato della telefonia mobile. [10]

La presentazione ufficiale della versione 1.0 Apple Pie avvenne nel settembre del 2008.

1.3.2 Versioni

Dal 2008 ad oggi sono uscite 10 versioni del sistema operativo, le quali presentano sia numerose innovazioni per quanto riguarda l'usabilità del dispositivo, sia l'introduzione di nuove funzionalità che la risoluzione degli svariati bug presenti. Una così rapida evoluzione del sistema ha portato Android ad essere uno dei più importanti sistemi operativi nel mercato degli smartphone essendo presente sul 53% dei dispositivi.[11]

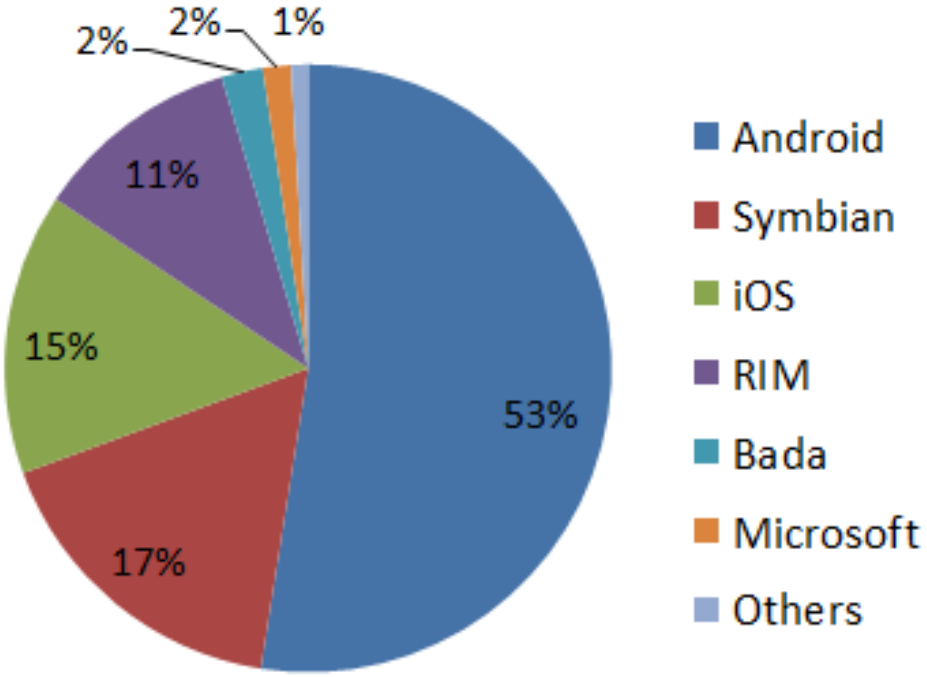

Figura 1.1: Distribuzione dei Sistemi Operativi per smarphone sul mercato

Tra le varie distribuzioni la più diffusa è sicuramente la Android 2.3.3, chiamata Gingerbread, presente su più del 50% dei *device*. [12] Ogni versione è retrocompatibile con le precedenti. Sviluppando quindi aplicazioni per la versione 2.3.3 si ha la certezza di coprire la stragrande maggioranza dei dispositivi, e di ottenere il miglior compromesso sviluppatore utente finale. Infatti lo sviluppatore può tenere conto di tutte le peculiarità introdotte da Gingerbread e al contempo raggiungere la maggior parte degli utenti.

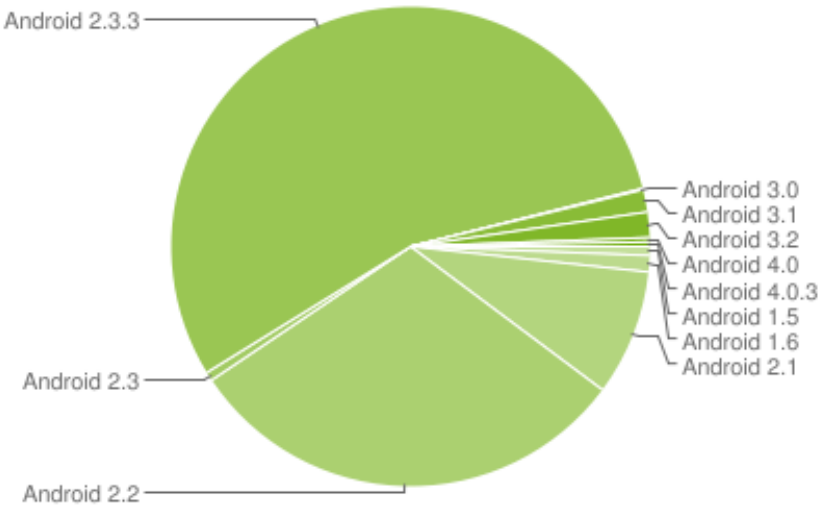

Figura 1.2: Diffusione delle varie distribuzioni Android nel mercato

Google offre agli sviluppatori la possibilità di aggiungere i loro nuovi programmi sul *Google Play Store*. Il *Google Play* è un gestore di contenuti multimediali di tutti i tipi, fornisce agli utenti la via maestra per reperire giochi, applicazioni, programmi di ogni genere ma anche libri, musica e filmati.

L'utente può installare senza difficoltà tutte le nuove funzionalità direttamente dallo smartphone, decidendo il metodo di pagamento che preferisce. Lo sviluppatore dal canto suo, seguendo un semplice processo di crittografia, può aggiungere la sua applicazione allo Store Virtuale in modo da raggiungere subito il largo pubblico di riferimento.

Oltre al *Google Play* è possibile installare gli apk anche da altre fonti, pertanto non si è necessariamente vincolati al *Google Play* ma anche le applicazioni che per qualche motivo sono state rimosse dal negozio o che si è deciso di non caricare nel negozio stesso possono comunque ricevere una buona diffusione per vie secondarie e mercati alternativi, rendendo così libero l'intero sistema di distribuzione.

Libertà è insita nel cuore di Android che è infatti un sistema operativo *open source* (eccettuate alcune versioni), un sistema perfetto per fare delle sperimentazioni poichè permette a noi studenti e sviluppatori di conoscere nel profondo ciò di cui ci stiamo occupando.

Pur essendo una piattaforma *open source* Android non si classifica molto bene nella classifica stilata dal *Open Governance Index*, ottenendo un misero 23% sulla scala di libertà, a causa di alcuni fattori rilevanti: l'accesso al codice sorgente non avviene in modo omogeneo tra tutti gli utenti, ma prima ricevono l'accesso i partners commerciali di

Google; gli sviluppatori attivi del progetto rimangono anonimi e non viene messo in luce chi contribuisce maggiormente al progetto; Android non permette l'utilizzo dei marchi, del codice e di servizi quali il market senza prima firmare un contratto; la comunity non è sufficentemente ben strutturata e incentivata.[13]

Android pur essendo meno *"open source"* della controparte *symbian* ha una diffusione davvero sorprendente. Infatti i dati sulle vendite e sulla diffusione dei dispositivi Android sul mercato dimostrano questa tendenza del mercato.

1.3.3 Ambiente di Sviluppo

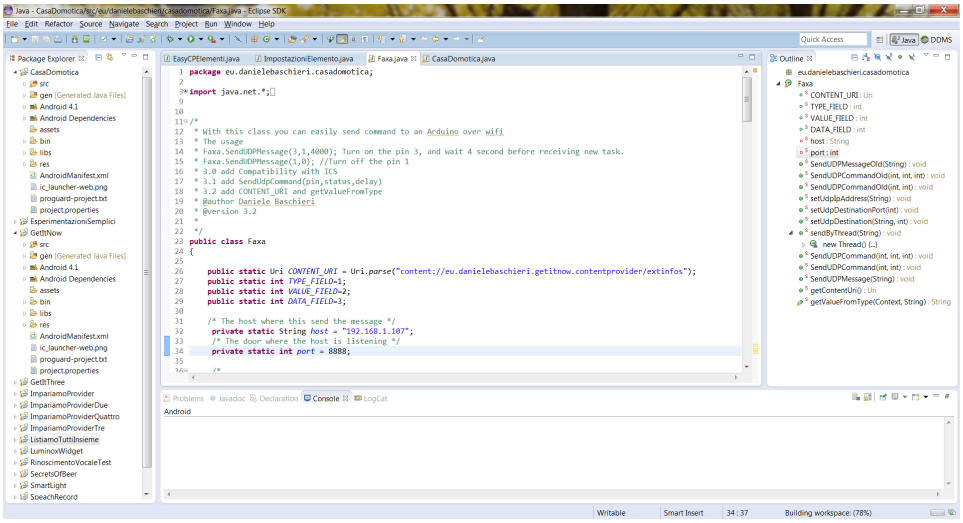

Figura 1.3: Ambiente di Sviluppo Eclipse

Eclipse è un ottimo abimente di sviluppo per applicazioni, in particolare per sviluppare programmi per Android. Infatti offre una ricca gamma di feature che rendono più agevole il lavoro, partendo dallo sviluppo, fino ad arrivare al *testing* del software prodotto.

Grazie ai numerosi *plug-in* di Eclipse, infatti, è possibile aggiungere al programma una macchina virtuale su cui provare le applicazioni appena scritte nell'interfaccia. Questo permette di vedere su diverse configurazioni possibili il funzionamento o meno del codice appena scritto.

Eclipse è un ambiente di sviluppo *open source* e ciò lo rende un ancor più valido strumento. Il fatto stesso che sia gratuito e multipiattaforma permette ad utenti di

qualsiasi sistema operativo di lavorare al proprio progetto prescindendo dalla macchina in cui ci si trova.

L'intero progetto è infatti da me stato scritto in questo ambiente di sviluppo in quanto, essendo ancora alle prime armi, quando ho cominciato avevo la necessità di tutti i suggerimenti *inline* che il programma fornisce, suggerimenti generati automaticamente dalla documentazione java. Mi sono avvalso anche della macchina virtuale che è stata strumento fondamentale almeno per le prime fasi di sviluppo.

1.4 Arduino

Arduino è un framework *open source* che permette la prototipazione rapida e l'apprendimento veloce dei principi fondamentali dell'elettronica e della programmazione. Si tratta di una piattaforma hardware o circuito stampato che integra un microcontrollore mediante pin connessi alle porte di I/O, un regolatore di tensione e una interfaccia USB.

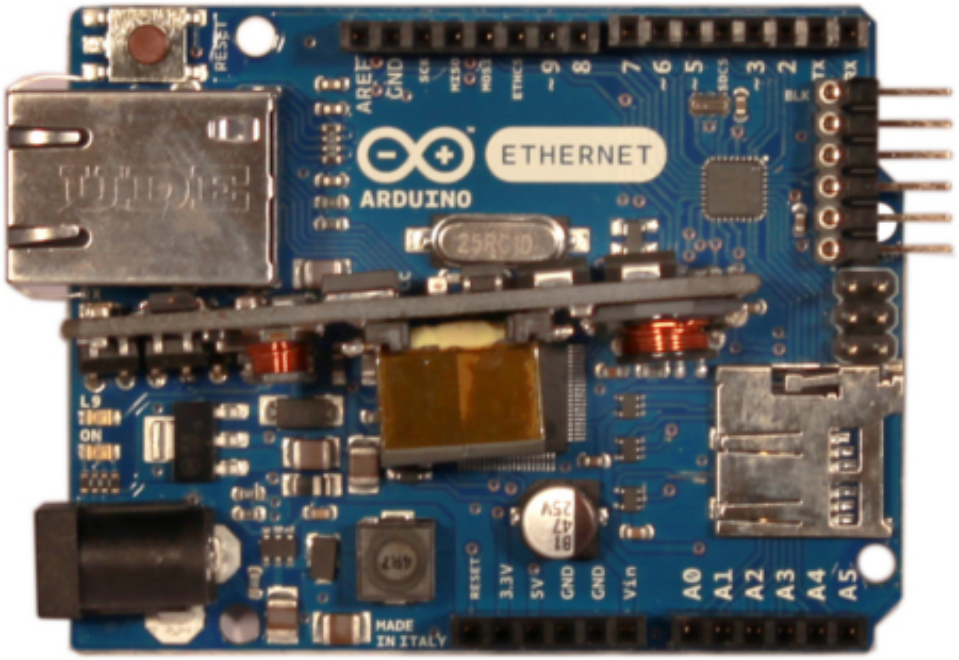

Figura 1.4: La board Arduino Ethernet with PoE

1.4.1 Storia

La piattaforma hardware è stata sviluppata presso l'*Interaction Design Institute*, da un gruppo di formazione post-dottorale con sede a Ivrea, fondato da Olivetti e Telecom Italia.[14]

Il nome della scheda deriva da un bar di Ivrea, frequentato da alcuni dei fondatori del progetto.[15]

Il team è formato da Massimo Banzi, Davi Cuartielles, Tom Igoe, Gianluca Martino e Davis Mellis. Il progetto nasce nel 2005 ad Ivrea con l'obiettivo di fornire agli studenti uno strumento più economico per realizzare progetti di *interaction design*.

1.4.2 Versioni

Vi sono tredici versioni del device, alcune delle quali completamente separate dal ramo principale.

Possiamo quindi individuare un filone principale che va dal Serial Arduino, Arduino Extreme, Arduino NG, Arduino NG Plus, Arduino Diecimila, Arduino Duemilaenove, Arduino Mega, Arduino Uno, Arduino Mega2560.

Molto simili ma con un approccio distinto si presentano invece LillyPad Arduino, Arduino Mini, Arduino Nano, Arduino Ethernet.

Tutti questi device hanno in comune queste caratteristiche: [16]

- Microcontrollore a 8-bit AVR con aggiunta di componenti complementari per facilitarne l'incorporazione in altri circuiti.
- Chip della serie megaAVR.
- Regolatore lineare di tensione a 5 volt.
- Oscillatore a cristallo a 16 MHz.

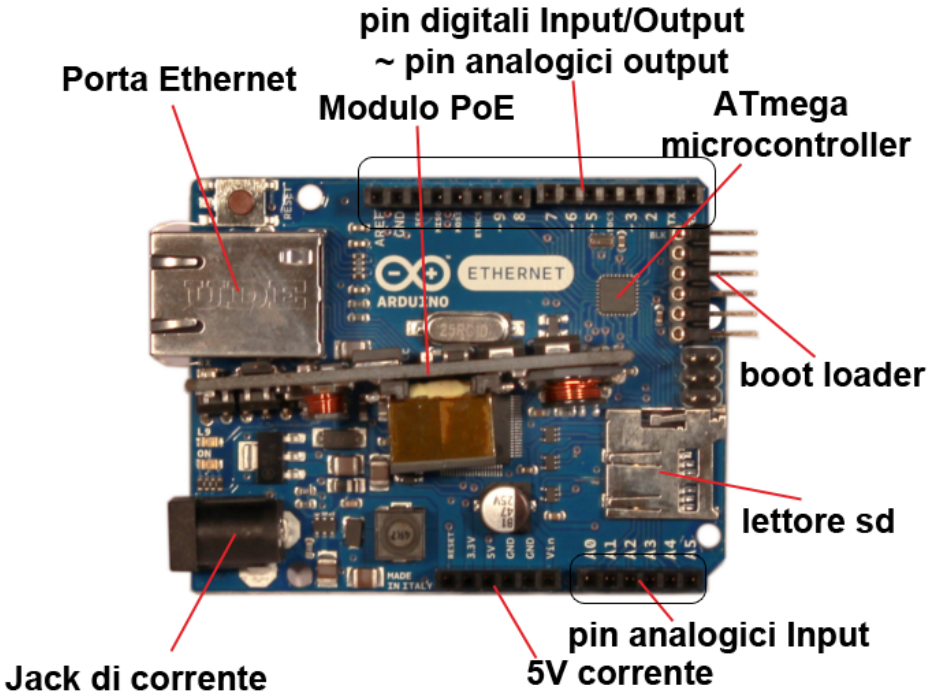

Figura 1.5: La board Arduino Ethernet with PoE scheda tecnica

La versione Arduino Ethernet ha alcune peculiarità che ne fanno il dispositivo d'eccezione di questa tesi. Il dispositivo è dotato di una porta Ethernet che ne permette la connessione ad un router fornendo la piattaforma di accesso alla rete internet. Altri significativi vantaggi sono la presenza *onboard* di un lettore di schede micro SD per poter archiviare i dati letti dai sensori e in alcune versioni un modulo PoE (*Power over Ethernet*) che permette di alimentare Arduino con la sola connessione ad internet.[17]

Il vero punto di forza di Arduino è la sua accessibilità: sia software che hardware sono *open source*, dando la possibilità di conoscere gli schemi tecnici del progetto per poterne apprendere appieno il funzionamento; pertanto si colloca come ottima piattaforma per apprendere ed imparare.

1.4.3 Sviluppo

L'ambiente di sviluppo integrato (IDE) di Arduino è una applicazione in java. Permette con semplicità di scrivere programmi anche a persone con una scarsa conoscenza della programmazione. Il codice infatti è ad alto livello e grazie ad i menu a tendina è possibile importare tutte le librerie necessarie. L'editor permette infine di compilare il programma e di eseguirlo su Arduino connesso al computer.

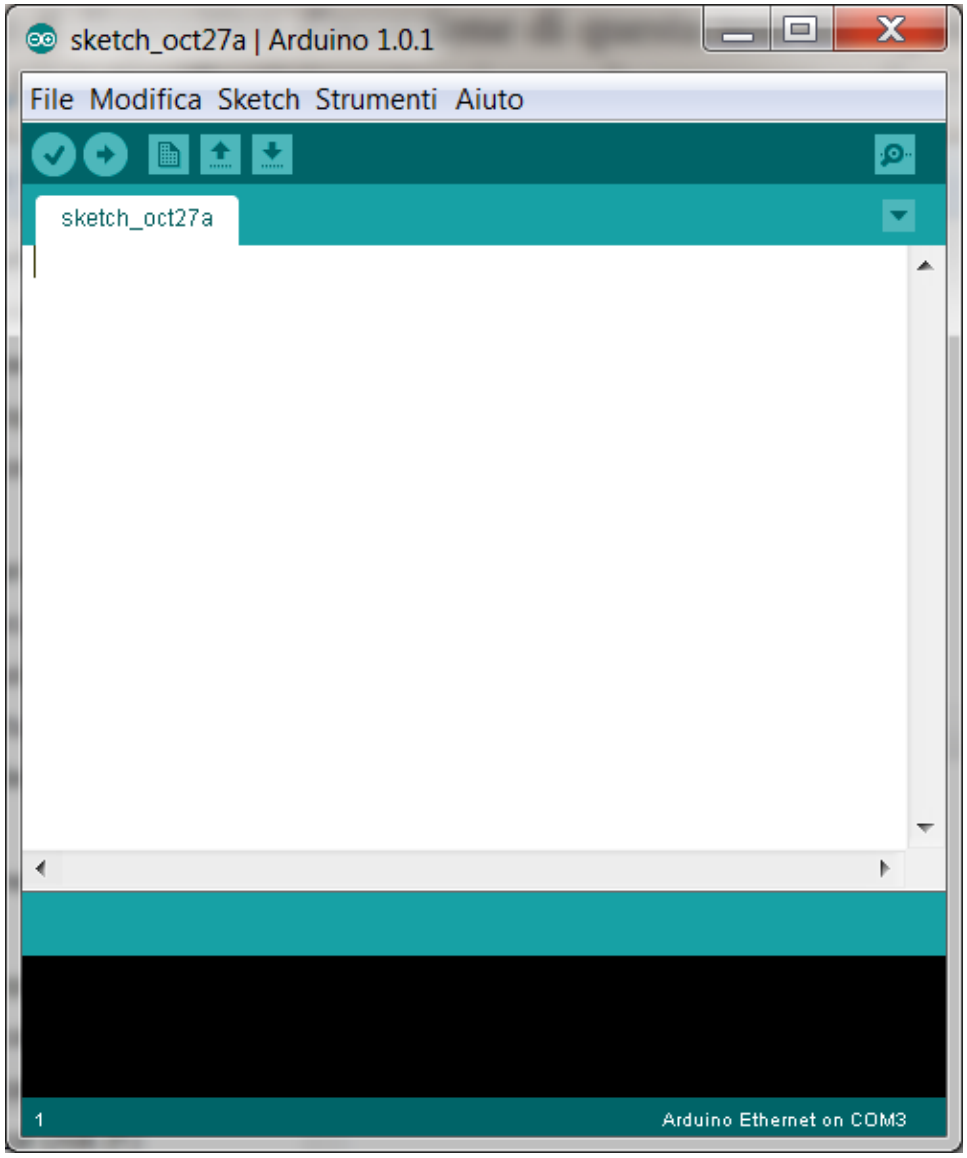

Figura 1.6: La board Arduino Ethernet with PoE scheda tecnica

Arduino può essere collegato tramite porta USB: infatti sulla maggior parte dei dispositivi è presente una porta USB o una porta seriale RS-232. Alcuni dispositivi invece usano una scheda o un cavo adattatore usb-seriale scollegabile dal dispositivo.

Sempre dalla piattaforma di sviluppo è possibile selezionare la porta alla quale è connesso il device e aprire un monitor seriale per valutare eventuali output di Arduino.

Le librerie sono molto ricche e con tantissime funzioni tra le più avanzate, ma le ridotte capacità di memoria del dispositivo non permettono grandi computazioni o programmi *realtime*. Ciononostante è possibile scrivere innumerevoli applicazioni e la ricca e virale flora di programmi reperibili online ne è la prova più evidente.

1.5 Arduino e Android

In occasione della conferenza Google I/O 2011, tenutasi il 10 e 11 maggio 2011 al Moscone Center di San Francisco, Google ha presentato una nuova piattaforma di sviluppo per Android basata su Arduino. [**?**]

Con questa scelta Google mostra di volere hardware basato su Arduino.

1.5.1 Amarino

Ben prima che Google annunciasse Arduino come piattaforma di sviluppo e prototipazione per applicazioni di controllo e gestione di apparecchiature per smartphone Android, un progetto indipendente aveva tentato con successo un approccio.

Nell'agosto 2010 nasce Amarino ovvero "*Android meets Arduino*", un primo incontro tra le due piattaforme. [19]

Si tratta di un *toolkit*, ovvero un insieme di strumenti software di base per facilitare e uniformare lo sviluppo di applicazioni derivate più complesse. In pratica consiste in una applicazione Android ed una libreria per Arduino: queste aiutano lo sviluppatore con interfacce grazie alle quali lo smartphone è in grado di comunicare con un dispositivo Arduino. La comunicazione tra i due dispositivi avviene grazie alla scheda *Bluethooth Shield* presente su Arduino. [20]

Amarino è composto da tre parti principali:

- Una applicazione Android chiamata Amarino.
- Una libreria per Android chiamata MeetAndroid.
- Un Plug-in opzionale.

Queste parti concorrono per permettere agli sviluppatori di scrivere applicazioni che accendano luci, innaffino piante, etc, direttamente dal dispositivo Android.

L'intero progetto è decisamente una buona soluzione al problema e presenta la peculiarità di essere il primo approccio. La scelta di effettuare la comunicazione tramite bluetooth però ne rappresenta la principale limitazione, infatti non tutti gli utenti mantengono acceso il bluetooth, che ha una funzione ormai sempre più marginale nei

moderni dispositivi, inoltre ha un costo in termini di batteria che non è controbilanciato dalla gittata del segnale di appena una decina di metri, tanto più ostacolato dai muri domestici.

Rimane però indiscussa l'importanza di questo progetto come dimostrazione che questo approccio è possibile e che è sentita la necessità di mettere in comunicazione questi dispositivi.

1.5.2 Soluzioni Personalizzate

Chiunque con una buona conoscenza dell'informatica è in grado di sviluppare una soluzione personalizzata al singolo problema di collegare un dispositivo Android ad un Arduino attraverso la rete internet.

Infatti esistono manuali che spiegano come accendere una luce o ricevere informazioni dall'ambiente circostante.

Queste soluzioni richiedono di conoscere il funzionamento dei protocolli di comunicazione TCP o UDP, della libreria Ethernet per Arduino, della libreria Socket per Android e rappresentano un approccio efficiente al problema.

D'altro canto questo approccio comporta una scarsa portabilità dei costrutti, che possono essere utilizzati solo per un ambito ristretto, e una altissima autonomia delle applicazioni, che non possono adattarsi alle mutevoli esigenze del cliente.

Soluzioni di questo tipo rappresentano la risposta ad un problema locale personale ma non possono diffondersi su larga scala.

Un ulteriore significativo elemento a sfavore di questo sistema è il costo in mesi-persona per scrivere un programma: se ogni soluzione è a sé stante, ogni tentativo di approccio al problema richiede una impiego di tempo non indifferente, specialmente per i neofiti.

Capitolo 2

Struttura del Progetto

2.1 Visione d'insieme e Obiettivi

2.1.1 Obiettivi

L'esigenza di questo progetto nasce anche dalla forte spinta alla domotica che si percepisce negli ultimi anni, pensiamo solo al Trentino Alto Adige che ha almeno un elemento di domotica nel 78% degli immobili di nuova costruzione. Ma la crescita è diffusa su tutto il territorio italiano con incrementi del 16% negli ultimi due anni. Ho quindi sentito l'esigenza di avere uno strumento per la prototipazione che tenesse in considerazione tutti questi fattori. [21]

Il mio lavoro ha come obiettivo ultimo quello di fornire agli sviluppatori un sistema semplice e veloce per scrivere programmi che permettano ad un device Android di manipolare lo spazio che lo circonda, ovvero di comunicare con un dispositivo Arduino ricevendo i dati dei suoi sensori e attivando gli attuatori eventualmente presenti. Valutando lo stato dell'arte ho verificato che non esiste un approccio al problema che soddisfi i requisiti di:

- Semplicità, intendendo sia la facilità d'uso per gli sviluppatori sia di impiego per gli utenti. Per questi ultimi in particolare il dispositivo mobile deve essere utilizzato in modalità standard, senza necessità di attivare il bluetooth o connettervi cavi o dover essere vicino alle apparecchiature da utilizzare.

- Portabilità del codice, senza dover ogni volta implementare tutto il programma per cambiare poche cose.

- Universalità: attivando questo sistema si potrebbe avere un vasto panorama di applicazioni che ne sfruttano le potenzialità, dando forza e ricchezza al progetto.

Grazie alla realizzazione di questo framework si potrà sviluppare un ampio spettro di applicazioni nell'ambito della domotica, sfruttando metodi ad alto livello e astrazioni sui dati.

2.1.2 Progetto

Per Android si è pensato di scrivere un programma demone, ovvero un programma sempre attivo sullo smartphone ma che non occupi una schermata principale e che lavori solamente in background. In questo modo l'utenza può svolgere altre operazioni senza doversi preoccupare dell'applicazione.

Questo programma attende in ascolto sulla rete la rete wifi aspettando i messaggi xml inviati dall'Arduino. Lo schema avviene come in figura.

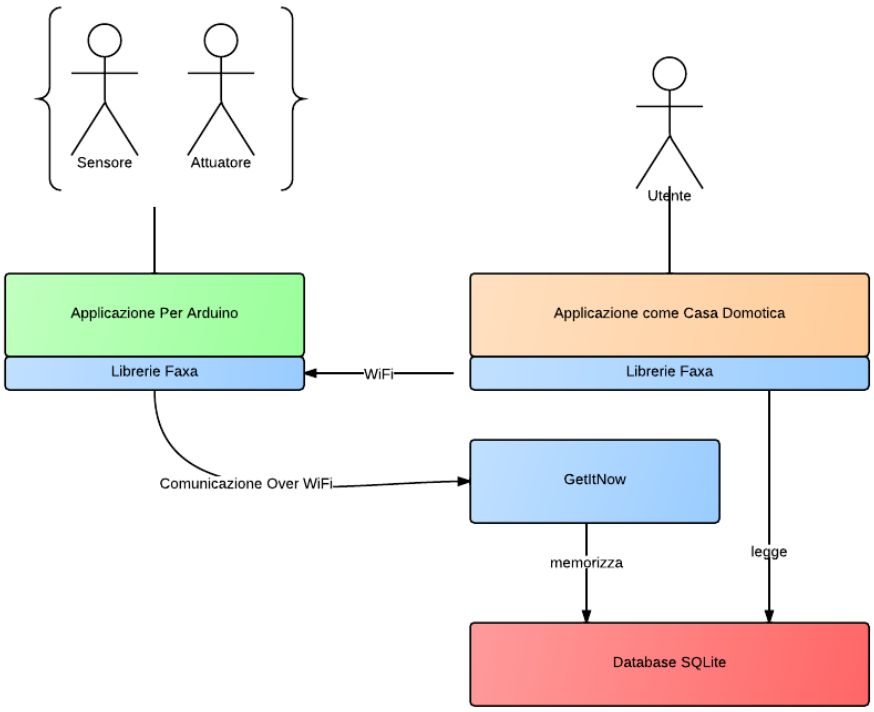

Figura 2.1: Struttura del Funzionamento di Faxa

Il demone GetItNow trae dall'xml le informazioni che poi va a memorizzare in un database SQLite.

Le informazioni presenti nel database sono disponibili a tutte le applicazioni del dispositivo che ne fanno richiesta, chiunque può scrivere un programma che manipola questi dati, con una singola query si ottengono i valori di tutti i sensori presenti sull'Arduino. [22]

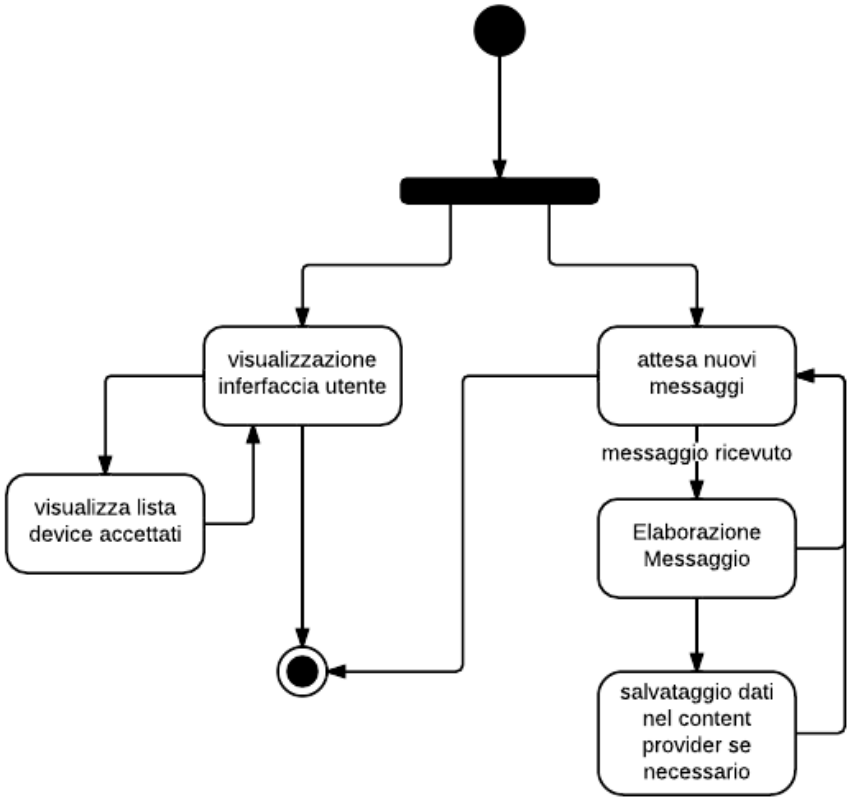

Figura 2.2: Diagramma di Stato di GetItNow

Il diagramma di stato in figura mostra il flusso del programma GetItNow. Inizialmente il programma si divide in due *thread*, quello principale, che è un thread in background, si occupa della ricezione degli xml e un thread secondario, che è un interfaccia utente, gestisce il programma.

Questa interfaccia permette alcuni semplici operazioni: filtrare i dispositivi Arduino in base all'id e decidere l'ip al quale mettersi in ascolto.

Invece per attivare gli attuatori si usa la classe Faxa per Android: essa è una libreria con alcune semplici funzioni che permettono di inviare ordini ad un indirizzo della rete e, se a quell'indirizzo è collegato un Arduino configurato in modo opportuno, è possibile grazie a questa libreria manipolare il dispositivo da remoto.

Sul lato Arduino invece è stata scritta la libreria Faxa per Arduino la quale, grazie ad alcuni metodi, permette di ricevere ordini remoti e inviare i dati dei sensori in formato xml.

I dati vengono inviati tramite il protocollo UDP, che è un servizio *best effort*: questo rende più rapida la comunicazione e permette la multiutenza.

Per mettere alla prova questo framework da me sviluppato ho progettato due applicazioni.

Per Android ho sviluppato l'applicazione Casa Domotica, un'interfaccia di semplice utilizzo per la gestione degli apparecchi elettronici domestici.

Per Arduino ho sviluppato un programma che permette di mappare tutti i pin del dispositivo e inviare i loro valori in rete.

2.1.3 Contributo Personale

Indispensabile dopo aver fissato i casi d'uso a cui il progetto si attiene e le entità coinvolte dare una precisazione sul mio contributo personale.

Esso si compone di:

- Progettazione e modellazione del programma Casa Domotica, come prototipo dell'applicazione mobile. Costituisce la dimostrazione del progetto, poichè in grado di comunicare con Arduino e fornire gli stati dei device connessi.
- Progettazione e modellazione del programma GetItNow.
- Progettazione e modellazione del programma BroadcastPin per Arduino.
- Valutazione, definizione e adattamento di librerie ed API necessarie per lo sviluppo dei componenti di questo progetto con conseguente sviluppo delle libreria Faxa sia per Arduino che per Android.
- Progettazione e modellazione di un'ontologia per la comunicazione via wifi.
- Testing su vari dispositivi per verificare il corretto funzionamento del progetto.

Capitolo 3

Faxa - Framework Android xml Arduino

3.1 Libreria per Arduino

Per Arduino si è scelto di implementare una libreria che offra allo sviluppatore un insieme di funzioni di alto livello.

3.1.1 Scelte Implementative

Per poter mettere in comunicazione il dispositivo Arduino con Android attraverso l'etere si è scelto Internet come via migliore, questo per alcuni motivi chiave:

- in primo luogo l'utente di uno smartphone ha sempre libero accesso ad Intenet ed è invogliato a mantenere attiva questa interfaccia per altri scopi, quindi non ha consumi ulteriori di batteria.
- la possibilità di avere una gestione anche solo locale attraverso il wifi se non si vuole aprire il servizio a terzi.
- l'efficenza e la gittata del segnale wifi.
- la possibilità di gestire in ogni luogo i propri apparecchi attraverso Internet.

Avendo per questi motivi deciso di gestire la comunicazione attraverso la rete Internet ovvero il wifi, si è scelto di effettuare scambi di pacchetti UDP.

La scelta del pacchetto UDP è stata una necessaria conseguenza delle limitate potenzialità dell'Arduino.

Infatti per gestire la multiutenza, si sarebbe dovuto aprire un thread parallelo per ogni connessione all'Arduino ma questo non è possibile poichè Arduino è *Single Thread* pertanto si sarebbe dovuto simulare il *MultiThreading* perdendo in prestazioni.

In secondo luogo non è nemmeno vantaggioso avere una comunicazione punto punto poichè si preferisce avere risposte rapide e immediate, piuttosto che attese per il *three way handshake* per instaurare una comunicazione sicura. Infine se l'Arduino è configurato per emettere i dati ogni secondo, non essendo un sistema *real time* un *delay* di pochi secondi non è visibile ad occhio nudo.

Arduino offre già una libreria per il controllo della rete Internet: infatti la libreria Ethernet.h permette la gestione delle interfacce di rete.

Proprio su questa prima libreria è stata indirizzata la mia scelta, ho però visto che non era sufficente per i miei scopi e che era necessario usare anche la libreria EthernetUdp.h per trasmettere un pacchetto UDP sulla rete Internet.

Il pacchettò così inviato però non gode di una struttura uniforme, infatti la modalità con la quale strutturare i dati inviati all'interno del pacchetto UDP può variare in base alle esigenze dello sviluppatore. Per dare una versione omogena ho deciso di implementare la libreria Faxa.

La scelta è caduta sull'xml che è semplice da usare, può essere manipolato sia come String che come Oggetto, ed esiste un Parser per ogni linguaggio di programmazione esistente.

Le ridotte capacità di memoria dell'Arduino mi hanno inoltre fatto capire che era necessario qualcosa di semplice da manipolare.

```
<?xml version='1.0' encoding='utf-8'?>
<arduino id="938" >
        <info type="PIN_0" value="145" />
        <info type="PIN_1" value="0" />
        <info type="PIN_13" value="1023" />
</arduino>
```

Vediamo quindi la struttura dell xml, la radice è Arduino che ha come attributo l'ID del dispositivo Arduino.

L'ID può essere scelto dallo sviluppatore oppure venir generato autonomamente dal dispositivo.

Questo permette la disambiguazione tra dispositivi diversi nella stessa sottorete, e anche permette ai device Android di filtrare i dispositivi Arduino.

Ogni nodo *info* contiene nei suoi attirbuti il tipo *type* di informazione che gestisce e il valore mappato tra 0 e 1024.

Teniamo presente che si è scelto di dare come tipo la chiave PIN_0 ma si può anche decidere di dare altri valori per specificare esattamente di cosa tratta quell'informazione: nel nostro caso ho scelto di assegnare un oggetto specifico a questo pin ma di lasciare all'applicazione Android la decisione. Maggiori dettagli sull'ontologia nel capitolo 3.1.2.

Per quanto riguarda la ricezione di comandi ho invece deciso di non affidarmi all'xml poichè non esiste un parser nativo per xml per Arduino. Esistono alcuni parser sviluppati a seguito per Arduino ma ho preferito ridurre il costo di computazione dei dati inviati ed evitare ulteriori sprechi in termini di tempo di computazione e spazio occupato, perciò ho pensato a una serie di triple così composte.

Soggetto verbo complemento

Il soggetto è il numero corrispondente ad uno dei 13 PIN di output.

Il verbo è l'azione da compiere su uno di questi PIN ovvero un numero tra 0 256. Dove 0 significa spento mentre 256 significa acceso.

Il complemento o durata è quanto tempo deve essere mantenuta questa operazione prima di passare alla successiva. Il valore va espresso in millisecondi.

Ad esempio la tripla 3 0 0 significa "Spegni il PIN_3 e non aspettare per la prossima operazione", mentre la tripla 3 255 1000 significa "Accendi il PIN_3 e aspetta un secondo prima della prossima operazione".

3.1.2 Ontologia

Nel xml sono presenti, all'interno del nodo info, l'attributo *type*; questo attributo accetta come parametro uno tra i seguenti valori presenti in questa ontologia:

temperature: temperatura espressa in gradi Celsius

light: luminosità espressa tra 0-1024

noise: rumore espresso tra 0-1024

speed: velocità di movimento espressa in m/s

pressure: pressione della stanza espressa in bar

map: url di una mappa di google

menu: menu del ristorante espresso come un insieme di coppie

text: testo libero

seating: numero di posti a sedere disponibili o più generalmente il numero di posti disponibili

offers: testo libero contenente offerte commerciali

context: contesto nel quale ci si trova. Esempi possibili per il campo value sono: hospital, restaurant, reunion, common room,...

PDI: punto d'interesse. Esempi possibili per il campo value sono: Cremlino, Santa Sofia, Colosseo.

password: password espressa come testo

music: titolo di una canzone

gps: coordinate gps del locale

app: link al *Google Play Store*

weather: condizioni meteo espresse come testo

services: elenco dei pin disponibili ai quali è connesso un input

PIN_0: uso generico per indicare il PIN_0 dal quale si riceve il valore

PIN_*: (* qualsiasi numero) indica un PIN generico al quale è connesso il dato ricevuto

Questa ontologia rappresenta solo uno standard per garantire compatibilità tra i vari sviluppatori e la possibilità di fare applicazioni personalizzate che possano sfruttare questi dati in più ambiti diversi. Nulla però vieta lo svilupparsi di un nuovo standard *de facto*, basato sulle molteplici e mutevoli esigenze emerse nel tempo. Potranno nascere nuove chiavi di tendenza che diventeranno cardini di questa ontologia in futuro. Tuttavia allontanandosi dallo standard per qualunque esigenza, è necessario tenere a mente che le applicazioni sviluppate non risulteranno compatibili con quelle *standard-compliant*.

3.1.3 Guida all'uso

Le librerie Faxa offrono agli sviluppatori alcuni agi per scrivere rapidamente il codice necessario a controllare i device connessi al dispositivo Arduino.

Per prima cosa è necessario inizializzare la classe Faxa con il comando

Listing 3.1: Estratto del codice, BroadcastPin

```
#include <Faxa.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
 Faxa faxa(1000);
void setup(){}
void loop{}
```

Con questo comando si dichiara la volontà di usare la classe Faxa e la si inizializza con il valore 1000 poichè tra un invio e l'altro si chede che debba passare almeno un secondo.

All'interno del setup sarà necessario inizializzare i costrutti della libreria, per fare questo bisogna chiamare la funzione init();

Listing 3.2: Estratto del codice, BroadcastPin

```
void setup()
{
        faxa.init();
}
```

In questo modo vengono inizializzare le interfacce di rete e viene predisposto il monitor seriale.

In questa fase viene anche generato l'ID univoco del dispostivo Arduino.

Per definire manualmente l'ID del device, si può chiamare la funzione:

Listing 3.3: Esempio, Cambiare id

```
int valore=125;
void setup()
{
        faxa.init();
        faxa.setUniqueId( valore );
}
```

Normalmente Arduino invia in broadcast il file xml all'indirizzo "192.168.1.255" alla porta "9876".

Ma è possibile riconfigurare sia l'indirizzo sia la porta con le seguenti istruzioni:

Listing 3.4: Esempio, Cambiare ip e porta di destinazione

```
IPAddress ip_add(2,14,234,104);
```

```
int port=7777;
void setup()
{
        faxa.setRemoteIpAddress( ip_add );
        faxa.setRemotePort( port );
        faxa.init();
}
```

Ora che abbiamo configurato correttamente il dispositivo possiamo passare all'invio di dati in *broadcast*.

Utilizzando i seguenti comandi si chiede al dispostivo di aggiungere i dati recepiti dai sensori nel xml. Con questi comandi non si inviano dati attraverso la rete ma ci si assicura che i dati letti dai sensori siano memorizzati nel file xml:

Listing 3.5: Estratto del codice, Faxa.h

```
        int addDigitalValue(String type,int pin);
        int addAnalogValue(String type,int pin);
        String addStringValue(String type,String value);
        int addIntValue(String type,int value);
```

Ognuna delle precedenti istruzioni ha un utilizzo e dei parametri specifici.

La *addDigitalValue* riceve in input una coppia nome del valore digitale selezionato "type", e il pin della board Arduino al quale è connesso il sensore.

Nella board Arduino Ethernet i pin da 0 a 13 sono parametri validi da passare alla *addDigitalValue()*.

La funzione *addAnalogValue* riceve in input gli stessi parametri con le stesse modalità, eccettuato il fatto che la board Arduino Ethernet ha i pin analogici che vanno da 0 a 5.

Nell'esempio seguente aggiungeremo all'xml i dati di un sensore di luminosità e poi li invieremo in broadcast.

Listing 3.6: Esempio, Inviare la luminostà

```
void setup()
{
        faxa.init();
}
void loop()
{
        faxa.addAnalogValue("light",0); //funzione che aggiunge i dati al xml
        faxa.sendXml(); //funzione che invia il file xml
}
```

A cui fa riferimento lo schema logico in figura 3.1 .

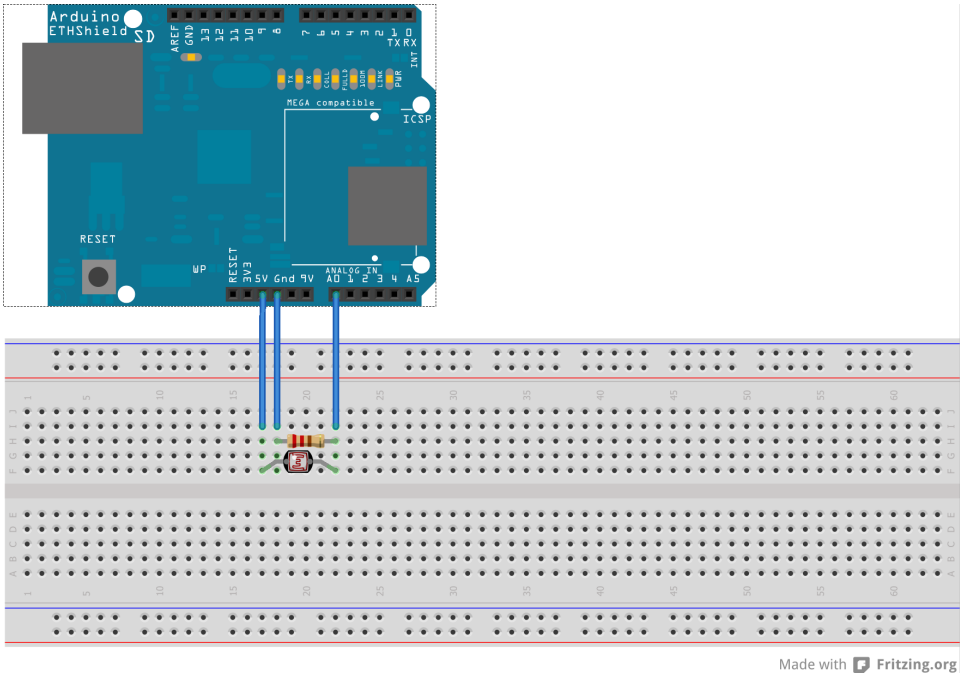

Figura 3.1: Schema logico sensore luminosità

Per ricevere comandi invece è presente la funzione così denominata:

Listing 3.7: Esempio, Ricevere un Ordine

```
boolean ordine_ricevuto=false;
void loop()
{
        ordine_ricevuto= faxa.reciveRawOrder();
}
```

Con questa istruzione è possibile ricevere un singolo ordine e elaborarlo. L'ordine dovrà essere nella forma a tripla, soggetto verbo complemento, e dovrà essere composto da soli numeri.

La chiamata non è bloccante, infatti se non vi sono ordini in arrivo il metodo restituisce *false* e continua.

In questo modo si riceve un solo ordine ogni ciclo di computazione. Nel caso si volesse ricevere tutti gli ordini in arrivo ed elaborarli immediatamente basta inserire un ciclo *while*.

Listing 3.8: Esempio, Ricevere una sequenza di Ordini

```
boolean ordine_ricevuto=false;
void loop()
{
        ordine_ricevuto= faxa.reciveRawOrder();
        while(ordine_ricevuto)
                ordine_ricevuto= faxa.reciveRawOrder();
}
```

3.1.4 BroadcastPin

Come abbiamo già detto, BroadcastPin non è altro che una delle possibili implementazioni che sfruttano le librerie Faxa.

Il programma attende tutti gli ordini in arrivo e li esegue finchè non li ha esauriti. Fatto ciò aggiunge all'xml il valore di 6 PIN 4 digitali e 2 analogici, infine invia tramite UDP i dati così immagazzinati.

Listing 3.9: Codice Sorgente , BroadcastPin

```
#include <Faxa.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
Faxa faxa(1000);
boolean ty;
void setup()
{
faxa.init();

}
void loop()
{
        ty= faxa.receiveRawOrder();
        while(ty)
                ty= faxa.receiveRawOrder();
        faxa.addAnalogValue("PIN_13",0);
        faxa.addAnalogValue("PIN_14",1);
        faxa.addDigitalValue("PIN_2",2);
        faxa.addDigitalValue("PIN_3",3);
        faxa.addDigitalValue("PIN_4",4);
```

```
        faxa.addDigitalValue("PIN_5",5);
23      faxa.sendXml();
}
```

Lo schema logico relativo a questo programma da montare su Arduino Ethernet è il seguente.

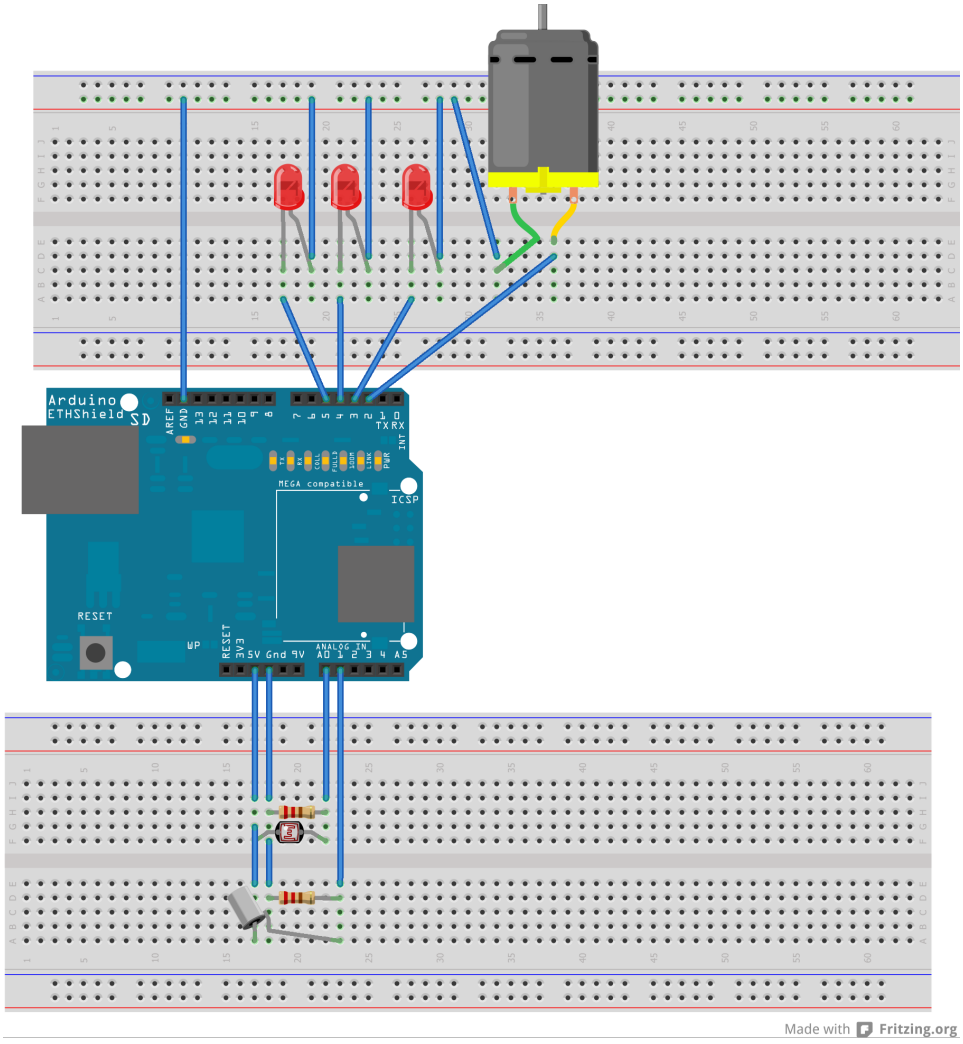

Figura 3.2: Schema logico BroadcastPin

3.2 Framework Android

Su Android invece era necessario qualcosa in più oltre alla libreria: ho quindi deciso di scrivere un'applicazione demone, che si occupasse della ricezione dei messaggi ed una libreria per inviare le risposte.

3.2.1 Scelte Implementative

Si è visto necessario semplificare al minimo la recezione dei messaggi inviati da ogni possibile Arduino. Inizialmente avevo pensato ad una soluzione locale, ove ogni applicazione si occupava della ricezione dei messaggi di cui aveva bisogno.

Questa soluzione mi è sembrata scorretta per alcuni motivi:

- non è trasferibile ad altre applicazioni, infatti ogni applicazione deve comunque decidere arbitrariamente come memorizzare e sfruttare i dati che riceve.
- non è omogenea, anche se si unificasse il salvataggio l'applicazione potrebbe alterare i dati che riceve prima di memorizzarli.
- non è efficiente, più applicazione aperte nello stesso momento che si occupano di dati diversi lavorano parallelamente in background occupando la memoria inutilmente.
- richiede più di qualche istruzione di codice da scrivere, e visto che il tentativo è quello di semplificare l'approccio al problema non faceva al caso mio.

Pertanto ho scelto di scrivere un'applicazione che si occupasse della ricezione di ogni possibile messaggio, i messagi vengono salvati così come sono, senza nessuna elaborazione intermedia, in questo modo il dato non viene corrotto. I messaggi ricevuti vengono salvati in un *content provider* ovvero un database *SQLite*. In questo modo ogni applicazione può fare riferimento a questi dati, elaborarli e riceverli semplicemente leggendo da un database.

- è scalabile, ogni applicazione può riusare il codice della precedente per leggere i dati.
- è omogenea, tutte le applicazioni leggono i dati allo stesso modo e leggono gli stessi dati.
- è efficente, più applicazioni aperte nello stesso momento che si occupano di dati diversi lavorano parallelamente solo per una lettura locale del dato, con tempi di accesso rapidi. Il demone invece si occupa della recezione di ogni dato dalla rete.

- richiede solo l'istruzione di lettura da un database, una qualsiasi guida *SQLite* è più che sufficiente per cominciare a ricevere dati remoti.
- privilegio minimo, le applicazioni possono leggere dati remoti senza per questo avere l'accesso ad internet.

Questa soluzione però ha dei possibili svantaggi:

- è necessario installare GetItNow sul dispositivo e lanciarlo nel momento in cui si vuole ricevere dati.
- qualora GetItNow presentasse dei bug, tutte le applicazione che fanno riferimento a questo demone ne sono affette.
- GetItNow rimane in eseguzione anche quando non ci sono applicazioni che fanno uso di quei dati.

Questi svantaggi però rappresentano il rovescio della medaglia, conseguenze necessarie ad una così alta astrazione e possono essere risolti in future *release* oppure attuando alcuni stratagemmi, come i seguenti:

- le applicazioni possono fare richiesta di installare GetItNow qualora non fosse presente.
- la natura OpenSource del prodotto dovrebbe ridurre il rischio di bug e il loro tempo di risoluzione a zero.
- si potrebbe aggiungere un contatore che viene incrementato da ogni applicazione che fa riferimento a GetItNow e che decresce ogni qual volta l'applicazione viene chiusa, in questo modo quando ogni applicazione è chiusa anche GetItNow si chiude. Questo sistema non è stato però inserito in questa versione perchè dipende troppo dalla buona condotta degli sviluppatori futuri: se qualcuno dimenticasse di aggiornare il contatore si perderebbe ogni beneficio di questo sistema, in oltre può essere interessante valutare l'ultimo dato ricevuto dal sensore, anche quando nessuna applicazione era in ascolto.

Facendo riferimento alla figura 2.2 si evince chiaramente il funzionamento del demone, infatti appena lanciato, il programma si occupa di avviare l'*Intent* di un servizio remoto.

Listing 3.10: Esempio, Ricevere una sequenza di Ordini

```
2 Intent myIntent = new Intent(getApplicationContext(), BroadcastWaitingService.class);
startService(myIntent);
```

La classe *BroadcastWaitingService.class* è la classe che si occupa dell'avvio del *Thread* principale che in figura 2.2.

Osservando il Diagramma delle Classi figura 3.3 si evince la struttura del programma. La classe chiave è ovviamente la *BWThread* iniziallizzata da BroadcastWaitingService. Questa classe si occupa di generare il *content provider* e di rifornirlo costantemente dei nuovi dati in arrivo.

Android Developer così definisce il *content provider [23]*

> *Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.*
>
> http://developer.android.com/guide/topics/providers/content-providers.html

Il *content provider* è quindi un database che permette la connessione dei dati da un processo che sta lavorando con un altro processo. Quindi i dati memorizzati in questo *content provider* sono disponibili a tutti i processi che ne fanno richiesta e permangono anche dopo la chiusura del processo che li ha generati.

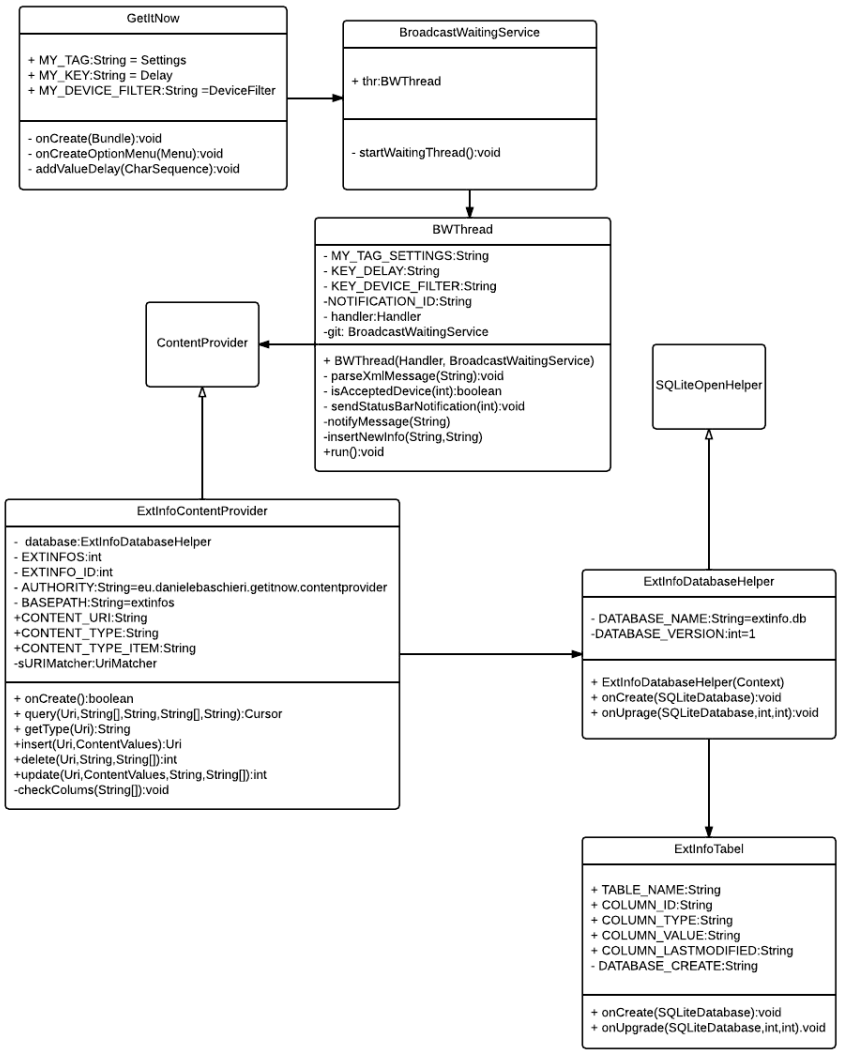

Figura 3.3: Diagramma delle Classi GetItNow

Terminate queste premesse sul demone possiamo passare alla libreria Faxa per Android: questa è composta da alcune funzioni chiave per l'invio di ordini sulla rete. Ho scelto di scrivere questa libreria per far fronte alle diverse possibili gestioni degli ordini,

fornendo uno strumento che con un'istruzione permetta di comunicare con un Arduino attraverso la rete.

Ho deciso di svincolare le modalità di invio da quelle di ricezione perchè non ho trovato vantaggi nell'avere una soluzione unita, infatti dare in gestione l'ordine di invio a GetItNow comporta una perdita di tempo non giustificata, dovuta alla scrittura e alla successiva lettura di un database SQLite.

Pertanto ho mantenuto le due modalità completamente separate, e a sé stanti. La comunicazione in invio avviene direttamente.

3.2.2 Guida all'uso

Per poter ricevere un dato è necessario effettuare i seguenti passaggi:

aggiungere al progetto l'uri del content provider.

Listing 3.11: Esempio, Ricevere un dato su android

```
private Uri CONTENT_URI =
        Uri.parse("content://eu.danielebaschieri.getitnow.contentprovider/extinfos");
```

Per fare ciò si può utilizzare il comando Faxa.getContentUri(); che restituisce l'uri del database SQLite.

A questo punto è possibile fare query al database per ottenere i dati dei sensori.

Listing 3.12: Esempio, Ricevere un dato su android

```
int pin=0;
String[] mProjection =null;
String mSelectionClause="type=?";
String mSelectionArg[]={"PIN_"+pin};
String mSortOrder=null;
Cursor mCursor = // The content URI of the words table
        git.getContentResolver().query(CONTENT_URI,
        mProjection, // The columns to return for each row
        mSelectionClause, // Selection criteria
        mSelectionArg, // Selection criteria
        mSortOrder);
if (null == mCursor) {
        Log.println(100,"IT_ERROR","IT_#003_database_assente");
}
else
        if (mCursor.getCount() < 1) {
```

```
                Log.println(100,"IT_ERROR","IT_#004_database_vuoto");
        }
            else {
                //Here you can use Cursor to get data
                int ime= mCursor.getColumnCount(); // obtain the number of column
                for (mCursor.moveToFirst();
                        !mCursor.isAfterLast();
                        mCursor.moveToNext()){
                                String value = mCursor.getString(Faxa.VALUE_FIELD);
                                String date= mCursor.getString(Faxa.DATA_FIELD);
                }
        }
```

In questo modo siamo riusciti a ottenere il valore del campo PIN_0 e la sua data, ovvero l'orario e il giorno in cui è stato modficato.

La variabile *mSelectionClause* permette di discriminare su quale colonna effettuare il filtraggio dei dati, nel nostro caso si è scelto di filtrare per *type*.

Il valore del filtro selezionato è contenuto nel vettore *mSelectionArg*.

Volendo invece ottenere il solo valore si può usare il medoto di libreria *Faxa.getValueFromType(Contesto , Tipo);* questo metodo è decisamente più semplice ma è più rigido e non permette richieste multiple di valori. Adatto solo per i primi test sul funzionamento.

Per poter invece inviare un comando bastano poche istruzioni per richiamare i metodi necessari.

Listing 3.13: Esempio, Inviare un singolo Ordine

```

Faxa.SendUDPCommand(int pin,int status,int delay);
Faxa.SendUDPCommand(int pin,int status);
```

Le due istruzioni presentate rappresentano ciascuna un modo per mandare un comando ad Arduino. La forma del comando è strutturata nella tripla soggetto verbo complemento.

pin il soggetto, rappresenta il pin d'accendere.

status un numero compreso tra 0 e 255.

delay rappresenta il tempo da attendere dopo aver compiuto questa operazione prima di eseguire la successiva.

Capitolo 4

Casa Domotica

Casa Domotica è una applicazione pensata per smartphone con sistema operativo Android, versione superiore o uguale alla 2.2 Froyo. Si propone di aiutare l'utente nella gestione della vita domestica, fornendo uno strumento intelligente e immediato per monitorare e controllare gli apparecchi elettronici.

4.1 Progetto e Implementazione

4.1.1 Casi d'uso

Casa Domotica nasce con il preciso obiettivo di mettere in luce le potenzialità del framework Faxa, pertanto mostrando le funzionalità di questo programma si riesce a vedere la grande varietà dei problemi che possono essere risolti grazie a Faxa.

Questo programma ha l'intento di diventare un "telecomando" o meglio uno "*smart assistant*" per la gestione della propria casa, offrendo lo stato dei sensori connessi, e la possibilità di manipolare con la voce o manualmente ogni dispositivo connesso.

Pertanto :

- l'utente deve poter visualizzare lo stato degli apparecchi connessi ad Arduino.
- l'utente deve poter variare con il tocco lo stato dei dispositivi connessi ad Arduino.
- l'utente deve poter variare con la voce lo stato dei dispositivi connessi ad Ardudino.
- l'utente deve poter aggiungere dispositivi .
- l'utente deve poter rimuovere dispositivi.

- l'utente deve poter decidere a che pin è connesso il dispositivo.
- l'utente deve poter rimappare i valori di un dispositivo.
- l'utente deve poter variare le parole chiave legate ad un dispositivo.
- l'utente deve ricevere un feedback vocale quando ha effettuato una richiesta vocale.

Più emblematica e significativa è la seguente immagine.

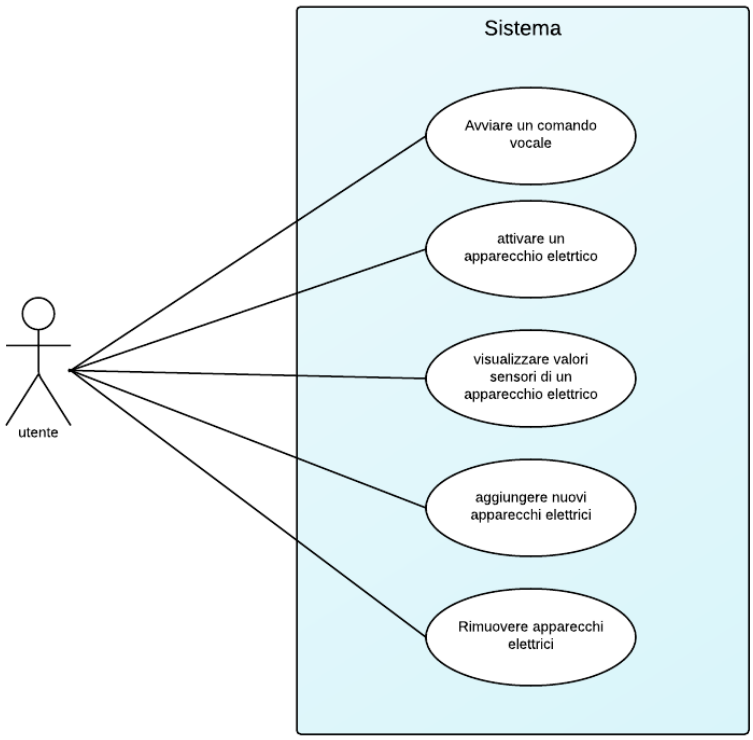

Figura 4.1: Diagramma dei casi d'uso

4.1.2 Identificazione delle principali entità

Lo studio delle funzionalità del software effettuato ha consentito l'individuazione delle principali entità che operano all'interno del sistema.

Da ciò segue il presente modello di dominio

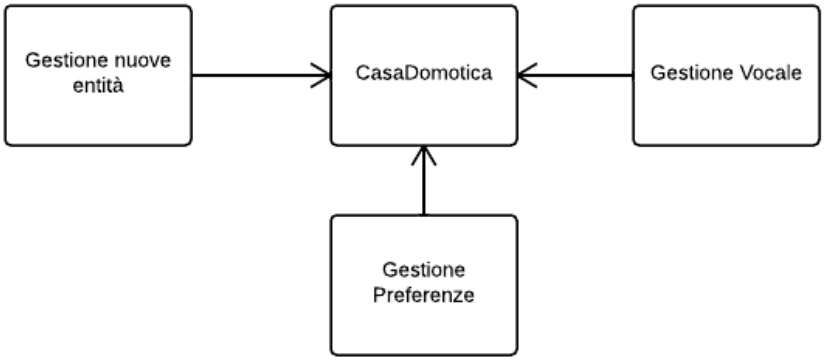

Figura 4.2: Modello di dominio

4.1.3 Progettazione di sistema

Di seguito maggiori specifiche di cosa si occupa ciascun dominio.

CasaDomotica

- Responsabilità: mostrare i dati dei sensori, permettere l'attivazione dei dispositivi, permettere l'avviamento dei comandi vocali, permettere la creazione di nuove entità, permettere la gestione delle preferenze.
- Interazione: modulo di creazione nuove entità, modulo gestione preferenze, modulo dei comandi vocali.
- Interfaccia: riceve e restituisce i dati sui sensori rendendoli visibili agli utenti.
- Coesione: sequenziale in quanto presenta una serie di passi.
- Accoppiamento: dati, in quanto l'unica sua funzione è quella di operare sui dati.

Modulo di creazione nuove entità

- Responsabilità: gestire la creazione di nuove entità, salvare queste entità nelle *shared preference*.

- Vincoli: necessita l'esistenza delle *shared preference.*
- Interazione: modulo di casa domotica.
- Risorse Necessarie: *shared preference.*
- Interfaccia: riceve e restituisce i dati.
- Coesione: sequenziale.
- Accoppiamento: dati poichè la sua funzione è operare sui dati dell'utente.

Gestione Preferenze

- Responsabilità: permette di impostare le preferenze dell'utente.
- Vincoli: necessita l'esistenza delle *shared preference.*
- Interazione: modulo di casa domotica.
- Risorse Necessarie: *shared preference.*
- Interfaccia: riceve e restituisce i dati.
- Coesione: sequenziale.
- Accoppiamento: dati poichè la sua funzione è operare sui dati dell'utente.

Gestione Vocale

- Resposabilità: permette di lanciare un comando grazie alla voce.
- Vincoli: necessita dei permessi di lettura dell'audio.
- Interazione: modulo di casa domotica.
- Risorse Necessarie: permessi Android, un microfono.
- Interfaccia: riceve il segnale audio.
- Coesione: sequenziale.

4.1.4 Diagramma di stato

Il diagramma nella seguente pagina illustra gli stati e le loro possibili transizione all'interno della applicazione.

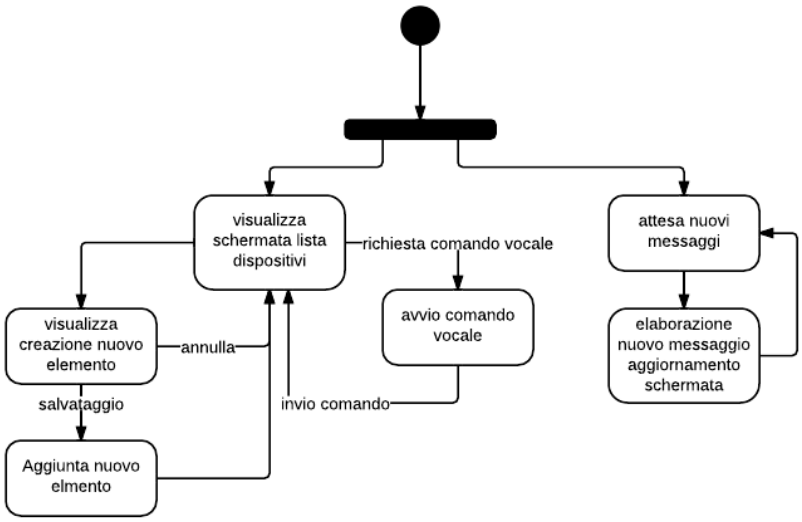

Figura 4.3: Diagramma di stato

4.1.5 Diagramma di sequenza

Di seguito sono riportati i diagrammi di sequenza dell'applicazione.

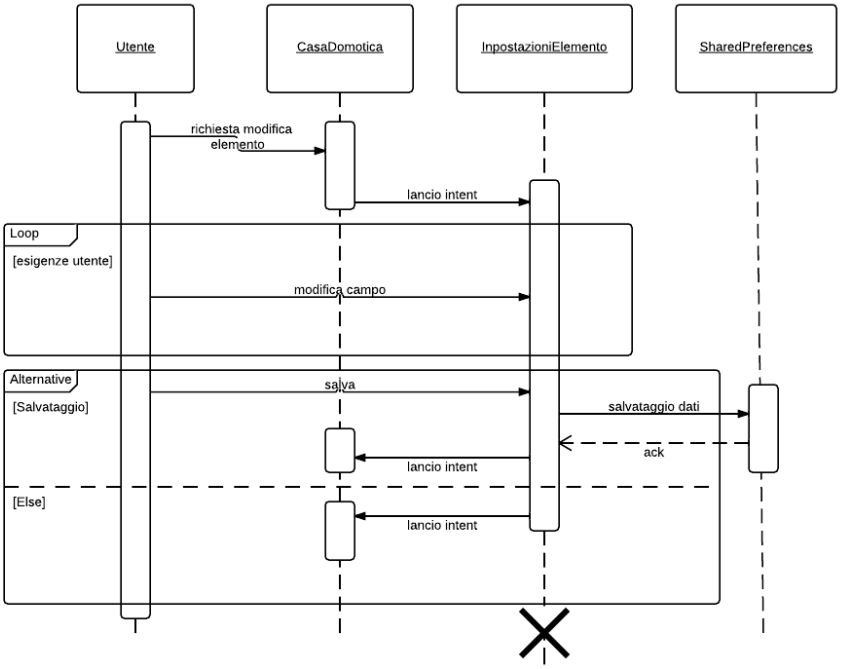

Figura 4.4: Diagramma di sequenza aggiunta nuovo elemento

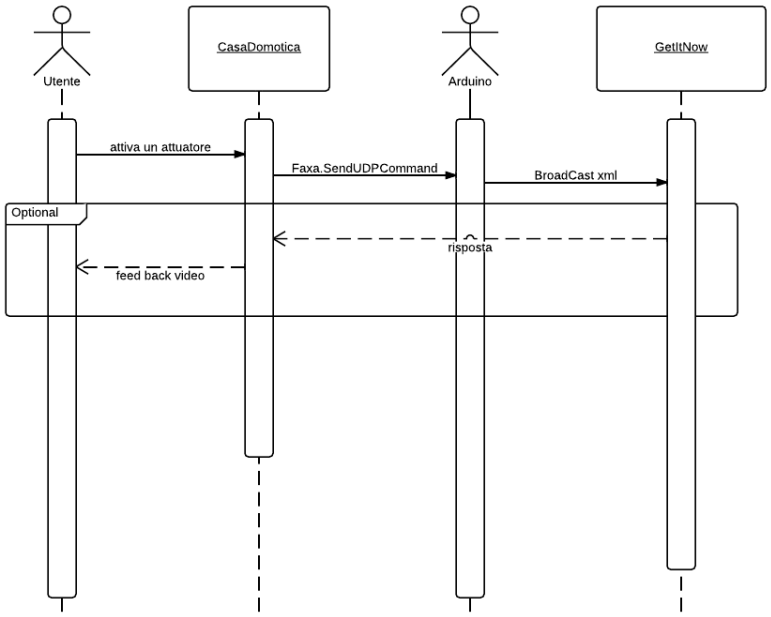

Figura 4.5: Diagramma di sequenza esegui un comando

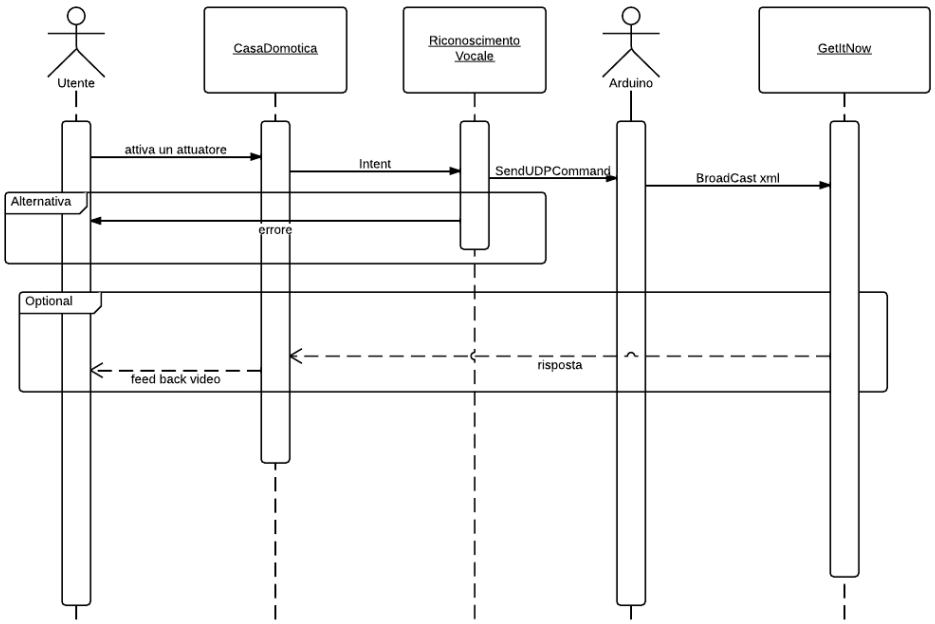

Figura 4.6: Diagramma di sequenza esegui un comando vocale

4.1.6 Strutturazione degli Oggetti

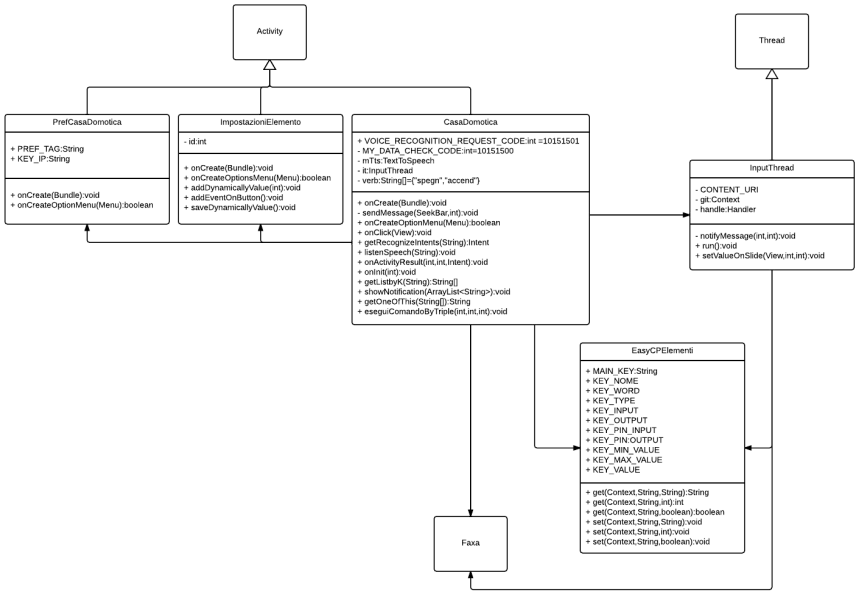

Figura 4.7: Diagramma delle classi

Il precedente diagramma delle classi ci fornisce la conclusione della progettazione di questa applicazione. Si evince dallo schema che vi saranno tre *Activity* principali nel programma:

CasaDomotica, *ImpostazioniElemento*, *PrefCasaDomotica*. Queste Activity permetteranno di operare sui dati nelle varie schermate.

CasaDomotica è connessa con *EasyCPElementi* una classe che si occupa di salvare i dati in uno *SharedPreferences* e di recuperarli.

La classe CasaDomotica lancerà il thread *InputThread* che si occuperà di recuperare i dati dal database SQLite e di aggiornare l'interfaccia attraverso un *Handler*.

4.1.7 Interfaccia utente

Per quanto riguarda l'interfaccia si è deciso di gestire delle schermate minimali, per non spaventare l'utente e rendere immediati i dati da presentare.

L'interfaccia è così pensata

Figura 4.8: Interfaccia grafica CasaDomotica home

In questa interfaccia sono presenti: in alto centrale il pulsante per attivare i comandi vocali, a sinistra i nomi di tutti i dispositivi connessi e a destra il loro stato rappresentato con una barra percentuale.

Trascinando le barre percentuali è possibile variare lo stato del dispositivo mentre premendo il bottone con scritto il nome si entra nel menù per la gestione di quell'entità.

Figura 4.9: Interfaccia grafica CasaDomotica menù gestione entità

In questo menù è invece possibile rinominare l'entità, variare la sua parola chiave e il tipo di entità coinvolta, determinare se si tratta di una periferica di input o di output e a quali pin sono connesse queste due interfacce; infine è possibile rimappare i valori ricevuti con una scala personale.

I valori *KeyWord* e *Tipo* sono coinvolti attivamente nei comandi vocali.

Figura 4.10: Interfaccia grafica CasaDomotica comandi vocali

Quanto si presenta la schermata precedente, è possibile comunicare al dispositivo le proprie esigenze.

Il programma è studiato per riconoscere alcune parole chiave che se presenti nella frase attiveranno le funzioni richieste.

In primo luogo bisogna ricordare che il programma lavora per triple: verbo, complemento di stato in luogo, complemento oggetto.

Il verbo è studiato in modo da permettere molteplici frasi possibili: si è scelta infatti come parola chiave "accen" che può essere pronunciata nelle varie forme di "accendi", "accendere", "accendiamo", "accensione". Questo comando permette l'accensione di un dispositivo, ovvero un segnale di 254. L'altro verbo utilizzato è destinato a spegnere il dispositivo "spegn" è la parola chiave: permette le possibili interpretazioni di "spegnere", "spegnete", "spegniamo", "spegnimento".

Per i complemento di stato in luogo si usa la chiave inserita nel menù delle entità, nel campo *KeyWork*. Alcuni esempi proposti sono, cucina, sala, bagno, etc.

Per il complemento oggetto invece si tiene presente la chiave inserita nel menù delle entità nel campo *Tipo.* Alcuni esempi potrebbero essere le parole luce, lampada, lampadario, termosifone, lavatrice.

Una delle possibili frasi accettate dall'applicazione è: "Accendi la luce in cucina".

Il programma andando a cercare le 3 parole chiave capirà che l'ordine richiesto è inviare il segnale di accensione al pin mappato con chiavi cucina e luce.

Non si è in alcun modo però vincolati a questa singola frase, la frase richiesta può essere molto più ricca proprio come in una normale conversazione: "Potresti accendere la luce giù in cucina" oppure "Mi farebbe comodo vederti accendere la luce nella cucina, perfavore" sono tutte alternative valide.

Questa strategia, sebbene origine di contraddizioni e quindi non accettabile come soluzione definitiva (anche la frase 'non accendere la luce in cucina' sortirà l'effetto di accenderla'), viene utilizzata in quanto è un'immediata soluzione al problema della comunicazione uomo-macchina, e per l'attuale livello di dettaglio dell'applicazione è l'ideale.

Volendo infatti andare più in profondità nel sistema anche la frase "Non accendere la luce in cucina" sortità l'effetto di accenderla, si tratta però di un problema non strettamente legato al programma ma alla mancanza di un sistema per il riconoscimento vocale per Android sufficientemente sofisticato da permettere l'interpretazione dei messaggi in ingresso. Google ha già comunicato che presto sarà disponibile uno strumento (simile a Siri di iOS) che permetterà un riconoscimento vocale più efficace e darà la possibilità a quest'applicazione di migliorarsi ed estendere i propri limiti di comprensione. [**?**]

Il programma quando riceve una richiesta vocale risponde vocalmente con una affermazione di successo nel caso il comando sia stato inviato correttamente.

Se la frase pronunciata dall'utente non dovesse contenere uno dei tre elementi necessari al fine di disambiguare tra le varie possibili alternative, il programma produrrà un segnale vocale chiedendo l'inserimento dei dati mancanti: alla richiesta incompleta "accendi la luce", verrà infatti posta la domanda "dove?" dall'applicazione; se l'utente dovesse chiedere "luce in cucina" l'applicazione risponderà con "non ho capito cosa devo fare", poiché non è stata esplicitata l'azione da intraprendere.

Infine vi è un ultimo menù che permette la gestione delle configurazioni di rete, ovvero l'ip al quale si fa riferimento durante l'invio dei messaggi, questo menù può essere raggunto dalla home.

Figura 4.11: Interfaccia grafica configurazioni di rete

Capitolo 5

Conclusioni e Prospettive

5.1 Conclusioni

Dal mio punto di vista il framework così sviluppato non è un punto d'arrivo ma costituisce una base efficace per sviluppare applicazioni di interoperabilità tra le due piattaforme, l'inizio di qualcosa da sviluppare e perfezionare ulteriormente. E' infatti ancora possibile migliorare il progetto aggiungendo nuove funzionalità al framework e al demone GetItNow e correggendo eventuali errori che dovessero venire scoperti in futuro.

Questo progetto ha però sicuramente messo in luce che esisteva il modo di rendere la comunicazione tra i dispositivi omogenea e aperta ad altre applicazioni.

Il sistema proposto si è dimostrato all'altezza degli obiettivi fissati, dimostrando il suo buon funzionamento nelle numerose prove effettuate. Nonostante ciò ulteriori test futuri saranno necessari per migliorare sempre più il sistema.

Sebbene l'applicazione destinata al testing sia solo un prototipo, si è dimostrata assolutamente efficace per verificare nella sua interezza il framework, mostrando tutte le possibili interazioni tra i due apparecchi.

Sul lato Arduino si sono provati solo circuiti semplici, con led e sensori di piccole dimensioni, ma che permettono di scorgere le reali possibilità del progetto; infatti inserendo un relé è possibile controllare la corrente alternata di un'abitazione reale, con la possibilità di operare attualmente sulla strumentazione elettronica, anche solo semplicemente accendendo o spegnendo la luce in una stanza.

5.2 Prospettive future

Sarebbe interessante valutare insieme ad un team di persone un ontologia rdf piuttosto che un semplice xml per aumentare l'indipendenza della piattaforma e la sua compatibilità con molte altre realtà.

In futuro potrebbe essere anche interessante integrare le librerie per Faxa con quelle di Amarino per fornire un approccio generico al problema che in base alle esigenze attuasse le soluzioni migliori per l'utente finale.

Infine come è già stato detto sarebbe utile introdurre una intelligenza artificiale che dopo aver ricevuto la frase in ingresso deduca le possibili richieste dell'utente.

Bibliografia

[1] Casaleggio Associati, "L'evoluzione di Internet of Things", Febbraio 2011, http://www.slideshare.net/casaleggioassociati/levoluzione-di-internet-of-things, 2012.

[2] Andrew S. Tanenbaum, *Reti di calcolatori*, Pearson, pp. 468. ISBN 978-8871921822, 1 gennaio 2003.

[3] Raspberry Pi Foundation, "Home page", 2012, http://www.raspberrypi.org/, 2012.

[4] The Laundruino, Venerdì 5 agosto 2011, http://blogs.fsfe.org/clemens/2011/08/05/the-arduino-enabled-washing-machine/, 2012.

[5] Autorità per le garanzie nelle comunicazioni AGCOM, "Bilancio di mandato", 2005-2012,

[6] L'altra pagina, "Il mondo ha 6 miliardi di abbonati per servizi del cellulare", 12 Ottobre 2012, *ttp://www.laltrapagina.it/mag/?p=11749*, 2012.

[7] Wikipedia, "Domotica", 2012, http://it.wikipedia.org/wiki/Domotica, 2012

[8] P. Magrassi, *Supranet in "Dizionario dell'economia digitale"*, V. Di Bari, Sole 24Ore Pirola, Milano, 2002.

[9] Wikipedia, "Android", 2012, http://it.wikipedia.org/wiki/Android, 2012.

[10] Businessweek, "Google Buys Android for its Mobile Arsenal", 16 Agosto 2005 *http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal* 2012.

[11] Gartner, "Gartner Says Sales of Mobile Devices Grew 5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percent", 15 novembre 2011, *http://www.gartner.com/it/page.jsp?id=1848514 , 2012.*

[12] Mark Brownlow, "Smartphone Statistic and Market Share", *Ottobre 2012, immaginewww.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm ,* 2012.

[13] VisionMobile, "Open Governance Index", Luglio 2011, `http://www.visionmobile.com/product/open-governance-index/`, 2012.

[14] Massimo Banzi, *BetaBook, il manuale di Arduino: Cap. 3 - Un po' di storia di Arduino.* Apogeo. `http://arduino.apogeolab.it/03-un-po-di-storia-di-arduino/`, 12 luglio 2011.

[15] Justin Lahart, "Taking an Open-Source Approach to Hardware", 27 novembre 2009, *The Wall Street Journal.*

[16] Wikipedia, "Arduino (hardware)", 2012 `http://it.wikipedia.org/wiki/Arduino_(hardware)`, 2012.

[17] Arduino foundation, "Arduino Ethernet", `http://www.arduino.cc/en/Main/ArduinoBoardEthernet`, 2012.

[18] Slovati, "Google sceglie Arduino come piattaforma di sviluppo per Android", 13 maggio 2011, `http://it.emcelettronica.com/google-sceglie-arduino-come-piattaforma-di-sviluppo-android`, 2012.

[19] Bonifaz Kaufmann, 2010, `http://www.amarino-toolkit.net` , 2012.

[20] Bonifaz Kaufmann, "Design and Implementation of a Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing", Alpen-Adria-Universität Klagenfurt, August 2010

[21] BCasa, "Analisi sulla Domotica: In Italia la diffusione dell'automazione della casa è in crescita", maggio 2012 , `http://tinyurl.com/analisi-domotica`, 2012.

[22] Luca Bedogni, Marco Di Felice, "Data Management", 2012, `http://www.cs.unibo.it/projects/android/slides/datamanagement.pdf`, 2012.

[23] Android Developers, "Content Providers", http://developer.android.com/guide/topics/providers/content-providers.html, 2012.

[24] Davide Falagna, "Google, arriva l'intelligenza artificiale", 07 Ottobre 2012, http://www.webnews.it/2012/10/07/google-arriva-lintelligenza-artificiale/, 2012.

Tool utilizzati

- **Eclipse:** Potente IDE utilizzata per sviluppare tutte le applicazioni Java di questo progetto, comprese applicazioni di testing non rilasciate.
- **Android SDK:** kit di sviluppo per poter implementare applicazioni Android.
- **Geany:** Un IDE leggero per sviluppare le librerie Faxa per Arduino.
- **Arduino IDE 1.0.1:** L'IDE per sviluppare applicazioni per Arduino, con il quale ho sviluppato il programma BroadcastPin.
- **LucidChart.com:** Sito web dedicato allo sviluppo di diagrammi standard, tra i quali UML.
- **Gimp:** programma con il quale ho editato la maggior parte delle immagini presenti nel documento.
- **Lyx:** Una IDE per la scrittura in LaTeX, necessario per acelerare la scrittura di questo documento.
- **Fritzing:** Un IDE per disegnare circuiti elettrici, necessario per mostrare le immagini dei componenti e le loro connessioni, presenti in questo documento.

Ringraziamenti

Questa tesi ed il progettono annesso sono stati possibili grazie all'aiuto di alcune persone che hanno speso il loro tempo fornendo i loro consigli e le loro conoscenze.

Ringrazio il mio relatore **Luciano Bononi** per la fiducia riposta in me, per avermi entusiasmato durante le lezioni e per avermi aiutato con disponibilità e attenzione, nonostante i molti impegni.

Ringrazio il mio corelatore **Luca Bedogni** per avermi consigliato e per aver concordato con me le prime parti di questo progetto, per avermi ascoltato parlare più volte di cio che avevo in mente e corretto con preziosi suggerimenti.

Ringrazio il mio correlatore **Marco di Felice** per avermi indirizzato verso questi argomenti e per avermi fornito alcune delle idee che poi hanno visto la luce nel progetto.

Un ringraziamento è d'obbligo a **Claudia Minardi** per avermi corretto la tesi nonostante il lavoro e lo studio.

Grazie a tutti i miei compagni di corso che mi hanno offerto aiuto e consiglio e mi hanno permesso di testare il progetto anche sui loro dispositivi, **Giulio**, **Morg**, **Fede**, **Alain** e **Stefano**.

Grazie a **mia madre** che ha letto la tesi e dato il suo aiuto fin dalle prime correzioni, e a **mio padre** per avermi aperto gli occhi al mondo dell'informatica fin da quando ero molto piccolo.

Grazie ai miei amici **Manuel**, **Simone**, **Eugenio** e **Antonino**, per avermi trascinato fuori di casa anche in quei giorni in cui vedevo tutto grigio e non avermi fatto perdere la speranza.